aikit Documentation

Release 0.2.3-dev

Lionel Massoulard

Aug 20, 2020

Contents:

aikit stands for Artificial Intelligent tool Kit and provides method to facilitate and accelerate the DataScientist job.

The optic is to provide tools to ease the repetitive part of the DataScientist job and so that he/she can focus on modelization. This package is still in alpha and more features will be added.

This library is intended for the user who knows machine learning, knows python its data-science environnement (sklearn, numpy, pandas, . . .) but doesn't want to spend too much time thinking about python technicallities in order to focus more on modelling. The idea is to automatize or at least accelerate parts of the DataScientist job so that he/she can focus on what he/she does best. The more time spend on coding the less time spent on asking the rights questions and solving the problems. It will also help to really use model in production and not just play with them.

The library is usefull if you ever asked yourself that type of questions :

- How do I handle different type of data ?
- I don't remember how to concatenate sparse array and dataframe ?
- How can I retrieve the name of my features now that everything is a numpy array ?
- I'd like to use sklearn but my data is in a DataFrame with strings object and I don't want to use 2 transformers just to encode the categorical features ?
- How do I deal with Data with several types like text, number and categorical data ?
- How can I quickly test models to see what work and what doesn't ?
- . . .

Here a quick summary of what is provided:

- additional sklearn-like transformers to facilitate operations (categories encoding, missing value handling, text encoding, . . .) : *Transformer*
- an extension of sklearn Pipeline that handle generic composition of transformations : *GraphPipeline*
- a framework to automatically test machine learning models : *Auto ML Overview*
- helper functions to accelerate the *day-to-day*
- . . .

CHAPTER 1

Installation

Using pip:

```
pip install aikit
```

In order to use the full functionnalities of aikit you can also install additionnal packages :

- graphviz : to have a nice representation of the graph of models
- lightgbm : to use lightgbm in the auto-ml
- nltk and gensim : to have advanced text encoder
- nltk corpus to clean text

To install everything you can do the following:

```
pip install lightgbm
pip install gensim
pip install nltk
python -m nltk.downloader punkt
python -m nltk.downloader stopwords
conda install graphviz
```

CHAPTER 2

Getting Started

This notebook will show you how to built a complexe pipeline using aikit and how to crossvalidated it

[3]:
```
from aikit.datasets.datasets import load_dataset, DatasetEnum
Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
Xtrain.head(10)
```

[3]:
```
   pclass                                               name     sex   age  \
0       1                           McCarthy, Mr. Timothy J    male  54.0
1       1                                 Fortune, Mr. Mark    male  64.0
2       1                              Sagesser, Mlle. Emma  female  24.0
3       3                     Panula, Master. Urho Abraham    male   2.0
4       1                             Maioni, Miss. Roberta  female  16.0
5       3                               Waelens, Mr. Achille    male  22.0
6       3                            Reed, Mr. James George    male   NaN
7       1  Swift, Mrs. Frederick Joel (Margaret Welles Ba...  female  48.0
8       1     Smith, Mrs. Lucien Philip (Mary Eloise Hughes)  female  18.0
9       1                                Rowe, Mr. Alfred G    male  33.0

   sibsp  parch    ticket      fare        cabin embarked boat   body  \
0      0      0     17463   51.8625          E46        S  NaN  175.0
1      1      4     19950  263.0000  C23 C25 C27        S  NaN    NaN
2      0      0  PC 17477   69.3000          B35        C    9    NaN
3      4      1   3101295   39.6875          NaN        S  NaN    NaN
4      0      0    110152   86.5000          B79        S    8    NaN
5      0      0    345767    9.0000          NaN        S  NaN    NaN
6      0      0    362316    7.2500          NaN        S  NaN    NaN
7      0      0     17466   25.9292          D17        S    8    NaN
8      1      0     13695   60.0000          C31        S    6    NaN
9      0      0    113790   26.5500          NaN        S  NaN  109.0

                     home_dest
0                Dorchester, MA
1                  Winnipeg, MB
2                           NaN
3                           NaN
```

(continues on next page)

(continued from previous page)

```
4                              NaN
5  Antwerp, Belgium / Stanton, OH
6                              NaN
7                    Brooklyn, NY
8                  Huntington, WV
9                          London
```

[4]:
```
y_train[0:10]
```

[4]:
```
array([0, 0, 1, 0, 1, 0, 0, 1, 1, 0], dtype=int64)
```

[13]:
```
from aikit.pipeline import GraphPipeline
from aikit.transformers import ColumnsSelector, NumericalEncoder, NumImputer,␣
↪CountVectorizerWrapper
from sklearn.ensemble import RandomForestClassifier

text_cols     = ["name","ticket"]
non_text_cols = [c for c in Xtrain.columns if c not in text_cols]

gpipeline = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect":CountVectorizerWrapper(analyzer="word",columns_to_use=text_cols),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                      },
              edges = [("sel","enc","imp","rf"),("vect","rf")])

gpipeline.fit(Xtrain,y_train)
gpipeline.graphviz
```

[13]:

[17]:
```
from aikit.cross_validation import cross_validation
from sklearn.model_selection import StratifiedKFold

cv = StratifiedKFold(10, shuffle=True, random_state=123)

cv_res, yhat_proba = cross_validation(gpipeline, Xtrain, y_train,cv=cv, scoring=[
↪"accuracy", "roc_auc", "neg_log_loss"], return_predict=True, method="predict_proba")

cv_res
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started
```

(continues on next page)

(continued from previous page)

```
cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   13.3s finished
```

[17]:

	test_accuracy	test_roc_auc	test_neg_log_loss	train_accuracy	train_roc_auc	train_neg_log_loss	fit_time	score_time	n_test_samples	fold_nb
0	0.980952	0.998095	-0.116852	1.0	1.0	-0.043928	0.841272	0.133642	105	0
1	0.961905	0.969512	-0.484764	1.0	1.0	-0.042055	0.903584	0.126663	105	1
2	0.961905	0.994474	-0.139215	1.0	1.0	-0.041864	0.747745	0.117702	105	2
3	0.990476	1.000000	-0.102579	1.0	1.0	-0.042316	0.735497	0.120638	105	3
4	0.952381	0.994284	-0.130034	1.0	1.0	-0.044032	0.772531	0.119271	105	4
5	0.961905	0.996570	-0.134116	1.0	1.0	-0.041499	0.725558	0.126695	105	5
6	0.971429	0.998476	-0.140661	1.0	1.0	-0.047236	0.761099	0.116698	105	6
7	0.961905	0.995617	-0.155353	1.0	1.0	-0.040947	0.734543	0.111288	105	7
8	0.961538	0.994386	-0.132630	1.0	1.0	-0.041335	0.740471	0.111782	104	8
9	0.980769	0.996903	-0.150282	1.0	1.0	-0.044830	0.749113	0.112777	104	9

[18]:
```
yhat_proba.head(10)
```

[18]:

	0	1
0	1.00	0.00
1	0.88	0.12
2	0.05	0.95
3	0.93	0.07
4	0.07	0.93
5	1.00	0.00
6	1.00	0.00
7	0.03	0.97
8	0.06	0.94

(continues on next page)

(continued from previous page)

```
9  0.98  0.02
```

2.1 Using cross_validation you get in one call :

- both train and test score
- all the metrics
- the probabilities predicted for each observation

CHAPTER 3

Examples

3.1 GraphPipeline getting started

This notebook is here to show a few things that can be done by the package.

It doesn't means that these are the things you should do on that particular dataset.

Let's load titanic dataset to test a few things

[1]:
```
import warnings
warnings.filterwarnings('ignore') # to remove gensim warning
```

[2]:
```
from aikit.datasets.datasets import load_dataset, DatasetEnum
Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
```

[3]:
```
Xtrain.head(20)
```

[3]:
```
    pclass                                                name     sex   age ␣
↪\
0        1                             McCarthy, Mr. Timothy J    male  54.0
1        1                                   Fortune, Mr. Mark    male  64.0
2        1                                 Sagesser, Mlle. Emma  female  24.0
3        3                       Panula, Master. Urho Abraham    male   2.0
4        1                              Maioni, Miss. Roberta  female  16.0
5        3                                Waelens, Mr. Achille    male  22.0
6        3                              Reed, Mr. James George    male   NaN
7        1  Swift, Mrs. Frederick Joel (Margaret Welles Ba...  female  48.0
8        1     Smith, Mrs. Lucien Philip (Mary Eloise Hughes)  female  18.0
9        1                                 Rowe, Mr. Alfred G    male  33.0
10       3                                   Meo, Mr. Alfonzo    male  55.5
11       3                        Abbott, Mr. Rossmore Edward    male  16.0
12       3                                    Elias, Mr. Dibo    male   NaN
13       2                      Reynaldo, Ms. Encarnacion  female  28.0
14       3                                 Khalil, Mr. Betros    male   NaN
```

(continues on next page)

(continued from previous page)

```
15       1                                Daniels, Miss. Sarah  female  33.0
16       3                    Ford, Miss. Robina Maggie 'Ruby'  female   9.0
17       3  Thorneycroft, Mrs. Percival (Florence Kate White)  female   NaN
18       3                                    Lennon, Mr. Denis    male   NaN
19       3                           de Pelsmaeker, Mr. Alfons    male  16.0

    sibsp  parch      ticket      fare        cabin embarked boat   body  \
0       0      0       17463   51.8625          E46        S  NaN  175.0
1       1      4       19950  263.0000  C23 C25 C27        S  NaN    NaN
2       0      0    PC 17477   69.3000          B35        C    9    NaN
3       4      1     3101295   39.6875          NaN        S  NaN    NaN
4       0      0      110152   86.5000          B79        S    8    NaN
5       0      0      345767    9.0000          NaN        S  NaN    NaN
6       0      0      362316    7.2500          NaN        S  NaN    NaN
7       0      0       17466   25.9292          D17        S    8    NaN
8       1      0       13695   60.0000          C31        S    6    NaN
9       0      0      113790   26.5500          NaN        S  NaN  109.0
10      0      0  A.5. 11206    8.0500          NaN        S  NaN  201.0
11      1      1   C.A. 2673   20.2500          NaN        S  NaN  190.0
12      0      0        2674    7.2250          NaN        C  NaN    NaN
13      0      0      230434   13.0000          NaN        S    9    NaN
14      1      0        2660   14.4542          NaN        C  NaN    NaN
15      0      0      113781  151.5500          NaN        S    8    NaN
16      2      2  W./C. 6608   34.3750          NaN        S  NaN    NaN
17      1      0      376564   16.1000          NaN        S   10    NaN
18      1      0      370371   15.5000          NaN        Q  NaN    NaN
19      0      0      345778    9.5000          NaN        S  NaN    NaN

                                    home_dest
0                             Dorchester, MA
1                               Winnipeg, MB
2                                        NaN
3                                        NaN
4                                        NaN
5              Antwerp, Belgium / Stanton, OH
6                                        NaN
7                               Brooklyn, NY
8                             Huntington, WV
9                                     London
10                                       NaN
11                       East Providence, RI
12                                       NaN
13                                     Spain
14                                       NaN
15                                       NaN
16  Rotherfield, Sussex, England Essex Co, MA
17                                       NaN
18                                       NaN
19                                       NaN
```

[4]:
```
y_train[0:20]
```

[4]:
```
array([0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
      dtype=int64)
```

For now let's ignore the Name and Ticket column which should probably be handled as text

[5]:
```
import pandas as pd
from aikit.transformers import TruncatedSVDWrapper, NumImputer,␣
↪CountVectorizerWrapper, NumericalEncoder
from aikit.pipeline import GraphPipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```
Matplotlib won't work
```

[6]:
```
non_text_cols = [c for c in Xtrain.columns if c not in ("ticket","name")] #␣
↪everything that is not text
non_text_cols
```

[6]:
```
['pclass',
 'sex',
 'age',
 'sibsp',
 'parch',
 'fare',
 'cabin',
 'embarked',
 'boat',
 'body',
 'home_dest']
```

[7]:
```
gpipeline = GraphPipeline(models = { "enc":NumericalEncoder(),
                                     "imp":NumImputer(),
                                     "forest":RandomForestClassifier(n_
↪estimators=100)
                                   },
                          edges = [("enc","imp","forest")])

gpipeline.fit(Xtrain.loc[:,non_text_cols],y_train)
gpipeline.graphviz
```

[7]:

3.1.1 Let's do a cross-validation

[8]:
```
from aikit.cross_validation import cross_validation
from sklearn.model_selection import StratifiedKFold
cv = StratifiedKFold(10, random_state=123, shuffle=True)

cv_result = cross_validation(gpipeline, Xtrain.loc[:,non_text_cols], y_train,
↪cv = cv,
                             scoring=["roc_auc","accuracy","neg_log_loss"])
cv_result
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started
```

(continues on next page)

(continued from previous page)

```
cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    3.8s finished
```

[8]:

```
   test_roc_auc  test_accuracy  test_neg_log_loss  train_roc_auc  \
0      0.997332       0.990476          -0.050391       0.999830
1      0.968369       0.961905          -0.723250       0.999986
2      0.983232       0.942857          -0.154483       0.999816
3      1.000000       1.000000          -0.035742       0.999707
4      0.996380       0.961905          -0.088300       0.999802
5      0.991806       0.952381          -0.125793       0.999797
6      1.000000       1.000000          -0.040940       0.999703
7      0.996380       0.980952          -0.088508       0.999842
8      0.992838       0.971154          -0.107017       0.999793
9      0.999613       0.980769          -0.072026       0.999764

   train_accuracy  train_neg_log_loss  fit_time  score_time  n_test_samples ␣
↪\
0        0.995758           -0.029559  0.200567    0.076803             105
1        0.997879           -0.022651  0.192597    0.066856             105
2        0.995758           -0.026256  0.200526    0.069852             105
3        0.995758           -0.030825  0.210022    0.070714             105
4        0.995758           -0.028642  0.199074    0.064868             105
5        0.997879           -0.025816  0.193644    0.070361             105
6        0.995758           -0.029609  0.215009    0.065831             105
7        0.996819           -0.026614  0.184709    0.077791             105
8        0.995763           -0.027610  0.187533    0.063796             104
9        0.995763           -0.028887  0.199505    0.062847             104

   fold_nb
0        0
1        1
2        2
3        3
4        4
5        5
6        6
7        7
8        8
9        9
```

This cross-validate the complete Pipeline. The difference with sklearn function is that : * you can score more than one metric at a time * you retrieve train and test score

[9]:

```
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

[9]:
```
test_roc_auc          0.992595
test_accuracy         0.974240
test_neg_log_loss    -0.148645
dtype: float64
```

We can do the same but selecting the columns directly in the pipeline :

[10]:
```
from aikit.transformers import ColumnsSelector
gpipeline2 = GraphPipeline(models = { "sel":ColumnsSelector(columns_to_
↪use=non_text_cols),
                                      "enc":NumericalEncoder(columns_to_use=
↪"object"),
                                      "imp":NumImputer(),
                                      "forest":RandomForestClassifier(n_
↪estimators=100, random_state=123)
                                    },
                           edges = [("sel","enc","imp","forest")])

gpipeline2.fit(Xtrain,y_train)
gpipeline2.graphviz
```

[10]:

3.1.2 Remark : 'columns_to_use="object"' tells aikit to encode the columns of type object, it will keep the rest untouched

[11]:
```
cv_result = cross_validation(gpipeline2,Xtrain,y_train,cv = cv,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    4.0s finished
```

[11]:
```
test_roc_auc          0.991698
test_accuracy         0.972335
```

(continues on next page)

(continued from previous page)

```
test_neg_log_loss   -0.178280
dtype: float64
```

Now let's see what we can do with the columns we excluded. We could craft features from them, but let's try to use them as text directly.

[12]:
```
text_cols = ["ticket","name"]
vect = CountVectorizerWrapper(analyzer="word", columns_to_use=text_cols)
vect.fit(Xtrain,y_train)
```

[12]:
```
CountVectorizerWrapper(analyzer='word', column_prefix='BAG',
                       columns_to_use=['ticket', 'name'],
                       desired_output_type='SparseArray',
                       drop_unused_columns=True, drop_used_columns=True,
                       max_df=1.0, max_features=None, min_df=1, ngram_
↪range=1,
                       regex_match=False, tfidf=False, vocabulary=None)
```

3.1.3 Remark : aikit CountVectorizer can direcly work on 2 (or more) columns, no need to use a FeatureUnion or something of the sort

[13]:
```
features = vect.get_feature_names()
features[0:20] + ["..."] + features[-20:]
```

[13]:
```
['ticket__BAG__10482',
 'ticket__BAG__110152',
 'ticket__BAG__110413',
 'ticket__BAG__110465',
 'ticket__BAG__110469',
 'ticket__BAG__110489',
 'ticket__BAG__110564',
 'ticket__BAG__110813',
 'ticket__BAG__111163',
 'ticket__BAG__111240',
 'ticket__BAG__111320',
 'ticket__BAG__111361',
 'ticket__BAG__111369',
 'ticket__BAG__111426',
 'ticket__BAG__111427',
 'ticket__BAG__112050',
 'ticket__BAG__112052',
 'ticket__BAG__112053',
 'ticket__BAG__112058',
 'ticket__BAG__11206',
 '...',
 'name__BAG__woolf',
 'name__BAG__woolner',
 'name__BAG__worth',
 'name__BAG__wright',
 'name__BAG__wyckoff',
 'name__BAG__yarred',
 'name__BAG__yasbeck',
 'name__BAG__ylio',
 'name__BAG__yoto',
 'name__BAG__young',
```

(continues on next page)

(continued from previous page)

```
 'name__BAG__youseff',
 'name__BAG__yousif',
 'name__BAG__youssef',
 'name__BAG__yousseff',
 'name__BAG__yrois',
 'name__BAG__zabour',
 'name__BAG__zakarian',
 'name__BAG__zebley',
 'name__BAG__zenni',
 'name__BAG__zillah']
```

The encoder directly encodes the 2 features

[14]:
```
xx_res = vect.transform(Xtrain)
xx_res
```

[14]:
```
<1048x2440 sparse matrix of type '<class 'numpy.int32'>'
        with 5414 stored elements in COOrdinate format>
```

Again let's create a GraphPipeline to cross-validate

[15]:
```
gpipeline3 = GraphPipeline(models = {"vect":CountVectorizerWrapper(analyzer=
↪"word",columns_to_use=text_cols),
                                     "logit":LogisticRegression(solver=
↪"liblinear", random_state=123)},
                           edges=[("vect","logit")])
gpipeline3.fit(Xtrain,y_train)
gpipeline3.graphviz
```

[15]:

[16]:
```
cv_result = cross_validation(gpipeline3, Xtrain,y_train,cv = cv,scoring=[
↪"roc_auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.9s finished
```

[16]:
```
test_roc_auc          0.850918
test_accuracy         0.819679
test_neg_log_loss    -0.451681
dtype: float64
```

We can also try we "bag of char"

[17]:
```
gpipeline4 = GraphPipeline(models = {
        "vect": CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
        "logit": LogisticRegression(solver="liblinear", random_state=123) },␣
↪edges=[("vect","logit")])
gpipeline4.fit(Xtrain,y_train)
gpipeline4.graphviz
```

[17]:

[18]:
```
cv_result = cross_validation(gpipeline4,Xtrain,y_train,cv = cv,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    5.9s finished
```

[18]:
```
test_roc_auc          0.849773
test_accuracy         0.813956
test_neg_log_loss    -0.559254
dtype: float64
```

3.1.4 Now let's use all the columns

[19]:
```
gpipeline5 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
```

(continues on next page)

(continued from previous page)

```
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect":CountVectorizerWrapper(analyzer="word",columns_to_use=text_cols),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
               edges = [("sel","enc","imp","rf"),("vect","rf")])
gpipeline5.fit(Xtrain,y_train)
gpipeline5.graphviz
```

[19]:

This model uses both set of columns: * bag of word * and categorical/numerical features

[20]:
```
cv_result = cross_validation(gpipeline5,Xtrain,y_train,cv = cv,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   11.0s finished
```

[20]:
```
test_roc_auc         0.992779
test_accuracy        0.968507
test_neg_log_loss   -0.173236
dtype: float64
```

We can also use both Bag of Char and Bag of Word

[21]:
```
gpipeline6 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect_char":CountVectorizerWrapper(analyzer="word",columns_to_use=text_
↪cols),
    "vect_word":CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
```

(continues on next page)

(continued from previous page)

```
                edges = [("sel","enc","imp","rf"),("vect_char","rf"),("vect_
↪word","rf")])
gpipeline6.fit(Xtrain,y_train)
gpipeline6.graphviz
```

[21]:

[22]:
```
cv_result = cross_validation(gpipeline6,Xtrain,y_train,cv = cv,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   13.9s finished
```

[22]:
```
test_roc_auc         0.947360
test_accuracy        0.843516
test_neg_log_loss   -0.325666
dtype: float64
```

Maybe we can try SVD to limit dimension of bag of char/word features

[23]:
```
gpipeline7 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect_word":CountVectorizerWrapper(analyzer="word",columns_to_use=text_
↪cols),
    "vect_char":CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
    "svd":TruncatedSVDWrapper(n_components=100, random_state=123),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
                edges = [("sel", "enc","imp","rf"),("vect_word","svd","rf"),(
↪"vect_char","svd","rf")])
gpipeline7.fit(Xtrain,y_train)
gpipeline7.graphviz
```

[23]:

[24]:
```
cv_result = cross_validation(gpipeline7,Xtrain,y_train,cv = 10,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   23.4s finished
```

[24]:
```
test_roc_auc         0.992953
test_accuracy        0.972326
test_neg_log_loss   -0.167037
dtype: float64
```

We can even add 'SVD' columns AND bag of word/char columns

[25]:
```
gpipeline8 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect_word":CountVectorizerWrapper(analyzer="word",columns_to_use=text_
↪cols),
    "vect_char":CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
    "svd":TruncatedSVDWrapper(n_components=100, random_state=123),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
            edges = [("sel","enc","imp","rf"),("vect_word","svd","rf"),(
↪"vect_char","svd","rf"),("vect_word","rf"),("vect_char","rf")])

gpipeline8.graphviz
```

[25]:

[26]:
```
cv_result = cross_validation(gpipeline8,Xtrain,y_train,cv = 10,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   22.1s finished
```

[26]:
```
test_roc_auc          0.941329
test_accuracy         0.834011
test_neg_log_loss    -0.334545
dtype: float64
```

Instead of 'SVD' we can add a layer that filter columns…

[27]:
```
from aikit.transformers import FeaturesSelectorClassifier
```

[28]:
```
gpipeline9 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect_word":CountVectorizerWrapper(analyzer="word",columns_to_use=text_
↪cols),
    "vect_char":CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
    "selector":FeaturesSelectorClassifier(n_components=20),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
               edges = [("sel","enc","imp","rf"),("vect_word","selector","rf
↪"),("vect_char","selector","rf")])

gpipeline9.graphviz
```

[28]:

Retrieve feature importance

Let's use that complicated example to show how to retrieve the feature importance

[29]:
```
gpipeline9.fit(Xtrain, y_train)

df_imp = pd.Series(gpipeline9.models["rf"].feature_importances_,
                   index = gpipeline9.get_input_features_at_node("rf"))
df_imp.sort_values(ascending=False,inplace=True)
df_imp
```

[29]:
```
boat____null__              3.839758e-01
sex__female                 3.816301e-02
name__BAG__mr               3.715979e-02
name__BAG__mr.              3.636483e-02
fare                        3.419880e-02
name__BAG__mr.              3.133609e-02
sex__male                   2.962421e-02
name__BAG__r.               2.910019e-02
name__BAG__s.               2.776609e-02
boat__15                    2.672268e-02
age                         2.643157e-02
name__BAG__s.               2.500470e-02
name__BAG__ mr.             2.249752e-02
boat__13                    1.863079e-02
boat____default__           1.711391e-02
pclass                      1.665125e-02
name__BAG__                 1.597853e-02
sibsp                       1.524516e-02
home_dest____null__         1.015056e-02
boat__7                     9.817018e-03
home_dest____default__      9.534058e-03
boat__C                     9.453317e-03
cabin____null__             8.265959e-03
cabin____default__          7.290940e-03
parch                       7.138940e-03
embarked__S                 6.643220e-03
boat__5                     6.206360e-03
name__BAG__iss.             6.139824e-03
embarked__C                 6.040638e-03
boat__3                     5.547742e-03
name__BAG__(                5.352397e-03
name__BAG__mr               5.260205e-03
body_isnull                 4.829877e-03
name__BAG__ (               4.360392e-03
boat__16                    4.245866e-03
boat__9                     4.224166e-03
boat__D                     4.194419e-03
name__BAG__ss               4.076246e-03
embarked__Q                 4.047912e-03
name__BAG__mrs              3.602001e-03
body                        2.955222e-03
name__BAG__rs               2.899086e-03
name__BAG__rs.              2.869114e-03
age_isnull                  2.859144e-03
boat__14                    2.809765e-03
boat__10                    2.695927e-03
name__BAG__rs.              2.165917e-03
boat__12                    2.103210e-03
name__BAG__mrs.             2.064884e-03
home_dest__New York, NY     1.799501e-03
boat__11                    1.495248e-03
```

(continues on next page)

(continued from previous page)

```
name__BAG__miss            1.054318e-03
name__BAG__mrs             9.616334e-04
boat__4                    9.420111e-04
boat__6                    8.184419e-04
home_dest__London          7.515602e-04
boat__8                    3.679950e-04
fare_isnull                2.438310e-09
dtype: float64
```

[30]:
```
cv_result = cross_validation(gpipeline9,Xtrain,y_train,cv = 10,scoring=["roc_
↪auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   15.0s finished
```

[30]:
```
test_roc_auc          0.994108
test_accuracy         0.973288
test_neg_log_loss    -0.153255
dtype: float64
```

[31]:
```
gpipeline10 = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect_word":CountVectorizerWrapper(analyzer="word",columns_to_use=text_
↪cols),
    "vect_char":CountVectorizerWrapper(analyzer="char",ngram_range=(1,4),
↪columns_to_use=text_cols),
    "svd":TruncatedSVDWrapper(n_components=10),
    "selector":FeaturesSelectorClassifier(n_components=10, random_state=123),
    "rf":RandomForestClassifier(n_estimators=100, random_state=123)
                        },
              edges = [("sel","enc","imp","rf"),
                       ("vect_word","selector","rf"),
```

(continues on next page)

(continued from previous page)

```
                        ("vect_char","selector","rf"),
                        ("vect_word","svd","rf"),
                        ("vect_char","svd","rf")])

gpipeline10.fit(Xtrain,y_train)
gpipeline10.graphviz
```

[31]:

In this model here is what is done : * categorical columns are encoded ('enc') * missing values are filled ('imp') * bag of word and bag of char are created, for the two text features * an SVD is done on those * a selector is called to select most important bag of word/char features * everything is given to a RandomForest

[32]:
```
cv_result = cross_validation(gpipeline10,Xtrain,y_train,cv = 10,scoring=[
↪"roc_auc","accuracy","neg_log_loss"])
cv_result.loc[:,("test_roc_auc","test_accuracy","test_neg_log_loss")].mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent␣
↪workers.
cv 0 started

cv 1 started

cv 2 started

cv 3 started

cv 4 started

cv 5 started

cv 6 started

cv 7 started

cv 8 started

cv 9 started

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   16.9s finished
```

[32]:
```
test_roc_auc          0.994333
test_accuracy         0.975201
test_neg_log_loss    -0.143788
dtype: float64
```

As we saw the GraphPipeline allow flexibility in the creation of models and several choices can be easily tested.

Again, it is not the best possible choices for that database, the example are here to illustrate the capabilities.

Better score could be obtained by adjusting hyper-parameters and/or models/transformers and creating some new features.

[ ]:

3.2 How to load a model from a json

This notebook shows how to save the definition into a json object and reload it to be trained

[1]:
```
import warnings
warnings.filterwarnings('ignore') # to remove gensim warning
```

[2]:
```
from aikit.model_definition import sklearn_model_from_param
```

```
Matplotlib won't work
```

The idea is to be able to define a model by its name and its parameters. The overral syntax is :

(ModelName , {hyperparameters})

3.2.1 Example : this is a RandomForestClassifier

[3]:
```
rf_json = ("RandomForestClassifier",{"n_estimators":100})
rf_json
```

[3]:
```
('RandomForestClassifier', {'n_estimators': 100})
```

... which you can create using 'sklearn_model_from_param'

[4]:
```
rf = sklearn_model_from_param(rf_json)
rf
```

[4]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_
↪nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

The idea is simple :

- sklearn model that klass(**kwargs)
- corresponds to 2-uple : 'klass',kwargs

3.2.2 You can create more complexe model, like GraphPipeline

[5]:
```
json_enc = ("NumericalEncoder",{"columns_to_use": ["BLOCK" + "NUMBERTOKEN",
↪"DATETOKEN","CURRENCYTOKEN"]})
json_vec = ("CountVectorizerWrapper",{"analyzer":"char","ngram_range":(1,4),
↪"columns_to_use":["STRINGLINE"]})
json_rf  = ("RandomForestClassifier",{"n_estimators":500})
json_des = ("GraphPipeline",{"models":{"encoder":json_enc,
                                       "vect":json_vec,
                                       "rf":json_rf},
            "edges":[("encoder","rf"),("vect","rf")]})
json_des
```

[5]:
```
('GraphPipeline',
 {'models': {'encoder': ('NumericalEncoder',
    {'columns_to_use': ['BLOCKNUMBERTOKEN', 'DATETOKEN', 'CURRENCYTOKEN']}),
   'vect': ('CountVectorizerWrapper',
    {'analyzer': 'char',
     'ngram_range': (1, 4),
     'columns_to_use': ['STRINGLINE']}),
   'rf': ('RandomForestClassifier', {'n_estimators': 500})},
  'edges': [('encoder', 'rf'), ('vect', 'rf')]})
```

... and again you can convert it to a real model :

[6]:
```
gpipe = sklearn_model_from_param(json_des)
gpipe
gpipe.graphviz
```

[6]:

[7]:
```
gpipe.models["encoder"]
```

[7]:
```
NumericalEncoder(columns_to_use=['BLOCKNUMBERTOKEN', 'DATETOKEN',
                                 'CURRENCYTOKEN'],
                 desired_output_type='DataFrame', drop_unused_columns=False,
                 drop_used_columns=True, encoding_type='dummy',
                 max_cum_proba=0.95, max_modalities_number=100,
                 max_na_percentage=0.05, min_modalities_number=20,
                 min_nb_observations=10, regex_match=False)
```

[8]:
```
gpipe.models["rf"]
```

[8]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_
↪nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[9]:
```
gpipe.models["vect"]
```

[9]:
```
CountVectorizerWrapper(analyzer='char', column_prefix='BAG',
                       columns_to_use=['STRINGLINE'],
                       desired_output_type='SparseArray',
                       drop_unused_columns=True, drop_used_columns=True,
                       max_df=1.0, max_features=None, min_df=1,
                       ngram_range=(1, 4), regex_match=False, tfidf=False,
                       vocabulary=None)
```

If a model has another has its parameters, like a 'BoxCoxTargetTransformer' or 'Stacker'... it works the same

[10]:
```
json_full = "BoxCoxTargetTransformer",{"model":json_des,"ll":0}
json_full
```

[10]:
```
('BoxCoxTargetTransformer',
 {'model': ('GraphPipeline',
   {'models': {'encoder': ('NumericalEncoder',
      {'columns_to_use': ['BLOCKNUMBERTOKEN', 'DATETOKEN', 'CURRENCYTOKEN']}
↪),
```

(continues on next page)

(continued from previous page)

```
    'vect': ('CountVectorizerWrapper',
     {'analyzer': 'char',
      'ngram_range': (1, 4),
      'columns_to_use': ['STRINGLINE']}),
    'rf': ('RandomForestClassifier', {'n_estimators': 500})},
   'edges': [('encoder', 'rf'), ('vect', 'rf')]}),
  'll': 0})
```

[11]:
```
model = sklearn_model_from_param(json_full)
model
```

[11]:
```
BoxCoxTargetTransformer(ll=0,
                        model=GraphPipeline(edges=[('encoder', 'rf'),
                                                   ('vect', 'rf')],
                                            models={'encoder':␣
↪NumericalEncoder(columns_to_use=['BLOCKNUMBERTOKEN',
                                                                                ␣
↪                  'DATETOKEN',
                                                                                ␣
↪                  'CURRENCYTOKEN'],
                                                                                ␣
↪   desired_output_type='DataFrame',
                                                                                ␣
↪   drop_unused_columns=False,
                                                                                ␣
↪   drop_used_columns=True,
                                                                                ␣
↪   encoding_type='dummy',
                                                                                ␣
↪   max_cum_proba=0.95,
                                                                                ␣
↪   max_modalities_number=100,
                                                                                ␣
↪   max_na_percenta...
                                                                                ␣
↪    random_state=None,
                                                                                ␣
↪    verbose=0,
                                                                                ␣
↪    warm_start=False),
                                                    'vect':␣
↪CountVectorizerWrapper(analyzer='char',
                                                                                ␣
↪      column_prefix='BAG',
                                                                                ␣
↪      columns_to_use=['STRINGLINE'],
                                                                                ␣
↪      desired_output_type='SparseArray',
                                                                                ␣
↪      drop_unused_columns=True,
                                                                                ␣
↪      drop_used_columns=True,
                                                                                ␣
↪      max_df=1.0,
                                                                                ␣
↪      max_features=None,
                                                                                ␣
↪      min_df=1,
```

(continues on next page)

(continued from previous page)

```
↪       ngram_range=(1,
↪                    4),
↪       regex_match=False,
↪       tfidf=False,
↪       vocabulary=None)},
                                          no_concat_nodes=None,
                                          verbose=False))
```

[12]:
```
model.model.graphviz
```

[12]:

[13]:
```
from aikit.model_definition import DICO_NAME_KLASS
```

For it to work : the model should be added within DICO_NAME_KLASS

[14]:
```
DICO_NAME_KLASS
```

[14]:
```
registered klasses :
AgglomerativeClusteringWrapper
BoxCoxTargetTransformer
CdfScaler
Char2VecVectorizer
ColumnsSelector
CountVectorizerWrapper
DBSCANWrapper
ExtraTreesClassifier
ExtraTreesRegressor
FeaturesSelectorClassifier
FeaturesSelectorRegressor
GraphPipeline
KMeansTransformer
KMeansWrapper
LGBMClassifier
LGBMRegressor
Lasso
LogisticRegression
NumImputer
NumericalEncoder
OutSamplerTransformer
PCAWrapper
PassThrough
Pipeline
RandomForestClassifier
RandomForestRegressor
Ridge
StackerClassifier
StackerRegressor
TargetEncoderClassifier
TargetEncoderEntropyClassifier
TargetEncoderRegressor
TextDefaultProcessing
TextDigitAnonymizer
```

(continues on next page)

(continued from previous page)

```
TextNltkProcessing
TruncatedSVDWrapper
Word2VecVectorizer
```

For it to work, each model should be registered

3.3 Auto ML

This notebook will explain the auto-ml capabilities of aikit.

It shows the several things involved. If you just want to run it you should use the automl launcher

Let's start by loading some small data

[1]:
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd

from aikit.datasets.datasets import load_dataset,DatasetEnum
dfX, y, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
dfX.head()
```

[1]:
```
   pclass                            name     sex    age  sibsp  parch  ␣
↪ticket  \
0       1          McCarthy, Mr. Timothy J    male  54.0      0      0     17463
1       1                Fortune, Mr. Mark    male  64.0      1      4     19950
2       1             Sagesser, Mlle. Emma  female  24.0      0      0  PC 17477
3       3  Panula, Master. Urho Abraham    male   2.0      4      1   3101295
4       1            Maioni, Miss. Roberta  female  16.0      0      0    110152

       fare        cabin embarked boat   body       home_dest
0   51.8625          E46        S  NaN  175.0  Dorchester, MA
1  263.0000  C23 C25 C27        S  NaN    NaN    Winnipeg, MB
2   69.3000          B35        C    9    NaN             NaN
3   39.6875          NaN        S  NaN    NaN             NaN
4   86.5000          B79        S    8    NaN             NaN
```

[2]:
```
y[0:5]
```

[2]:
```
array([0, 0, 1, 0, 1])
```

Now let's load what is needed

[4]:
```
from aikit.ml_machine import AutoMlConfig, JobConfig,  MlJobManager,␣
↪MlJobRunner, AutoMlResultReader
from aikit.ml_machine import FolderDataPersister, SavingType,␣
↪AutoMlModelGuider
```

3.3.1 AutoML configuration object

This object will contain all the relevant information about the problem at hand : * it's type : REGRESSION or CLASSIFICATION * the information about the column in the data * the steps that are needed in the processing pipeline (see explanation after) * the models that are to be tested * ...

By default the model will guess everything but everything can be changed if needed

[5]:
```
auto_ml_config = AutoMlConfig(dfX = dfX, y = y, name = "titanic")
auto_ml_config.guess_everything()
auto_ml_config
```

[5]:
```
<aikit.ml_machine.ml_machine.AutoMlConfig object at 0x10c90fa90>
type of problem : CLASSIFICATION
```

type of problem

[6]:
```
auto_ml_config.type_of_problem
```

[6]:
```
'CLASSIFICATION'
```

The config guess that it was a Classification problem

information about columns

[7]:
```
auto_ml_config.columns_informations
```

[7]:
```
OrderedDict([('pclass',
               {'TypeOfVariable': 'NUM', 'HasMissing': False, 'ToKeep': True}
↪),
              ('name',
               {'TypeOfVariable': 'TEXT', 'HasMissing': False, 'ToKeep': True}
↪),
              ('sex',
               {'TypeOfVariable': 'CAT', 'HasMissing': False, 'ToKeep': True}
↪),
              ('age',
               {'TypeOfVariable': 'NUM', 'HasMissing': True, 'ToKeep': True}),
              ('sibsp',
               {'TypeOfVariable': 'NUM', 'HasMissing': False, 'ToKeep': True}
↪),
              ('parch',
               {'TypeOfVariable': 'NUM', 'HasMissing': False, 'ToKeep': True}
↪),
              ('ticket',
               {'TypeOfVariable': 'TEXT', 'HasMissing': False, 'ToKeep': True}
↪),
              ('fare',
               {'TypeOfVariable': 'NUM', 'HasMissing': True, 'ToKeep': True}),
              ('cabin',
               {'TypeOfVariable': 'CAT', 'HasMissing': True, 'ToKeep': True}),
              ('embarked',
               {'TypeOfVariable': 'CAT', 'HasMissing': True, 'ToKeep': True}),
              ('boat',
               {'TypeOfVariable': 'CAT', 'HasMissing': True, 'ToKeep': True}),
              ('body',
               {'TypeOfVariable': 'NUM', 'HasMissing': True, 'ToKeep': True}),
              ('home_dest',
               {'TypeOfVariable': 'CAT', 'HasMissing': True, 'ToKeep': True}
↪)])
```

[8]:
```
pd.DataFrame(auto_ml_config.columns_informations).T
```

[8]:

	HasMissing	ToKeep	TypeOfVariable
pclass	False	True	NUM
name	False	True	TEXT
sex	False	True	CAT
age	True	True	NUM
sibsp	False	True	NUM
parch	False	True	NUM
ticket	False	True	TEXT
fare	True	True	NUM
cabin	True	True	CAT
embarked	True	True	CAT
boat	True	True	CAT
body	True	True	NUM
home_dest	True	True	CAT

For each column in the DataFrame, its type were guess among the three possible values : * NUM : for numerical columns * TEXT : for columns that contains text * CAT : for categorical columns

Remarks: * The difference between TEXT and CAT is based on the number of different modalities * Be careful with categorical value that are encoded into integers (algorithm won't know that it is really a categorical feature)

columns block

[9]:
```
auto_ml_config.columns_block
```

[9]:
```
OrderedDict([('NUM', ['pclass', 'age', 'sibsp', 'parch', 'fare', 'body']),
             ('TEXT', ['name', 'ticket']),
             ('CAT', ['sex', 'cabin', 'embarked', 'boat', 'home_dest'])])
```

the ml machine has the notion of block of columns. For some use-case features naturally falls into blocks. By default the tool will use the type of feature has blocks. But other things can be used.

The ml machine will sometimes try to create a model without a block

needed steps

[10]:
```
auto_ml_config.needed_steps
```

[10]:
```
[{'step': 'TextPreprocessing', 'optional': True},
 {'step': 'TextEncoder', 'optional': False},
 {'step': 'TextDimensionReduction', 'optional': True},
 {'step': 'CategoryEncoder', 'optional': False},
 {'step': 'MissingValueImputer', 'optional': False},
 {'step': 'Scaling', 'optional': True},
 {'step': 'DimensionReduction', 'optional': True},
 {'step': 'FeatureExtraction', 'optional': True},
 {'step': 'FeatureSelection', 'optional': True},
 {'step': 'Model', 'optional': False}]
```

The ml machine will create processing pipeline by assembling different steps. Here are the steps it will use for that use case :

- TextPreprocessing

- TextEncoder : encoding of text into numerical values
- TextDimensionReduction : specific dimension reduction for text based features
- CategoryEncoder : encoder of categorical data
- MissingValueImputer : since there are missing value they need to be filled
- Scaling : step to re-scale features
- DimensionReduction : generic dimension reduction
- FeatureExtraction : create new features
- FeatureSelction : select feature
- Model : the final classification/regression model

models to keep

[11]:
```
auto_ml_config.models_to_keep
```

[11]:
```
[('Model', 'LogisticRegression'),
 ('Model', 'RandomForestClassifier'),
 ('Model', 'ExtraTreesClassifier'),
 ('Model', 'LGBMClassifier'),
 ('FeatureSelection', 'FeaturesSelectorClassifier'),
 ('TextEncoder', 'CountVectorizerWrapper'),
 ('TextEncoder', 'Word2VecVectorizer'),
 ('TextEncoder', 'Char2VecVectorizer'),
 ('TextPreprocessing', 'TextNltkProcessing'),
 ('TextPreprocessing', 'TextDefaultProcessing'),
 ('TextPreprocessing', 'TextDigitAnonymizer'),
 ('CategoryEncoder', 'NumericalEncoder'),
 ('CategoryEncoder', 'TargetEncoderClassifier'),
 ('MissingValueImputer', 'NumImputer'),
 ('DimensionReduction', 'TruncatedSVDWrapper'),
 ('DimensionReduction', 'PCAWrapper'),
 ('TextDimensionReduction', 'TruncatedSVDWrapper'),
 ('DimensionReduction', 'KMeansTransformer'),
 ('Scaling', 'CdfScaler')]
```

This give us the list of models/transformers to test at each steps.

Remarks: * some steps are removed because they have no transformer yet

3.3.2 job configuration

[12]:
```
job_config = JobConfig()
job_config.guess_cv(auto_ml_config = auto_ml_config, n_splits = 10)
job_config.guess_scoring(auto_ml_config = auto_ml_config)
job_config.score_base_line = None
```

[13]:
```
job_config.scoring
```

[13]:
```
['accuracy', 'log_loss_patched', 'avg_roc_auc', 'f1_macro']
```

[14]:
```
job_config.cv
```

[14]:
```
StratifiedKFold(n_splits=10, random_state=123, shuffle=True)
```

[15]:
```
job_config.main_scorer
```

[15]:
```
'accuracy'
```

[16]:
```
job_config.score_base_line
```

The baseline can be setted if we know what a good performance is. It will be used to specify the threshold bellow which we stop crossvalidation in the first fold

This object has the specific configuration for the job to do : * how to cross validate * what scoring/benchmark to use

3.3.3 Data Persister

To synchronize processes and to save values, we need an object to take of that.

This object is a DataPersister, which save everything on disk (Other persister using database might be created)

[ ]:
```
base_folder = # INSERT PATH HERE
data_persister = FolderDataPersister(base_folder = base_folder)
```

3.3.4 controller

[ ]:
```
result_reader = AutoMlResultReader(data_persister)
auto_ml_guider = AutoMlModelGuider(result_reader = result_reader,
                                   job_config = job_config,
                                   metric_transformation="default",
                                   avg_metric=True
                                   )

job_controller = MlJobManager(auto_ml_config = auto_ml_config,
                              job_config = job_config,
                              auto_ml_guider = auto_ml_guider,
                              data_persister = data_persister)
```

the search will be driven by a controller process. This process won't actually train models but it will decide what models should be tried.

Here three object are actually created : * result reader : its job is to read the result of the auto-ml process and aggregate them

- auto_ml_guider : its job is to help the controller guide the seach (using a bayesian technic)
- job_controller : the controller

All those objects need the 'data_persister' object to write/read data

Now the controller can be started using:

job_controller.run() You need to launch in a subprocess

3.3.5 Worker(s)

The last things needed is to create worker(s) that will do the actual cross validation. Those worker will : * listen to the controller * does the cross validation of the models they are told * save result

```
[ ]: job_runner = MlJobRunner(dfX = dfX ,
                              y = y,
                              groups = None,
                              auto_ml_config = auto_ml_config,
                              job_config = job_config,
                              data_persister = data_persister)
```

as before the controller can be started using : job_runner.run()

You need to launcher that in a Subprocess or a Thread

3.3.6 Result Reader

After a few models were tested you can see the result, for that you need the 'result_reader' (which I re-create here for simplicity)

```
[ ]: base_folder = # INSERT path here
     data_persister = FolderDataPersister(base_folder = base_folder)

     result_reader = AutoMlResultReader(data_persister)
```

```
[ ]: df_results = result_reader.load_all_results()
     df_params  = result_reader.load_all_params()
     df_errors  = result_reader.load_all_errors()
```

- df_results : DataFrame with the scoring results
- df_params : DataFrame with the parameters of the complete processing pipeline
- df_errors : DataFrame with the errors

All those DataFrames can be joined using the common 'job_id' column

```
[ ]: df_merged_result = pd.merge( df_params, df_results, how = "inner",on = "job_
     ↪id")
     df_merged_error  = pd.merge( df_params, df_errors , how = "inner",on = "job_
     ↪id")
```

And result can be writted in an Excel file (for example)

```
[ ]: try:
         df_merged_result.to_excel(base_folder + "/result.xlsx",index=False)
     except OSError:
         print("I couldn't save excel file")

     try:
         df_merged_error.to_excel(base_folder + "/result_error.xlsx",index=False)
     except OSError:
         print("I couldn't save excel file")
```

3.3.7 Load a given model

```
[ ]: from aikit.ml_machine import FolderDataPersister, SavingType
     from aikit.model_definition import sklearn_model_from_param

     base_folder = # INSERT path here
     data_persister = FolderDataPersister(base_folder = base_folder)
```

```
[ ]: job_id    = # INSERT job_id here
     job_param = data_persister.read(job_id, path = "job_param", write_type =␣
     ↪SavingType.json)
     job_param
```

```
[ ]: model = sklearn_model_from_param(job_param["model_json"])
     model
```

```
[ ]:
```

3.4 Default Pipeline

This notebook shows how you can use aikit to directly get a default pipeline that you can fit on your data

```
[1]: from aikit.datasets.datasets import load_dataset, DatasetEnum
     Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
```

```
[2]: from aikit.ml_machine import get_default_pipeline
     model = get_default_pipeline(Xtrain, y_train)
     model
```

```
Matplotlib won't work
```

```
C:\HOMEWARE\Anaconda3-Windows-x86_64\lib\site-packages\gensim\utils.py:1197:␣
↪UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
[2]: GraphPipeline(edges=[('ColumnsSelector', 'NumImputer'),
                          ('CountVectorizerWrapper', 'NumImputer'),
                          ('NumericalEncoder', 'NumImputer',
                           'RandomForestClassifier')],
                   models={'ColumnsSelector': ColumnsSelector(columns_to_
     ↪drop=None,
                                                              columns_to_use=[
     ↪'pclass',
                                                                              'age
     ↪',
     ↪'sibsp',
     ↪'parch',
     ↪'fare',
     ↪'body'],
```

(continues on next page)

(continued from previous page)

```
                                                                raise_if_shape_
↪differs=True,
                                                                regex_match=False),
                      'CountVectorizerWrapper'...
                      'RandomForestClassifier':␣
↪RandomForestClassifier(bootstrap=True,
                                                                 class_
↪weight=None,
                                                                ␣
↪criterion='gini',
                                                                 max_
↪depth=None,
                                                                 max_
↪features='auto',
                                                                 max_
↪leaf_nodes=None,
                                                                 min_
↪impurity_decrease=0.0,
                                                                 min_
↪impurity_split=None,
                                                                 min_
↪samples_leaf=1,
                                                                 min_
↪samples_split=2,
                                                                 min_
↪weight_fraction_leaf=0.0,
                                                                 n_
↪estimators=100,
                                                                 n_
↪jobs=None,
                                                                 oob_
↪score=False,
                                                                ␣
↪random_state=123,
                                                                ␣
↪verbose=0,
                                                                 warm_
↪start=False)},
              no_concat_nodes=None, verbose=False)
```

[3]:
```
model.graphviz
```

[3]:

[4]:
```
model.fit(Xtrain, y_train)
```

[4]:
```
GraphPipeline(edges=[('ColumnsSelector', 'NumImputer'),
                     ('CountVectorizerWrapper', 'NumImputer'),
                     ('NumericalEncoder', 'NumImputer',
                      'RandomForestClassifier')],
              models={'ColumnsSelector': ColumnsSelector(columns_to_
↪drop=None,
                                                         columns_to_use=[
↪'pclass',
                                                                         'age
↪',

↪'sibsp',
```

(continues on next page)

(continued from previous page)

```
↪'parch',

↪'fare',

↪'body'],
                                                                raise_if_shape_
↪differs=True,
                                                                regex_match=False),
                     'CountVectorizerWrapper'...
                     'RandomForestClassifier':␣
↪RandomForestClassifier(bootstrap=True,
                                                                              class_
↪weight=None,
                                                                             ␣
↪criterion='gini',
                                                                              max_
↪depth=None,
                                                                              max_
↪features='auto',
                                                                              max_
↪leaf_nodes=None,
                                                                              min_
↪impurity_decrease=0.0,
                                                                              min_
↪impurity_split=None,
                                                                              min_
↪samples_leaf=1,
                                                                              min_
↪samples_split=2,
                                                                              min_
↪weight_fraction_leaf=0.0,
                                                                              n_
↪estimators=100,
                                                                              n_
↪jobs=None,
                                                                              oob_
↪score=False,
                                                                             ␣
↪random_state=123,
                                                                             ␣
↪verbose=0,
                                                                              warm_
↪start=False)},
             no_concat_nodes=None, verbose=False)
```

[ ]:

```
[1]: %load_ext autoreload
     %autoreload 2
```

```
[26]: import warnings
      warnings.filterwarnings('ignore') # to remove gensim warning
```

3.5 Auto clustering

This notebook will explain the auto-clustering capabilities of aikit.

It shows the several things involved. If you just want to run it you should use the automl launcher

Custom random search

[3]:
```
import pandas as pd
from sklearn.datasets import load_iris
```

[4]:
```
iris = load_iris()
```

[5]:
```
X = iris.data
y = iris.target
```

[6]:
```
X = pd.DataFrame(X, columns=iris.feature_names)
X.head()
```

[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[7]:
```
y = pd.DataFrame(y, columns=['label'])
```

[9]:
```
from aikit.ml_machine import AutoMlConfig, JobConfig,  MlJobManager,␣
↪MlJobRunner, AutoMlResultReader
from aikit.ml_machine import FolderDataPersister, SavingType,␣
↪AutoMlModelGuider
```

3.5.1 AutoML configuration object

This object will contain all the relevant information about the problem at hand : * it's type : REGRESSION or CLASSIFICATION * the information about the column in the data * the steps that are needed in the processing pipeline (see explanation after) * the models that are to be tested * ...

By default the model will guess everything but everything can be changed if needed

If y is set to `None`, it will guess that it is a clustering problem.

[10]:
```
from aikit.ml_machine.ml_machine_registration import MODEL_REGISTER
```

[11]:
```
auto_ml_config = AutoMlConfig(dfX=X, y=None, name = "iris")
auto_ml_config.guess_everything()
```

[11]:
```
<aikit.ml_machine.ml_machine.AutoMlConfig object at 0x12eb44630>
type of problem : CLUSTERING
```

[12]:
```
auto_ml_config.type_of_problem
```

[12]:
```
'CLUSTERING'
```

[13]:
```
pd.DataFrame(auto_ml_config.columns_informations).T
```

[13]:
```
                  HasMissing ToKeep TypeOfVariable
sepal length (cm)      False   True            NUM
sepal width (cm)       False   True            NUM
petal length (cm)      False   True            NUM
petal width (cm)       False   True            NUM
```

[14]:
```
auto_ml_config.needed_steps
```

[14]:
```
[{'step': 'Scaling', 'optional': True},
 {'step': 'DimensionReduction', 'optional': True},
 {'step': 'FeatureExtraction', 'optional': True},
 {'step': 'FeatureSelection', 'optional': True},
 {'step': 'Model', 'optional': False}]
```

[15]:
```
auto_ml_config.models_to_keep
```

[15]:
```
[('TextEncoder', 'CountVectorizerWrapper'),
 ('TextEncoder', 'Word2VecVectorizer'),
 ('TextEncoder', 'Char2VecVectorizer'),
 ('TextPreprocessing', 'TextNltkProcessing'),
 ('TextPreprocessing', 'TextDefaultProcessing'),
 ('TextPreprocessing', 'TextDigitAnonymizer'),
 ('CategoryEncoder', 'NumericalEncoder'),
 ('MissingValueImputer', 'NumImputer'),
 ('DimensionReduction', 'TruncatedSVDWrapper'),
 ('DimensionReduction', 'PCAWrapper'),
 ('TextDimensionReduction', 'TruncatedSVDWrapper'),
 ('DimensionReduction', 'KMeansTransformer'),
 ('Scaling', 'CdfScaler'),
 ('Model', 'KMeansWrapper'),
 ('Model', 'AgglomerativeClusteringWrapper'),
 ('Model', 'DBSCANWrapper')]
```

3.5.2 Manual clustering pipeline

[16]:
```
from aikit.pipeline import GraphPipeline
```

[17]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[18]:
```
gpipeline = GraphPipeline(models={"scaler": StandardScaler(),
                                  "kmeans": KMeans(n_clusters=3)},
                          edges=[("scaler", "kmeans")])

gpipeline.fit(X)
```

[18]:
```
GraphPipeline(edges=[('scaler', 'kmeans')],
              models={'kmeans': KMeans(algorithm='auto', copy_x=True,
                                       init='k-means++', max_iter=300,
                                       n_clusters=3, n_init=10, n_jobs=None,
                                       precompute_distances='auto',
                                       random_state=None, tol=0.0001,
```

(continues on next page)

(continued from previous page)

```
                                          verbose=0),
                       'scaler': StandardScaler(copy=True, with_mean=True,
                                                with_std=True)},
                 no_concat_nodes=None, verbose=False)
```

[19]:
```
labels = gpipeline.predict(X)
```

[20]:
```
labels
```

[20]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2], dtype=int32)
```

[27]:
```
from aikit.cross_validation import score_from_params_clustering
cv_result = score_from_params_clustering(gpipeline, X,
                                y=None,
                                scoring=["silhouette", 'calinski_harabaz'],
                                verbose=1)
cv_result
```

[27]:

	test_silhouette	test_calinski_harabaz	fit_time	score_time
0	0.506153	505.957631	0.016606	0.002193

[28]:
```
cv_result = score_from_params_clustering(KMeans(n_clusters=3), X,
                                y=None,
                                scoring=["silhouette", 'calinski_harabaz'],
                                verbose=1)
cv_result
```

[28]:

	test_silhouette	test_calinski_harabaz	fit_time	score_time
0	0.552819	561.627757	0.017693	0.002682

3.5.3 Auto-ML

[ ]:
```
job_config = JobConfig()
job_config.guess_scoring(auto_ml_config = auto_ml_config)

job_config.score_base_line = None
```

[ ]:
```
job_config.scoring
```

[ ]:
```
base_folder = # INSERT PATH HERE
data_persister = FolderDataPersister(base_folder = base_folder)
```

[ ]:
```
result_reader = AutoMlResultReader(data_persister)
auto_ml_guider = AutoMlModelGuider(result_reader = result_reader,
                                   job_config = job_config,
                                   metric_transformation="default",
                                   avg_metric=True
```

(continues on next page)

(continued from previous page)

```
                                        )

job_controller = MlJobManager(auto_ml_config = auto_ml_config,
                              job_config = job_config,
                              auto_ml_guider = auto_ml_guider,
                              data_persister = data_persister)
```

[ ]:
```
job_runner = MlJobRunner(dfX = X ,
                         y = None,
                         groups = None,
                         auto_ml_config = auto_ml_config,
                         job_config = job_config,
                         data_persister = data_persister)
```

[ ]:
```
def my_function(u):
    if u==0:
        job_controller.run()
    if u==1:
        job_runner.run()
```

3.5.4 Carefull : this will stare 2 deamon thread that won't stop until you stop them

from multiprocessing.dummy import Pool as ThreadPool pool = ThreadPool(2) results = pool.map(my_function, [0,1])

3.6 Choice of columns

[1]:
```
from aikit.datasets.datasets import load_dataset, DatasetEnum
Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)

from aikit.transformers import NumericalEncoder
```

```
C:\HOMEWARE\Anaconda3-Windows-x86_64\lib\site-packages\gensim\utils.py:1197:␣
↪UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

[2]:
```
Xtrain
```

[2]:

	pclass	name	sex	age	sibsp	parch	\
0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	
1	1	Fortune, Mr. Mark	male	64.0	1	4	
2	1	Sagesser, Mlle. Emma	female	24.0	0	0	
3	3	Panula, Master. Urho Abraham	male	2.0	4	1	
4	1	Maioni, Miss. Roberta	female	16.0	0	0	
...	...	...	...	...	...	...	
1043	2	Sobey, Mr. Samuel James Hayden	male	25.0	0	0	
1044	1	Ryerson, Master. John Borie	male	13.0	2	2	
1045	2	Lahtinen, Rev. William	male	30.0	1	1	
1046	3	Drazenoic, Mr. Jozef	male	33.0	0	0	
1047	2	Hosono, Mr. Masabumi	male	42.0	0	0	

(continues on next page)

(continued from previous page)

```
          ticket      fare            cabin embarked boat   body  \
0          17463   51.8625              E46        S  NaN  175.0
1          19950  263.0000      C23 C25 C27        S  NaN    NaN
2       PC 17477   69.3000              B35        C    9    NaN
3        3101295   39.6875              NaN        S  NaN    NaN
4         110152   86.5000              B79        S    8    NaN
...          ...       ...              ...      ...  ...    ...
1043  C.A. 29178   13.0000              NaN        S  NaN    NaN
1044    PC 17608  262.3750  B57 B59 B63 B66        C    4    NaN
1045      250651   26.0000              NaN        S  NaN    NaN
1046      349241    7.8958              NaN        C  NaN   51.0
1047      237798   13.0000              NaN        S   10    NaN

                          home_dest
0                     Dorchester, MA
1                       Winnipeg, MB
2                                NaN
3                                NaN
4                                NaN
...                              ...
1043          Cornwall / Houghton, MI
1044  Haverford, PA / Cooperstown, NY
1045                  Minneapolis, MN
1046        Austria Niagara Falls, NY
1047                     Tokyo, Japan

[1048 rows x 13 columns]
```

[3]:
```
encoder = NumericalEncoder(columns_to_use=["sex","home_dest"])
Xencoded = encoder.fit_transform(Xtrain)
Xencoded.head()
```

[3]:
```
   pclass                            name   age  sibsp  parch    ticket  \
0       1       McCarthy, Mr. Timothy J  54.0      0      0     17463
1       1             Fortune, Mr. Mark  64.0      1      4     19950
2       1          Sagesser, Mlle. Emma  24.0      0      0  PC 17477
3       3  Panula, Master. Urho Abraham   2.0      4      1   3101295
4       1         Maioni, Miss. Roberta  16.0      0      0    110152

       fare        cabin embarked boat   body  sex__male  sex__female  \
0   51.8625          E46        S  NaN  175.0          1            0
1  263.0000  C23 C25 C27        S  NaN    NaN          1            0
2   69.3000          B35        C    9    NaN          0            1
3   39.6875          NaN        S  NaN    NaN          1            0
4   86.5000          B79        S    8    NaN          0            1

   home_dest____null__  home_dest__New York, NY  home_dest__London  \
0                    0                        0                  0
1                    0                        0                  0
2                    1                        0                  0
3                    1                        0                  0
4                    1                        0                  0

   home_dest____default__
0                       1
```

(continues on next page)

(continued from previous page)

```
1                        1
2                        0
3                        0
4                        0
```

3.6.1 Called like this the transformer encodes “sex” and “home_dest” and keeps th other columns untouched

[4]:
```
encoder = NumericalEncoder(columns_to_use=["sex","home_dest"], drop_unused_
↪columns=True)
Xencoded = encoder.fit_transform(Xtrain)
print(Xencoded.shape)
Xencoded.head()
```

```
(1048, 6)
```

[4]:
```
   sex__male  sex__female  home_dest____null__  home_dest__New York, NY  \
0          1            0                     0                        0
1          1            0                     0                        0
2          0            1                     1                        0
3          1            0                     1                        0
4          0            1                     1                        0

   home_dest__London  home_dest____default__
0                  0                       1
1                  0                       1
2                  0                       0
3                  0                       0
4                  0                       0
```

Called like this the transformer encodes “sex” and “home_dest” and drop the other columns

[5]:
```
Xtrain["home_dest"].value_counts()
```

[5]:
```
New York, NY                                47
London                                      11
Cornwall / Akron, OH                         9
Winnipeg, MB                                 7
Montreal, PQ                                 7
                                            ..
London / Birmingham                          1
Folkstone, Kent / New York, NY               1
Treherbert, Cardiff, Wales                   1
Devonport, England                           1
Buenos Aires, Argentina / New Jersey, NJ     1
Name: home_dest, Length: 333, dtype: int64
```

Only the most frequent modalities are kept (this can be changed)

[6]:
```
encoder = NumericalEncoder(columns_to_use=["sex","home_dest"],
                           drop_unused_columns=True,
                           min_modalities_number=400)
Xencoded = encoder.fit_transform(Xtrain)
print(Xencoded.shape)
Xencoded.head()
```

```
(1048, 336)
```

[6]:

	sex__male	sex__female	home_dest____null__	home_dest__New York, NY	\
0	1	0	0	0	
1	1	0	0	0	
2	0	1	1	0	
3	1	0	1	0	
4	0	1	1	0	

	home_dest__London	home_dest__Cornwall / Akron, OH	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	home_dest__Winnipeg, MB	home_dest__Montreal, PQ	\
0	0	0	
1	1	0	
2	0	0	
3	0	0	
4	0	0	

	home_dest__Philadelphia, PA	home_dest__Paris, France	...	\
0	0	0	...	
1	0	0	...	
2	0	0	...	
3	0	0	...	
4	0	0	...	

	home_dest__Deer Lodge, MT	home_dest__Bristol, England / New Britain, CT	␣
↪\			
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	home_dest__Holley, NY	home_dest__Bryn Mawr, PA, USA	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	home_dest__Tokyo, Japan	home_dest__Oslo, Norway Cameron, WI	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	

(continues on next page)

(continued from previous page)

```
4                         0                                          0

   home_dest__Cambridge, MA  home_dest__Ireland Brooklyn, NY  \
0                         0                                  0
1                         0                                  0
2                         0                                  0
3                         0                                  0
4                         0                                  0

   home_dest__England  \
0                   0
1                   0
2                   0
3                   0
4                   0

   home_dest__Aughnacliff, Co Longford, Ireland New York, NY
0                                                          0
1                                                          0
2                                                          0
3                                                          0
4                                                          0

[5 rows x 336 columns]
```

If I specify 'min_modalities_number': 400, all the modalities are kept.

I'll start filtering the modalities only if I have more than 400)

[7]:
```
encoder = NumericalEncoder(columns_to_use=["sex","home_dest"], drop_used_
↪columns=False)
Xencoded = encoder.fit_transform(Xtrain)
print(Xencoded.shape)
Xencoded.head()
```

```
(1048, 19)
```

[7]:
```
   pclass                          name     sex    age  sibsp  parch    ␣
↪ticket  \
0       1       McCarthy, Mr. Timothy J    male  54.0      0      0     17463
1       1            Fortune, Mr. Mark    male  64.0      1      4     19950
2       1          Sagesser, Mlle. Emma  female  24.0      0      0  PC 17477
3       3  Panula, Master. Urho Abraham    male   2.0      4      1   3101295
4       1         Maioni, Miss. Roberta  female  16.0      0      0    110152

       fare        cabin embarked boat   body       home_dest  sex__male  \
0   51.8625          E46        S  NaN  175.0  Dorchester, MA          1
1  263.0000  C23 C25 C27        S  NaN    NaN    Winnipeg, MB          1
2   69.3000          B35        C    9    NaN             NaN          0
3   39.6875          NaN        S  NaN    NaN             NaN          1
4   86.5000          B79        S    8    NaN             NaN          0

   sex__female  home_dest____null__  home_dest__New York, NY  \
0            0                    0                        0
1            0                    0                        0
```

(continues on next page)

(continued from previous page)

```
2            1                    1                          0
3            0                    1                          0
4            1                    1                          0

   home_dest__London  home_dest____default__
0                  0                       1
1                  0                       1
2                  0                       0
3                  0                       0
4                  0                       0
```

Called like this the transformer encodes "sex" and "home_dest" but also keep them in the final result

[8]:
```
from aikit.transformers import TruncatedSVDWrapper

X = pd.DataFrame(np.random.randn(100,20), columns=[f"COL_{j}" for j in␣
↪range(20)])

svd = TruncatedSVDWrapper(n_components=2, drop_used_columns=True)
Xencoded = svd.fit_transform(X)
Xencoded.head()
```

[8]:
```
     SVD__0    SVD__1
0  2.075079 -0.858158
1  0.332307 -0.970121
2  2.279417  1.340435
3 -0.563442  0.551599
4 -1.640313 -1.569441
```

[9]:
```
svd = TruncatedSVDWrapper(n_components=2, drop_used_columns=False)
Xencoded = svd.fit_transform(X)
Xencoded.head()
```

[9]:
```
      COL_0     COL_1     COL_2     COL_3     COL_4     COL_5     COL_6  \
0  0.858982 -0.655989 -0.028417 -0.357398  0.569531  0.145816  0.552368
1 -0.187936  0.041684  0.941944  1.898925  0.179125  0.636418  2.050173
2  1.097298 -0.136058 -0.323606 -1.096158 -0.009371 -0.945267  1.455854
3  0.928934  0.269935 -1.274605 -0.287077  0.279328 -0.320871  0.802277
4 -0.714467  1.637883 -0.451313  0.409956  0.565926  0.448906 -0.128214

      COL_7     COL_8     COL_9  ...    COL_12    COL_13    COL_14    COL_15␣
↪ \
0  1.983438  1.092890 -0.453562  ...  0.285189 -0.604234 -1.053623 -0.291745
1  0.229349 -1.910368  0.702720  ... -0.533445 -0.371779 -0.401205  0.231492
2 -0.108160  1.141867 -1.407562  ...  2.310153  2.414735 -0.184708 -1.486121
3 -0.713909 -1.039250  1.227245  ...  0.020298  0.259960 -0.885320  0.014820
4 -0.845320  0.433473 -0.416148  ...  0.758863 -1.702709 -0.000005 -0.293631

     COL_16    COL_17    COL_18    COL_19    SVD__0    SVD__1
0 -1.646335 -0.215531  0.008500  1.100297  2.076530 -0.854836
1 -1.043176  1.842388  0.329271  0.882017  0.346758 -0.951460
2 -0.676003 -0.686621 -0.836830  0.972978  2.330389  1.407472
3  0.268819 -0.432435  1.254164  0.031453 -0.572056  0.539412
4 -0.859405 -0.167067  0.400996 -1.095900 -1.603850 -1.532443
```

(continues on next page)

(continued from previous page)

```
[5 rows x 22 columns]
```

Another example of the usage of 'drop_used_columns' and 'drop_unused_columns': * in the first case (drop_used_columns = True) : only the SVD columns are retrieved * in the second case (drop_used_columns = False) : I retrive the original columns AND the svd columns

[ ]:

3.7 Transformers

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
from aikit.datasets.datasets import load_dataset,DatasetEnum
Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
```

[3]:
```
Xtrain.head(20)
```

[3]:

	pclass	name	sex	age
0	1	McCarthy, Mr. Timothy J	male	54.0
1	1	Fortune, Mr. Mark	male	64.0
2	1	Sagesser, Mlle. Emma	female	24.0
3	3	Panula, Master. Urho Abraham	male	2.0
4	1	Maioni, Miss. Roberta	female	16.0
5	3	Waelens, Mr. Achille	male	22.0
6	3	Reed, Mr. James George	male	NaN
7	1	Swift, Mrs. Frederick Joel (Margaret Welles Ba...	female	48.0
8	1	Smith, Mrs. Lucien Philip (Mary Eloise Hughes)	female	18.0
9	1	Rowe, Mr. Alfred G	male	33.0
10	3	Meo, Mr. Alfonzo	male	55.5
11	3	Abbott, Mr. Rossmore Edward	male	16.0
12	3	Elias, Mr. Dibo	male	NaN
13	2	Reynaldo, Ms. Encarnacion	female	28.0
14	3	Khalil, Mr. Betros	male	NaN
15	1	Daniels, Miss. Sarah	female	33.0
16	3	Ford, Miss. Robina Maggie 'Ruby'	female	9.0
17	3	Thorneycroft, Mrs. Percival (Florence Kate White)	female	NaN
18	3	Lennon, Mr. Denis	male	NaN
19	3	de Pelsmaeker, Mr. Alfons	male	16.0

	sibsp	parch	ticket	fare	cabin	embarked	boat	body
0	0	0	17463	51.8625	E46	S	NaN	175.0
1	1	4	19950	263.0000	C23 C25 C27	S	NaN	NaN
2	0	0	PC 17477	69.3000	B35	C	9	NaN
3	4	1	3101295	39.6875	NaN	S	NaN	NaN
4	0	0	110152	86.5000	B79	S	8	NaN
5	0	0	345767	9.0000	NaN	S	NaN	NaN
6	0	0	362316	7.2500	NaN	S	NaN	NaN
7	0	0	17466	25.9292	D17	S	8	NaN
8	1	0	13695	60.0000	C31	S	6	NaN
9	0	0	113790	26.5500	NaN	S	NaN	109.0

(continues on next page)

(continued from previous page)

```
10      0      0  A.5. 11206    8.0500          NaN        S  NaN  201.0
11      1      1   C.A. 2673   20.2500          NaN        S  NaN  190.0
12      0      0        2674    7.2250          NaN        C  NaN    NaN
13      0      0      230434   13.0000          NaN        S    9    NaN
14      1      0        2660   14.4542          NaN        C  NaN    NaN
15      0      0      113781  151.5500          NaN        S    8    NaN
16      2      2  W./C. 6608   34.3750          NaN        S  NaN    NaN
17      1      0      376564   16.1000          NaN        S   10    NaN
18      1      0      370371   15.5000          NaN        Q  NaN    NaN
19      0      0      345778    9.5000          NaN        S  NaN    NaN

                                    home_dest
0                             Dorchester, MA
1                               Winnipeg, MB
2                                        NaN
3                                        NaN
4                                        NaN
5               Antwerp, Belgium / Stanton, OH
6                                        NaN
7                               Brooklyn, NY
8                             Huntington, WV
9                                     London
10                                       NaN
11                        East Providence, RI
12                                       NaN
13                                     Spain
14                                       NaN
15                                       NaN
16  Rotherfield, Sussex, England Essex Co, MA
17                                       NaN
18                                       NaN
19                                       NaN
```

[4]:
```
columns_block = {"CAT":["sex","embarked","cabin"],
                 "NUM":["pclass","age","sibsp","parch","fare"],
                 "TEXT":["name","ticket"]}
columns_block
```

[4]:
```
{'CAT': ['sex', 'embarked', 'cabin'],
 'NUM': ['pclass', 'age', 'sibsp', 'parch', 'fare'],
 'TEXT': ['name', 'ticket']}
```

3.7.1 Numerical Encoder

[5]:
```
from aikit.transformers import NumericalEncoder

encoder = NumericalEncoder(columns_to_use=columns_block["CAT"] + columns_
↪block["NUM"])
Xtrain_cat = encoder.fit_transform(Xtrain)
```

[6]:
```
Xtrain_cat.head()
```

[6]:
```
                          name    ticket boat   body       home_dest  \
0       McCarthy, Mr. Timothy J     17463  NaN  175.0  Dorchester, MA
1             Fortune, Mr. Mark     19950  NaN    NaN    Winnipeg, MB
```

(continues on next page)

(continued from previous page)

```
2          Sagesser, Mlle. Emma  PC 17477    9    NaN              NaN
3  Panula, Master. Urho Abraham   3101295  NaN    NaN              NaN
4         Maioni, Miss. Roberta    110152    8    NaN              NaN

   sex__male  sex__female  embarked__S  embarked__C  embarked__Q  ...  \
0          1            0            1            0            0  ...
1          1            0            1            0            0  ...
2          0            1            0            1            0  ...
3          1            0            1            0            0  ...
4          0            1            1            0            0  ...

   fare__7.925  fare__7.225  fare__7.25  fare__8.6625  fare__0.0  fare__69.
↪55  \
0            0            0           0             0          0            0
1            0            0           0             0          0            0
2            0            0           0             0          0            0
3            0            0           0             0          0            0
4            0            0           0             0          0            0

   fare__15.5  fare__7.8542  fare__21.0  fare____default__
0           0             0           0                 1
1           0             0           0                 1
2           0             0           0                 1
3           0             0           0                 1
4           0             0           0                 1

[5 rows x 80 columns]
```

[7]:
```
list(Xtrain_cat.columns)
```

[7]:
```
['name',
 'ticket',
 'boat',
 'body',
 'home_dest',
 'sex__male',
 'sex__female',
 'embarked__S',
 'embarked__C',
 'embarked__Q',
 'cabin____null__',
 'cabin____default__',
 'pclass__3',
 'pclass__1',
 'pclass__2',
 'age____null__',
 'age__24.0',
 'age__18.0',
 'age__22.0',
 'age__21.0',
 'age__30.0',
 'age__36.0',
 'age__28.0',
 'age__19.0',
 'age__27.0',
 'age__25.0',
 'age__29.0',
```

(continues on next page)

(continued from previous page)

```
 'age__23.0',
 'age__31.0',
 'age__26.0',
 'age__35.0',
 'age__32.0',
 'age__33.0',
 'age__39.0',
 'age__17.0',
 'age__42.0',
 'age__45.0',
 'age__16.0',
 'age__20.0',
 'age__50.0',
 'age__40.0',
 'age__34.0',
 'age__38.0',
 'age__1.0',
 'age__47.0',
 'age____default__',
 'sibsp__0',
 'sibsp__1',
 'sibsp__2',
 'sibsp__4',
 'sibsp__3',
 'sibsp__8',
 'sibsp__5',
 'parch__0',
 'parch__1',
 'parch__2',
 'parch__5',
 'parch__4',
 'parch__3',
 'parch__9',
 'parch__6',
 'fare__13.0',
 'fare__7.75',
 'fare__8.05',
 'fare__7.8958',
 'fare__26.0',
 'fare__10.5',
 'fare__7.775',
 'fare__7.2292',
 'fare__26.55',
 'fare__7.925',
 'fare__7.225',
 'fare__7.25',
 'fare__8.6625',
 'fare__0.0',
 'fare__69.55',
 'fare__15.5',
 'fare__7.8542',
 'fare__21.0',
 'fare____default__']
```

3.7.2 Target Encoder

[8]:
```
from aikit.transformers import TargetEncoderClassifier

encoder = TargetEncoderClassifier(columns_to_use=columns_block["CAT"] +␣
↪columns_block["NUM"])
Xtrain_cat = encoder.fit_transform(Xtrain, y_train)

Xtrain_cat.head()
```

[8]:
```
                            name    ticket boat   body       home_dest  \
0       McCarthy, Mr. Timothy J     17463  NaN  175.0  Dorchester, MA
1             Fortune, Mr. Mark     19950  NaN    NaN    Winnipeg, MB
2          Sagesser, Mlle. Emma  PC 17477    9    NaN             NaN
3  Panula, Master. Urho Abraham   3101295  NaN    NaN             NaN
4         Maioni, Miss. Roberta    110152    8    NaN             NaN

   sex__target_1  embarked__target_1  cabin__target_1  pclass__target_1  \
0       0.184564            0.341945         0.195652          0.612766
1       0.184564            0.341945         0.597354          0.612766
2       0.746398            0.576720         0.695652          0.612766
3       0.184564            0.341945         0.314483          0.264822
4       0.746398            0.341945         0.391304          0.612766

   age__target_1  sibsp__target_1  parch__target_1  fare__target_1
0       0.547457         0.353941         0.328632        0.185878
1       0.453744         0.524017         0.276773        0.597354
2       0.481185         0.353941         0.328632        0.695652
3       0.463933         0.267929         0.653542        0.166522
4       0.427970         0.353941         0.328632        0.710857
```

3.7.3 CountVectorizer

[9]:
```
from aikit.transformers import CountVectorizerWrapper

encoder = CountVectorizerWrapper(analyzer="char",columns_to_use=columns_
↪block["TEXT"])

Xtrain_enc = encoder.fit_transform(Xtrain)
Xtrain_enc
```

[9]:
```
<1048x62 sparse matrix of type '<class 'numpy.int32'>'
        with 21936 stored elements in COOrdinate format>
```

[10]:
```
encoder.get_feature_names()
```

[10]:
```
['name__BAG__ ',
 "name__BAG__'",
 'name__BAG__(',
 'name__BAG__)',
 'name__BAG__,',
 'name__BAG__-',
 'name__BAG__.',
 'name__BAG__/',
 'name__BAG__a',
 'name__BAG__b',
```

(continues on next page)

(continued from previous page)

```
 'name__BAG__c',
 'name__BAG__d',
 'name__BAG__e',
 'name__BAG__f',
 'name__BAG__g',
 'name__BAG__h',
 'name__BAG__i',
 'name__BAG__j',
 'name__BAG__k',
 'name__BAG__l',
 'name__BAG__m',
 'name__BAG__n',
 'name__BAG__o',
 'name__BAG__p',
 'name__BAG__q',
 'name__BAG__r',
 'name__BAG__s',
 'name__BAG__t',
 'name__BAG__u',
 'name__BAG__v',
 'name__BAG__w',
 'name__BAG__x',
 'name__BAG__y',
 'name__BAG__z',
 'ticket__BAG__ ',
 'ticket__BAG__.',
 'ticket__BAG__/',
 'ticket__BAG__0',
 'ticket__BAG__1',
 'ticket__BAG__2',
 'ticket__BAG__3',
 'ticket__BAG__4',
 'ticket__BAG__5',
 'ticket__BAG__6',
 'ticket__BAG__7',
 'ticket__BAG__8',
 'ticket__BAG__9',
 'ticket__BAG__a',
 'ticket__BAG__c',
 'ticket__BAG__e',
 'ticket__BAG__f',
 'ticket__BAG__h',
 'ticket__BAG__i',
 'ticket__BAG__l',
 'ticket__BAG__n',
 'ticket__BAG__o',
 'ticket__BAG__p',
 'ticket__BAG__q',
 'ticket__BAG__r',
 'ticket__BAG__s',
 'ticket__BAG__t',
 'ticket__BAG__w']
```

3.7.4 Truncated SVD

[11]:
```
from aikit.transformers import TruncatedSVDWrapper
svd = TruncatedSVDWrapper(n_components=0.1)

xx_train_small_svd = svd.fit_transform(Xtrain_enc)
xx_train_small_svd
```

[11]:
```
        SVD__0    SVD__1    SVD__2    SVD__3    SVD__4    SVD__5
0     5.087193 -0.825855 -1.777827  1.079631  1.131004 -1.943098
1     4.329638 -1.584227 -1.592165  0.356413  0.598414 -0.024379
2     5.899164 -0.197573  2.482450 -0.058293 -1.766613 -0.709567
3     7.419224  0.221492 -2.534401  2.470868 -1.714951  0.337166
4     5.771280  1.272598 -0.836011  0.126549  1.036792 -0.689974
...        ...       ...       ...       ...       ...       ...
1043  7.751914 -0.199622  0.934052 -0.541200 -1.632366 -0.380562
1044  7.063821 -1.730135 -1.130117 -2.086049  0.066611 -0.866250
1045  5.595593  1.029825  1.534740  1.149602  1.487406  1.612092
1046  4.625180 -1.261425 -0.873728 -0.566604  1.032036  0.202452
1047  5.020200  1.633941 -2.131337 -0.567731  0.226184 -1.410670

[1048 rows x 6 columns]
```

3.7.5 Column selector

[12]:
```
from aikit.transformers import ColumnsSelector
selector = ColumnsSelector(columns_to_use=columns_block["TEXT"])
Xtrain_subset = selector.fit_transform(Xtrain)
Xtrain_subset.head(10)
```

[12]:
```
                                                 name    ticket
0                             McCarthy, Mr. Timothy J     17463
1                                   Fortune, Mr. Mark     19950
2                                Sagesser, Mlle. Emma  PC 17477
3                          Panula, Master. Urho Abraham   3101295
4                                 Maioni, Miss. Roberta    110152
5                                  Waelens, Mr. Achille    345767
6                                Reed, Mr. James George    362316
7  Swift, Mrs. Frederick Joel (Margaret Welles Ba...     17466
8     Smith, Mrs. Lucien Philip (Mary Eloise Hughes)     13695
9                                    Rowe, Mr. Alfred G    113790
```

[ ]:
```

```

3.8 Stacking

This notebook will show you how to stacks model using aikit

3.8.1 Regression with OutSamplerTransformer

Let's start by creating a simple Regression dataset

[1]:
```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

X, y = make_regression(n_samples=1000, n_features=30, n_informative=10, n_
↪targets=1, random_state=123)

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.1,␣
↪shuffle=True, random_state=123)
```

[2]:
```
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from sklearn.linear_model import Ridge

from aikit.pipeline import GraphPipeline
from aikit.models import OutSamplerTransformer

cv = 10

stacking_model = GraphPipeline(models = {
    "rf"    : OutSamplerTransformer(RandomForestRegressor(random_state=123, n_
↪estimators=10) , cv = cv),
    "lgbm" : OutSamplerTransformer(LGBMRegressor(random_state=123, n_
↪estimators=10)          ,  cv = cv),
    "ridge": OutSamplerTransformer(Ridge(random_state=123)     , cv = cv),
    "blender":Ridge()
    }, edges = [("rf","blender"),("lgbm","blender"),("ridge","blender")])


stacking_model.graphviz
```

```
Matplotlib won't work
```

```
C:\HOMEWARE\Anaconda3-Windows-x86_64\lib\site-packages\gensim\utils.py:1197:␣
↪UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

[2]:

This model behaves like a regular sklearn regressor.

It can be fitted :

[3]:
```
stacking_model.fit(Xtrain, ytrain)
```

[3]:
```
GraphPipeline(edges=[('rf', 'blender'), ('lgbm', 'blender'),
                     ('ridge', 'blender')],
              models={'blender': Ridge(alpha=1.0, copy_X=True,
                                       fit_intercept=True, max_iter=None,
                                       normalize=False, random_state=None,
                                       solver='auto', tol=0.001),
                      'lgbm': OutSamplerTransformer(columns_prefix=None,␣
↪cv=10,
                                                    desired_output_type=None,
                                                    ␣
↪model=LGBMRegressor(boosting_type='gbdt',
                                                                            ␣
↪class_weight=...
                                                                             ␣
↪ n_jobs=None,
                                                                             ␣
↪ oob_score=False,
```

(continues on next page)

(continued from previous page)

```
↪ random_state=123,
↪ verbose=0,
↪ warm_start=False),
                                                    random_state=123),
                      'ridge': OutSamplerTransformer(columns_prefix=None,␣
↪cv=10,
                                                     desired_output_
↪type=None,
                                                     model=Ridge(alpha=1.0,
                                                                 copy_X=True,
                                                                 fit_
↪intercept=True,
                                                                 max_
↪iter=None,
                                                                ␣
↪normalize=False,
                                                                 random_
↪state=123,
                                                                 solver='auto
↪',
                                                                 tol=0.001),
                                                     random_state=123)},
              no_concat_nodes=None, verbose=False)
```

You can predict

[4]:
```
yhat_test = stacking_model.predict(Xtest)
```

[5]:
```
from sklearn.metrics import mean_squared_error
10_000 * mean_squared_error(ytest, yhat_test)
```

[5]:
```
9.978737586369565
```

Let's describe what goes on during the fit:

1. cross_val_predict is called on each of the model => This create *out-of-sample* predictions for each observation
2. the each model is re-fitted on the full data => To be ready when called for a new prediction
3. The blender is given the out-of-sample prediction of the 3 models to fit a final model

The 'OutSamplerTransformer' object implements the logic to create out-of-sample prediction whereas GraphPipeline act to pass transformation from one node to the next(s).

With that logic you can do more complexe things

Let's say you have missing value to fill before feeding the data to the models (Remark : this is not the case here).

You can just add a node at the top of the pipeline:

[6]:
```
from aikit.transformers import NumImputer
```

(continues on next page)

(continued from previous page)

```
stacking_model = GraphPipeline(models = {
    "imp"   : NumImputer(),
    "rf"    : OutSamplerTransformer(RandomForestRegressor(random_state=123, n_
↪estimators=10) , cv = cv),
    "lgbm" : OutSamplerTransformer(LGBMRegressor(random_state=123, n_
↪estimators=10)          ,  cv = cv),
    "ridge": OutSamplerTransformer(Ridge(random_state=123)      , cv = cv),
    "blender":Ridge()
    }, edges = [("imp", "rf","blender"),("imp", "lgbm","blender"),("imp",
↪"ridge","blender")])

stacking_model.graphviz
```

[6]:

Let's say you want to pass to the blender the predictions of the model **along with** the original features.

You can just add another edge :

[7]:
```
from aikit.transformers import NumImputer

stacking_model = GraphPipeline(models = {
    "imp"   : NumImputer(),
    "rf"    : OutSamplerTransformer(RandomForestRegressor(random_state=123, n_
↪estimators=10) , cv = cv),
    "lgbm" : OutSamplerTransformer(LGBMRegressor(random_state=123, n_
↪estimators=10)          ,  cv = cv),
    "ridge": OutSamplerTransformer(Ridge(random_state=123)      , cv = cv),
    "blender":Ridge()
    }, edges = [("imp", "rf","blender"),("imp", "lgbm","blender"),("imp",
↪"ridge","blender"), ("imp", "blender")])

stacking_model.graphviz
```

[7]:

3.8.2 Example on a Classification task

[8]:
```
from aikit.datasets.datasets import load_dataset, DatasetEnum
Xtrain, y_train, _ ,_ , _ = load_dataset(DatasetEnum.titanic)
Xtrain.head(10)
```

[8]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body
0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S	NaN	175.0
1	1	Fortune, Mr. Mark	male	64.0	1	4	19950	263.0000	C23 C25 C27	S	NaN	NaN
2	1	Sagesser, Mlle. Emma	female	24.0	0	0	PC 17477	69.3000	B35	C	9	NaN
3	3	Panula, Master. Urho Abraham	male	2.0	4	1	3101295	39.6875	NaN	S	NaN	NaN
4	1	Maioni, Miss. Roberta	female	16.0	0	0	110152	86.5000	B79	S	8	NaN
5	3	Waelens, Mr. Achille	male	22.0	0	0	345767	9.0000	NaN	S	NaN	NaN
6	3	Reed, Mr. James George	male	NaN	0	0	362316	7.2500	NaN	S	NaN	304.0
7	1	Swift, Mrs. Frederick Joel (Margaret Welles Ba...	female	48.0	0	0	17466	25.9292	D17	S	8	NaN
8	1	Smith, Mrs. Lucien Philip (Mary Eloise Hughes)	female	18.0	1	0	13695	60.0000	C31	S	6	NaN
9	1	Rowe, Mr. Alfred G	male	33.0	0	0	113790	26.5500	NaN	S	NaN	287.0

(continues on next page)

(continued from previous page)

```
1      1      4     19950  263.0000  C23 C25 C27        S  NaN    NaN
2      0      0  PC 17477   69.3000          B35        C    9    NaN
3      4      1   3101295   39.6875          NaN        S  NaN    NaN
4      0      0    110152   86.5000          B79        S    8    NaN
5      0      0    345767    9.0000          NaN        S  NaN    NaN
6      0      0    362316    7.2500          NaN        S  NaN    NaN
7      0      0     17466   25.9292          D17        S    8    NaN
8      1      0     13695   60.0000          C31        S    6    NaN
9      0      0    113790   26.5500          NaN        S  NaN  109.0

                        home_dest
0                  Dorchester, MA
1                    Winnipeg, MB
2                             NaN
3                             NaN
4                             NaN
5  Antwerp, Belgium / Stanton, OH
6                             NaN
7                    Brooklyn, NY
8                  Huntington, WV
9                          London
```

You can also stack models that works on different part of the data, for example :

- a CountVectorizer + Logit model that works on "text-like" columns
- along with a NumericalEncoder + Random Forest RandomForestClassifier for the other columns

[9]:
```
from aikit.transformers import CountVectorizerWrapper, NumericalEncoder,␣
↪NumImputer, ColumnsSelector
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from aikit.models import OutSamplerTransformer
from aikit.pipeline import GraphPipeline

from sklearn.model_selection import StratifiedKFold

cv = StratifiedKFold(10, random_state=123, shuffle=True)


text_cols     = ["name","ticket"]
non_text_cols = [c for c in Xtrain.columns if c not in text_cols]

gpipeline = GraphPipeline(models = {
    "sel":ColumnsSelector(columns_to_use=non_text_cols),
    "enc":NumericalEncoder(columns_to_use="object"),
    "imp":NumImputer(),
    "vect":CountVectorizerWrapper(analyzer="word",columns_to_use=text_cols),
    "rf":OutSamplerTransformer(RandomForestClassifier(n_estimators=100,␣
↪random_state=123),cv=cv),
    "logit":OutSamplerTransformer(LogisticRegression(random_state=123),
↪cv=cv),
    "blender":LogisticRegression(random_state=123)
},
```

(continues on next page)

(continued from previous page)

```
                edges = [("sel","enc","imp","rf", "blender"),("vect","logit",
↪"blender")])

gpipeline.fit(Xtrain,y_train)
gpipeline.graphviz
```

```
C:\HOMEWARE\Anaconda3-Windows-x86_64\lib\site-packages\sklearn\linear_
↪model\logistic.py:432: FutureWarning: Default solver will be changed to
↪'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

[9]:

3.8.3 Probabilities calibration (Platt's scaling)

Using that object you can also re-calibrate your probabilities. This can be done by using a method called 'Platt's scaling' https://en.wikipedia.org/wiki/Platt_scaling/)

Which consist in feeding the probability of one model to a Logistic Regression which will re-scale them.

[10]:
```
rf_rescaled = GraphPipeline(models = {
    "sel"  : ColumnsSelector(columns_to_use=non_text_cols),
    "enc"  : NumericalEncoder(),
    "imp"  : NumImputer(),
    "rf"   : OutSamplerTransformer( RandomForestClassifier(class_weight =
↪"auto"), cv = cv),
    "scaling":LogisticRegression()
    }, edges = [('sel','enc','imp','rf','scaling')]
)
rf_rescaled.graphviz
```

[10]:

3.9 Download notebooks

- Getting Started
- GraphPipeline
- ModelJson
- DefaultPipeline
- AutoML
- Clustering
- NumericalEncoder
- Transformers
- Stacking

CHAPTER 4

GraphPipeline

The GraphPipeline object is an extension of `sklearn.pipeline.Pipeline` but the transformers/models can be chained with any directed graph.

The objects takes as input two arguments:

- models : dictionary of models (each key is the name of a given node, and each corresponding value is the transformer corresponding to that node)
- edges : list of tuples that link the nodes to each other

The created object will behave like a regular sklearn model :

- it can be cloned and thus cross-validated
- it can pickled
- set and get params works
- it can obviously be fitted and returns predictions
- ...

See full example here :

Example:

```
gpipeline = GraphPipeline(models = {"vect" : CountVectorizerWrapper(analyzer="char",
↪ngram_range=(1,4)),
                                    "svd"  : TruncatedSVDWrapper(n_
↪components=400) ,
                                    "logit" : LogisticRegression(class_weight=
↪"balanced")},
                           edges = [("vect","svd","logit")]
                           )
```

This is a model on which we do the following steps:

1. do a Bag of Char CountVectorizer

2. apply an SVD on the result
3. fit a Logistic Regression on the result

The edges parameters just says that there are edges from "vect" to "svd" to "logit". By convention if a tuple is of length more than 2 it is assumed that all consecutive elements forms an edge (same convention as in DOT graph language)

This model could have be built using sklearn Pipeline. If `graphviz` is installed you can get a nice vizualization of the graph using:

```
gpipeline.graphviz
```

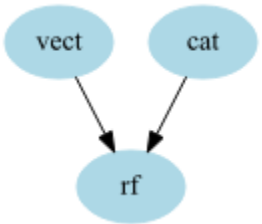
Now the aims of the GraphPipeline object is to be able to handle more complexe chaining of transformers. Let's assume that you have 2 texts columns in your dataset "Text1","Text2" and 2 categorical columns "Cat1","Cat2" and that you want to do the following:

- CountVectorizer on your text
- Categorical Encoder on your categories
- merging the 2 encoded variables and give the result to a `RandomForestClassifier`

You would need to create a GraphPipeline like that:

```
gpipeline = GraphPipeline(models = {"vect" : CountVectorizerWrapper(analyzer="char",
↪ngram_range=(1,4), columns_to_use=["text1","text2"]),
                                    "cat"  : NumericalEncoder(columns_to_use=["cat1",
↪"cat2"]) ,
                                    "rf"   : RandomForestClassifier(n_estimators=100)}
↪  ,
                                edges = [("vect","rf"),("cat","rf")]
                                )
```

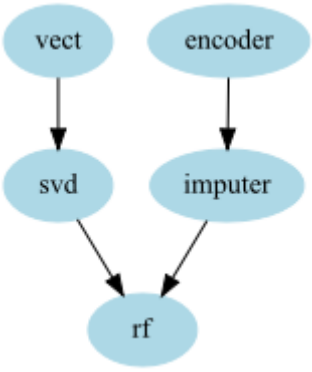
Note that this time, the edges parameters tells that there are 2 edges : "vect" to "rf" and "cat" to "rf"



This particular graph could have been built using a combination of `sklearn.pipeline.Pipeline` and `sklearn.pipeline.FeatureUnion`, however the syntax is easier here.

Let's take a slightly more complexe exemple:

```
gpipeline = GraphPipeline(models = {"encoder" : NumericalEncoder(columns_to_use = [
↪"cat1","cat2"]),
                                    "imputer" : NumImputer(),
                                    "vect"    : CountVectorizerWrapper(analyzer="word
↪",columns_to_use=["cat1","cat2"]),
                                    "svd"     : TruncatedSVDWrapper(n_components=50),
                                    "rf"      : RandomForestClassifier(n_
↪estimators=100)
                                        },
                          edges = [("encoder","imputer","rf"),("vect","svd","rf")] )
```



In that example we have 2 chains of transformers :

- one for the text on which we apply a CountVectorizer and then do an SVD
- one for the categorical variable that we encode and then input the missing values

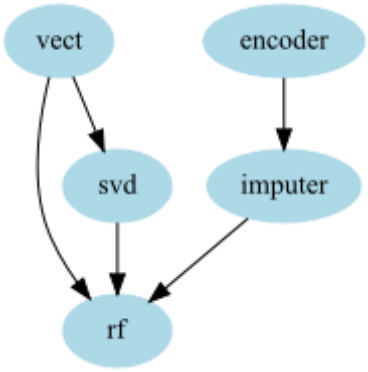
Now let's take one last example of something that couldn't be done using FeatureUnion and Pipeline without computing some transformations twice.

Imagine that you want the RandomForestClassifier to have both the CountVectorizer features and the SVD features, using the Graphpipeline you can do the following model:: You can do that by just adding another edge between "vect" and "rf":

```
gpipeline = GraphPipeline(models = {"encoder" : NumericalEncoder(columns_to_use = [
↪"cat1","cat2"]),
                                    "imputer" : NumImputer(),
                                    "vect"    : CountVectorizerWrapper(analyzer="word
↪",columns_to_use=["cat1","cat2"]),
                                    "svd"     : TruncatedSVDWrapper(n_components=50),
                                    "rf"      : RandomForestClassifier(n_
↪estimators=100)
                                        },
                          edges = [("encoder","imputer","rf"),("vect","rf"),("vect","svd
↪","rf")] )
```

A variation of that would be to include a node to select feature after the CountVectorizer, that way you could :

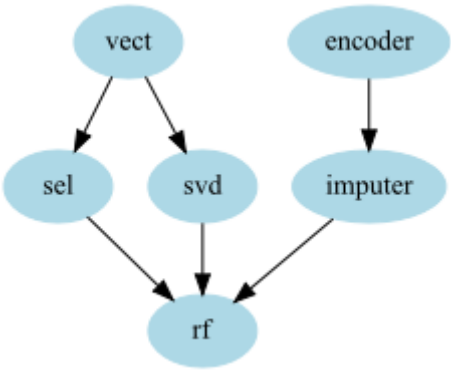
- keep the most important text features *as is*
- retain some of the other information but reduce the dimension via an SVD
- keep the other features



- feed everything to a given model

So code would be:

```
gpipeline = GraphPipeline(models = {"encoder" : NumericalEncoder(columns_to_use = [
↪"cat1","cat2"]),
                                    "imputer" : NumImputer(),
                                    "vect"    : CountVectorizerWrapper(analyzer="word
↪",columns_to_use=["cat1","cat2"]),
                                    "sel"     : FeaturesSelectorClassifier(n_
↪components = 25),
                                    "svd"     : TruncatedSVDWrapper(n_components=50),
                                    "rf"      : RandomForestClassifier(n_
↪estimators=100)
                                        },
                        edges = [("encoder","imputer","rf"),("vect","sel","rf"),("vect
↪","svd","rf")] )
```

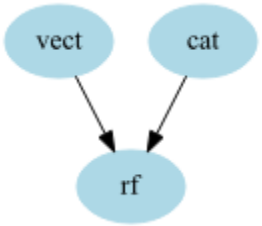


4.1 Merging nodes

The GraphPipeline automatically knows how to merge data of different types. For example the output of a CountVectorizer is most of time a sparse matrix whereas the output of an encoder usually is a DataFrame or a numpy array. This is the case for the pipeline

before applying the RandomForest a concatenation should be made. The GraphPipeline uses the generic concatenation functions `aikit.data_structure_helper.generic_hstack()` to handle that.

Remark : in some cases you don't want the model to handle concatenation, for example because you want to treat the



two (or more) inputs separately yourself in the following transformer. In that case you can add specify the argument *no_concat_nodes* to specify the nodes at which you don't want the GraphPipeline to concatenate. In that case the following node will receive a **dictionnary of transformed data** the keys are the name of the preceding nodes and the values the corresponding transformations.

Remark : the order on which the node appear in the graphviz representation is not necessary the order of the concatenation of the features made within the GraphPipeline. The order of concatenation is actually fixed by the order of the edges : the first node that appear within the edges will be on the *left* of the concatenated data.

4.2 Exposed methods

The GraphPipeline exposed the methods that are available in the final node of the estimator (Pipeline is doing exactly the same)

- if the final estimator has a decision_functions method, the GraphPipeline will be able to call it (applying transformations on each node and then decision_function on the final node)
- if that is not the case an AttributeError will be raised

4.3 Features Names

The GraphPipeline also makes it easier to handle features names. Indeed when using it the names of the features are passed down the pipeline using the "get_feature_names" of each node and whenever possible using the "input_features" argument to tell each node the name of the features in input. That way each node *knows* the name of the feature it has as input and can *give* the names of its features. Those name are used are columns names whenever a DataFrame is retrieved.

You can also retrieve the features at each node if needed (for Example to look at features importances on the last node). The two methods to do that are:

- get_feature_names_at_node : retrieve the name of the features at the **exit** of a given node
- get_input_features_at_node : retrive the name of the features at the **entry** of a given node (this can be called at the last node to know to which features corresponds which feature importance for example)

4.4 Complete Documentation

class aikit.pipeline.**GraphPipeline**(*models*, *edges=None*, *verbose=False*, *no_concat_nodes=None*)

sklearn Transformer that act like a pipeline but on a more generic graph structure

models [dict] dictionnary of models, keys = name of models, values = the models

edges [list of tuple] in each tuple each consecutives elements is an edge

verbose [boolean, default = False] level of verbosity

no_concat_nodes [list or None, default = None] if not None contains, the node on that list will be called with a dictionnary of Data : key = parent node and values = data at precedding node

CHAPTER 5

Auto ML Overview

5.1 Auto ML Launcher

To help to launch the ml machine jobs you can use the `MlMachineLauncher` object.

it :

- contains the configurations of the auto-ml
- has methods to launch a controller, a worker, ... (See after)
- has a method to process command line arguments to quickly create a script that can be used to drive the ml-process (See after)

The easiest way is to create a script like the following one.

Example:

```
from aikit.datasets import load_dataset, DatasetEnum
from aikit.ml_machine import MlMachineLauncher

def loader():
    """ modify this function to load the data

    Returns
    -------
    dfX, y

    Or
    dfX, y, groups

    """
    dfX, y, *_ = load_dataset(DatasetEnum.titanic)
    return dfX, y

def set_configs(launcher):
```

(continues on next page)

(continued from previous page)

```
    """ modify that function to change launcher configuration """

    launcher.job_config.score_base_line = 0.75
    launcher.job_config.allow_approx_cv = True

    return launcher

if __name__ == "__main__":
    launcher = MlMachineLauncher(base_folder = "C:/automl/titanic",
                                 name = "titanic",
                                 loader = loader,
                                 set_configs = set_configs)

    launcher.execute_processed_command_argument()
```

(in what follows we will assume that this is the content of the *"automl_launcher.py"* file)

Here is what is going on:

1. first import the launcher
2. define a loader function : it is better to define a loading function that can be called **if needed** instead of just loading the data (because you don't always need the data)
3. create a launcher, with a base folder and the loading function

4. (Optional) : you can change a few things in the configurations. Here we set the base line to 75% and tell the auto-ml that it can do approximate cross-validation. (See the 'advanced functionnalities' section) To do that pass a function that change the configuration. 5. Process the command argument to actually start a command

Remarks:

- if no change in the default configurations are needed you can use set_configs = None
- the loading function can also return 3 things : dfX, y and groups (if a group cross-validation is needed)
- don't forget the 'if __name__ == "__main__"' part, since the code uses subprocess it is really needed

Having created a script like that you can now use the script to drive the auto-ml process :

- to start a controller and *n* workers
- to aggregate the result
- to separately start a controller
- to separately start worker(s)
- fit a specific model
- ...

5.1.1 what you need to specify ?

For the automl to work you need to specify a few things:

- a loader function

This function will load your data, it should return a DataFrame with features (dfX), the target (y), and optionnaly the groups (if you want to use a GroupedCV) It will be called only once during the initialisation phase. So if you're loading data you don't need to save it a shared folder accessible by all the worker. (After it is called, the auto-ml will persist everything needed)

- a base folder : the folder on which the automl will work.

This folder should be accessible by all the workers and the controller. It will be used to save result, save the queue of jobs, the logs, ...

- set_configs function : a function to modify the settings of the automl

You can modify the cv, the base line, the scoring, ... (See ml_machine_launcher_advanced for details).

5.1.2 run command

This is the main command, it will start everything that is needed. To start the whole process, you should use the 'run' command, in a command windows you can run:

```
python automl_launcher.py run
```

This is the main command, it will

1. load the data using the loader
2. initialize everything
3. modify configuration
4. save everything needed to disk
5. start one controller in a subprocess
6. start one worker

You can also start more than one worker, to do that, the "-n" command should be used:

```
python automl_launcher.py run -n 4
```

This will create a total of 4 workers (and also 1 controller), so at the end you'll have 5 python processes running

5.1.3 manual start

You can also use this script to start everything manually. That way you can

- do the initialization manually
- have one console for the controller
- have separate consoles for workers

To do that you need the same steps as before.

5.1.4 init command

If you only want to initialize everything, you can run the 'init' command:

```
python automl_launcher.py init
```

This won't start anything (no worker, no controller), but will load the data, prepare the configuration and apply the change and persist everything to disk.

5.1.5 manual init

alternatively you can do that manually in a notebook or your favorite IDE. That way you can actually see what the default configuration, prepare the data, etc.

Here is the code to do that:

```
launcher.MlMachineLauncher(base_folder="C:/automl/titanic", loader=loader)
launcher.initialize()
launcher.job_config.base_line = 0.75
launcher.auto_ml_config.columns_informations["Pclass"]["TypeOfVariable"] = "TEXT"

# ... here you can take a look at job_config and auto_ml_config
# ... any other change

launcher.persist()
```

5.1.6 controller command

If you only want to start a controller, you should use the 'controller' command:

```
python automl_launcher.py controller
```

This will start one controller (in the main process)

5.1.7 worker command

If you only want to start worker(s) you should use the 'worker' command:

```
python automl_launcher.py worker -n 2
```

This will start 2 workers (one in main process and one in a subprocess). For it to do anything a controller needs to be started elsewhere. This command is useful to add new workers to an existing task, or to add new worker on another computer (assuming the controller is running elsewhere).

5.1.8 result command

If you want to launch the aggregation of result, you can use the 'result' command:

```
python automl_launcher.py result
```

This will trigger the results aggregations and generate the excel result file

5.1.9 stop command

If you want to stop every process, you can use the 'stop' command:

```
python automl_launcher.py stop
```

It will create the stop file that will trigger the exit of all process listening to that folder

5.1.10 fit command

If you want to fit one or more specific model(s), you can use the 'fit' command. You'll need to specify the job_id(s) to fit:

```
python automl_launcher.py fit --job_ids 77648ab95306e564c4c230e8469e9470
```

Or:

```
python automl_launcher.py fit --job_ids 77648ab95306e564c4c230e8469e9470,
↪469ee473a55a4d1376d3c3186c95f048
```

To fit more that one model. The models will be saved within 'saved_models' along with their json.

5.1.11 Summary

To start a new experiment, first create the script with the example above then use run command.

If you want to split everything you can use

1. launcher.initialize()
2. apply modifications
3. launcher.persist()
4. controller command
5. worker command

Whenever you want an aggregation of results : result command

5.2 Auto Ml Advanced functionnalities

The launcher script can be used to specify lots of things in the ml machine. The auto-ml makes lots of choices by default which can be changed. To change those you need to modify the 'job_config' object or the 'auto_ml_config' object.

Within the launcher, you can use the 'set_configs' function to do just that.

Here is a complete example:

```
def set_configs(launcher):
    """ this is the function that will set the different configurations """
    # Change the CV here :
    launcher.job_config.cv = StratifiedKFold(n_splits=10, shuffle=True, random_
↪state=123) # specify CV

    # Change the scorer to use :
    launcher.job_config.scoring = ['accuracy', 'log_loss_patched', 'avg_roc_auc', 'f1_
↪macro']

    # Change the main scorer (linked with )
    launcher.job_config.main_scorer = 'accuracy'

    # Change the base line (for the main scorer)
    launcher.job_config.score_base_line = 0.8
```

(continues on next page)

(continued from previous page)

```
    # Allow 'approx cv or not :
    launcher.job_config.allow_approx_cv = False

    # Allow 'block search' or not :
    launcher.job_config.do_blocks_search = True

    # Start with default models or not :
    launcher.job_config.start_with_default = True

    # Change default 'columns block' : use for block search
    launcher.auto_ml_config.columns_block = OrderedDict([
          ('NUM', ['pclass', 'age', 'sibsp', 'parch', 'fare', 'body']),
         ('TEXT', ['name', 'ticket']),
         ('CAT', ['sex', 'cabin', 'embarked', 'boat', 'home_dest'])])

    # Change the list of models/transformers to use :
    launcher.auto_ml_config.models_to_keep = [
                #('Model', 'LogisticRegression'),
                ('Model', 'RandomForestClassifier'),
                #('Model', 'ExtraTreesClassifier'),

                # Example : keeping only RandomForestClassifer

                ('FeatureSelection', 'FeaturesSelectorClassifier'),

                ('TextEncoder', 'CountVectorizerWrapper'),

                #('TextPreprocessing', 'TextNltkProcessing'),
                #('TextPreprocessing', 'TextDefaultProcessing'),
                #('TextPreprocessing', 'TextDigitAnonymizer'),

                # => Example: removing TextPreprocessing

                ('CategoryEncoder', 'NumericalEncoder'),
                ('CategoryEncoder', 'TargetEncoderClassifier'),

                ('MissingValueImputer', 'NumImputer'),

                ('DimensionReduction', 'TruncatedSVDWrapper'),
                ('DimensionReduction', 'PCAWrapper'),

                ('TextDimensionReduction', 'TruncatedSVDWrapper'),
                ('DimensionReduction', 'KMeansTransformer'),
                ('Scaling', 'CdfScaler')
                ]

    # Specify the type of problem
    launcher.auto_ml_config.type_of_problem = 'CLASSIFICATION'

    # Specify special hyper parameters : Example
    launcher.auto_ml_config.specific_hyper = {
            ('Model', 'RandomForestClassifier') : {"n_estimators":[10,20]}
            }
    # Example : only test n_estimators to be 10 or 20

    return launcher
```

5.2.1 Job Config Option

The 'job_config' object stores information related to the way we will test the models : like the Cross-Validation object to use, the base-line, ...

1. Cross-Validation

You can change the 'cv' to use. Simply change the 'cv' attribute.

Remark : if you specify 'cv = 10', the type of CV will be guessed (StratifiedKFold, KFold, ...)

Remark : you can use a special 'RandomTrainTestCv' or 'IndexTrainTestCv' object to do only a 'train/test' split and not a full cross-validation.

2. Scorings

Here you can change the scorers to be used. Simplify specify a list with the name of scorers. You can also directly pass sklearn scorers.

3. Main Scorer

This specify the main scorer, it is used to compute the base-line. It can also be used to 'guide' the auto-ml.

4. Approximate CV

If you want to allow that or not. The idea of 'approximate cv' is to gain time by by-passing the CV of some transformers : if a transformers doesn't depend on the target you can reasonably skip cross-validation without having much leakage.

5. Do Block Search

If True, the auto-ml will do special jobs where the model is fixed, the preprocessing pipeline is the default one, but it tries to remove some of the block of columns. It also tries to use only one block. Those jobs helps figure out what block of features are important or not.

6. Starts with default

If True, the auto-ml starts by 'default' models and transformers before doing the full bayesian random search.

5.2.2 Auto Ml Config

The 'auto_ml_config' object stores the information related to the data, the problem, ...

1. Columns Blocks

Using that attribute you can change the blocks of columns, by default the blocks corresponds to the type of variable (Numerical, Categories and Text) but you can specify what you want. Those blocks will be used for the 'block search' jobs.

2. Models to Keep

Here you can filter the models/transformers to test.

Remark : you need to keep required preprocessing steps. For example, if you have text columns you need to keep at least one text encoder.

3. Type of Problem

You can chage the type of problem. This is needed if the guessing was wrong.

4. Specific Hyper Parameters

You can change the hyper parameters used, simply pass a dictionnary with keys being the models to change, and values the new hyper-parameters. The new hyper-parameters can either be a dict (as in the example above) or an object of the HyperCrossProduct class.

5.2.3 Usage of groups

Sometime your data falls into different groups. Sklearn allow you to pass those information to the cross-validation object to make sure the folds respect the groups. Aikit also allow you to use those groups for custom scorer. To use groups in the auto-ml the 'loader' function needs to returns three things instead of two : 'dfX, y, groups'

You can then specify a special CV or a special scorer that uses the groups.

- `Getting Started`
- `GraphPipeline`

5.3 Auto ML Manual Launch

5.3.1 Simple Launch

Here are the steps to launch a test. For simplicity you can also use an `MlMachineLauncher` object (see _ml_machine_launcher)

Load a dataset 'dfX' and its target 'y' and decide a 'name' for the experiment. Then create an `AutoMlConfig` object:

```
from aikit.ml_machine import AutoMlConfig, JobConfig
auto_ml_config = AutoMlConfig(dfX=dfX, y=y, name=name)
auto_ml_config.guess_everything()
```

This object will contain all the configurations of the problem. Then create a `JobConfig` object:

```
job_config = JobConfig()
job_config.guess_cv(auto_ml_config=auto_ml_config, n_splits=10)
job_config.guess_scoring(auto_ml_config=auto_ml_config)
```

The data (dfX and y) can be deleted from the AutoMlConfig after everything is guessed to save memory:

```
auto_ml_config.dfX = None
auto_ml_config.y = None
```

If you have an idea of a base line score you can set it:

```
job_config.score_base_line = 0.95
```

Create a `DataPersister` object (for now only a `FolderDataPersister` object is available but other persisters using database should be possible):

```
from aikit.ml_machine import FolderDataPersister
base_folder = "automl_folder_experiment_1"
data_persister = FolderDataPersister(base_folder=base_folder)
```

Now everything is ready, a `JobController` can be created and started (this controller can use a `AutoMlModelGuider` to help drive the random search):

```
from aikit.ml_machine import AutoMlResultReader, AutoMlModelGuider, MlJobManager,␣
↪MlJobRunner
result_reader = AutoMlResultReader(data_persister)

auto_ml_guider = AutoMlModelGuider(result_reader=result_reader,
```

(continues on next page)

(continued from previous page)

```
                                    job_config=job_config,
                                    metric_transformation="default",
                                    avg_metric=True)

job_controller = MlJobManager(auto_ml_config=auto_ml_config,
                              job_config=job_config,
                              auto_ml_guider=auto_ml_guider,
                              data_persister=data_persister)
```

Same thing one (or more workers) can be created:

```
job_runner = MlJobRunner(dfX=dfX,
                         y=y,
                         auto_ml_config=auto_ml_config,
                         job_config=job_config,
                         data_persister=data_persister)
```

To start the controller just do:

```
job_controller.run()
```

To start the worker just do:

```
job_runner.run()
```

Remark :

- the runner and the worker(s) should be started seperately (for example the controller in a special thread, or in a special process).
- the controller doesn't need the data (dfX and y) and they can be deletted from the `AutoMlConfig` after everything is guessed

5.3.2 Result Aggregation

After a while (whenever you want actually) you can aggregate the results and see them. The most simple way to do that is to aggregate everything into an Excel file. Create an `AutoMlResultReader`:

```
from aikit.ml_machine import AutoMlResultReader, FolderDataPersister

base_folder = "automl_folder_experiment_1"
data_persister = FolderDataPersister(base_folder = base_folder)

result_reader = AutoMlResultReader(data_persister)
```

Then retrieve everything:

```
   df_results = result_reader.load_all_results()
   df_params  = result_reader.load_all_params()
   df_errors  = result_reader.load_all_errors()

* df_results is a DataFrame with all the results (cv scores)
* df_params is a DataFrame with all the parameters of each models
* df_error is a DataFrame with all the errors that arises when fitting the models
```

Everything is linked by the 'job_id' column and can be merged:

```
df_merged_result = pd.merge(df_params, df_results, how="inner", on="job_id")
df_merged_error  = pd.merge(df_params, df_errors , how="inner", on="job_id")
```

And saved into an Excel file:

```
df_merged_result.to_excel(base_folder + "/result.xlsx", index=False)
```

5.3.3 Result File

Here are the parts of the file:

- job_id : idea of the current model (this is id is used everywhere)
- hasblock_** columns : indicate whether or not a given block of column were used
- steps columns indicating which transformer/model were used for each step (Example : "CategoricalEncoder":"NumericalEncoder")
- hyper-parameter columns : indicate the hyperparameters for each model/transformers
- test_** : scoring metrics on testing data (ie: out of fold)
- train_** : scoring metrics on training data (ie: in fold)
- time, time_score : time to fit and score each model
- nb : number of fold (not always max because sometime we don't do the full crossvalidation if performance are not good)

aikit proposes a tool to automatically search among machine learning models and preprocessings to find the best one(s).

To do that the algorithm needs an 'X' DataFrame and a target 'y' and that is to be predicted. The algorithm starts to guess everything that is needed:

- the type of problem (regression, classification)
- the type of each variable (categorical, text or numerical)
- the models/transformers to use
- the scorer to use
- the type of cross-validation to use
- ...

Everything can be overrided by the the user if needed. (See *Auto Ml Advanced functionnalities*)

A folder also needs to be set because everything will transit on disk and be saved in that folder.

Once everything is set a job controller should be launched. Its job will be to create new models to try. Then, one (or more) workers should be launched to actually do the job and test the model.

After a while the result can be seen and a model can be chosen.

The process is more or less the following (see after for detailed)

1. the controller creates a random model (see detailed after)

2. one worker picks up that model and cross validates it
3. the controller picks up the result to help drive the random search
4. after a while the result can be aggregated to choose the model

Everything can be driven via a script.

The easiest way to start the auto ml is to create a script like the following one (and save it as 'automl_launcher.py' for example)

Example:

```
from aikit.datasets import load_dataset, DatasetEnum
from aikit.ml_machine import MlMachineLauncher

def loader():
    """ modify this function to load the data

    Returns
    -------
    dfX, y

    Or
    dfX, y, groups

    """
    dfX, y, *_ = load_dataset(DatasetEnum.titanic)
    return dfX, y

def set_configs(launcher):
    """ modify that function to change launcher configuration """

    launcher.job_config.score_base_line = 0.75
    launcher.job_config.allow_approx_cv = True

    return launcher

if __name__ == "__main__":
    launcher = MlMachineLauncher(base_folder = "C:/automl/titanic",
                                 name = "titanic",
                                 loader = loader,
                                 set_configs = set_configs)

    launcher.execute_processed_command_argument()
```

The only thing to do is to replace the loader function by a function that loads your own data. Once that is done, just run the following command

```
python automl_launcher.py run -n 4
```

It will starts the process using 4 workers (you can change that number if you have more or less processes available).

Here is a diagram that summarize what is going and explained the different functionnalities. For a complete explanation of all the command please look at the *Auto ML Launcher* page.

For a detailed explanation about how the auto-ml is working you can go here:

5.4 Auto ML Inner Workings

Here is a more detailed explanation about what the Ml Machine is doing.

5.4.1 Steps

Each transformation are grouped in steps. There can be several steps needed for each DataFrame, and the needed steps depend on the type of problem/variable. Some steps are optional some are needed.

Example of such steps:

- TextPreprocessing (optional) this steps has all the text preprocessing transformer
- TextEncoder : (needed) this step encodes the text into numerical value (Example using CountVectorizer or Word2Vec)
- MissingValueImputer (needed if some variable have missing values)
- CategorieEncoder (needed if some variables are categorical)
- ...
- Model (needed) : last step consisting of the prediction model

see complete list with `aikit.enums.StepCategories`

The Ml Machine will randomly draw one model per step and merge them into a complex processing pipeline. Optional steps are sometimes drawn and sometime not.

(The transformers that are drawn are the one in the ml machine registry : *Model Register*)

5.4.2 First Rounds

Before randomly selected pipelines, a first round of models are tested using:

- default parameters
- without all the optional steps

Usually those pipelines should perform *relatively* well and gives a good idea about what work and what doesn't.

5.4.3 Next Rounds

Once that is done, random rounds are started. For those, random models are drawn:

- for each step, draw a random transformation (or ignore the step if it is optional)
- for each model draw random hyper-parameters
- if block of variable were setted, randomly draw a subset of those blocks
- merge everythihg into a complexe graph

That model is then *send* to the worker to be cross-validated.

5.4.4 Stopping Threshold

For each given model the worker aims to do a full cross-validation. However the cross-validation can be stopped after the first fold if the result are too low (bellow a threshold fixed by the controller).

That threshold is computed using :

- the base line score if it exists
- a quantile on already done result

(See `aikit.cross_validation.cross_validation()` which is used to compute the cross-validation)

5.4.5 Guided Job

After a few models, with a given random probability, the controller will start to create *Guided Jobs*. Those jobs are not random anymore but uses BayesianOptimization to try to guess a model that will perform correctly.

Concretely a *meta model* is fitted to try to predict performance based on hyper-paramters and transformers/models choices. And instead we use that meta model to predict wheither or not a candidate model will perform or not.

5.4.6 Random Model Generator

The random model generator can be used outside of the Ml Machine:

```
from aikit.ml_machine.ml_machine import RandomModelGenerator
generator = RandomModelGenerator( auto_ml_config = auto_ml_config)

Graph, all_models_params, block_to_use = generator.draw_random_graph()
```

The generator returns three things:

1. Graph : networkx graph of the model
2. all_models_params : dictionnary with all the hyper-parameters of all the transformers/models
3. block_to_use : the block of columns to use

With this 3 objects the json of a model can be created:

```
from aikit.ml_machine.model_graph import convert_graph_to_code
model_json_code = convert_graph_to_code(Graph, all_models_params
```

And then a working model can be created:

```
from aikit.model_definition import sklearn_model_from_param
skmodel = aikit.model_definition.sklearn_model_from_param(model_json_code)
```

You can download and adapt the scripts here

- `Auto Ml Default (Titanic)`
- `Auto Ml Advanced (Titanic)`

CHAPTER 6

Transformer

aikit offers some transformers to help process the data. Here is a brief description of them.

Some of those transformers are just relatively thin wrapper around what exists in sklearn, some are existing techniques packaged as transformers and some things built from scratch.

aikit transformers are built using a Model Wrapper

There is a more detailed explanation about the *aikit.transformers.model_wrapper.ModelWrapper* class. It will explain what the wrapper is doing and how to wrap new models. It will also explain some common functionnalities of all the transformers in aikit.

6.1 Wrapper Goal

The aim of the wrapper is to provide a generic class to handle most of the redondant operations that we might want to apply in a transformer. In particular it aims at making regular 'sklearn like' model more generic and more 'user friendly'.

Here are a few things the wrapper offer to aikit transformers :

- automatic conversion of input/output into a given format (which is useful when chaining models and some on them accepts DataFrame, some don't, . . .)
- verification of type, shape of new data
- shape conversion for model that only accept '1-dimensional' input
- automatic splits and concatenation of result for models that only work one column at a time (See : CountVectorizerWrapper)
- generation of features_names and usage of those names when the output is a DataFrame
- delay the creation of underlying model until the `fit()` is called. This allow to customize hyper-parameters based on the data (Ex : `n_components` can be a percentage of the number of columns given in input).
- automatic or manual selection the columns the transformers is supposed to work on.

Let's take sklearn `sklearn.feature_extraction.text.CountVectorizer` as an example. The transformer has the logic implemented however it can sometimes be a little difficult to use :

- if your data has more than one text column you need more than once CountVectorizer and you need to concatened the result

Indeed CountVectorizer work only on 1 dimensional input (corresponding to a text Serie or a text list)

- if your data is relatively small you might want to retrieve a regular pandas DataFrame and not a scipy.sparse matrix which might not work with your following steps
- you might want to have feature_names that not only correspond to the 'word/char' but also tells from which column it comes from. Example of such column name : 'text1_BAG_dog'
- you might want to tell the CountVectorizer to work on specific columns (so that you don't have to take care of manually splitting your data)

As a consequence it also make the creation of a "sklearn compliant" model (ie : a model that works well within the sklearn infrastructure easy : clone, set_params, hyper-parameters search, ...)

Wrapping the model makes the creation of complexe pipleline like the in *GraphPipeline* a lot easier.

To sum up the aim of the wrapper is to separate :

1. the logic of the transformer
2. the *mechanical* data transformation, checks, ... needed to make the transformer robust and easy to use

6.2 Selection of the columns

The transformers present in aikit are able to select the columns they work on via an hyper-parameter called 'columns_to_use'.

For example:

```
from aikit.transformers import CountVectorizerWrapper vectorizer = CountVectorizerWrapper(columns_to_use=["text1","text2"])
```

the preceding vectorizer will encode "text1" and "text2" using bag-of-word.

The parameter 'columns_to_use' can be of several type :

- list of strings : list of columns by name (assuming a DataFrame input)
- list of integers : list of columns by position (either a numpy array or a DataFrame
- special string "all" : means all the columns are used
- DataTypes.CAT : use aikit detection of columns type to keep only *categorical* columns
- DataTypes.TEXT : use aikit detection of columns type to keep only *textual* columns
- DataTypes.NUM : use aikit detection of columns type to keep only *numerical* columns
- other string like 'object' : use pandas.select_dtype to filter based on type of column

Remark : when a list of string is used the 'use_regex' attribute can be set to true. In that case the 'columns_to_use' are regexes and the columns retrieved are the one that match one of the regexes.

Here are a few examples:

Encoding only one or two columns by name:

```
from aikit.transformers import CountVectorizerWrapper
vectorizer = CountVectorizerWrapper(columns_to_use=["text1","text2"])
```

Encoding only 'TEXT' columns:

```
from aikit.transformers import CountVectorizerWrapper
from aikit.enums import DataTypes
vectorizer = CountVectorizerWrapper(columns_to_use=DataTypes.TEXT)
```

Encoding only 'object' columns:

```
from aikit.transformers import CountVectorizerWrapper
from aikit.enums import DataTypes
vectorizer = CountVectorizerWrapper(columns_to_use="object")
```

6.3 Drop Unused Columns

aikit transformer can also decided what you do with the columns you didn't encode. By default most transformer drop those columns. That way at the end of the transformer you retrieve only the encoded columns.

The behavior is setted by the parameter 'drop_used_columns':

- True : means you have only the encoded 'columns_to_use' result at the end
- False : means you have the encoded 'columns_to_use' + the other columns (un-touched by the transformer)

This can make it easier to transformed part of the data.

Remark : the only transformers that have 'drop_used_columns = False' as default are categorical encoder. That way they automatically encoded the categorical columns but keep the numerical column un-touched. Which means you can plug that at the begginning of your pipeline.

6.4 Drop Used Columns

You can also decided if you want to keep the 'columns_to_use' in their original format (pre-encoding). To do that you need to specify 'drop_used_columns=False'. If you do that you'll have both encoded and non-encoded value after the transformers. This can be usefull sometimes.

For example, let's say that you want to do an SVD but you also want to keep the original columns (so the SVD is not reducing the dimension but adding new compositie features). You can do it like that:

```
from aikit.transformers import TruncatedSVDWrapper
svd = TruncatedSVDWrapper(columns_to_use="all", n_components=5, drop_used_
↪columns=False)
```

6.5 You can wrap your own model

You can use aikit wrapper for your own model, this is useful if you want to code a new transformer but you don't want to think about all the details to make it robust.

See :

6.5.1 How to Wrap a transformer

To wrap a new model you should

1. Create a new class that inherit from ModelWrapper
2. In the __init__ of that class specify the *rules* of the wrapper (see just after)
3. create a _get_model method to specify the underlying transformers

class aikit.transformers.model_wrapper.**ModelWrapper**(*columns_to_use*, *work_on_one_column_only*, *all_columns_at_once*, *accepted_input_types*, *remove_sparse_serie*, *column_prefix*, *desired_output_type*, *must_transform_to_get_features_name*, *dont_change_columns*, *drop_used_columns=True*, *drop_unused_columns=True*, *regex_match=False*)

This is a generic class to help wrapping existing transformer and make them more robust

Parameters

- **columns_to_use** (*None or list of string*) – this parameters will allow the wrapped transformer to select its columns
- **work_on_one_column_only** (*boolean*) – if True tells that the underlying transformer works with 1 dimensinal data (pd.Serie for example)
- **all_columns_at_once** (*boolean*) – if False it tells that the underlying transformer only know how to work one a singular column This is the case for sklearn CountVectorizer for example. If that is the case the wrapped model will work one several column has well (a clone of the underlying model will be create for each column)
- **accepted_input_types** (*list of DataType*) – tells what is accepted by the underlying transformer, a conversion will be made if the input type is not among that list if None nothing is done
- **remove_sparse_serie** (*bool*) – if True will remove Sparse Serie from DataFrame
- **column_prefix** (*str or None*) – if we want the features_names to be prefixed by something like '**SVD_**' or '**BAG_**' (for TruncatedSVD or CountVectorizer)
- **desired_output_type** (*None or DataType*) – specify the desired output type of transformer, a conversion will be made if necesary
- **must_transform_to_get_features_name** (*boolean*) – specify if the transformer should transform its data in order to get its features names. Ideally the underlying transformer should implement a 'get_features_names' method but sometimes the features names are only retrieve from the column of the transformed DataFrame
- **dont_change_columns** (*boolean*) – indicate that the transformer doesn't change the column (for example a StandardScaler) if that is the case you know that the resulting feature are the input feature
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (untransformed) If True, only the transformed columns are in the result

- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result
- **regex_match** (*boolean, default = False*) – if True will use a regex to match columns otherwise exact match

A few notes:

- must_transform_to_get_features_name and dont_change_columns are here to help the wrapped transformers to implement a correct 'get_feature_names'
- the wrapped model has a 'model' attribute that retrieves the underlying transformer(s)
- the wrapped model will generate a NotFittedError error when called without being fit first (this behavior is not consistent across all transformers)

Here is an example of how to wrap sklearn CountVectorizer:

```
class CountVectorizerWrapper(ModelWrapper):
    """ wrapper around sklearn CountVectorizer with additionnal capabilities

     * can select its columns to keep/drop
     * work on more than one columns
     * can return a DataFrame
     * can add a prefix to the name of columns

    """
    def __init__(self,
                 columns_to_use = "all",
                 analyzer = "word",
                 max_df = 1.0,
                 min_df = 1,
                 ngram_range = 1,
                 max_features = None,
                 columns_to_use = None,
                 regex_match    = False,
                 desired_output_type = DataTypes.SparseArray
                 ):

        self.analyzer = analyzer
        self.max_df = max_df
        self.min_df = min_df
        self.ngram_range = ngram_range
        self.columns_to_use = columns_to_use
        self.regex_match    = regex_match
        self.desired_output_type = desired_output_type

        super(CountVectorizerWrapper,self).__init__(
            columns_to_use = columns_to_use,
            regex_match    = regex_match,

            work_on_one_column_only = True,
            all_columns_at_once = False,
            accepted_input_types = (DataTypes.DataFrame,DataTypes.NumpyArray),
            column_prefix = "BAG",
            desired_output_type = desired_output_type,
            must_transform_to_get_features_name = False,
            dont_change_columns = False)
```

(continues on next page)

(continued from previous page)

```
    def _get_model(self,X,y = None):

        if not isinstance(self.ngram_range,(tuple,list)):
            ngram_range = (1,self.ngram_range)
        else:
            ngram_range = self.ngram_range

        ngram_range = tuple(ngram_range)

        return CountVectorizer(analyzer = self.analyzer,
                               max_df = self.max_df,
                               min_df = self.min_df,
                               ngram_range = ngram_range)
```

And here is an example of how to wrap TruncatedSVD to make it work with DataFrame and create columns features:

```
class TruncatedSVDWrapper(ModelWrapper):
    """ wrapper around sklearn TruncatedSVD

    * can select its columns to keep/drop
    * work on more than one columns
    * can return a DataFrame
    * can add a prefix to the name of columns

    n_components can be a float, if that is the case it is considered to be a␣
↪percentage of the total number of columns

    """
    def __init__(self,
                 n_components = 2,
                 columns_to_use = "all",
                 regex_match  = False
                 ):
        self.n_components = n_components
        self.columns_to_use = columns_to_use
        self.regex_match    = regex_match

        super(TruncatedSVDWrapper,self).__init__(
            columns_to_use = columns_to_use,
            regex_match    = regex_match,

            work_on_one_column_only = False,
            all_columns_at_once = True,
            accepted_input_types = None,
            column_prefix = "SVD",
            desired_output_type = DataTypes.DataFrame,
            must_transform_to_get_features_name = True,
            dont_change_columns = False)


    def _get_model(self,X,y = None):

        nbcolumns = _nbcols(X)
        n_components = int_n_components(nbcolumns, self.n_components)

        return TruncatedSVD(n_components = n_components)
```

6.5.2 What append during the fit

To help understand a little more what goes on, here is a brief summary the fit method

1. if 'columns_to_use' is set, creation and fit of a `aikit.transformers.model_wrapper.ColumnsSelector` to subset the column
2. type and shape of input are stored
3. input is converted if it is not among the list of accepted input types
4. input is converted to be 1 or 2 dimensions (also depending on what is accepted by the underlying transformer)
5. underlying transformer is created (using '_get_model') and fitted
6. logic is applied to try to figure out the features names

6.6 Other Transformers

For a full Description of aikit Transformers go there :

6.6.1 All Transformers

Here is a list of some of aikit transformers

Text Transformer

TextDigitAnonymizer

class aikit.transformers.text.**TextDigitAnonymizer**(*concat=False*)
Text transformer to anonymize digits.

TextNltkProcessing

This is another text pre-processing transformers that does classical text transformations.

class aikit.transformers.text.**TextNltkProcessing**(*lower=True*, *digit_anonymize=True*, *digit_character='#'*, *remove_non_words=True*, *remove_stopwords=True*, *stem=True*, *concat=False*)
Text transformer using NLKT. It can perform the following steps:

- put the text in lower case
- anonymize digits
- tokenize the words
- remove every words that doesn't contain any letter
- remove stopwords
- stem the rest

Parameters

- **lower** (*boolean, default = True*) – if True will put the string in lowercase
- **digit_anonymize** (*boolean, default = True*) – if True will anonymize digits, replacing them with 'digit_character'
- **digit_character** (*string, default = '#'*) – character to use to replace digits
- **remove_non_words** (*boolean, default = True*) – if True will remove tokens that are not sequences of letters (aka word)

remove_stopwords [boolean, default = True] if True will remove word that are stop words

stem [boolean, default = True] if True will perform stemming

Example

```
>>> texts = ["A stemmer for English operating on the stem cat should␣
↪identify such strings as cats, catlike, and catty",
"A stemming algorithm might also reduce the words fishing, fished, and␣
↪fisher to the stem fish"]
>>> transformer = TextNltkProcessing()
>>> transformer.fit_transform(texts)
>>> ['stemmer english oper stem cat identifi string cat catlik catti',
     'stem algorithm might also reduc word fish fish fisher stem fish']
```

CountVectorizerWrapper

Wrapper around sklearn `CountVectorizer`.

class aikit.transformers.text.**CountVectorizerWrapper**(*analyzer='word'*, *max_df=1.0*, *min_df=1*, *ngram_range=1*, *max_features=None*, *vocabulary=None*, *tfidf=False*, *columns_to_use='all'*, *regex_match=False*, *desired_output_type='SparseArray'*, *column_prefix='BAG'*, *drop_used_columns=True*, *drop_unused_columns=True*, ***other_count_vectorizer_arguments*)

Wrapper around sklearn `CountVectorizer` with additional capabilities:

- can select its columns to keep/drop
- work on more than one columns
- can return a DataFrame

- can add a prefix to the name of columns

  Parameters

  - **sklearn.CountVectorizer for complete list** (*See*) –
  - **analyzer** (*str, default = "word"*) – type of analyzer ("char","word","char wb")

max_df [float in range [0.0, 1.0] or int, default=1.0] When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_df [float in range [0.0, 1.0] or int, default=1] When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

ngram_range [tuple (min_n, max_n) or int (1, ngram_range)] The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.

max_features [int or None, default=None] If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.

This parameter is ignored if vocabulary is not None.

vocabulary [Mapping or iterable, optional] Either a Mapping (e.g., a dict) where keys are terms and values are indices in the feature matrix, or an iterable over terms. If not given, a vocabulary is determined from the input documents. Indices in the mapping should not be repeated and should not have any gap between 0 and the largest index.

tfidf [boolean, default = False] if True will use a TfIdfVectorizer, otherwise regular CountVectorizer

columns_to_use [None or list of string] this parameter will allow the wrapped transformer to select its columns

regex_match [boolean, default = False] if True will use a regex to match columns otherwise exact match

column_prefix [str or None, default = "BAG"] prefix of the column

drop_used_columns [boolean, default=True] what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result

drop_unused_columns: boolean, default=True what to do with the column that were not used. if False, will drop them if True, will keep them in the result

desired_output_type [None or DataType] specify the desired output type of transformer, a conversion will be made if necesary

Word2VecVectorizer

This model does *Continuous bag of word*, meaning that it will fit a Word2Vec model and then average the word vectors of each text.

class aikit.transformers.text.**Word2VecVectorizer**(*size=100*, *window=5*, *min_count=5*, *text_preprocess='default'*, *same_embedding_all_columns=True*, *use_fast_text=False*, *random_state=None*, *other_params=None*, *columns_to_use='all'*, *desired_output_type='DataFrame'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*)

Word2Vec vectorizer, this model does an average of the embedding of each word :

it is sometimes called 'Continuous Bag of Word'

size [int, default = 100] the size of the embedding

window [int, default = 5] the size of the training window of the word2vec model

text_preprocess: string, default = 'default' type of text preprocessing to use, possible choices are : * 'default' : TextDefaultProcessing : put everything in lower case and remove some punctuation * 'digit' : TextDigitAnonymizer : anonymize digits * 'nltk' : TextNltkProcessing : lower, stemming, remove stopwords, ... * None : do nothing

same_embedding_all_columns [boolean, default = True] if True will fit ONE embedding for ALL the text columns, otherwise will fit one word2vec PER text column

use_fast_text [boolean, default = False] if True will use fasttext instead of gensim

random_state [None or int] state of random generator

other_params [dict or None, default = None]

if not None, additional parameters to be passed to the word2vec model

columns_to_use [list,] columns to encode

desired_output_type [data type, default = DataFrame] desired output type

drop_used_columns [boolean, default=True] what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result

drop_unused_columns: boolean, default=True what to do with the column that were not used. if False, will drop them if True, will keep them in the result

Char2VecVectorizer

This model is the equivalent of a "Bag of N-gram" characters but using embedding. It is fitting embedding for sequence of caracters and then average all those embeddings.

class aikit.transformers.text.**Char2VecVectorizer**(*size=100*, *window=5*, *ngram=3*, *text_preprocess='default'*, *same_embedding_all_columns=True*, *use_fast_text=False*, *random_state=None*, *other_params=None*, *columns_to_use='all'*, *desired_output_type='DataFrame'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*)

Char2Vec vectorizer, this model does an average of the embedding of each ngram :

it is sometimes called 'Continuous Bag of Word'

size [int, default = 50] the size of the embedding

window [int, default = 5] the size of the training window of the word2vec model

ngram [int, default = 3] the size of the ngram on which we will fit embedding

text_preprocess: string, default = 'default' type of text preprocessing to use, possible choices are : * 'default' : TextDefaultProcessing : put everything in lower case and remove some punctuation * 'digit' : TextDigitAnonymizer : anonymize digits * 'nltk' : TextNltkProcessing : lower, stemming, remove stopwords, ... * None : do nothing

same_embedding_all_columns [boolean, default = True] if True will fit ONE embedding for ALL the text column, otherwise will fit one word2vec PER text column

random_state [None or int] state of random generator

other_params [dict or None, default = None]

if not None, additional parameters to be passed to the word2vec model

columns_to_use [list,] columns to encode

desired_output_type [data type, default = DataFrame] desired output type

drop_used_columns [boolean, default=True] what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result

drop_unused_columns: boolean, default=True what to do with the column that were not used. if False, will drop them if True, will keep them in the result

Dimension Reduction

TruncatedSVDWrapper

Wrapper around sklearn `TruncatedSVD`.

class aikit.transformers.base.**TruncatedSVDWrapper**(*n_components=2*, *columns_to_use='all'*, *regex_match=False*, *random_state=None*, *drop_used_columns=True*, *drop_unused_columns=True*)

Wrapper around sklearn `TruncatedSVD` with additional capabilities:

- can select its columns to keep/drop
- work on more than one columns
- can return a DataFrame
- can add a prefix to the name of columns

`n_components` can be a float, if that is the case it is considered to be a percentage of the total number of columns.

KMeansTransformer

This transformers does a KMeans clustering and uses the cluster to generate new features (based on the distance between each cluster). Remark : for the 'probability' result_type, since KMeans isn't a probabilistic model the probability is computed using an heuristic.

class aikit.transformers.base.**KMeansTransformer**(*n_clusters=10*, *result_type='probability'*, *temperature=1*, *scale_input=True*, *random_state=None*, *columns_to_use='all'*, *regex_match=False*, *desired_output_type='DataFrame'*, *drop_used_columns=True*, *drop_unused_columns=True*, *kmeans_other_params=None*)

Transformer that apply a KMeans and output distance from cluster center

Parameters

- **n_clusters** (*int, default = 10*) – the number of clusters
- **result_type** (*str, default = 'probability'*) – determines what to output. Possible choices are
  - 'probability' : number between 0 and 1 with 'probability' to be in a given cluster
  - 'distance' : distance to each center
  - 'inv_distance' : inverse of the distance to each cluster
  - 'log_disantce' : logarithm of distance to each cluster
  - 'cluster' : 0 if in cluster, 1 otherwise
- **temperature** (*float, default = 1*) – used to shift probability :unormalized proba = proba ^ temperature
- **scale_input** (*boolean, default = True*) – if True the input will be scaled using StandardScaler before applying KMeans

- **random_state** (*int or None, default = None*) – the initial random_state of KMeans
- **columns_to_use** (*list of str*) – the columns to use
- **regex_match** (*boolean, default = False*) – if True use regex to match columns
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result
- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result
- **desired_output_type** (*DataType*) – the type of result

Feature Selection

FeaturesSelectorRegressor

This transformer will perform feature selection. Different strategies are available:

- "default" : uses sklearn default selection, using correlation between target and variable
- "linear" : uses absolute value of scaled parameters of a linear regression between target and variables
- "forest" : uses `feature_importances` of a RandomForest between target and variables

class aikit.transformers.base.**FeaturesSelectorRegressor**(*n_components=0.5*, *selector_type='forest'*, *component_selection='number'*, *model_params=None*, *columns_to_use='all'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*)

Features Selection based on RandomForest, LinearModel or Correlation.

Parameters

- **n_components** (*int or float, default = 0.5*) – number of component to keep, if float interpreted as a percentage of X size
- **component_selection** (*str, default = "number"*) – if "number" : will select the first 'n_components' features if "elbow" : will use a tweaked 'elbow' rule to select the number of features
- **selector_type** (*string, default = 'forest'*) – 'default' : using sklearn f_regression/f_classification 'forest' : using RandomForest features importances 'linear' : using Ridge/LogisticRegression coefficient
- **random_state** (*int, default = None*) –
- **model_params** – Model hyper parameters

FeaturesSelectorClassifier

Exactly as *aikit.transformers.base.FeaturesSelectorRegressor* but for classification.

class aikit.transformers.base.**FeaturesSelectorClassifier**(*n_components=0.5*, *selector_type='forest'*, *component_selection='number'*, *random_state=None*, *model_params=None*, *columns_to_use='all'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*)

Features Selection based on RandomForest, LinearModel or Correlation.

Parameters

- **n_components** (*int or float, default = 0.5*) – number of component to keep, if float interpreted as a percentage of X size
- **component_selection** (*str, default = "number"*) – if "number" : will select the first 'n_components' features if "elbow" : will use a tweaked 'elbow' rule to select the number of features
- **selector_type** (*string, default = 'forest'*) – 'default' : using sklearn f_regression/f_classification 'forest' : using RandomForest features importances 'linear' : using Ridge/LogisticRegression coefficient
- **random_state** (*int, default = None*) –
- **model_params** – Model hyper parameters

Missing Value Imputation

NumImputer

Numerical value imputer for numerical features.

class aikit.transformers.base.**NumImputer**(*strategy='mean'*, *add_is_null=True*, *fix_value=0*, *allow_unseen_null=True*, *columns_to_use='all'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*)

Missing value imputer for numerical features.

Parameters

- **strategy** (*str, default = 'mean'*) – how to fill missing value, possibilities ('mean', 'fix' or 'median')
- **add_is_null** (*boolean, default = True*) – if this is True of 'is_null' columns will be added to the result
- **fix_value** (*float, default = 0*) – the fix value to use if needed

- **allow_unseen_null** (*boolean, default = True*) – if not True an error will be generated on testing data if a column has missing value in test but didn't have one in train
- **columns_to_use** (*list of str or None*) – the columns to use
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result
- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result
- **regex_match** (*boolean, default = False*) – if True, use regex to match columns

Categories Encoding

NumericalEncoder

This is a transformer to encode categorical variable into numerical values.

The transformer handle two types of encoding:

- 'dummy' : dummy encoding (aka : one-hot-encoding)
- 'num' : simple numerical encoding where each modality is transformed into a number

The transformer includes also other capabilities to simplify encoding pipeline:

- merging of modalities with too few observations to prevent huge result dimension and overfitting,
- treating `None` has a special modality if many `None` are present,
- if the columns are not specified, guess the columns to encode based on their type

class aikit.transformers.categories.**NumericalEncoder**(*columns_to_use='CAT'*, *min_modalities_number=20*, *max_modalities_number=100*, *max_cum_proba=0.95*, *min_nb_observations=10*, *max_na_percentage=0.05*, *encoding_type='dummy'*, *regex_match=False*, *desired_output_type='DataFrame'*, *drop_used_columns=True*, *drop_unused_columns=False*)

Numerical Encoder of categorical variables

Parameters

- **columns_to_use** (*list of str*) – the columns to use
- **min_modalities_number** (*int, default = 20*) – if less that 'min_modalities_number' modalities no modalities will be filtered
- **max_modalities_number** (*int, default = 100,*) – the number of modalities kept will never be more than 'max_modalities_number'

- **max_cum_proba** (*float, default = 0.95*) – if modalities should be filtered, first filter applied is removing modalities that account for less than 1-'max_cum_proba'
- **min_nb_observations** (*int, default = 10*) – if modalities should be filtered, modalities with less thant 'min_nb_observations' observations will be removed
- **max_na_percentage** (*float, default = 0.05*) – if more than 'max_na_percentage' percentage of missing value, None will be treated as a special modality named '__null__' otherwise, will just put -1 (for encoding_type == 'num') or 0 everywhere (for encoding_type == 'dummy')
- **encoding_type** (*'dummy' or 'num', default = 'dummy'*) – type of encoding between a numerical encoding and a dummy encoding
- **regex_match** (*boolean, default = False*) – if True use regex to match columns
- **desired_output_type** (*DataType*) – the type of result
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result
- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result

CategoricalEncoder

This is a wrapper around module:categorical_encoder package.

class aikit.transformers.categories.**CategoricalEncoder**(*columns_to_use='CAT'*, *encoding_type='dummy'*, *basen_base=2*, *hashing_n_components=10*, *regex_match=False*, *desired_output_type='DataFrame'*, *drop_used_columns=True*, *drop_unused_columns=False*)

Wrapper around categorical encoder package encoder

Parameters

- **columns_to_encode** (*None or list of str*) – the columns to encode (if None will guess)
- **encoding_type** (*str, default = 'dummy'*) –

  the type of encoding, possible choices :

  - dummy
  - binary
  - basen

– hashing

- **basen_base** (*int, default = 2*) – the base when using encoding_type == 'basen'
- **hashing_n_components** (*int, default = 10*) – the size of hashing when using encoding_type == 'hashing'
- **columns_to_use** (*list of str or None*) – the columns to use for that encoder
- **regex_match** (*boolean*) – if True will use regex to match columns
- **desired_output_type** (*list of DataType*) – the type of output wanted
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result
- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result

TargetEncoderRegressor

This transformer also handles categorical encoding but uses the target to do that. The idea is to encode each modality into the mean of the target on the given modality. To do that correctly special care should be taken to prevent leakage (and overfitting).

The following techniques can be used to limit the issue :

- use of an inner cross validation loop (so an observation in a given fold will be encoded using the average of the target computed on other folds)
- noise can be added to encoded result
- a prior corresponding to the global mean is apply, the more observations in a given modality the less weight the prior has

class aikit.transformers.target.**TargetEncoderRegressor**(*columns_to_use='CAT', max_na_percentage=0.05, smoothing_min=1, smoothing_value=10, noise_level=None, cv=10, random_state=None, regex_match=False, desired_output_type='DataFrame', drop_used_columns=True, drop_unused_columns=False*)

Class to encode categorical value using the target

Parameters

- **max_na_percentage** (*float, default = 0.05*) – if more than 'max_na_percentage' None within a column, None will be treated as a special modality, otherwise it will default to the global aggregat

- **smoothing_min** (*float, default = 1*) – handle the prior weight, see formula bellow
- **smoothing_value** (*float, default = 10*) – handle the *speed* with which the prior is forgotten (see formula bellow)
- **noise_level** (*float or None, default = None*) – degree of noise to add within the fit_transform
- **cv** (*int, None, or CV object, default = 10*) – the cv to use within fit_transform

Those parameters handles the prior weight, WEIGHT = 1/[1 + EXP(- (nb - smoothing_min) / smoothing_value)] where 'nb' is the number of observations of the corresponding modality

The precautions explained above causes the transformer to have a different behavior when doing:

- `fit` then `transform`
- `fit_transform`

When doing `fit` then `transform`, no noise is added during the transformation and the `fit` save the global average of the target. This is what you'd typically want to do when fitting on a training set and then applying the transformation on a testing set.

When doing `fit_transform`, noise can be added to the result (if `noise_level != 0`) and the target aggregats are computed fold by fold.

To understand better here is what append when `fit` is called :

1. variables to encode are guessed (if not specified)
2. global average per modality is computed
3. global average (for all dataset) is computed (to use as prior)
4. global standard deviation of target is computed (used to set noise level)
5. for each variable and each modality compute the encoded value using the global aggregat and the modality aggregat (weighted by a function of the number of observations for that modality)

Now here is what append when `transform` is called :

1. for each variable and each modality retrieve the corresponding value and use that numerical feature

Now when doing a `fit_transform` :

1. call `fit` to save everything needed to later be able to transform unseen data
2. do a cross validation and for each fold compute aggregat and the remaining fold
3. use that value to encode the modality
4. add noise to the result : proportional to `noise_level * global standard deviation`

TargetEncoderClassifier

This transformer handles categorical encoding and uses the target value to do that. It is the same idea as `TargetEncoderRegressor` but for classification problems. Instead of computing the average of the target, the probability of each target classes is used.

The same techniques are used to prevent leakage.

class aikit.transformers.target.**TargetEncoderClassifier**(*columns_to_use='CAT'*, *max_na_percentage=0.05*, *smoothing_min=1*, *smoothing_value=10*, *noise_level=None*, *cv=10*, *random_state=None*, *regex_match=False*, *desired_output_type='DataFrame'*, *drop_used_columns=True*, *drop_unused_columns=False*)

Class to encode categorical value using the target

Parameters

- **max_na_percentage** (*float, default = 0.05*) – if more than 'max_na_percentage' None within a column, None will be treated as a special modality, otherwise it will default to the global aggregat
- **smoothing_min** (*float, default = 1*) – handle the prior weight, see formula bellow
- **smoothing_value** (*float, default = 10*) – handle the *speed* with which the prior is forgotten (see formula bellow)
- **noise_level** (*float or None, default = None*) – degree of noise to add within the fit_transform
- **cv** (*int, None, or CV object, default = 10*) – the cv to use within fit_transform

Those parameters handles the prior weight, WEIGHT = 1/[1 + EXP(- (nb - smoothing_min) / smoothing_value)] where 'nb' is the number of observations of the corresponding modality

Other Target Encoder

Any new target encoder can easily be created using the same technique. The new target encoder class must inherit from `_AbstractTargetEncoder`, then the `aggregating_function` can be overloaded to compute the needed aggregat.

The `_get_output_column_name` can also be overloaded to specify feature names.

Scaling

CdfScaler

This transformer is used to re-scale feature, the re-scaling is non linear. The idea is to fit a cdf for each feature and use it to re-scale the feature to be either a uniform distribution or a gaussian distribution.

class aikit.transformers.base.**CdfScaler**(*distribution='auto-kernel'*, *output_distribution='uniform'*, *copy=True*, *verbose=False*, *sampling_number=1000*, *random_state=None*, *columns_to_use='all'*, *regex_match=False*, *drop_used_columns=True*, *drop_unused_columns=True*, *desired_output_type=None*)

Scaler based on the distribution

Each variable is scaled according to its law. The law can be approximated using :

- parametric law : distribution = "normal", "gamma", "beta"
- kernel approximation : distribution = "kernel"
- rank approximation : "rank"
- if distribution = "none" : no distribution is learned and no transformation is applied (useful to not transform some of the variables)
- if distribution = "auto-kernel" : automatic guessing on which column to use a kernel (columns whith less than 5 differents values are un-touched)
- if distribution = "auto-param" : automatic guessing on which column to use a parametric distribution (columns with less than 5 differents valuee are un-touched)

for other columns choice among "normal", "gamma" and "beta" law based on values taken

After the law is learn, the result is transformed into :

- a uniform distribution (output_distribution = 'uniform')
- a gaussian distribution (output_distribution = 'normal')

Parameters

- **distribution** (*str or list of str, default = "auto-kernel"*) – the distribution to use for each variable, if only one string the same transformation is applied everything where
- **output_distribution** (*str, default = "uniform"*) – type of output, either "uniform" or "normal"
- **copy** (*boolean, default = True*) – if True wil copy the data then modify it
- **verbose** (*boolean, default = True*) – set the verbosity level
- **sampling_number** (*int or None, default = 1000*) – if set subsample of size 'sampling_number' will be drawn to estimate kernel densities
- **random_state** (*int or None*) – state of the random generator
- **columns_to_use** (*list of str*) – the columns to use
- **regex_match** (*boolean, default = False*) – if True use regex to match columns
- **drop_used_columns** (*boolean, default=True*) – what to do with the ORIGINAL columns that were transformed. If False, will keep them in the result (un-transformed) If True, only the transformed columns are in the result

- **drop_unused_columns** (*boolean, default=True*) – what to do with the column that were not used. if False, will drop them if True, will keep them in the result
- **desired_output_type** (*DataType*) – the type of result

Target Transformation

BoxCoxTargetTransformer

This transformer is a regression model that modify that target by applying it a boxcox transformation. The target can be positive or negative. This transformation is useful to *flatten* the distribution of the target which can help underlying model (especially those who are not robust to outliers).

Remark : It is important to note that when predicting the inverse transformation will be applied. If what is important to you is the error on the logarithm of the error you should:

- directly transform you target before anything
- use a customized scorer

class aikit.transformers.base.**BoxCoxTargetTransformer**(*model*, *ll=0*)

BoxCoxTargetTransformer, it is used to fit the underlying model on a transformation of the target

the model does the following :

1. transform target using 'target_transform'
2. fit the underlying model on transformation
3. when prediction, apply 'inverse_transformation' to result

Here the transformation is in the 'box-cox' family.

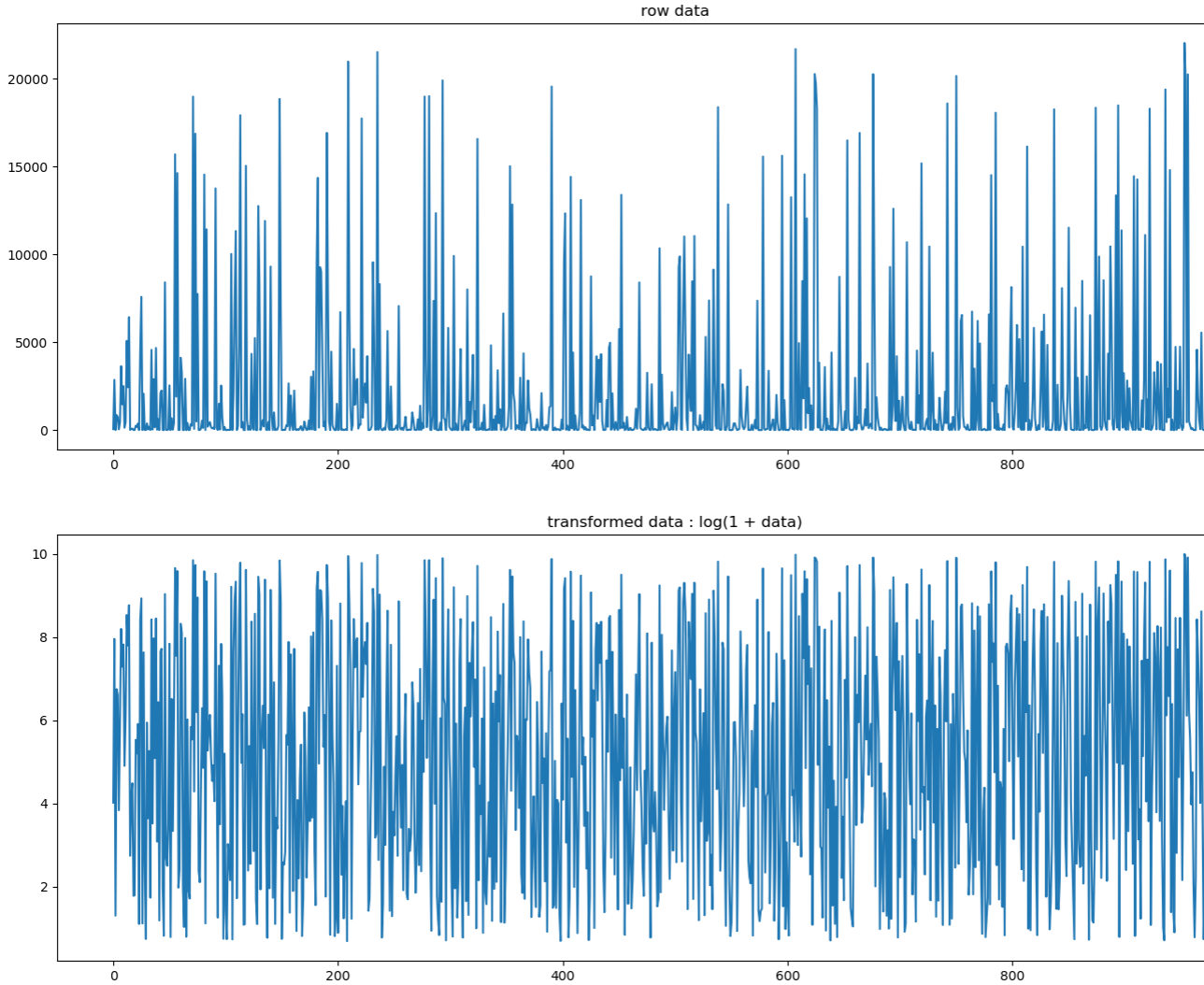
- ll = 0 means this transformation : sign(x) * log(1 + abs(x))
- ll > 0 sign(x) * exp(log(1 + ll * abs(xx)) / ll - 1)

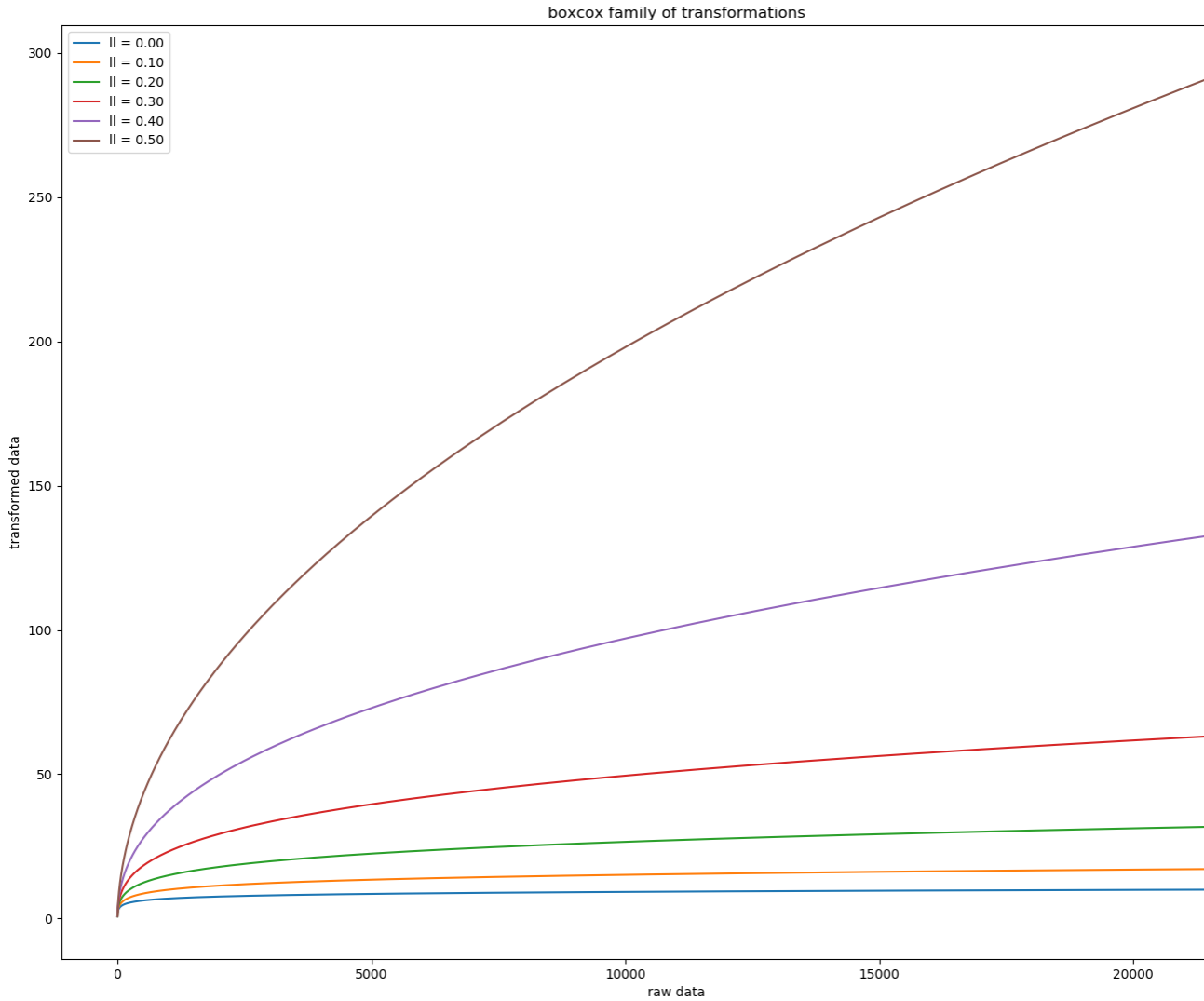
Parameters

- **model** (*sklearn like model*) – the model to use
- **ll** (*float, default = 0*) – the boxcox parameter

Example of transformation using `ll = 0`:

When `ll` increases the *flattenning* effect diminishes :


row data
20000
15000
10000
5000
0
0
200
400
600
800
transformed data : log(1 + data)
10
8
6
4
2
0
200
400
600
800


boxcox family of transformations
ll = 0.00
ll = 0.10
ll = 0.20
ll = 0.30
ll = 0.40
ll = 0.50
transformed data
raw data
0
50
100
150
200
250
300
0
5000
10000
15000
20000

CHAPTER 7

Other Functionnalities

Aikit offers other functionnalities that might be usefull for the DataScientist job.

Model Json

A function to back and forth between a json-like object and a sklearn-like model that can be fitted. This can be used to store the definition of a complete model into a json file for your projects.

7.1 ModelJson

7.1.1 Model representation

It is sometime useful to specify the model to use for a given use-case outside of the main code of the project. For example in a json like object. This can have several advantages :

- allow the change of the underlying model without change any code (example : shift from a `RandomForestClassifier` to a `LGBMClassifier`)
- allow the same code to be used for different sub problem *BUT* allowing specific hyper-parameters/models for each sub problems
- easier to incorporate model that were found automatically by an ml_machine

To be able to do that we need to save the description of a complex model into a simple json like format.

The syntax is easy : a model is represented by a tuple with its name and its hyper-parameters.

Example, the model:

```
RandomForestClassifier(n_estimators=100)
```

is represented by the object:

```
("RandomForestClassifier",{"n_estimators":100})
```

So : klass(**kwargs) is equivalent to ('klass',kwargs)

Let's take a more complexe example using a GraphPipeline:

```
gpipeline = GraphPipeline(models = {"vect" : CountVectorizerWrapper(analyzer="char",
↪ngram_range=(1,4)),
                                    "svd"  : TruncatedSVDWrapper(n_
↪components=400) ,
                                    "logit" : LogisticRegression(class_weight=
↪"balanced")},
                          edges = [("vect","svd","logit")]
                          )
```

is represented by:

```
json_object = ("GraphPipeline", {"models": {"vect" : ("CountVectorizerWrapper"  , {
↪"analyzer":"char","ngram_range":(1,4)} ),
                                 "svd"  : ("TruncatedSVDWrapper"     , {"n_components":
↪400}) ,
                                 "logit": ("LogisticRegression" , {"class_weight":
↪"balanced"}) },
                    "edges":[("vect","svd","logit")]
                    })
```

So if a given model uses other models as parameters it works as well.

7.1.2 Model conversion

Once the object is create you can convert it to a real (unfitted) model using `aikit.model_definition.sklearn_model_from_param()`

```
sklearn_model_from_param(json_object)
```

which gives a model that can be fitted.

7.1.3 Json saving

That representation uses only simple types that are json serializable (string, number, list, dictionnary) and the json can be saved on disk.

Remark [since json doesn't allow :]

- tuple (only list are known)
- dictionnary with non string keys

it is best to overried the json serializer to handle those type. The special encoder is found in **:module:'aikit.tools.json_helper'** and 'save_json' and 'load_json' can be used directly

Example saving the 'json_object' above:

```
from aikit.json_helper import save_json
save_json(json_object, fname ="model.json")

realoaded_json_object = load_json("model.json")
```

The special serializer works by transforming un-handle type into a dictionnary with

- a '__items__' key with a list of object
- a '__type__' key with the original type

Example:

```
("a","b")
```

is transformed into:

```
{"__items__":["a","b"], "__type__":"__tuple__"}
```

The handle types are :

- dict : '__dict__'
- tuple

7.1.4 Model Register

To be able to use a given model using only its name all the models should be registred in a dictionnary.

This is done within `aikit.simple_model_registration`, in that file you have a DICO_NAME_KLASS object which stored the classes of every model. To add a new model simple use the add_klass method.

Example:

```
DICO_NAME_KLASS.add_klass(LGBMClassifier)
DICO_NAME_KLASS.add_klass(LGBMRegressor)
```

Remark : this registrer is different from the one used for the automatic machine learning part (ml_machine) which contain more informations (hyper-parameters, type, . . .)

Stacking

Tools to do stacking easily without having to re-code everything from scratch.

7.2 Model Stacking

Here you'll find the explanation about how to fit stacking models using aikit.

You can also see the notbook :

- `Stacking`

7.2.1 Stacking How To

Stacking (or Blending) is done when you want to aggregate the result of several models into an aggregated prediction. More precisely you want to use the predictions of the models as input to another blending model. (In what follows I'll call *models* the models that we want to use as features and *blender* the model that uses those as features to make a final predictions)

To prevent overfitting you typically want to use out-sample predictions to fit the blender (otherwise the blender will just learn to trust the model that overfit the most. . .).

To generate out-sample predictions you can do a cross-validation:

for fold (i) and a given model :

1. fit model on all the data from the other folds (1,. . . ,i-1,i+1, .. N)
2. call fitted model on data from fold (i)
3. store the predictions

If that is done on all the folds, you can get predictions for every sample in your database, but each prediction were generated using out sample training set.

Now you can take all those predictions and fit a blender model.

7.2.2 StackerClassifier

This class does exactly what is describeb above. It takes as input a list of un-fitted models as well as a blending model and will generate a stacking model. In that case what is given to the blending model are the probabilities of each class for each model.

class aikit.models.stacking.**StackerClassifier**(*models*, *cv*, *blender*, *random_state=None*)

generic class to handle stacking

This class takes a list of models and does the following during its fitting phase

1. does a cross-validation on each model to output out-sample predictions
2. use those out-sample prediction to fit a blending model
3. re-fit the models on all the datas

During test: 1. call each models to retrieve predictions 2. call the blender to retrieve final aggregated prediction

Parameters

- **models** (*list of model*) – the models that we want to stacked
- **cv** (*cv object or int*) – the cross-validation to use to fit the blender
- **blender** (*model*) – the blending model

7.2.3 StackerRegressor

Same class but for regression problems

class aikit.models.stacking.**StackerRegressor**(*models*, *cv*, *blender*, *random_state=None*)

generic class to handle stacking

This class takes a list of models and does the following during its fitting phase

1. does a cross-validation on each model to output out-sample predictions
2. use those out-sample prediction to fit a blending model
3. re-fit the models on all the datas

During test: 1. call each models to retrieve predictions 2. call the blender to retrieve final aggregated prediction

Parameters

- **models** (*list of model*) – the models that we want to stacked
- **cv** (*cv object or int*) – the cross-validation to use to fit the blender
- **blender** (*model*) – the blending model

7.2.4 OutSamplerTransformer

The library offers another way to create stacking model. A stacking model can be viewed as another kind of pipeline : instead of a *transformation* it just uses other models as a special kind of transformers. And so GraphPipeline can be used to chain this transformation with a blending model.

To be able to do that two things are needed:

1. a model is not a transformer, and so we need to transform a model into a transformer (basically saying that the **:function:'transform'** method should use **:function:'predict'** or **:function:'predict_proba'**)
2. to do correct stacking we need to generate out-sample predictions

The `OutSamplerTransformer` does just that. It takes care of generating out-sample predictions, and the blending can be done in another node of the GraphPipeline (see example bellow)

class aikit.models.stacking.**OutSamplerTransformer**(*model*, *cv=10*, *random_state=123*, *desired_output_type=None*, *columns_prefix=None*)

This class is used to transform a model in a transformers that makes out of sample predictions

This transformation can be used to easily use model in part of a GraphPipeline.

fit method : 1. simply fit the underlying model

fit_transform method : 1. do a cross-validation on the underlying model to output out-of-sample prediction 2. re-fit underlying model on all the data

transform method : 1. just output prediction of underlying model

Parameters

- **model** (*a model*) – the model that we want to 'transform'
- **cv** (*cv object or int*) – which crossvalidation to use
- **random_state** (*int or None*) – specify the random state (to force the CV the be fixed)
- **desired_output_type** (*None or type of output*) – the output type of the result of the transformation
- **columns_prefix** (*None or str*) – each column will be prefixed by it

7.2.5 Example

Let's say we want to stack a RandomForestClassifier, an LGBMClassifier and a LogisticRegression. You can create the model like that:

```
stacker = StackerClassifier( models = [RandomForestClassifier() , LGBMClassifier(),␣
↪LogisticRegression()],
                             cv = 10,
                             blender = LogisticRegression()
                            )
```

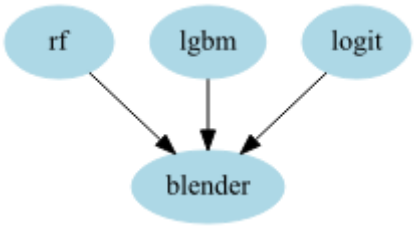
and then fit it as you would a regular model:

```
stacker.fit(X, y)
```

Using OutSamplerTransformer we can achieve the same thing but with a Pipeline instead:

```
from sklearn.model_selection import StratifiedKFold
from aikit.models import OutSamplerTransformer
from aikit.pipeline import GraphPipeline
cv = StratifiedKFold(10, shuffle=True, random_state=123)

stacker = GraphPipeline(models = {
    "rf"    : OutSamplerTransformer(RandomForestClassifier() , cv = cv),
    "lgbm" : OutSamplerTransformer(LGBMClassifier()          ,  cv = cv),
    "logit": OutSamplerTransformer(LogisticRegression()      , cv = cv),
    "blender":LogisticRegression()
    }, edges = [("rf","blender"),("lgbm","blender"),("logit","blender")])
```



Remark:

1. the 2 models are equivalents
2. to have *regular* stacking the same cvs should be used every where (either by creating it before hand, by setting the random state or using a non shuffle cv)

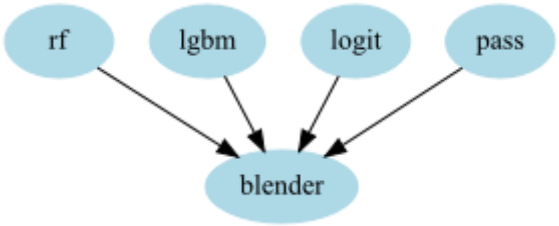
With this idea we can do more complicated things like :

1. *deep* stacking with more than one layer : simply add other layer to the GraphPipeline
2. create a blender that uses both predictions of models as well as the features (or part of it) : simply add another node linked to the blender (a PassThrough node for example)
3. do pre-processing before doing any stacking (and so doing it out-side of the cv loop)

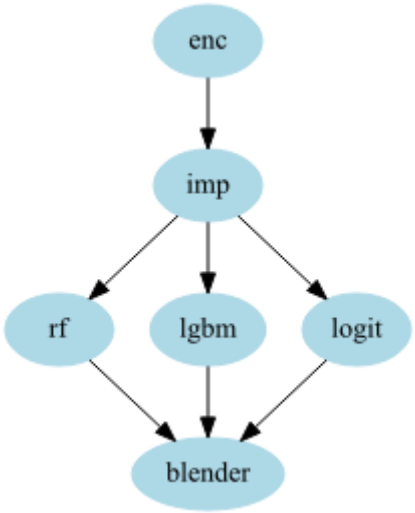
For example:

```
stacker = GraphPipeline(models = {
    "rf"    : OutSamplerTransformer(RandomForestClassifier() , cv = cv),
    "lgbm" : OutSamplerTransformer(LGBMClassifier() , cv = cv),
    "logit": OutSamplerTransformer(LogisticRegression(), cv = cv),
    "pass" : PassThrough(),
    "blender":LogisticRegression()
    }, edges = [("rf","blender"),
                ("lgbm","blender"),
                ("logit","blender"),
                ("pass", "blender")
                ])
```

Or:




```
stacker = GraphPipeline(models = {
    "enc"   : NumericalEncoder(),
    "imp"   : NumImputer(),
    "rf"    : OutSamplerTransformer(RandomForestClassifier() , cv = cv),
    "lgbm"  : OutSamplerTransformer(LGBMClassifier() , cv = cv),
    "logit": OutSamplerTransformer(LogisticRegression(), cv = cv),
    "blender":LogisticRegression()
    }, edges = [("enc","imp"),
                ("imp","rf","blender"),
                ("imp","lgbm","blender"),
                ("imp","logit","blender")
                ])
```

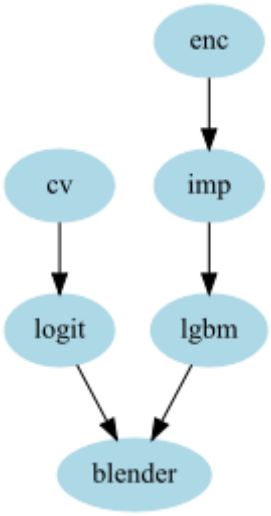


And lastly:

```
stacker = GraphPipeline(models = {
    "enc"   : NumericalEncoder(columns_to_use= ["cat1","cat2","num1","num2"]),
    "imp"   : NumImputer(),
    "cv"    : CountVectorizerWrapper(columns_to_use = ["text1","text2"]),
    "logit": OutSamplerTransformer(LogisticRegression(), cv = cv),
    "lgbm"  : OutSamplerTransformer(LGBMClassifier() , cv = cv),
    "blender":LogisticRegression()
    }, edges = [("enc","imp","lgbm","blender"),
                ("cv","logit","blender")
                ])
```

This last example shows a model where you have:

- categorical and numerical data, on which you apply a classical categorie encoder, you fill missing value



and use gradient boosting

- textual data which you can encode using a CountVectorizer and use LogisticRegression

Both models can be mixted with a Logistic Regression blending. Doing that just create an average between the predictions, the only difference is that since the blender is fitted, weights are in a sence optimal. (In some cases it might work better than just concatenate everything : especially since the 2 sets of features are highly different).

Another thing that can be done is to *calibrate probabilities*. This can be useful if your model generate meaningful scores but you

- if you skewed your training set to solve imbalance
- if your model is not probabilistic
- ...

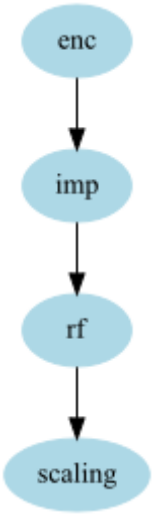
One method to re-calibrate probabilities, call Platt's scaling , is to fit a LogisticRegression on the output of your model. If your predictions are not completely wrong, this will usualy just compute an increasing function that recalibrate your probabilities but won't change change the ordering of the output. (roc_auc won't be affected, but logloss, or accuracy can change).

This can also be done using `OutSamplerTransformer`:

```
rf_rescaled = GraphPipeline(models = {
    "enc"   : NumericalEncoder(),
    "imp"   : NumImputer(),
    "rf"    : OutSamplerTransformer( RandomForestClassifier(class_weight = "auto"), cv␣
↪= 10),
    "scaling":LogisticRegression()
    }, edges = [('enc','imp','rf','scaling')]
)
```

7.2.6 OutSamplerTransformer regression mode

You can do exactly the same thing but for regresion tasks, only difference is that cross-validation uses `predict()` instead of `predict_proba()`.



Data Structure Helper

You also have tools to help you with things like :

- conversion of datatypes
- concatenation of heterogenous datatypes
- guess of type of columns
- ...

See :

7.3 Data Structure Helper

7.3.1 Data Types

Aikit helps dealing with the multiple type of data that coexist within the scikit-learn, pandas, numpy and scipy environments. Mainly :

- pandas DataFrame
- pandas Sparse DataFrame
- numpy array
- scipy sparse array (csc, csr, coo, ...)

The library offers tools to easily convert between each type.

Within aikit.enums there is a DataType enumeration with the following values :

- 'DataFrame'
- 'SparseDataFrame'
- 'Series'
- 'NumpyArray'
- 'SparseArray'

This is better use as an enumeration but the values are actual strings so you can use the string directly if needed.

The function *aikit.tools.data_structure_helper.get_type* retrieve the type (one element of the element).

Example of use:

```
from aikit.tools.data_structure_helper import get_type, DataTypes
df = pd.DataFrame({"a":[0,1,2],"b":["aaa","bbb","ccc"]})
mapped_type = get_type(df)

if mapped_type == DataTypes.DataFrame:
    first_column = df.loc[:,"a"]
else:
    first_column = df[:,0]
```

7.3.2 Generic Conversion

You can also convert each type to the desired type. This can be useful if a transformer only accepts DataFrames, or doesn't work with a Sparse Array, ... For that use the function *aikit.tools.data_structure_helper.convert_generic*

aikit.tools.data_structure_helper.**convert_generic**(*xx*, *mapped_type=None*, *output_type=None*)

generic conversion function

Parameters

- **xx** (*array, DataFrame, ..*) – the object to convert
- **mapped_type** (*enumeration from enums.DataTypes or None*) – if not None, the type enumeration of xx
- **output_type** (*enumeration from enums.DataTypes or None*) – if not None the desired output tpye

Example:

```
from aikit.tools.data_structure_helper import convert_generic, DataTypes
df = pd.DataFrame({"a":[0,1,2],"b":["aaa","bbb","ccc"]})

arr = convert_generic(df, output_type = DataTypes.NumpyArray)
```

7.3.3 Generic Horizontal Stacking

You can also horizontally concatenate multiple datas together (assuming they have the same number of rows). You can either specify the output type you want, if that is not the case the algorithm will guess :

- if same type will use that type
- if DataFrame and Array use DataFrame
- if Sparse and Non Sparse : convert to full if not to big otherwise keep Sparse

(See *aikit.tools.data_structure_helper.guess_output_type*)

The function to concatenate everything is *aikit.tools.data_structure_helper.generic_hstack*

Example:

```
df1 = pd.DataFrame({"a":list(range(10)),"b":["aaaa","bbbbb","cccc"]*3 + ["ezzzz"]})
df2 = pd.DataFrame({"c":list(range(10)),"d":["aaaa","bbbbb","cccc"]*3 + ["ezzzz"]})

df12 = generic_hstack((df1,df2))
```

aikit.tools.data_structure_helper.**generic_hstack**(*all_datas*, *output_type=None*, *all_columns_names=None*, *max_number_of_cells_for_non_sparse=10000000*)

generic function to concatenate horizontaly some datas objects

All datas should have the same number of rows

Parameters

- **all_datas** (*list of data object*) – the things that we want to concatenate
- **output_type** (*None or type of data*) – if None will guess the type (See 'guess_output_type') otherwise will concatenate using that format
- **= None or list of names** (*all_columns_names*) – if not None it corresponds to the list of columns of each sub datas

Returns

Return type aggregated object

7.3.4 Other

Two other functions that can be useful are *aikit.tools.data_structure_helper.make1dimension* and *aikit.tools.data_structure_helper.make2dimensions*. It convert to a 1 dimensional or 2 dimensional object whenever possible.

aikit.tools.data_structure_helper.**make1dimension**(*X*)

generic function to make an object uni dimensional

aikit.tools.data_structure_helper.**make2dimensions**(*X*)

generic function to make a data object at least bi-dimensional

Example

```
>>> df = pd.DataFrame({"a":np.arange(10),"b":["aa","bb","cc"]*3 + ["dd"]})
>>> assert make2dimensions(df).shape == (10,2)
>>> assert make2dimensions(df["a"]).shape == (10,1)
>>> assert make2dimensions(df.values).shape == (10,2)
>>> assert make2dimensions(df["a"].values).shape == (10,1)
```

Block Manager

A special data storage that is able to contains several data of different types (but same number of observation) into one single object that is indexable

7.4 Block Selector

In order to facilitate use-cases where several types of data are presented (as well as for internal use) two things were created

- a `BlockSelector` selector transformer whose job is to select a given *block of data* (See after)
- a `BlockManager` object to store and retrieve blocks of Data.

Here is the explanation. Let's imagine you have a use case where data has more than one type. Example you have text and regular features, or text and image. Typically to handle that you would need to put everyting into a DataFrame and then use selectors to retrieve different part of the data. This is already greatly facilitated by aikit with the GraphPipeline, the fact that wrapped models can select the variables they work on and the fact that this selection can be done uses regex (and consequently prefix). You could also deal with it manually but then cross-validation, splitting or train and test, would also have to be done manually.

However sometime it is just not praticle or possible to merge everyting into a DataFrame :

- you might have a big sparse entry and other regular features : not praticle to merge everything into either a dense or sparse object
- you might have a picture and regular feature and typically the pictures will be stored in a 3 or 4 dimensionals (observation x height x width x channels) tensor and not the classical 2 dimensions object
- .... any other reason

What you can do is put ALLs your data into a dictionnary (or a BlockManager, see just after) : one key per type of data and one value per data. You you pass that object to a block selector it can retrieve each block seperately:

```
Xtrain = {"regular_features":pandas_dataframe_object,
          "sparse_features" :scipy_sparse_object
          }

block_selector = BlockSelector("regular_features")
df = block_selector.fit_transform(Xtrain)
```

Here *df* is just the *pandas_dataframe_object* object. Remark : fit_transform is used as this work like a classical transformer. Even if the transformer doesn't do anything during the fit.

So you can just put those objects at the top of your pipeline and uses a dictionnary of data inplace of X.

The BlockSelector object also work with a list of datas (in that case the block to select is the integer corresponding to the position):

```
Xtrain = [pandas_dataframe_object,
          scipy_sparse_object]

block_selector = BlockSelector(0)
df = block_selector.fit_transform(Xtrain)
```

Example of such pipeline:

```
GraphPipeline(models = {
        "sel_sparse":BlockSelector("sparse_features"),
        "svd":TruncatedSVDWrapper(),
        "sel_other":BlockSelector("regular_features"),
        "rf":RandomForestClassifier()
        },
edges = [("sel_sparse","svd","rf"),("sel_other","rf")])
```

That model can be fitted however, you can't cross-validate it easily. That's is because Xtrain (which is a dictionnary of datas or a list of datas) isn't subsetable : you can't do Xtrain[index,:] or Xtrain.loc[index,:].

7.5 Block Manager

To solve this a new type of object is needed : the `BlockManager`. This object is conceptually exactly like the Xtrain object before, it can either be a dictionnary of data or a list of data. However it has a few additionnal things that allow it to work well within sklearn environnement.

- it has a shape attribute
- it can be subsetted using 'iloc'

Example:

```
df = pd.DataFrame({"a":np.arange(10),"b":["aaa","bbb","ccc"] * 3 + ["ddd"]})
arr = np.random.randn(df.shape[0],5)
X = BlockManager({"df":df, "arr":arr})

X["df"]        # this retrieves the DataFrame df (no copy)
X["arr"]       # this retrieves the numpy array arr (no copy)
X.iloc[0:5,:]  # this create a smaller BlockManager object with only 5 observations

block_selector = BlockSelector("arr")
block_selector.fit_transform()         #This retrieve the numpy array "arr" as well
```

CHAPTER 8

How to add new models

One of the idea of the package is to offer ways to quickly test ideas without much burden. It is thus relatively easy to add new models/transformers to the framework.

Those models/transformers can be included in the search of the auto-ml component to be tested on different databases and with other transformers/models.

A model needs to be added at two different places in order to be fully integrated within the framework.

Let's see what's need to be done to include an hypothetic new models:

```
class ReallyCoolNewTransformer(BaseEstimator, TransformerMixin):
    """ This is a great new  transformer """
    def __init__(self, super_choice):
        self.super_choice = super_choice

    def fit(self,X, y = None):
        pass

    def transform(self,X):
        pass
```

8.1 Add model to Simple register

This will allow the function 'sklearn_model_from_param' to be able to use your new model. The class simply needs to be added to the DICO_NAME_KLASS object:

```
from aikit.model_definition import DICO_NAME_KLASS
DICO_NAME_KLASS.add_klass(ReallyCoolNewTransformer)
```

Now that this is done, you can call the transformer by its *name*:

```
from aikit.model_definition import sklearn_model_from_param
model = sklearn_model_from_param(("ReallyCoolNewTransformer",{}))
```

model is an instance of ReallyCoolNewTransformer

8.2 Add model to Auto-Ml framework

This is a little more complicated, a few more informations need to be entered:

- type of model
- type of variable it uses
- hyper-parameters

To do that you need to use the @register decorator:

```
from aikit.ml_machine.ml_machine_registration import register, _
↪AbstractModelRepresentationDefault, StepCategories
import aikit.ml_machine.hyper_parameters as hp

@register
class DimensionReduction_ReallyCoolNewTransformer(_
↪AbstractModelRepresentationDefault):
    klass = ReallyCoolNewTransformer

    category = StepCategories.DimensionReduction
    type_of_variable = None
    type_of_model = None # Used for all problem

    custom_hyper = {"super_parameters":hp.HyperChoice(("superchoice_a","superchoice_b
↪"))}
```

See *Model Register* for complete description of register. See *Hyperparameters* for complete description of register.

Remark: The registers behaves like singletons so you can modify them in any part of the code. You just need the code to be executed somewhere for it to work.

If a model is stable and tested enough the new entry can be added to the python files :

- 'model_definition.py' : for the simple register
- ml_machine/ml_machine_registration.py : for the full auto-ml register

(See *Contribution* for detailed about how to contribute to the evolution of the library)

Remark : you don't need to use the wrapper for your model to be incorporated in the framework. However, it is best to do so. That way you can focus on the logic and let the wrapper make your model more generic.

8.2.1 Model Register

To be able to randomly create complex processing pipelines the Ml Machine needs to know a few things.

1. The list of models/transformers to use
2. Information about each models/transformations

For each models/transformers here are the information needed:

1. the step on which is will be used : CategorieEncoder, TextProcessing, ...
2. the type of variable it will be used on : Text, Numerical, Categorie or None (for everything)
3. can it be used for regression, classification or both

4. the list of hyper-parameters and their values
5. ... any other needed piece of information

To be able to that each transformers and models should be register and everything is stored into an object.

All that is done within `aikit.ml_machine.ml_machine_registration`

To register a new model simply create a class and decorate it:

```
@register
class RandomForestClassifier_Model(_AbstractModelRepresentationDefault):

    klass = RandomForestClassifier
    category = StepCategories.Model
    type_of_variable = None

    custom_hyper = {"criterion": ("gini","entropy") }

    is_regression = False


    default_parameters = {"n_estimators":100}
```

Remark : the name of the class doesn't matter, no instance will ever be created. It is just a nice way to write information.

A few things are needed within that class:

- klass : should contain the actual class of the underlying model
- category : one of the StepCategories choice
- type_of_variable : if None it means that the model should be applied to the complete Dataset (this field will be used to create branches with the different type of variable in the pipeline)
- is_regression : True, False or None (None means both)
- default_parameters : to override the klass default parameters. Those parameters are use during the First Round of the Ml Machine

HyperParameters

Each class should be able to generate its hyper-parameters, that is done by default with the 'get_hyper_parameter' class method.

Here is what is done to generate the hyper-parameters:

For each parameters within the signature of the __init__ method of the klass:

- if the parameters is present within 'custom_hyper', use that to generate the corresponding values
- if it is present within the 'default_hyper', use that to generate the corresponding values
- if not, don't include it in the hyperparameters (and consequently the default value will be kept).

Default Hyper

Here is the list of the default hyper-parameters:

```
default_hyper = {
    "n_components" : hp.HyperRangeFloat(start = 0.1,end = 1,step = 0.05),

    # Forest like estimators
    "n_estimators" : hp.HyperComposition(
                        [(0.75,hp.HyperRangeInt(start = 25, end = 175, step = 25)),
                         (0.25,hp.HyperRangeInt(start = 200, end = 1000, step = 
↪100))]),

    "max_features":hp.HyperComposition([ ( 0.25 , ["sqrt","auto"]),
                                         ( 0.75 , hp.HyperRangeBetaFloat(start = 0, 
↪end = 1,alpha = 3,beta = 1) )
                                       ]),

    "max_depth":hp.HyperChoice([None,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50,
↪100]),
    "min_samples_split":hp.HyperRangeBetaInt(start = 1,end = 100, alpha = 1, beta = 
↪5),

    # Linear model
    "C":hp.HyperLogRangeFloat(start = 0.00001, end = 10, n = 50),

    "alpha":hp.HyperLogRangeFloat(start = 0.00001,end = 10, n = 50),

    # CV
    "analyzer":hp.HyperChoice(['word','char','char_wb']),
 }
```

This helps when hyper-parameters are common across many models.

Special HyperParameters

In some cases, hyper-parameters can't be drawn independently from each other.

For example, by default, we might want to test either for a CountVectorizer:

- 'char' encoding with bag of char of size 1,2,3 or 4
- 'word' encoding only with bag of word of size 1

In that case we need to create a custom global hyper-parameter, that can be done by overriding the `get_hyper_parameter()` classmethod.

Example:

```
@register
class CountVectorizer_TextEncoder(_AbstractModelRepresentationDefault):
    klass = CountVectorizerWrapper
    category = StepCategories.TextEncoder
    type_of_variable = TypeOfVariables.TEXT


    @classmethod
    def get_hyper_parameter(cls):
        ### Specific function to handle the fact that I don't want ngram != 1 IF 
↪analyzer = word ###
        res = hp.HyperComposition([(0.5 , hp.HyperCrossProduct({"ngram_range":1,
                                                                "analyzer":"word",
```

(continues on next page)

(continued from previous page)

```
                                                                    "min_df":[1,0.001, 0.
↪01, 0.05],
                                                                    "max_df":[0.999, 0.99,␣
↪0.95]
                                                                    }) ),
                                        (0.5  , hp.HyperCrossProduct({
                                                    "ngram_range": hp.HyperRangeInt(start =␣
↪1,end = 4),
                                                     "analyzer": hp.HyperChoice(("char",
↪"char_wb")) ,
                                                     "min_df":[1, 0.001, 0.01, 0.05],
                                                     "max_df":[0.999, 0.99, 0.95]
                                                     }) )
                                         ])



        return res
```

This tells that the global hyperparameter is a composition between bag of word with ngram_range = 1 and bag of char with ngram between 1 and 4

(See *Hyperparameters* for detailed explanation on how to specify hyper-parameters)

8.2.2 Hyperparameters

Here you'll find the explanation of how to specify random hyper-parameters. Those hyper-parameters are used to generate random values of the parameters of a given model.

Example, to generate a random integer between 1 and 10 (included) you can use HyperRangeInt:

```
hyper = HyperRangeInt(start = 1, end = 10)
[hyper.get_rand() for _ in range(5)]

>> [4,1,10,3,3]
```

The complete list of HyperParameters are available here **:module:'aikit.ml_machine.hyper_parameters'** Each class implements a 'get_rand' method.

HyperCrossProduct

This class is used to combine hyper-parameters together which is needed to generate complexe dictionnary-like hyper-parameters

class aikit.ml_machine.hyper_parameters.**HyperCrossProduct**(*list_of_hyperparameters*, *random_state=None*)

Cartesian Product of Distribution

The list of hyperameters can be :

- dict or OrderedDict : in that case the object will draw dictionnary with them key and value from the hyper-parameters
- list or tuple : in that case the object will draw list (resp. tuple) from each element within the list (resp. tuple)

Parameters **list_of_hyperparameters** (*list of hyperameters like object*) –

Examples

```
>>> hp = HyperCrossProduct([HyperRangeInt(0,10), HyperRangeFloat(0,1)])
>>> hp.get_rand()
```

```
>>> hp = HyperCrossProduct({"int_value":HyperRangeInt(0,10), "float_value
↪":HyperRangeFloat(0,1)})
>>> hp.get_rand()
```

```
>>> hp = HyperCrossProduct({"int_value":scipy.stats.randint(0,10),
↪"float_value":HyperRangeFloat(0,1) , "choice":("a","b","c"),"constant":
↪10})
>>> hp.get_rand()
```

Example:

```
hp = HyperCrossProduct({"int_value":HyperRangeInt(0,10), "float_value":
↪HyperRangeFloat(0,1)})
hp.get_rand()
```

This will generate random dictionnary with keys 'int_value' and 'float_value'

HyperComposition

This class is used to include dependency between parameters or to create an hyper parameters from two different distributions

class aikit.ml_machine.hyper_parameters.**HyperComposition**(*dict_vals*, *random_state=None*)

Composition of Distributions : randomly choice among several distributions

the size of the values can be : * if size 1 : list of HyperParameters * if size 2 : list of weight * HyperParameters

Parameters **dict_vals** (*list or tuple of size 2 or 1*) –

Examples

```
>>> hp = HyperComposition([ HyperRangeInt(0,100) , HyperRangeInt(100,
↪1000)])
>>> hp.get_rand()
```

```
>>> hp = HyperComposition([ (0.9,HyperRangeInt(0,100)) ,(0.1,
↪HyperRangeInt(100,1000)) ])
>>> hp.get_rand()
```

```
>>> hp = HyperComposition([ (0.9,"choice_a")  ,(0.1, "choice_b") ])
>>> hp.get_rand()
```

Example:

```
hp = HyperComposition([ (0.9,HyperRangeInt(0,100)) ,(0.1,HyperRangeInt(100,1000)) ])
hp.get_rand()
```

This will generate a random number between 0 and 100 with probability 0.9 and one between 100 and 1000 with probability 0.1

```
hp = HyperComposition([
    (0.5 , HyperComposition({"analyzer":"char" , "n_gram_range":[1,4]})),
    (0.5 , HyperComposition({"analyzer":"word" , "n_gram_range":1}) )

])
hp.get_rand()
```

This will generate with probability:

- 1/2 a dictionnary with “analyzer”:”char” and “n_gram_range”: random between 1 and 4
- 1/2 a dictionnary with “analyzer”:”word” and “n_gram_range”: 1

CHAPTER 9

Contribution

You can contribute aikit source code. To do that you need to do the following steps :

1. fork aikit
2. clone the fork
3. create a branch with your new development
4. push the branch and create a pull request

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Index