Install AutoML

```
!apt-get install swig -y
!pip install Cython numpy

# sometimes you have to run the next command twice on colab
# I haven't figured out why
!pip install auto-sklearn
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
swig is already the newest version (3.0.12-1).
The following package was automatically installed and is no longer required:
  libnvidia-common-410
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: auto-sklearn in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: smac==0.8 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: ConfigSpace<0.5,>=0.4.0 in /usr/local/lib/pytho
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pynisher>=0.4.2 in /usr/local/lib/python3.6/dis
Requirement already satisfied: pyrfr<0.8,>=0.7 in /usr/local/lib/python3.6/dis
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: scikit-learn<0.20,>=0.19 in /usr/local/lib/pyth
Requirement already satisfied: xgboost>=0.80 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: lockfile in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: nose in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: psutil in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: liac-arff in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: scipy>=0.14.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: typing in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: sphinx-rtd-theme in /usr/local/lib/python3.6/dis
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: sphinx in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pyparsing in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: docutils>=0.3 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: imagesize in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: sphinxcontrib-websupport in /usr/local/lib/pyth
Requirement already satisfied: Pygments>=2.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: Jinja2>=2.3 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: snowballstemmer>=1.1 in /usr/local/lib/python3.
Requirement already satisfied: babel!=2.0,>=1.3 in /usr/local/lib/python3.6/di
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: alabaster<0.8,>=0.7 in /usr/local/lib/python3.6
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/di
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/
```

```
!pip install auto-sklearn
```

```
Requirement already satisfied: auto-sklearn in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: pynisher>=0.4.2 in /usr/local/lib/python3.6/dis
Requirement already satisfied: pyrfr<0.8,>=0.7 in /usr/local/lib/python3.6/dis
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: scikit-learn<0.20,>=0.19 in /usr/local/lib/pyth
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: lockfile in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: nose in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: scipy>=0.14.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: psutil in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: liac-arff in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: xgboost>=0.80 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: smac==0.8 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: ConfigSpace<0.5,>=0.4.0 in /usr/local/lib/pytho
Requirement already satisfied: docutils>=0.3 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: sphinx in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: sphinx-rtd-theme in /usr/local/lib/python3.6/di
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: typing in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: pyparsing in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: Pygments>=2.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: imagesize in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: alabaster<0.8,>=0.7 in /usr/local/lib/python3.6
Requirement already satisfied: Jinja2>=2.3 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: sphinxcontrib-websupport in /usr/local/lib/pyth
Requirement already satisfied: snowballstemmer>=1.1 in /usr/local/lib/python3.
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: babel!=2.0,>=1.3 in /usr/local/lib/python3.6/di
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/di
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3
```

```python
import autosklearn.classification
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/weight_boosting.py:29:
  from numpy.core.umath_tests import inner1d
```

AutoML

Functions

```python
import pandas as pd
import numpy as np
import random as rnd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn import preprocessing


#%% md
#### MyPCA
#%%
def myPCA(data,n):
    pca = PCA(n_components=n)
    pca.fit(data)
    df = pca.transform(data)
    PCA_Data = pd.DataFrame(df)
    return PCA_Data

#%% md
#### myNormalize
#%%
def myNormalize(data):
    min_max_scaler = preprocessing.MinMaxScaler()
    Normalized_Data = min_max_scaler.fit_transform(data)
    Normalized_Data = pd.DataFrame(Normalized_Data)
    return Normalized_Data

#%% md
#### myEncode
#%%
def myEncode(data,col):
    NewData_Encode = data.copy()
    NewData_Encode = pd.get_dummies(NewData_Encode, columns=col, prefix = col)
    return NewData_Encode


#%% md
#### myCleanAndTransformData
#%%
def myCleanAndTransformData(data):

    #Drop null rows
    NewData = data.dropna()
    #Remove unknown ata
    NewData = NewData[NewData['episodes']!='Unknown']
    #Add a new column rating class
    NewData['Class']=1
    # 1: High
    # or 0: Low based on rating
    NewData.loc[NewData['rating'] >= NewData['rating'].mean(), 'Class'] = 1
    NewData.loc[NewData['rating'] < NewData['rating'].mean(), 'Class'] = 0

    #Split genre values into rows
    NewData = pd.DataFrame(NewData.genre.str.split(',').tolist(), index=[NewData.anime_id
    NewData = NewData.reset_index([0,'anime_id','type','episodes','rating','members','Cla
    NewData.columns=['anime_id','type','episodes','rating','members','Class','genre']

    #Encode type feature: 6 unique values
    NewData = myEncode(NewData,['type'])

    #Encode genre feature: 82 unique values
    NewData = myEncode(NewData,['genre'])

     #Drop anmie_id,rating,Class
    NewData = NewData.drop(['rating'],axis=1)
    NewData = NewData.drop(columns=['anime_id'])
    #NewData = NewData.drop(columns=['episodes'])

    return NewData


#%% md
#### mySplitData
#%%
```

```
def mySplitData(X_Data,Y_Data,test_size,random_state):
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_Data, Y_Data, test_size=test_si
    return X_train, X_test, y_train, y_test

def mySplitDataByTrainSize(X_Data,Y_Data,train_size,random_state):
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_Data, Y_Data, train_size=train_
    X_train, X_test, y_train, y_test = mySplitData(X_train,y_train,0.33,random_state)
    return X_train, X_test, y_train, y_test
```

```
#%% md
# Load data from files
#%%
RawData = pd.read_csv('anime.csv')
RawData.head()
```

	anime_id	name	genre	t
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	M
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	
3	9253	Steins;Gate	Sci-Fi, Thriller	
4	9969	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	

```
#%% md
#### Clean and Transform Data
#%%
Cleaned_Data = myCleanAndTransformData(RawData)
Y_Data = Cleaned_Data['Class']
X_Data = Cleaned_Data.drop(columns=['Class'])

#%% md
#### Normalize  Data
#%%
Normalized_Data = myNormalize(X_Data)
#%% md
#### PCA
#%%
n_components=40
PCA_Data = myPCA(Normalized_Data,n_components)
PCA_Data.head()
```

	0	1	2	3	4	5	6	7	
0	-0.311566	0.786508	-0.420821	0.005236	-0.078664	-0.049645	-0.062636	0.007171	-
1	-0.284842	0.763991	-0.412009	-0.010872	-0.110067	-0.087028	-0.096769	0.054629	-
2	-0.284838	0.767910	-0.395570	-0.007614	-0.091869	-0.059765	-0.062085	0.036505	-
3	-0.291600	0.777175	-0.408316	0.000301	-0.080828	-0.049799	-0.056889	0.019143	-
4	0.732145	-0.153155	-0.102203	-0.458230	0.816867	0.046174	0.015773	-0.064781	

```
#%% md
####-----------------------------------------------------------------
#### Split  PCA_Data
####-----------------------------------------------------------------
```

```
#%%
PCA_X_train, PCA_X_test, PCA_y_train, PCA_y_test  = mySplitData(PCA_Data,Y_Data,0.33,42)

PCA_X_train.head()
#%%
PCA_X_test.head()
#%%
PCA_y_train.head()
#%%
PCA_y_test.head()
```

```
22373    0
10508    1
11570    1
22262    0
734      1
Name: Class, dtype: int64
```

Train and Test Model

```
# configure auto-sklearn
anmie_automl = autosklearn.classification.AutoSklearnClassifier(
          time_left_for_this_task=120, # run auto-sklearn for at most 2min
          per_run_time_limit=30, # spend at most 30 sec for each model training
          include_preprocessors=["no_preprocessing"],
          )

# train model(s)
anmie_automl.fit(PCA_X_train, PCA_y_train)

from sklearn.metrics import accuracy_score
PCA_y_predicted = anmie_automl.predict(PCA_X_test)
test_acc = accuracy_score(PCA_y_test, PCA_y_predicted)

print("Test Accuracy score {0}".format(test_acc))
```

```
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
[WARNING] [2019-07-31 16:40:50,472:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 16:40:50,483:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 16:40:52,490:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 16:40:54,495:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 16:40:56,507:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 16:40:58,517:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
1
['/tmp/autosklearn_tmp_137_7300/.auto-sklearn/ensembles/1.0000000000.ensemble'
Test Accuracy score 0.804424550228114
```

Inspecting the results

```
# evaluate
from sklearn.metrics import accuracy_score
PCA_y_predicted = anmie_automl.predict(PCA_X_test)
test_acc = accuracy_score(PCA_y_test, PCA_y_predicted)

print("Test Accuracy score {0}".format(test_acc))
```

```
Test Accuracy score 0.804424550228114
```

```
anmie_automl.sprint_statistics()
```

```
'auto-sklearn results:\n  Dataset name: 64b53ca9ba24ac1e45ad29eb0951a812\n  Me
```

```
anmie_automl.show_models()
```

```
e_hot_encoding', 'classifier:__choice__': 'random_forest', 'imputation:strategy'
```

```
anmie_automl.cv_results_
```

```
{'mean_fit_time': array([ 9.36614823,  7.68715644, 30.03748894, 11.33386064,
        29.3589673 , 18.02530432]),
 'mean_test_score': array([0.75960427, 0.79660799, 0.        , 0.75767699, 0.6
        0.78350251, 0.        ]),
 'param_balancing:strategy': masked_array(data=['none', 'none', 'none', 'weigh
                    'weighting', 'none'],
              mask=[False, False, False, False, False, False, False],
        fill_value='N/A',
             dtype='<U9'),
 'param_categorical_encoding:__choice__': masked_array(data=['one_hot_encoding
                    'one_hot_encoding', 'one_hot_encoding',
                    'one_hot_encoding', 'one_hot_encoding',
                    'one_hot_encoding'],
              mask=[False, False, False, False, False, False, False],
        fill_value='N/A',
             dtype='<U16'),
 'param_categorical_encoding:one_hot_encoding:minimum_fraction': masked_array(
                    0.00034835629696198427, 0.010000000000000004, --],
              mask=[False, False,  True, False, False, False,  True],
        fill_value=1e+20),
 'param_categorical_encoding:one_hot_encoding:use_minimum_fraction': masked_ar
                    'False'],
              mask=[False, False, False, False, False, False, False],
        fill_value='N/A',
             dtype='<U5'),
 'param_classifier:__choice__': masked_array(data=['random_forest', 'random_fo
                    'random_forest', 'gaussian_nb', 'gradient_boosting',
                    'random_forest'],
              mask=[False, False, False, False, False, False, False],
        fill_value='N/A',
             dtype='<U17'),
 'param_classifier:adaboost:algorithm': masked_array(data=[--, --, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:adaboost:learning_rate': masked_array(data=[--, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:adaboost:max_depth': masked_array(data=[--, --, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:adaboost:n_estimators': masked_array(data=[--, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:bernoulli_nb:alpha': masked_array(data=[--, --, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:bernoulli_nb:fit_prior': masked_array(data=[--, --, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:criterion': masked_array(data=[--, --, --, --
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:max_depth_factor': masked_array(data=[--, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:max_features': masked_array(data=[--, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:max_leaf_nodes': masked_array(data=[--, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:min_impurity_decrease': masked_array(data=[--
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:min_samples_leaf': masked_array(data=[--, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:min_samples_split': masked_array(data=[--, --
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:decision_tree:min_weight_fraction_leaf': masked_array(data=
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:bootstrap': masked_array(data=[--, --, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:criterion': masked_array(data=[--, --, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:max_depth': masked_array(data=[--, --, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:max_features': masked_array(data=[--, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:max_leaf_nodes': masked_array(data=[--, --, --,
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:min_impurity_decrease': masked_array(data=[--, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:min_samples_leaf': masked_array(data=[--, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:min_samples_split': masked_array(data=[--, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:min_weight_fraction_leaf': masked_array(data=[-
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:extra_trees:n_estimators': masked_array(data=[--, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:gradient_boosting:criterion': masked_array(data=[--, --, --
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value='N/A',
             dtype='<U32'),
 'param_classifier:gradient_boosting:learning_rate': masked_array(data=[--, --
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:loss': masked_array(data=[--, --, --, --,
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value='N/A',
             dtype='<U32'),
 'param_classifier:gradient_boosting:max_depth': masked_array(data=[--, --, --
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:max_features': masked_array(data=[--, --,
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:max_leaf_nodes': masked_array(data=[--, -
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value='N/A',
             dtype='<U32'),
 'param_classifier:gradient_boosting:min_impurity_decrease': masked_array(data=
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:min_samples_leaf': masked_array(data=[--,
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:min_samples_split': masked_array(data=[--
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:min_weight_fraction_leaf': masked_array(da
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:n_estimators': masked_array(data=[--, --,
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:gradient_boosting:subsample': masked_array(data=[--, --, --
              mask=[ True,  True,  True,  True,  True, False,  True],
        fill_value=1e+20),
 'param_classifier:k_nearest_neighbors:n_neighbors': masked_array(data=[--, --
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:k_nearest_neighbors:p': masked_array(data=[--, --, --, --, 
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:k_nearest_neighbors:weights': masked_array(data=[--, --, --
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:lda:n_components': masked_array(data=[--, --, --, --, --, -
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20,
             dtype=float64),
 'param_classifier:lda:shrinkage': masked_array(data=[--, --, --, --, --, --, 
              mask=[ True,  True,  True,  True,  True,  True,  True],
        fill_value=1e+20
```

```
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:lda:shrinkage_factor': masked_array(data=[--, --, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:lda:tol': masked_array(data=[--, --, --, --, --, --, --],
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:C': masked_array(data=[--, --, --, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:dual': masked_array(data=[--, --, --, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:fit_intercept': masked_array(data=[--, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:intercept_scaling': masked_array(data=[--, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:loss': masked_array(data=[--, --, --, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:multi_class': masked_array(data=[--, --, --, 
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:penalty': masked_array(data=[--, --, --, --, 
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:liblinear_svc:tol': masked_array(data=[--, --, --, --, --, 
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:libsvm_svc:C': masked_array(data=[--, --, 6.342897164595882
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value=1e+20),
        'param_classifier:libsvm_svc:coef0': masked_array(data=[--, --, --, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:libsvm_svc:degree': masked_array(data=[--, --, --, --, --, 
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:libsvm_svc:gamma': masked_array(data=[--, --, 0.22298706233
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value=1e+20),
        'param_classifier:libsvm_svc:kernel': masked_array(data=[--, --, 'rbf', --, -
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value='N/A',
                    dtype='<U32'),
        'param_classifier:libsvm_svc:max_iter': masked_array(data=[--, --, -1.0, --, 
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value=1e+20),
        'param_classifier:libsvm_svc:shrinking': masked_array(data=[--, --, 'False', 
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value='N/A',
                    dtype='<U32'),
        'param_classifier:libsvm_svc:tol': masked_array(data=[--, --, 2.0063452643810
                     mask=[ True,  True, False,  True,  True,  True,  True],
               fill_value=1e+20),
        'param_classifier:multinomial_nb:alpha': masked_array(data=[--, --, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:multinomial_nb:fit_prior': masked_array(data=[--, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:passive_aggressive:C': masked_array(data=[--, --, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:passive_aggressive:average': masked_array(data=[--, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:passive_aggressive:fit_intercept': masked_array(data=[--, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:passive_aggressive:loss': masked_array(data=[--, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:passive_aggressive:tol': masked_array(data=[--, --, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:qda:reg_param': masked_array(data=[--, --, --, --, --, --, 
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:random_forest:bootstrap': masked_array(data=['True', 'True'
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value='N/A',
                    dtype='<U4'),
        'param_classifier:random_forest:criterion': masked_array(data=['gini', 'gini'
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value='N/A',
                    dtype='<U4'),
        'param_classifier:random_forest:max_depth': masked_array(data=['None', 'None'
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value='N/A',
                    dtype='<U4'),
        'param_classifier:random_forest:max_features': masked_array(data=[0.5, 0.5240
                           --, 0.9260795160807372],
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:random_forest:max_leaf_nodes': masked_array(data=['None', '
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value='N/A',
                    dtype='<U4'),
        'param_classifier:random_forest:min_impurity_decrease': masked_array(data=[0.
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:random_forest:min_samples_leaf': masked_array(data=[1.0, 10
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:random_forest:min_samples_split': masked_array(data=[2.0, 1
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:random_forest:min_weight_fraction_leaf': masked_array(data=
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:random_forest:n_estimators': masked_array(data=[100.0, 100.
                     mask=[False, False,  True, False,  True,  True, False],
               fill_value=1e+20),
        'param_classifier:sgd:alpha': masked_array(data=[--, --, --, --, --, --, --],
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:average': masked_array(data=[--, --, --, --, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:epsilon': masked_array(data=[--, --, --, --, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:eta0': masked_array(data=[--, --, --, --, --, --, --],
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:fit_intercept': masked_array(data=[--, --, --, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:l1_ratio': masked_array(data=[--, --, --, --, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:learning_rate': masked_array(data=[--, --, --, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:loss': masked_array(data=[--, --, --, --, --, --, --],
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:penalty': masked_array(data=[--, --, --, --, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:power_t': masked_array(data=[--, --, --, --, --, --, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:sgd:tol': masked_array(data=[--, --, --, --, --, --, --],
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:base_score': masked_array(data=[--, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:booster': masked_array(data=[
         'param_classifier:xgradient_boosting:booster': masked_array(data=[--, --, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:colsample_bylevel': masked_array(data=[-
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:colsample_bytree': masked_array(data=[--
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:gamma': masked_array(data=[--, --, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:learning_rate': masked_array(data=[--, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:max_delta_step': masked_array(data=[--,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:max_depth': masked_array(data=[--, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:min_child_weight': masked_array(data=[--
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:n_estimators': masked_array(data=[--, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:normalize_type': masked_array(data=[--,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:rate_drop': masked_array(data=[--, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:reg_alpha': masked_array(data=[--, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:reg_lambda': masked_array(data=[--, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:sample_type': masked_array(data=[--, --,
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:scale_pos_weight': masked_array(data=[--
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_classifier:xgradient_boosting:subsample': masked_array(data=[--, --, -
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_imputation:strategy': masked_array(data=['mean', 'mean', 'most_frequen
                           'mean', 'mean', 'mean'],
                     mask=[False, False, False, False, False, False, False],
               fill_value='N/A',
                    dtype='<U13'),
        'param_preprocessor:__choice__': masked_array(data=['no_preprocessing', 'no_p
                           'no_preprocessing', 'no_preprocessing',
                           'no_preprocessing', 'no_preprocessing',
                           'no_preprocessing'],
                     mask=[False, False, False, False, False, False, False],
               fill_value='N/A',
                    dtype='<U16'),
        'param_rescaling:__choice__': masked_array(data=['standardize', 'normalize',
                           'standardize', 'robust_scaler', 'standardize',
                           'minmax'],
                     mask=[False, False, False, False, False, False, False],
               fill_value='N/A',
                    dtype='<U13'),
        'param_rescaling:quantile_transformer:n_quantiles': masked_array(data=[--, --
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_rescaling:quantile_transformer:output_distribution': masked_array(data
                     mask=[ True,  True,  True,  True,  True,  True,  True],
               fill_value=1e+20,
                    dtype=float64),
        'param_rescaling:robust_scaler:q_max': masked_array(data=[--, --, --, --, 0.8
                     mask=[ True,  True,  True,  True, False,  True,  True],
               fill_value=1e+20),
        'param_rescaling:robust_scaler:q_min': masked_array(data=[--, --, --, --, 0.0
                     mask=[ True,  True,  True,  True, False,  True,  True],
               fill_value=1e+20),
        'params': [{'balancing:strategy': 'none',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:minimum_fraction': 0.01,
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
          'classifier:__choice__': 'random_forest',
          'classifier:random_forest:bootstrap': 'True',
          'classifier:random_forest:criterion': 'gini',
          'classifier:random_forest:max_depth': 'None',
          'classifier:random_forest:max_features': 0.5,
          'classifier:random_forest:max_leaf_nodes': 'None',
          'classifier:random_forest:min_impurity_decrease': 0.0,
          'classifier:random_forest:min_samples_leaf': 1,
          'classifier:random_forest:min_samples_split': 2,
          'classifier:random_forest:min_weight_fraction_leaf': 0.0,
          'classifier:random_forest:n_estimators': 100,
          'imputation:strategy': 'mean',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'standardize'},
         {'balancing:strategy': 'none',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:minimum_fraction': 0.000125865724289
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
          'classifier:__choice__': 'random_forest',
          'classifier:random_forest:bootstrap': 'True',
          'classifier:random_forest:criterion': 'gini',
          'classifier:random_forest:max_depth': 'None',
          'classifier:random_forest:max_features': 0.5240592829918601,
          'classifier:random_forest:max_leaf_nodes': 'None',
          'classifier:random_forest:min_impurity_decrease': 0.0,
          'classifier:random_forest:min_samples_leaf': 10,
          'classifier:random_forest:min_samples_split': 16,
          'classifier:random_forest:min_weight_fraction_leaf': 0.0,
          'classifier:random_forest:n_estimators': 100,
          'imputation:strategy': 'mean',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'normalize'},
         {'balancing:strategy': 'none',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
          'classifier:__choice__': 'libsvm_svc',
          'classifier:libsvm_svc:C': 6.342897164595882,
          'classifier:libsvm_svc:gamma': 0.2229870623330047,
          'classifier:libsvm_svc:kernel': 'rbf',
          'classifier:libsvm_svc:max_iter': -1,
          'classifier:libsvm_svc:shrinking': 'False',
          'classifier:libsvm_svc:tol': 2.006345264381097e-05,
          'imputation:strategy': 'most_frequent',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'standardize'},
         {'balancing:strategy': 'weighting',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:minimum_fraction': 0.410946144307535
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
          'classifier:__choice__': 'random_forest',
          'classifier:random_forest:bootstrap': 'True',
          'classifier:random_forest:criterion': 'gini',
          'classifier:random_forest:max_depth': 'None',
          'classifier:random_forest:max_features': 0.5686453602598863,
          'classifier:random_forest:max_leaf_nodes': 'None',
          'classifier:random_forest:min_impurity_decrease': 0.0,
          'classifier:random_forest:min_samples_leaf': 1,
          'classifier:random_forest:min_samples_split': 2,
          'classifier:random_forest:min_weight_fraction_leaf': 0.0,
          'classifier:random_forest:n_estimators': 100,
          'imputation:strategy': 'most_frequent',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'standardize'},
         {'balancing:strategy': 'weighting',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:minimum_fraction': 0.000348356296961
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
          'classifier:__choice__': 'gaussian_nb',
          'imputation:strategy': 'mean',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'robust_scaler',
          'rescaling:robust_scaler:q_max': 0.8245132980938538,
          'rescaling:robust_scaler:q_min': 0.08947420373097192},
         {'balancing:strategy': 'weighting',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:minimum_fraction': 0.010000000000000
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
          'classifier:__choice__': 'gradient_boosting',
          'classifier:gradient_boosting:criterion': 'mse',
          'classifier:gradient_boosting:learning_rate': 0.051832615669195795,
          'classifier:gradient_boosting:loss': 'deviance',
          'classifier:gradient_boosting:max_depth': 6,
          'classifier:gradient_boosting:max_features': 0.8807456180216267,
          'classifier:gradient_boosting:max_leaf_nodes': 'None',
          'classifier:gradient_boosting:min_impurity_decrease': 0.0,
          'classifier:gradient_boosting:min_samples_leaf': 7,
          'classifier:gradient_boosting:min_samples_split': 19,
          'classifier:gradient_boosting:min_weight_fraction_leaf': 0.0,
          'classifier:gradient_boosting:n_estimators': 366,
          'classifier:gradient_boosting:subsample': 0.7314831276137047,
          'imputation:strategy': 'mean',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'standardize'},
         {'balancing:strategy': 'none',
          'categorical_encoding:__choice__': 'one_hot_encoding',
          'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
          'classifier:__choice__': 'random_forest',
          'classifier:random_forest:bootstrap': 'True',
          'classifier:random_forest:criterion': 'gini',
          'classifier:random_forest:max_depth': 'None',
          'classifier:random_forest:max_features': 0.9260795160807372,
          'classifier:random_forest:max_leaf_nodes': 'None',
          'classifier:random_forest:min_impurity_decrease': 0.0,
          'classifier:random_forest:min_samples_leaf': 17,
          'classifier:random_forest:min_samples_split': 7,
          'classifier:random_forest:min_weight_fraction_leaf': 0.0,
          'classifier:random_forest:n_estimators': 100,
          'imputation:strategy': 'mean',
          'preprocessor:__choice__': 'no_preprocessing',
          'rescaling:__choice__': 'minmax'}],
        'rank_test_scores': array([3, 1, 6, 4, 5, 2, 6]),
        'status': ['Success',
         'Success',
         'Timeout',
         'Success',
         'Success',
         'Success',
         'Timeout']}
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=9)

# train model(s)
knn_m = knn.fit(PCA_X_train, PCA_y_train)

# evaluate
knn_test_acc = knn_m.score(PCA_X_test,PCA_y_test)
print("Test Accuracy score {0}".format(knn_test_acc))
```

```
Test Accuracy score 0.797279848497891
```

SVM

```
#Import svm model
from sklearn import svm

# Create a svm Classifier with PCA data
```

```
svc = svm.SVC(C=1.0, gamma=0.1, kernel='rbf') # Linear Kernel

# train model(s)
svm_m = svc.fit(PCA_X_train, PCA_y_train)

# evaluate
svm_test_acc = svm_m.score(PCA_X_test,PCA_y_test)
print("Test Accuracy score {0}".format(svm_test_acc))
```

```
Test Accuracy score 0.6980287509684083
```

Decision Tree

```
#Import svm model
from sklearn.tree import DecisionTreeClassifier


# Create a DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=0,max_depth=30,min_samples_leaf=20)


# train model(s)
dt_m = dt.fit(PCA_X_train, PCA_y_train)

# evaluate
dt_test_acc = dt_m.score(PCA_X_test,PCA_y_test)
print("Test Accuracy score {0}".format(dt_test_acc))
```

```
Test Accuracy score 0.7936644572609107
```

Random Forest

```
#Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest Classifier with original data
rf = RandomForestClassifier(criterion ='gini', max_depth= 15, max_features= 'sqrt', min_s


# train model(s)
rf_m = rf.fit(PCA_X_train, PCA_y_train)

# evaluate
rf_test_acc = rf_m.score(PCA_X_test,PCA_y_test)
print("Test Accuracy score {0}".format(rf_test_acc))
```

```
Test Accuracy score 0.8053714384092279
```

Neural Network

```
#Import svm model
from sklearn.neural_network import MLPClassifier

# Create a NN Classifier with PCA data
nn = MLPClassifier(max_iter=500)
```

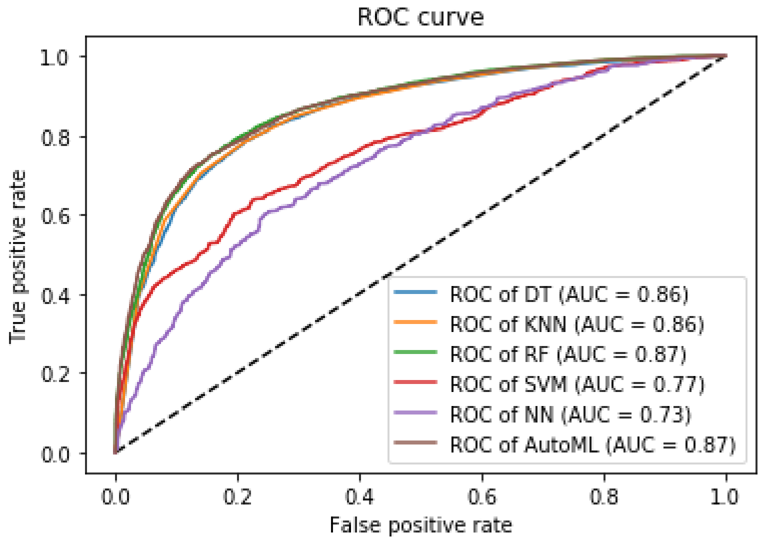
```
# train model(s)
nn_m = nn.fit(PCA_X_train, PCA_y_train)

# evaluate
nn_test_acc = nn_m.score(PCA_X_test,PCA_y_test)
print("Test Accuracy score {0}".format(nn_test_acc))
```

```
Test Accuracy score 0.7021606266678144
```

Comparison between all classifiers (including AutoML)

```
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
import autosklearn.classification

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve,auc

def roc_auc_curve(model,X_train,Y_train,X_test,Y_test):
  model.fit(X_train, Y_train)

  dt_lm = LogisticRegression(solver='lbfgs', max_iter=1000)
  dt_lm.fit(X_test, Y_test)

  y_pred_dt = model.predict_proba(X_test)[:, 1]
  fpr_dt, tpr_dt, _ = roc_curve(Y_test, y_pred_dt)
  roc_auc = auc(fpr_dt, tpr_dt)

  test_acc_score = model.score(X_test,Y_test)

  return fpr_dt, tpr_dt,roc_auc,test_acc_score

# Create a DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0,max_depth=30,min_samples_leaf=20)
fpr_dt, tpr_dt,roc_auc,dt_test_acc_score = roc_auc_curve(clf,PCA_X_train,PCA_y_train,PCA_


#KNN
knn = KNeighborsClassifier(n_neighbors=9)
knn_fpr, knn_tpr,knn_roc_auc,knn_test_acc_score = roc_auc_curve(knn,PCA_X_train,PCA_y_tra

#Random Forest
rf = RandomForestClassifier(criterion ='gini', max_depth= 15, max_features= 'sqrt', min_s
rf_fpr, rf_tpr,rf_roc_auc,rf_test_acc_score = roc_auc_curve(rf,PCA_X_train,PCA_y_train,PC

#svm
svmModel = svm.SVC(C=1.0, gamma=0.1, kernel='rbf',probability=True) # Linear Kernel
svm_fpr, svm_tpr,svm_roc_auc,svm_test_acc_score = roc_auc_curve(svmModel,PCA_X_train,PCA_

#NN
nn = MLPClassifier(alpha= 0.05, hidden_layer_sizes =(50, 100, 50),max_iter=500)
nn_fpr, nn_tpr,nn_roc_auc,nn_test_acc_score = roc_auc_curve(nn,PCA_X_train,PCA_y_train,PC

#AutoML
AutoML = autosklearn.classification.AutoSklearnClassifier(
          time_left_for_this_task=120, # run auto-sklearn for at most 2min
          per_run_time_limit=30, # spend at most 30 sec for each model training
          include_preprocessors=["no_preprocessing"]
          )
AutoML_fpr, AutoML_tpr,AutoML_roc_auc,AutoML_test_acc_score = roc_auc_curve(AutoML,PCA_X_


plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')

plt.plot(fpr_dt, tpr_dt, label='ROC of DT (AUC = %0.2f)' % roc_auc)
plt.plot(knn_fpr, knn_tpr, label='ROC of KNN (AUC = %0.2f)' % knn_roc_auc)
plt.plot(rf_fpr, rf_tpr, label='ROC of RF (AUC = %0.2f)' % rf_roc_auc)
plt.plot(svm_fpr, svm_tpr, label='ROC of SVM (AUC = %0.2f)' % svm_roc_auc)
plt.plot(nn_fpr, nn_tpr, label='ROC of NN (AUC = %0.2f)' % nn_roc_auc)
plt.plot(AutoML_fpr, AutoML_tpr, label='ROC of AutoML (AUC = %0.2f)' % AutoML_roc_auc)


plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
[WARNING] [2019-07-31 15:57:54,019:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 15:57:54,031:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 15:57:56,035:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 15:57:58,040:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 15:58:00,051:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
[WARNING] [2019-07-31 15:58:02,063:EnsembleBuilder(1):64b53ca9ba24ac1e45ad29eb
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.]
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
1
['/tmp/autosklearn_tmp_137_3027/.auto-sklearn/ensembles/1.0000000000.ensemble'
```



Testing Accuracy Scores of all classifiers

```
scores = pd.DataFrame({"score":[dt_test_acc_score,
                                knn_test_acc_score,
                                rf_test_acc_score,
                                svm_test_acc_score,
                                nn_test_acc_score,
                                AutoML_test_acc_score],
                              "name":["dt_test_acc_score",
                                "knn_test_acc_score",
                                "rf_test_acc_score",
                                "svm_test_acc_score",
                                "nn_test_acc_score",
                                "AutoML_test_acc_score"]})
scores.sort_values(by=['score'])
scores
```

	score	name
0	0.793664	dt_test_acc_score
1	0.797280	knn_test_acc_score
2	0.805371	rf_test_acc_score
3	0.698029	svm_test_acc_score
4	0.705001	nn_test_acc_score
5	0.804425	AutoML_test_acc_score