train documentation

train Documentation

- train_and_pickle

built with pdoc

train

View Source

```
__authors__ = 'Abdullah + Vinayak'

import pickle
import pandas as pd
import time
from sklearn.metrics import classification_report
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from pprint import pprint
from sklearn.neural_network import MLPClassifier
from MLSOTE import *




def train_and_pickle(model, name, X_train, y_train):
    """
    The function calls the models and trains them with the training dataset and then saves t
    in a pickle file

    :param model:
    :param name: the classifier name
    :param X_train: The training inputs (the feature texts)
```

```
    :param y_train: The expected training outputs (the refactoring labels)
    :return: The training time for each model along with its training accuracy.
    """
    model.fit(X_train, y_train)
    y_pred = model.predict(X_train)
    print(classification_report(y_pred, y_train))
    print('Done with the training ')
    with open('../models/' + name + '.pickle', 'wb') as f:
        pickle.dump(model, f)


if __name__ == "__main__":
    """
    main method to use the split train data set and send it to train_and_pickle method with
    to be trained and generate  the classification_report and lastly pickle the model.
    Also, there is an option to train the data set with MLSMOTE function for the data imbala
    just uncomment the line where MLSMOTE methods is called.

    """
    start = time.time()
    # Load data
    X_train = pd.read_csv(r'../Data/Train_Features.csv', index_col=0)
    y_train = pd.read_csv(r'../Data/Train_Labels.csv', index_col=0)

    #Uncomment here to apply MLSOTE
    #Getting minority instance of that datframe
    #X_sub, y_sub = get_minority_instace(X_train, y_train)
    #Applying MLSMOTE to augment the dataframe
    #X_res,y_res =MLSMOTE(X_sub, y_sub, 1000)



    # perapare the classifiers with their parameters to be send to train_and_pickle
    classifiers = [{'classifierName': 'RF', 'classifier':RandomForestClassifier(class_weight
                   {'classifierName': 'SVM', 'classifier': svm.SVC()},
                   {'classifierName': 'LR', 'classifier': LogisticRegression()} ,
                   {'classifierName': 'MLP', 'classifier': MLPClassifier(random_state=1, max
                   {'classifierName': 'MNB', 'classifier':MultinomialNB()}]

    # send the classifiers one by one using for loop
    for x in classifiers:
        print(x['classifierName'], x['classifier'])
        classifierChain = ClassifierChain(x['classifier'])
        #before the data balancing
        train_and_pickle(classifierChain, x['classifierName'], X_train, y_train)
```

```
        #use this comment with data balancing MLSMOTE
        #train_and_pickle(classifierChain, x['classifierName'], X_res, y_res)
        print((time.time() - start), 'sec')
```

def train_and_pickle(model, name, X_train, y_train):

View Source

```
def train_and_pickle(model, name, X_train, y_train):
    """
    The function calls the models and trains them with the training dataset and then saves t
    in a pickle file

    :param model:
    :param name: the classifier name
    :param X_train: The training inputs (the feature texts)
    :param y_train: The expected training outputs (the refactoring labels)
    :return: The training time for each model along with its training accuracy.
    """
    model.fit(X_train, y_train)
    y_pred = model.predict(X_train)
    print(classification_report(y_pred, y_train))
    print('Done with the training ')
    with open('../models/' + name + '.pickle', 'wb') as f:
        pickle.dump(model, f)
```

The function calls the models and trains them with the training dataset and then saves the trained model and the weights in a pickle file

:param model: :param name: the classifier name :param X_train: The training inputs (the feature texts) :param y_train: The expected training outputs (the refactoring labels) :return: The training time for each model along with its training accuracy.