Tuning documentation

Tuning Documentation

- RandomizedSearch
- Gridsearch
- CustomTuning

built with pdoc

Tuning

View Source

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
import pandas as pd
from skmultilearn.problem_transform import ClassifierChain
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from pprint import pprint


def RandomizedSearch(X_train,y_train):
    """
        Random Search great approach for checking every value combination from a
        predefined list and evaluating the result of each combination.
        On the other hand ,the Random Search method tries random
        combinations of a range of values to discover new hyper parameter
        combinations to find the best parameter solution.We tuned the random forest model
        since this model gave us the best F1 SCORE. We defined  a list pf random forest para
        and let the RandomizedSearch iterate through them to find best score.

    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """
```

```
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
    max_features = ['auto', 'sqrt']
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    min_samples_split = [2, 5, 10]
    min_samples_leaf = [1, 2, 4]
    bootstrap = [True, False]
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'bootstrap': bootstrap}
    pprint(random_grid)

    # Use the random grid to search for best hyperparameters
    # First create the base model to tune
    clf=RandomForestClassifier()
    # Random search of parameters, using 3 fold cross validation,
    rf_random = RandomizedSearchCV(clf, random_grid, n_iter = 100, cv = 3,verbose=2, random_
    # Fit the random search model
    rf_random.fit(X_train, y_train)

    print(rf_random.best_params_)

def Gridsearch(X_train,y_train):
    """
    Grid search is a great approach for checking every value combination from a
    predefined list and evaluating the result of each combination.
    We have combined the two methods together to find the best
    parameter that will enhance the predictions' results. First, we
    utilized a random search with a wide range of parameters.
    Second, the best parameters found by the random search will
    be used as an input to the Grid search. This approach helps
    to reduce the execution time of the Grid search.
    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """

    param_grid = {
            'bootstrap': [False],
            'max_depth': [None],
            'max_features': [2,3],
            'min_samples_leaf': [1, 3, 5],
```

```
            'min_samples_split': [2, 4, 8],
            'n_estimators': [100, 200, 300, 1000]
        }
    # Create a based model
    rf =RandomForestClassifier()
    # Instantiate the grid search model
    grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                               cv = 3, n_jobs = -1, verbose = 2)


    grid_search.fit(X_train, y_train)

    print(grid_search.best_params_)

def CustomTuning(X_train,y_train):
    """
    We have tried another solution to find
    the best parameters for our two models that achieved the best
    results random forest .We built our custom function which will
    iterate through predefined parameter values of a specific model
    and try every combination. The function will evaluate and
    present the Accuracy, Recall, Precision, and F1 Scores metric
    for each combination
    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """

    max_features = ['auto', 'sqrt']
    bootstrap = [True, False]
    min_samples_split = [2, 5, 10]
    min_samples_leaf = [1, 2, 4]
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    best_f1 = 0
    best_model = None
    for m in max_features:
        for b in bootstrap:
            for min_sp in min_samples_split:
                for min_leaf in min_samples_leaf:
                    for n in n_estimators:
                        for max_d in max_depth:
                            classifier = ClassifierChain(RandomForestClassifier(
                                n_estimators= n,min_samples_split=min_sp,min_samples_leaf=mi
                                ,max_depth=max_d,bootstrap=b))
```

```
                            classifier.fit(X_train, y_train)
                            predictions = classifier.predict(X_test)
                            if f1_score(y_test,predictions,average='micro') >= best_f1:
                                best_f1 = f1_score(y_test,predictions,average='micro')
                                print(f'the parameter of best model is n_estimators = {n} mi
                                best_model = classifier

if __name__ == "__main__":
    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)
    X_test = pd.read_csv(r'../Data/Test_Features.csv', index_col=0)
    y_test = pd.read_csv(r'../Data/Test_Labels.csv', index_col=0)

    #pass the data to the one/all methods after retrived them from excel sheet
    #RandomizedSearch(X_train,y_train)
    #Gridsearch(X_train,y_train)
    CustomTuning(X_train,y_train)
```

def RandomizedSearch(X_train, y_train):

View Source

```
def RandomizedSearch(X_train,y_train):
    """
        Random Search great approach for checking every value combination from a
        predefined list and evaluating the result of each combination.
        On the other hand ,the Random Search method tries random
        combinations of a range of values to discover new hyper parameter
        combinations to find the best parameter solution.We tuned the random forest model
        since this model gave us the best F1 SCORE. We defined  a list pf random forest para
        and let the RandomizedSearch iterate through them to find best score.

    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """

    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
    max_features = ['auto', 'sqrt']
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    min_samples_split = [2, 5, 10]
    min_samples_leaf = [1, 2, 4]
    bootstrap = [True, False]
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
```

```
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf,
                'bootstrap': bootstrap}
    pprint(random_grid)

    # Use the random grid to search for best hyperparameters
    # First create the base model to tune
    clf=RandomForestClassifier()
    # Random search of parameters, using 3 fold cross validation,
    rf_random = RandomizedSearchCV(clf, random_grid, n_iter = 100, cv = 3,verbose=2, random_
    # Fit the random search model
    rf_random.fit(X_train, y_train)

    print(rf_random.best_params_)
```

Random Search great approach for checking every value combination from a predefined list and evaluating the result of each combination. On the other hand ,the Random Search method tries random combinations of a range of values to discover new hyper parameter combinations to find the best parameter solution.We tuned the random forest model since this model gave us the best F1 SCORE. We defined a list pf random forest parameters and let the RandomizedSearch iterate through them to find best score.

:param X_train:Train_Features :param y_train:Train_labels :return: list of the best parameters that achived the highest result.

def Gridsearch(X_train, y_train):

View Source

```
def Gridsearch(X_train,y_train):
    """
    Grid search is a great approach for checking every value combination from a
    predefined list and evaluating the result of each combination.
    We have combined the two methods together to find the best
    parameter that will enhance the predictions' results. First, we
    utilized a random search with a wide range of parameters.
    Second, the best parameters found by the random search will
    be used as an input to the Grid search. This approach helps
    to reduce the execution time of the Grid search.
    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """

    param_grid = {
            'bootstrap': [False],
            'max_depth': [None],
```

```
        'max_features': [2,3],
        'min_samples_leaf': [1, 3, 5],
        'min_samples_split': [2, 4, 8],
        'n_estimators': [100, 200, 300, 1000]
    }
# Create a based model
rf =RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)


grid_search.fit(X_train, y_train)

print(grid_search.best_params_)
```

Grid search is a great approach for checking every value combination from a predefined list and evaluating the result of each combination. We have combined the two methods together to find the best parameter that will enhance the predictions' results. First, we utilized a random search with a wide range of parameters. Second, the best parameters found by the random search will be used as an input to the Grid search. This approach helps to reduce the execution time of the Grid search. :param X_train:Train_Features :param y_train:Train_labels :return: list of the best parameters that achived the highest result.

def CustomTuning(X_train, y_train):

View Source

```
def CustomTuning(X_train,y_train):
    """
    We have tried another solution to find
    the best parameters for our two models that achieved the best
    results random forest .We built our custom function which will
    iterate through predefined parameter values of a specific model
    and try every combination. The function will evaluate and
    present the Accuracy, Recall, Precision, and F1 Scores metric
    for each combination
    :param X_train:Train_Features
    :param y_train:Train_labels
    :return: list of the best parameters that achived the highest result.
    """

    max_features = ['auto', 'sqrt']
    bootstrap = [True, False]
    min_samples_split = [2, 5, 10]
    min_samples_leaf = [1, 2, 4]
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
best_f1 = 0
best_model = None
for m in max_features:
    for b in bootstrap:
        for min_sp in min_samples_split:
            for min_leaf in min_samples_leaf:
                for n in n_estimators:
                    for max_d in max_depth:
                        classifier = ClassifierChain(RandomForestClassifier(
                            n_estimators= n,min_samples_split=min_sp,min_samples_leaf=mi
                            ,max_depth=max_d,bootstrap=b))
                        classifier.fit(X_train, y_train)
                        predictions = classifier.predict(X_test)
                        if f1_score(y_test,predictions,average='micro') >= best_f1:
                            best_f1 = f1_score(y_test,predictions,average='micro')
                            print(f'the parameter of best model is n_estimators = {n} mi
                            best_model = classifier
```

We have tried another solution to find the best parameters for our two models that achieved the best results random forest .We built our custom function which will iterate through predefined parameter values of a specific model and try every combination. The function will evaluate and present the Accuracy, Recall, Precision, and F1 Scores metric for each combination :param X_train:Train_Features :param y_train:Train_labels :return: list of the best parameters that achived the highest result.