Random Forest - BoW

May 10, 2024

1 Initialization

Connect to Google Drive:

[ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')

# %cd '/content/drive/MyDrive/GitHub/emotion-dectection-from-text'
```

Preparing necessary packages:

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import accuracy_score
from joblib import dump

from preset_function import evaluate_model, draw_learning_curve,␣
 ↪load_processed_data

X_train_bow, X_test_bow, X_train_tfidf, X_test_tfidf, \
    X_train_bow_L1, X_test_bow_L1, X_train_tfidf_L1, X_test_tfidf_L1=␣
 ↪load_processed_data('input')

y_train, y_test = load_processed_data('output')

%matplotlib inline
```

1.1 Select dataset

At first, we choose the dataset to be used for training and testing the model.

[ ]:
```
X_train = X_train_bow
X_test = X_test_bow
```

2 Basic training

We define the model with the default parameters and train it.

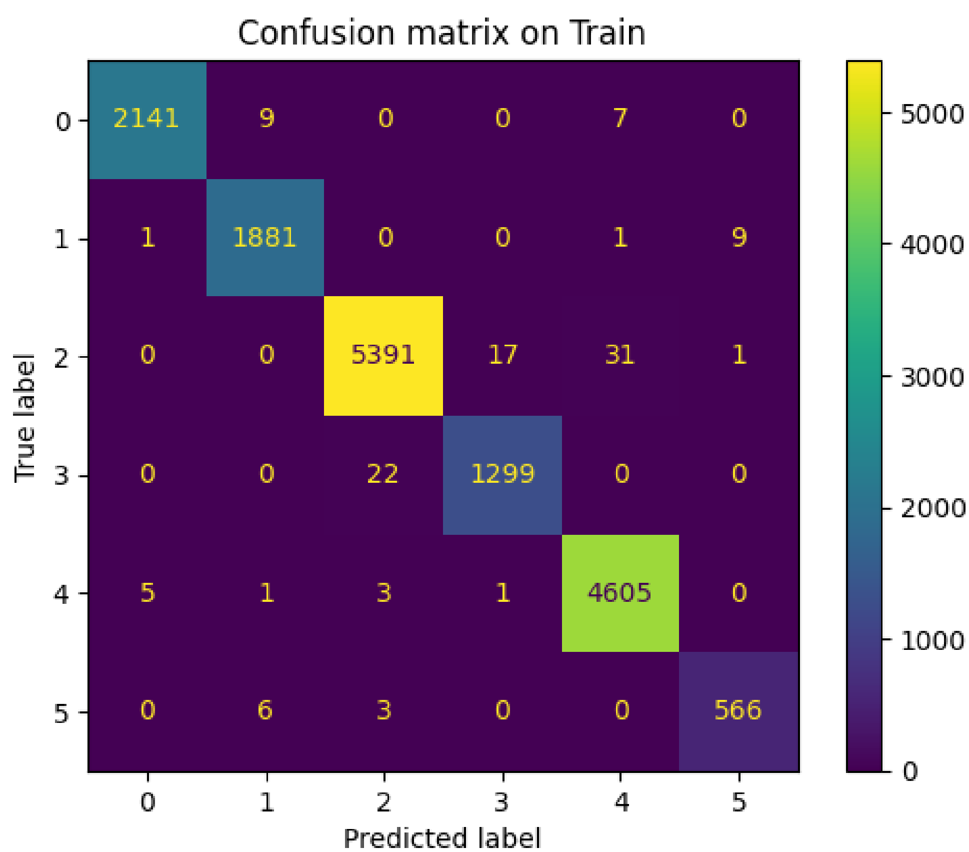
```
[ ]: RF = RandomForestClassifier()
     RF.fit(X_train , y_train)
```

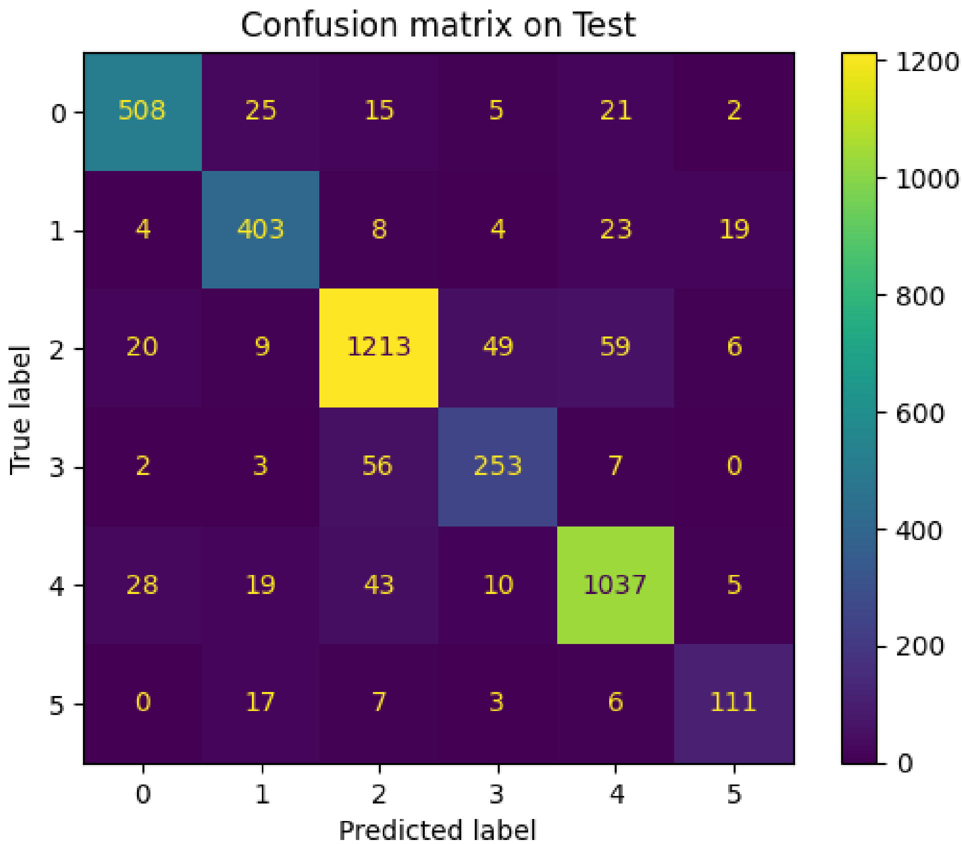
```
[ ]: RandomForestClassifier()
```

Evaluate this model using a preset function:

```
[ ]: evaluate_model(RF, X_train, X_test, y_train, y_test, include_training=True)
```

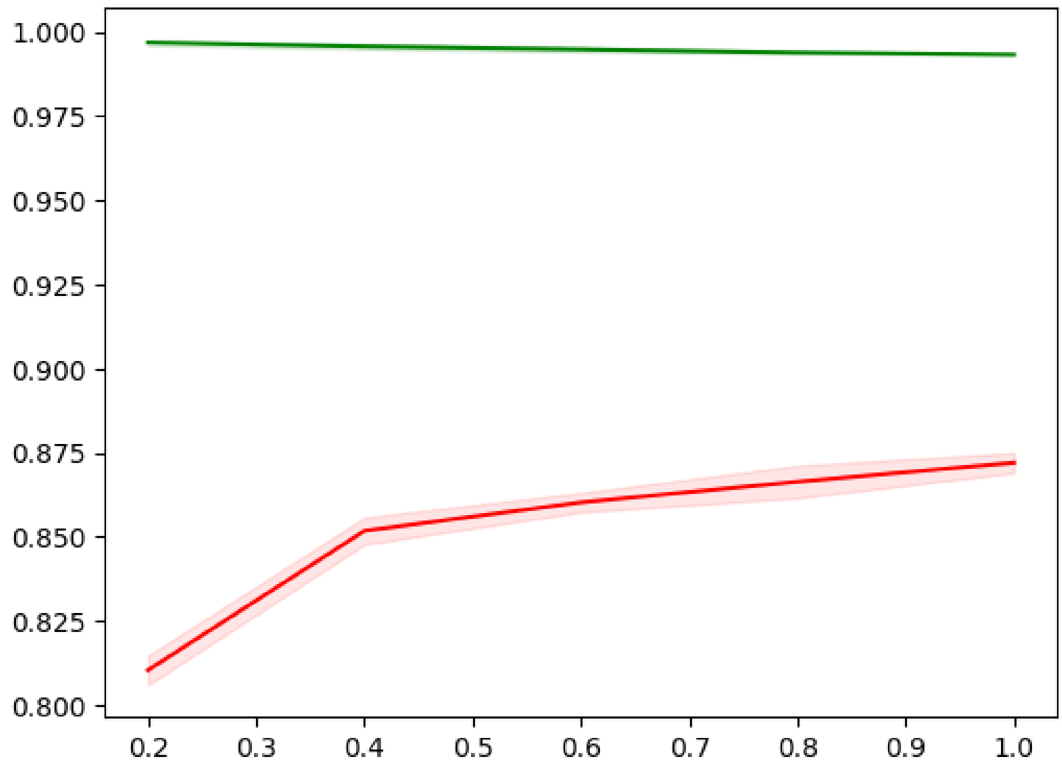
```
Score of on train are:
        - Accuracy score: 0.9927
        - Micro F1 score: 0.9927
        - Macro F1 score: 0.9906
Score of on test are:
        - Accuracy score: 0.8812
        - Micro F1 score: 0.8812
        - Macro F1 score: 0.8523
```





Draw learning curve using a preset function:

[ ]:
```
draw_learning_curve(RF, X_train, y_train)
```



3 Single tuning

This section examines the best range for each parameters by plotting the performance of the model with a range of value for each parameters.

3.1 N_estimator

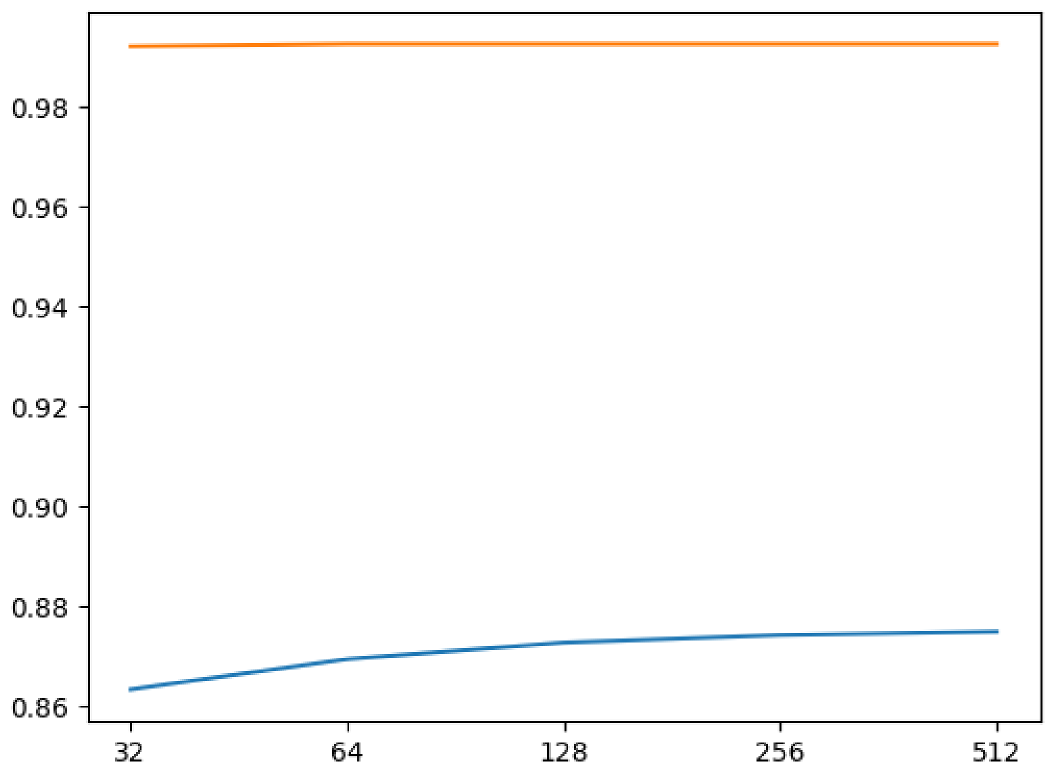
The number of trees in the forest.

[ ]:
```
# Setting the possible value for n_estimators
n_estimators_list = [32, 64, 128, 256, 512]

trs_list = list()
cvs_list = list()

for n_estimators in n_estimators_list:
    # Define model for each n_estimators
    rf_model = RandomForestClassifier(n_estimators=n_estimators)
    rf_model.fit(X_train, y_train)

    # Calculate the cross validation score
    train_score = accuracy_score(y_train, rf_model.predict(X_train))
```

```
    cvs_score = np.mean(cross_val_score(rf_model, X_train, y_train, cv=5,␣
↪n_jobs=6))

    trs_list.append(train_score)
    cvs_list.append(cvs_score)
```

[ ]:
```
# Draw the plot for n_estimators
fig = sns.lineplot(x=list(range(len(n_estimators_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(n_estimators_list))), y=trs_list)
fig.set_xticks(range(len(n_estimators_list)))
fig.set_xticklabels(n_estimators_list)
```

[ ]:
```
[Text(0, 0, '32'),
 Text(1, 0, '64'),
 Text(2, 0, '128'),
 Text(3, 0, '256'),
 Text(4, 0, '512')]
```



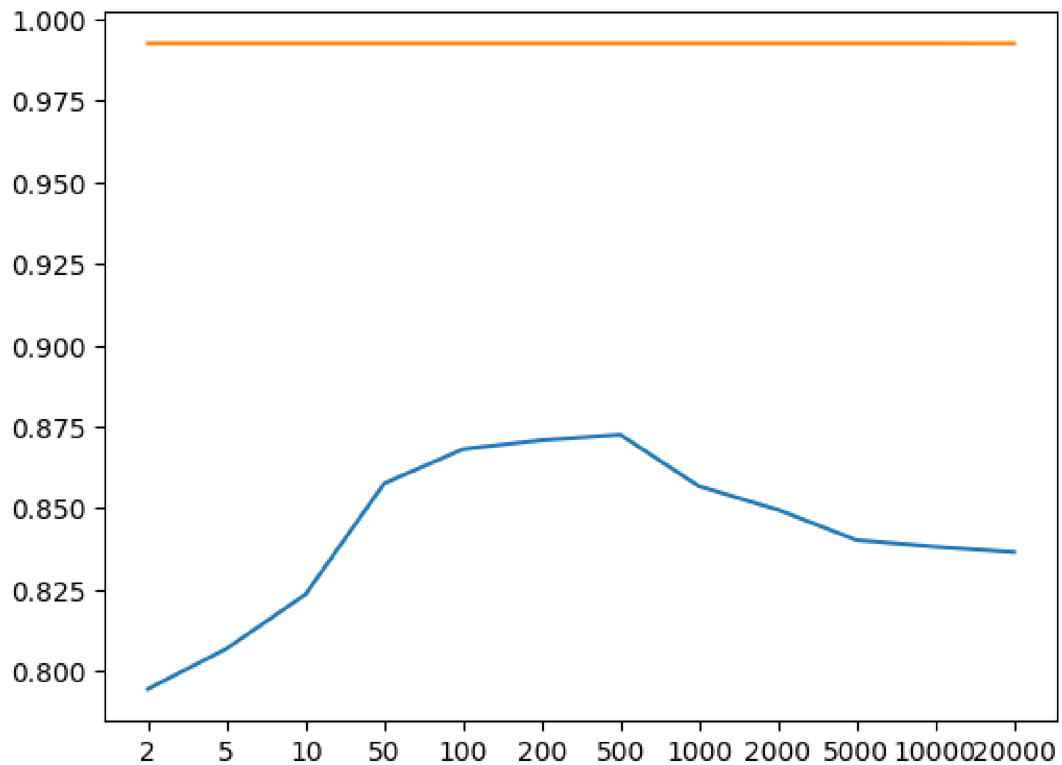
3.2 Max_features

The number of features to consider when looking for the best split.

[ ]:
```
# Setting the max_features range
max_features_list = [2, 5, 10, 50, 100, 200, 500, 1000, 2000, 5000, 10000,␣
↪20000]
trs_list = list()
cvs_list = list()

for max_features in max_features_list:
  # Define model for each max_features
  rf_model = RandomForestClassifier(max_features=max_features)
  rf_model.fit(X_train, y_train)

  # Calculate the cross validation score
  train_score = accuracy_score(y_train, rf_model.predict(X_train))
  cv_score = np.mean(cross_val_score(rf_model, X_train, y_train, cv=5,␣
↪n_jobs=8))

  trs_list.append(train_score)
  cvs_list.append(cv_score)
```

[ ]:
```
# Draw the plot for max_features
fig = sns.lineplot(x=list(range(len(max_features_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(max_features_list))), y=trs_list)
fig.set_xticks(range(len(max_features_list)))
fig.set_xticklabels(max_features_list)
```

[ ]:
```
[Text(0, 0, '2'),
 Text(1, 0, '5'),
 Text(2, 0, '10'),
 Text(3, 0, '50'),
 Text(4, 0, '100'),
 Text(5, 0, '200'),
 Text(6, 0, '500'),
 Text(7, 0, '1000'),
 Text(8, 0, '2000'),
 Text(9, 0, '5000'),
 Text(10, 0, '10000'),
 Text(11, 0, '20000')]
```



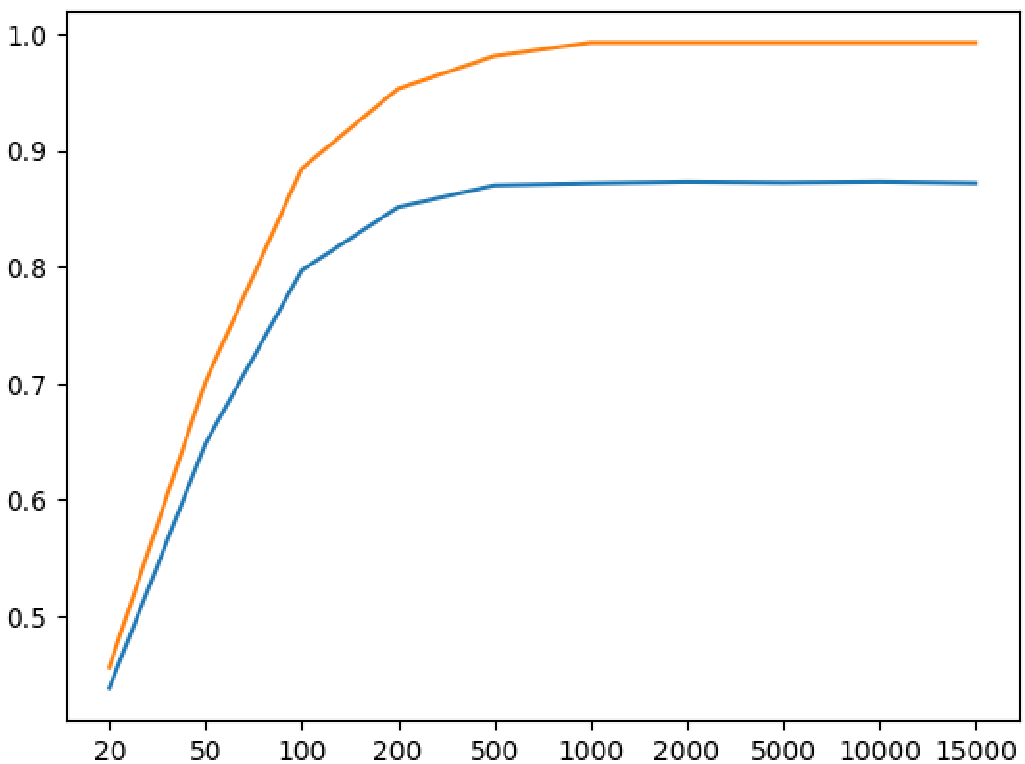
3.3 Max_depth

max_depth is the maximum depth of the tree.

[ ]:
```
# Setting the possible value for max depth
max_depth_list = [20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 15000]

trs_list = list()
cvs_list = list()

for max_depth in max_depth_list:
    # Define model for each max_depth
    rf_model = RandomForestClassifier(max_depth=max_depth)
    rf_model.fit(X_train, y_train)

    # Calculate the cross validation score
    train_score = accuracy_score(y_train, rf_model.predict(X_train))
    cvs_score = np.mean(cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪n_jobs=-1))

    trs_list.append(train_score)
    cvs_list.append(cvs_score)
```

```
[ ]: # Draw the plot for max depth
fig = sns.lineplot(x=list(range(len(max_depth_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(max_depth_list))), y=trs_list)
fig.set_xticks(range(len(max_depth_list)))
fig.set_xticklabels(max_depth_list)
```

```
[ ]: [Text(0, 0, '20'),
 Text(1, 0, '50'),
 Text(2, 0, '100'),
 Text(3, 0, '200'),
 Text(4, 0, '500'),
 Text(5, 0, '1000'),
 Text(6, 0, '2000'),
 Text(7, 0, '5000'),
 Text(8, 0, '10000'),
 Text(9, 0, '15000')]
```



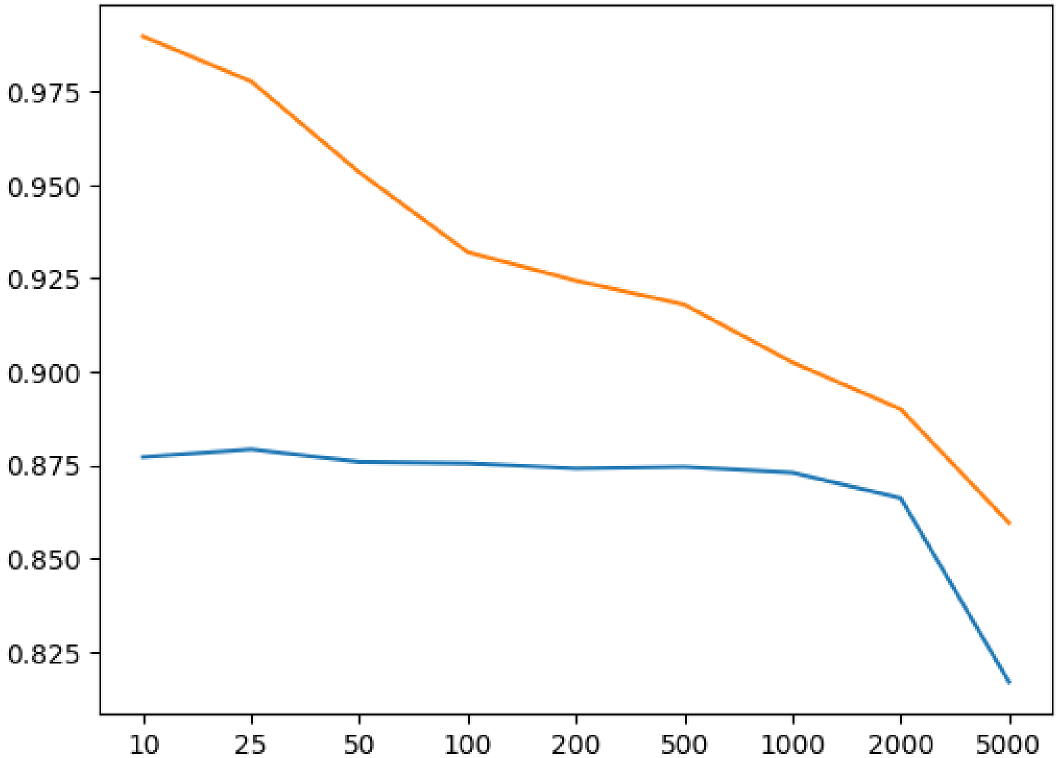
3.4 Min_samples_split

min_samples_split is the minimum number of samples required to split an internal node.

```
[ ]: # Setting the possible value for min_samples_split
min_samples_split_list = [10, 25, 50, 100, 200, 500, 1000, 2000, 5000]

trs_list = list()
cvs_list = list()

for min_samples_split in min_samples_split_list:
    # Define model for each min_samples_split
    rf_model = RandomForestClassifier(min_samples_split=min_samples_split)
    rf_model.fit(X_train, y_train)

    # Calculate the cross validation score
    train_score = accuracy_score(y_train, rf_model.predict(X_train))
    cvs_score = np.mean(cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪n_jobs=6))

    trs_list.append(train_score)
    cvs_list.append(cvs_score)
```

```
[ ]: # Draw the plot for min_samples_split
fig = sns.lineplot(x=list(range(len(min_samples_split_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(min_samples_split_list))), y=trs_list)
fig.set_xticks(range(len(min_samples_split_list)))
fig.set_xticklabels(min_samples_split_list)
```

```
[ ]: [Text(0, 0, '10'),
 Text(1, 0, '25'),
 Text(2, 0, '50'),
 Text(3, 0, '100'),
 Text(4, 0, '200'),
 Text(5, 0, '500'),
 Text(6, 0, '1000'),
 Text(7, 0, '2000'),
 Text(8, 0, '5000')]
```



3.5 Min_samples_leaf

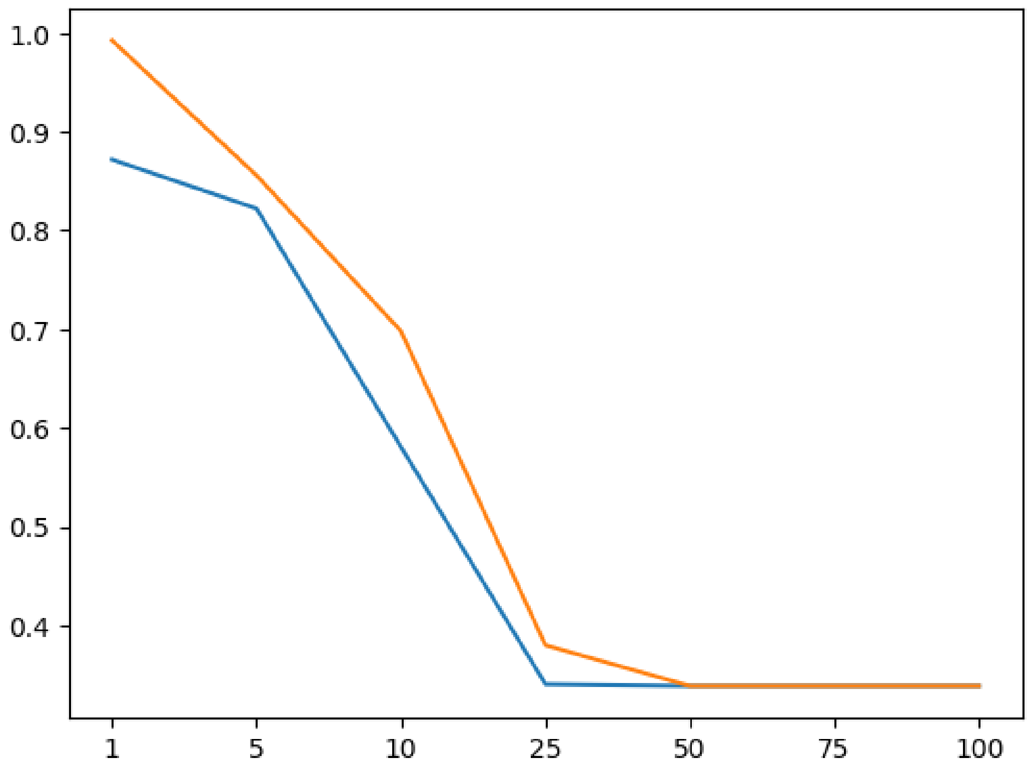
min_samples_leaf is the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least ***min_samples_leaf*** training samples in each of the left and right branches.

```
[ ]: # Setting the min_samples_leaf range
min_samples_leaf_list = [1, 5, 10, 25, 50, 75, 100]
trs_list = list()
cvs_list = list()

for min_samples_leaf in min_samples_leaf_list:
  # Define model for each min_samples_leaf
  rf_model = RandomForestClassifier(min_samples_leaf=min_samples_leaf)
  rf_model.fit(X_train, y_train)

  # Calculate the cross validation score
  train_score = accuracy_score(y_train, rf_model.predict(X_train))
  cv_score = np.mean(cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪n_jobs=6))

  trs_list.append(train_score)
```

```
    cvs_list.append(cv_score)
```

[ ]:
```
# Draw the plot for min_samples_leaf
fig = sns.lineplot(x=list(range(len(min_samples_leaf_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(min_samples_leaf_list))), y=trs_list)
fig.set_xticks(range(len(min_samples_leaf_list)))
fig.set_xticklabels(min_samples_leaf_list)
```

[ ]:
```
[Text(0, 0, '1'),
 Text(1, 0, '5'),
 Text(2, 0, '10'),
 Text(3, 0, '25'),
 Text(4, 0, '50'),
 Text(5, 0, '75'),
 Text(6, 0, '100')]
```



From the plot, we can see that the higher this parameter is, the lower the accuracy for both training and testing are.

4 Multiple tuning

First, we use grid search to help tuning this model.

[ ]:
```python
dict_param = {
    'max_depth' : np.asarray([200, 500, 2000]),
    'min_samples_split': np.asarray([10, 200, 2000]),
    'min_samples_leaf': np.arange(1, 4),
    'max_features': np.asarray([50, 200, 500]),
}

grid_search = GridSearchCV(RandomForestClassifier(n_estimators=512),␣
 ↪dict_param, cv = 5, n_jobs=5)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(n_estimators=512), n_jobs=5,
             param_grid={'max_depth': array([ 200,  500, 2000]),
                         'max_features': array([ 50, 200, 500]),
                         'min_samples_leaf': array([ 1,  5, 10]),
                         'min_samples_split': array([  10,  200, 2000])})
```

We elminate all parameters that appear in models with the validation accuracy < 0.85

[ ]:
```python
df = pd.DataFrame(
  dict(
    max_depth = [val['max_depth'] for val in grid_search.cv_results_['params']],
    min_samples_split = [val['min_samples_split'] for val in grid_search.
 ↪cv_results_['params']],
    min_samples_leaf = [val['min_samples_leaf'] for val in grid_search.
 ↪cv_results_['params']],
    max_features = [val['max_features'] for val in grid_search.
 ↪cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)

df = df[df['score'] <= 0.86]
print(df)

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 81 // len(dict_param[param]) :
      print(param, value)
```

```
    max_depth  min_samples_split  min_samples_leaf  max_features     score
0         200                 10                 1            50  0.654875
1         200                200                 1            50  0.638312
2         200               2000                 1            50  0.638375
3         200                 10                 5            50  0.338688
4         200                200                 5            50  0.338688
..        …                  …                 …             …       …
76       2000                200                 5           500  0.854313
```

```
77        2000              2000                5             500   0.853750
78        2000                10               10             500   0.816125
79        2000               200               10             500   0.815188
80        2000              2000               10             500   0.817125

[67 rows x 5 columns]
max_depth 200
min_samples_leaf 5
min_samples_leaf 10
```

We repeat this process again, this time with the domain narrowed down.

[ ]:
```python
dict_param = {
    'max_depth' : np.asarray([500, 1000, 2000]),
    'min_samples_split': np.asarray([25, 200, 500]),
    'min_samples_leaf': np.asarray([1, 5, 10]),
    'max_features': np.asarray([100, 200, 500]),
}

grid_search = GridSearchCV(RandomForestClassifier(n_estimators=512),␣
 ↪dict_param, cv = 5, n_jobs=5)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(n_estimators=512), n_jobs=5,
             param_grid={'max_depth': array([ 500, 1000, 2000]),
                         'max_features': array([100, 200, 500]),
                         'min_samples_leaf': array([ 1,  5, 10]),
                         'min_samples_split': array([ 25, 200, 500])})
```

[ ]:
```python
df = pd.DataFrame(
  dict(
    max_depth = [val['max_depth'] for val in grid_search.cv_results_['params']],
    min_samples_split = [val['min_samples_split'] for val in grid_search.
 ↪cv_results_['params']],
    min_samples_leaf = [val['min_samples_leaf'] for val in grid_search.
 ↪cv_results_['params']],
    max_features = [val['max_features'] for val in grid_search.
 ↪cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)

df = df[df['score'] <= 0.86]

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 81 // len(dict_param[param]) :
      print(param, value)
```

```
min_samples_leaf 5
min_samples_leaf 10
```

Find the best combination of parameters for the model:

```
[ ]: print(grid_search.best_estimator_, grid_search.best_score_)
```

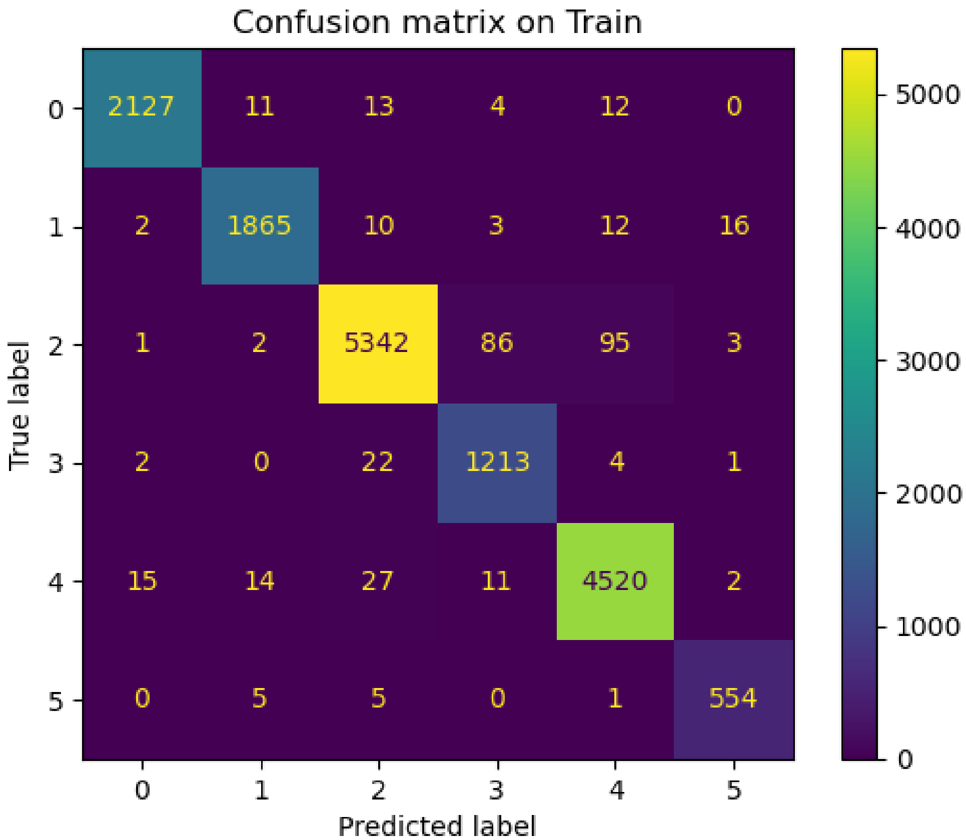
```
RandomForestClassifier(max_depth=1000, max_features=200, min_samples_split=25,
                       n_estimators=512) 0.8781875000000001
```

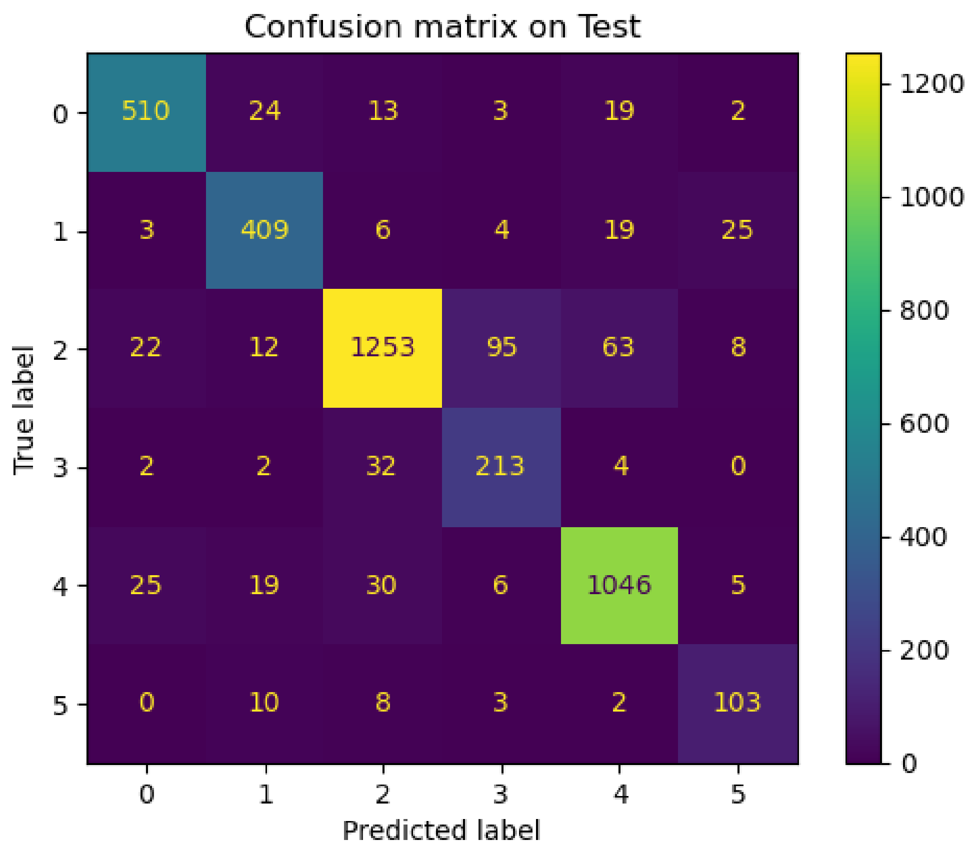
5 Conclusion

We use all the parameters from the last section to define the best model and then evaluate it using the preset functions.

```
[ ]: best_rf_model = RandomForestClassifier(max_depth=1000, max_features=200,
                         min_samples_split=25, n_estimators=512)
     best_rf_model.fit(X_train, y_train)

     evaluate_model(best_rf_model, X_train, X_test, y_train, y_test,␣
      ↪include_training=True)
```

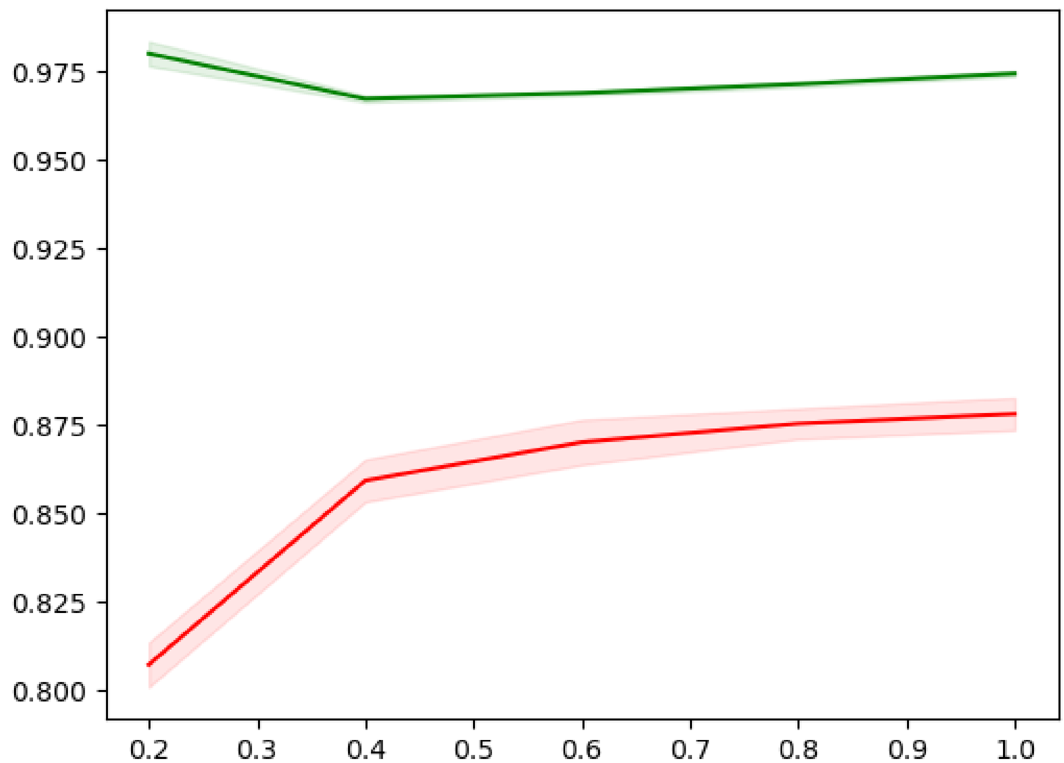
```
Score of on train are:
        - Accuracy score: 0.9763
        - Micro F1 score: 0.9763
        - Macro F1 score: 0.9734
Score of on test are:
        - Accuracy score: 0.8835
        - Micro F1 score: 0.8835
        - Macro F1 score: 0.8475
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2127 11 13 4 12 0
2 1865 10 3 12 16
1 2 5342 86 95 3
2 0 22 1213 4 1
15 14 27 11 4520 2
0 5 5 0 1 554
5000
4000
3000
2000
1000
0



After that, we draw the learning curve of this Random forest model.

[ ]:
```
draw_learning_curve(best_rf_model, X_train, y_train)
```



Finally, we export the model.

[ ]:
```
directory = "data/models/"

dump(best_rf_model, directory + "best_rf_model_bow.joblib")
```

[ ]:
```
['data/models/best_rf_model_bow.joblib']
```