Decision Tree tfidf

May 7, 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.tree import DecisionTreeClassifier
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_tfidf
X_test = X_test_tfidf
```

2 Basic training

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

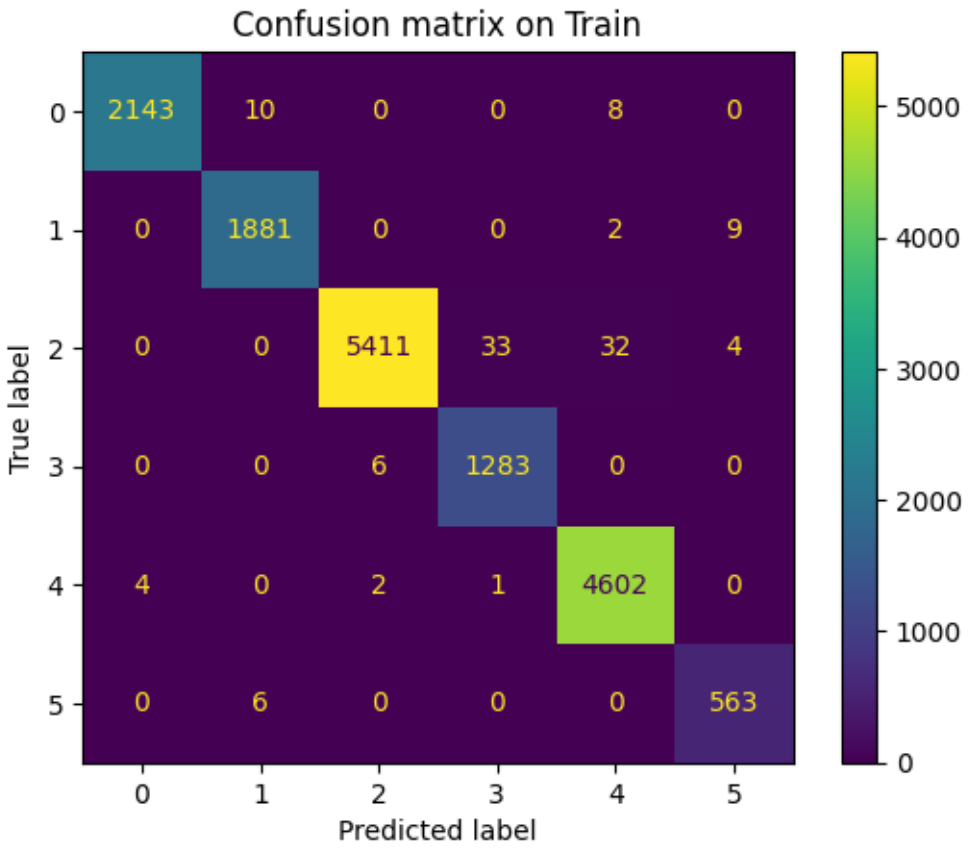
```
[ ]: DT = DecisionTreeClassifier()
     DT.fit(X_train , y_train)
```

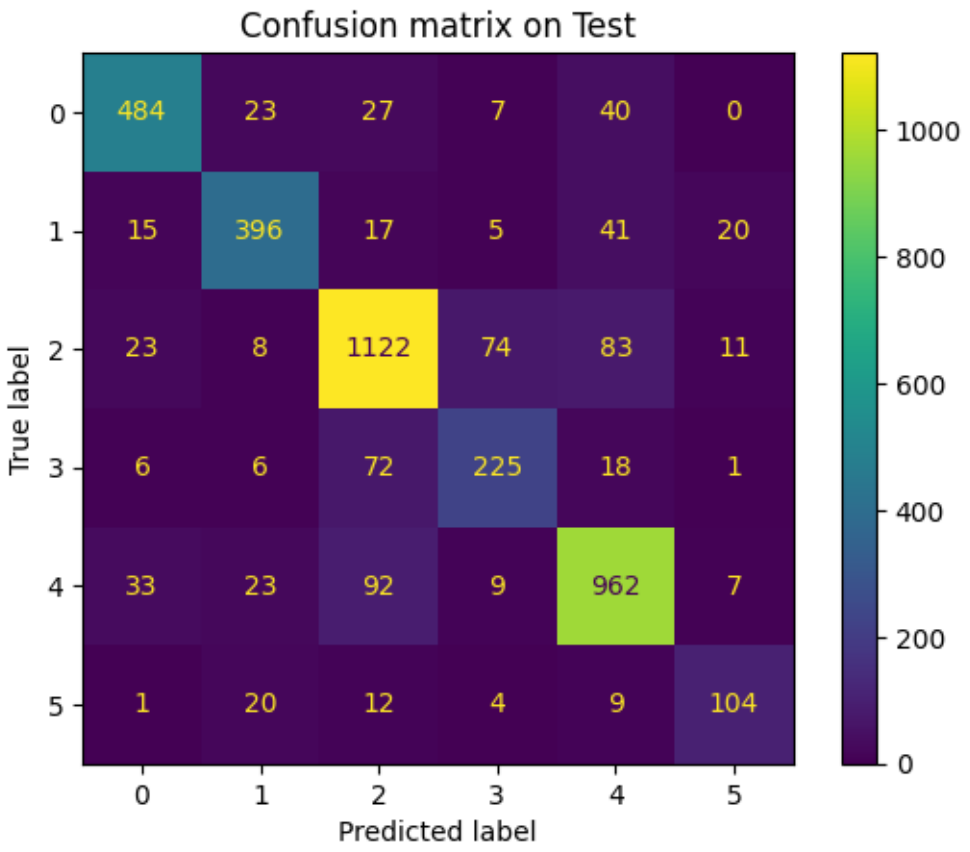
```
[ ]: DecisionTreeClassifier()
```

Evaluate this model using a preset function:

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

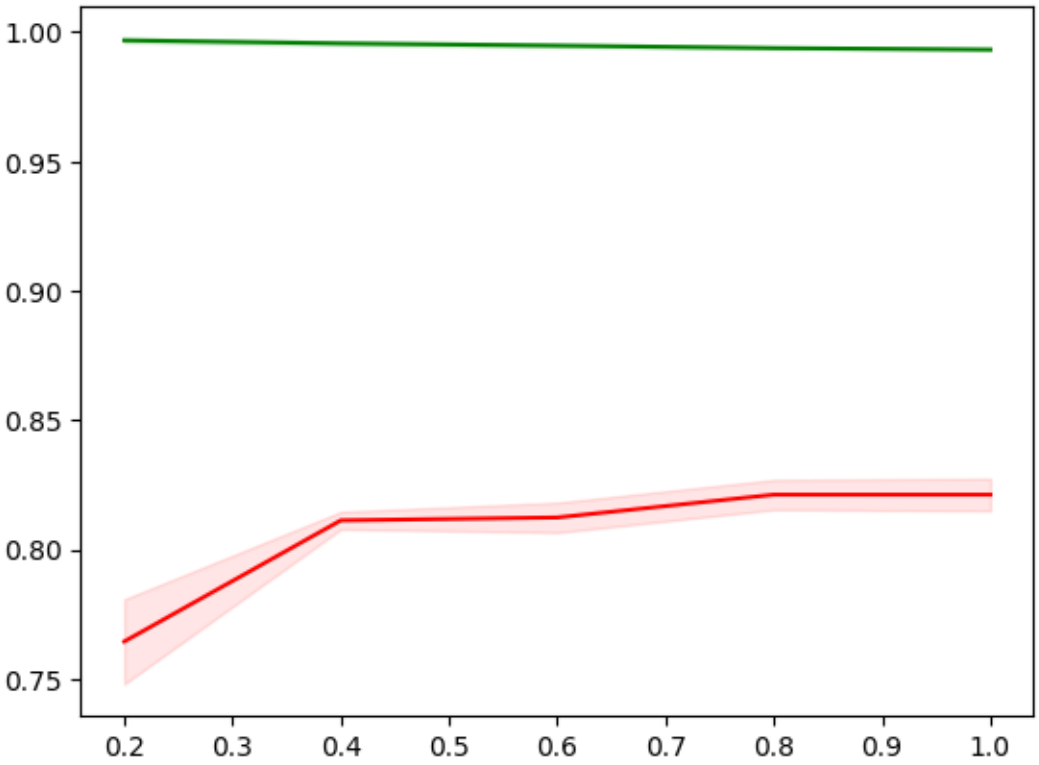
```
Score of on train are:
        - Accuracy score: 0.9927
        - Micro F1 score: 0.9927
        - Macro F1 score: 0.9906
Score of on test are:
        - Accuracy score: 0.8233
        - Micro F1 score: 0.8233
        - Macro F1 score: 0.7917
```





Draw learning curve using a preset function:

[ ]:
```
draw_learning_curve(DT, 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 Max_depth

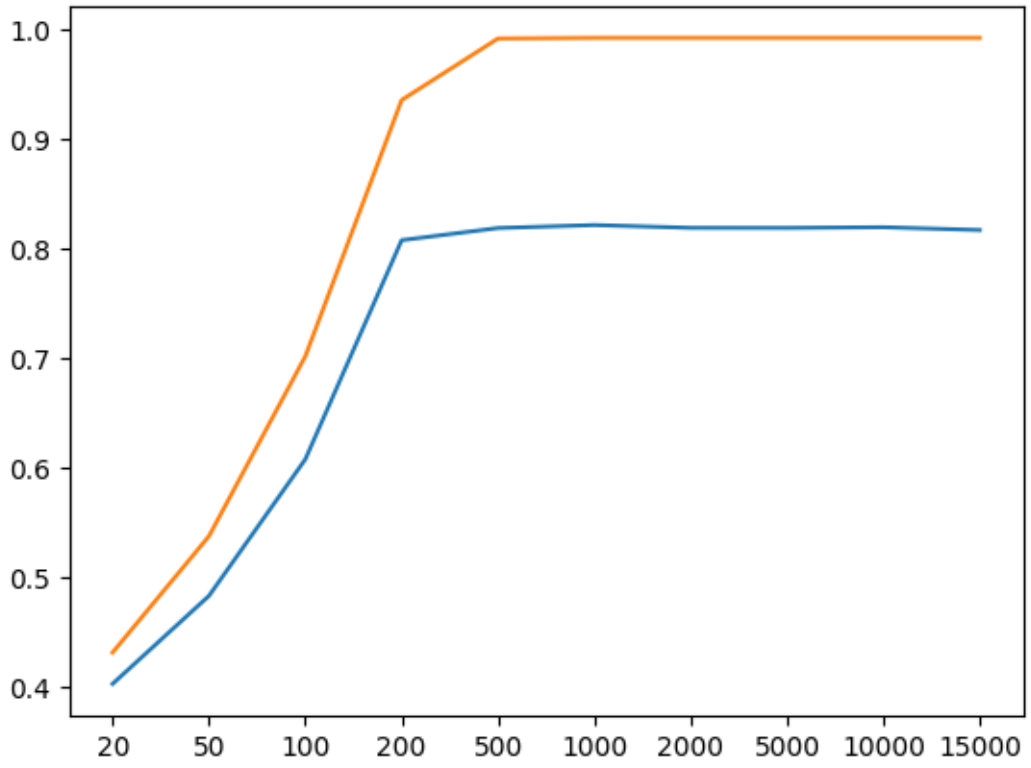
max_depth is the maximum depth of the tree.

[ ]:
```
# 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
    dt_model = DecisionTreeClassifier(max_depth=max_depth)
    dt_model.fit(X_train, y_train)

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

```
    cvs_score = np.mean(cross_val_score(dt_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')]
```



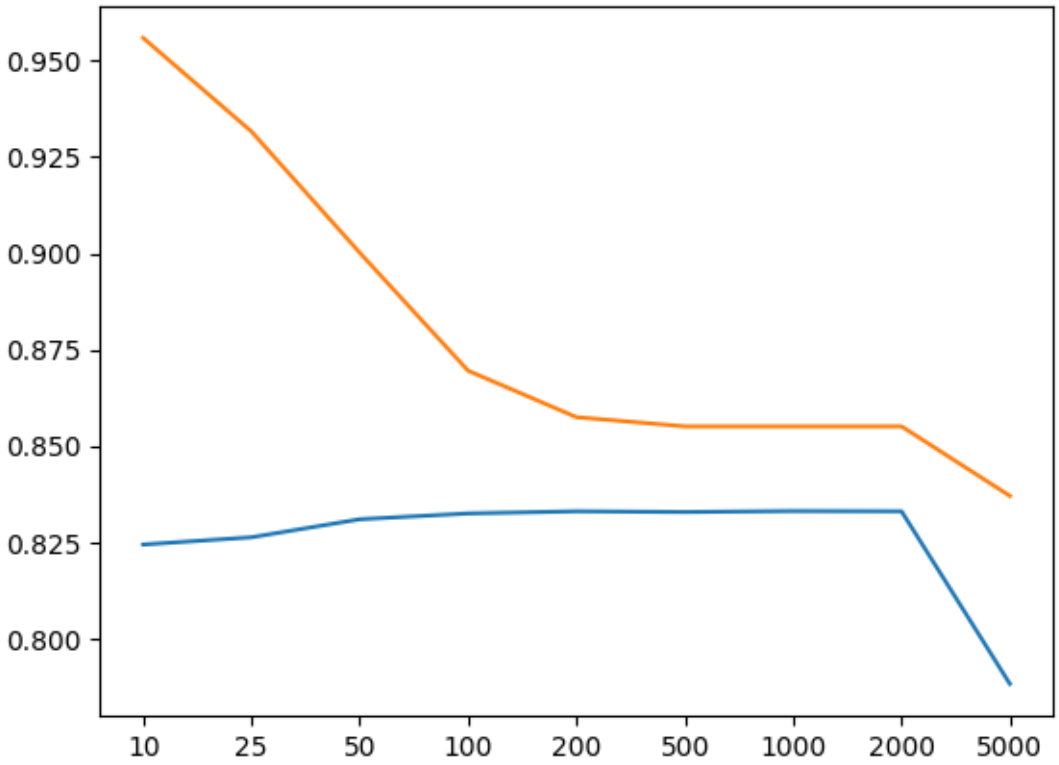
3.2 Min_samples_split

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

[ ]:
```
# Setting the possible value for max depth
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 max_depth
    dt_model = DecisionTreeClassifier(max_depth=276,␣
 ↪min_samples_split=min_samples_split)
    dt_model.fit(X_train, y_train)

    # Calculate the cross validation score
    train_score = accuracy_score(y_train, dt_model.predict(X_train))
    cvs_score = np.mean(cross_val_score(dt_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(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')]
```



3.3 Min_samples_leaf

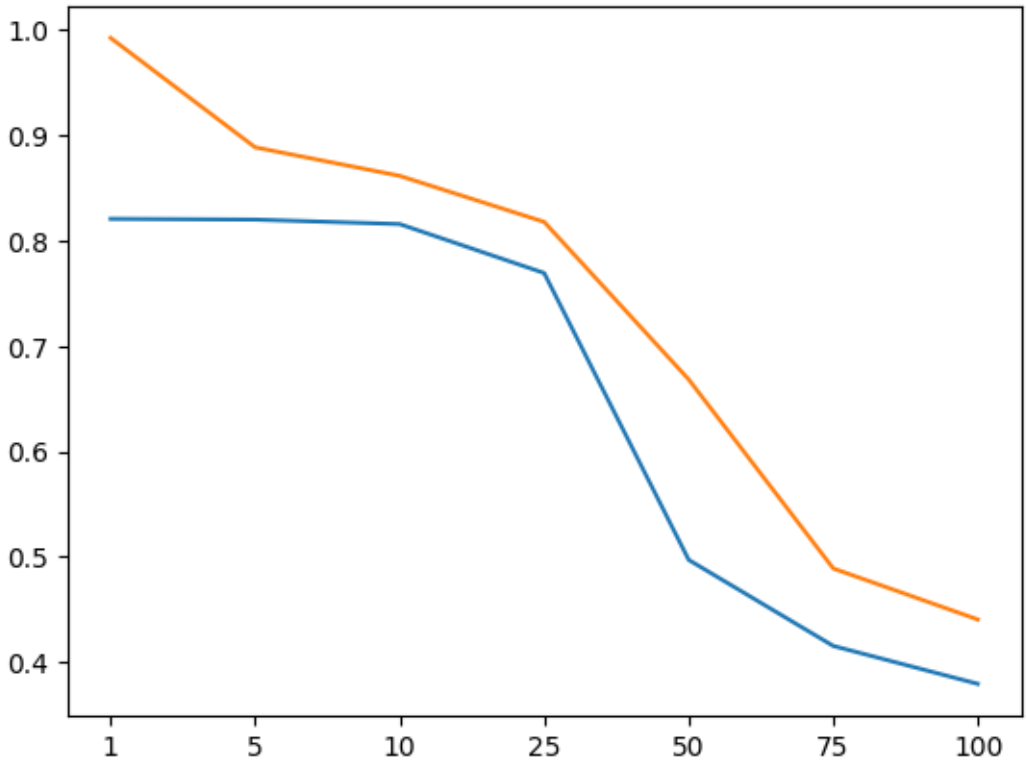
min_samples_leaf is the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least ***min_samples_leaf*** training samples in each of the left and right branches.

[ ]:
```
# Setting the min sample 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 k
  dt_model = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf)
  dt_model.fit(X_train, y_train)

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

  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.

[ ]:
```
dict_param = {
    'max_depth' : np.asarray([100, 1000, 5000]),
    'min_samples_split': np.asarray([10, 100, 1000, 5000]),
    'min_samples_leaf': np.asarray([5, 25, 100]),
}

grid_search = GridSearchCV(DecisionTreeClassifier(), dict_param, cv = 5,␣
 ↪n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=8,
             param_grid={'max_depth': array([ 100, 1000, 5000]),
                         'min_samples_leaf': array([  5,  25, 100]),
                         'min_samples_split': array([  10,  100, 1000, 5000])})
```

We elminate all parameters that appear in models with the validation accuracy < 0.82

[ ]:
```
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']],
    score = grid_search.cv_results_['mean_test_score']
  )
)

df = df[df['score'] <= 0.82]

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 36 // len(dict_param[param]) :
      print(param, value)
```

```
max_depth 100
min_samples_split 10
min_samples_split 5000
min_samples_leaf 25
min_samples_leaf 100
```

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

[ ]:
```
dict_param = {
    'max_depth' : np.asarray([1000, 2000, 5000]),
    'min_samples_split': np.asarray([500, 1000, 2000]),
    'min_samples_leaf': np.asarray([i for i in range(1, 10, 2)])
}
```

```
grid_search = GridSearchCV(DecisionTreeClassifier(), dict_param, cv = 5,␣
 ↪n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=8,
             param_grid={'max_depth': array([1000, 2000, 5000]),
                         'min_samples_leaf': array([1, 3, 5, 7, 9]),
                         'min_samples_split': array([ 500, 1000, 2000])})
```

[ ]:
```
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']],
    score = grid_search.cv_results_['mean_test_score']
  )
)

df = df[df['score'] <= 0.83]

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 45 // len(dict_param[param]) :
      print(param, value)
```

```
min_samples_leaf 5
min_samples_leaf 7
min_samples_leaf 9
```

Find the best combination of parameters for the model:

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

```
DecisionTreeClassifier(max_depth=2000, min_samples_leaf=3,
                       min_samples_split=500) 0.8310625
```

5 Max_leaf_nodes

This parameter is tuned separately after finding the best combination of other parameters because it is very time-consuming.

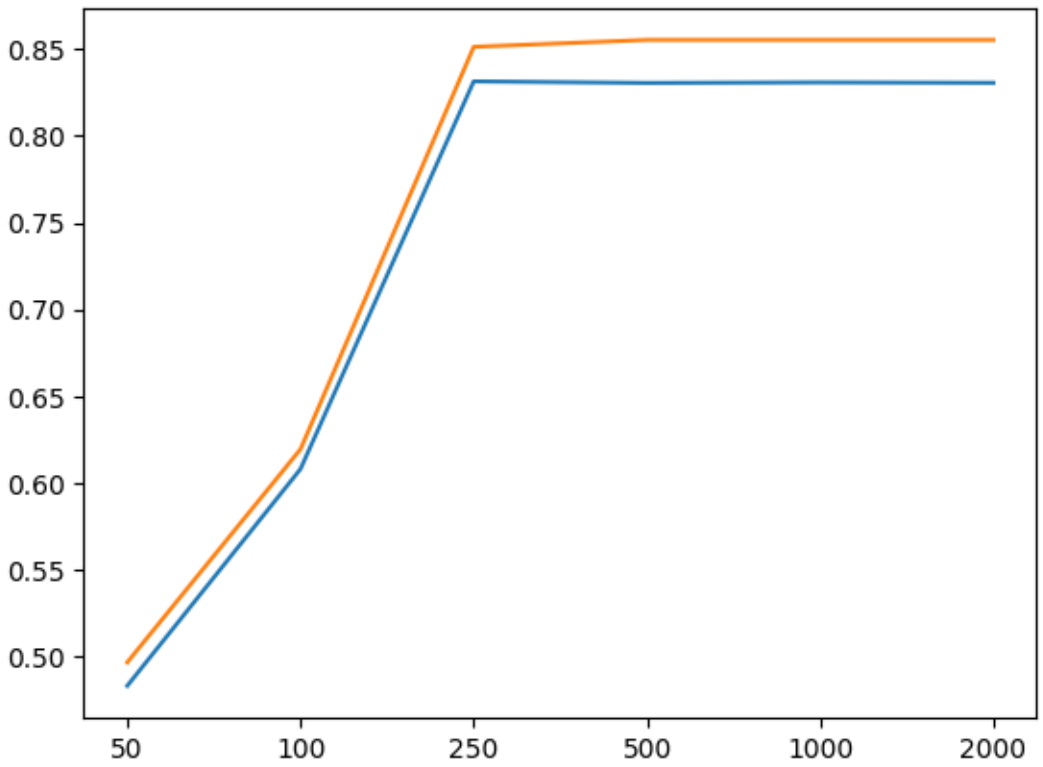
First, we examine this parameter in a wide range of value.

[ ]:
```
# Setting the max_leaf_nodes range
max_leaf_nodes_list = [50, 100, 250, 500, 1000, 2000]
trs_list = list()
cvs_list = list()

for max_leaf_nodes in max_leaf_nodes_list:
  # Define model for each k
  dt_model = DecisionTreeClassifier(max_depth=2000, min_samples_leaf=3,
                       min_samples_split=500, max_leaf_nodes=max_leaf_nodes)
  dt_model.fit(X_train, y_train)

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

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

Then, we plot the result.

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

[ ]:
```
[Text(0, 0, '50'),
 Text(1, 0, '100'),
 Text(2, 0, '250'),
 Text(3, 0, '500'),
 Text(4, 0, '1000'),
 Text(5, 0, '2000')]
```



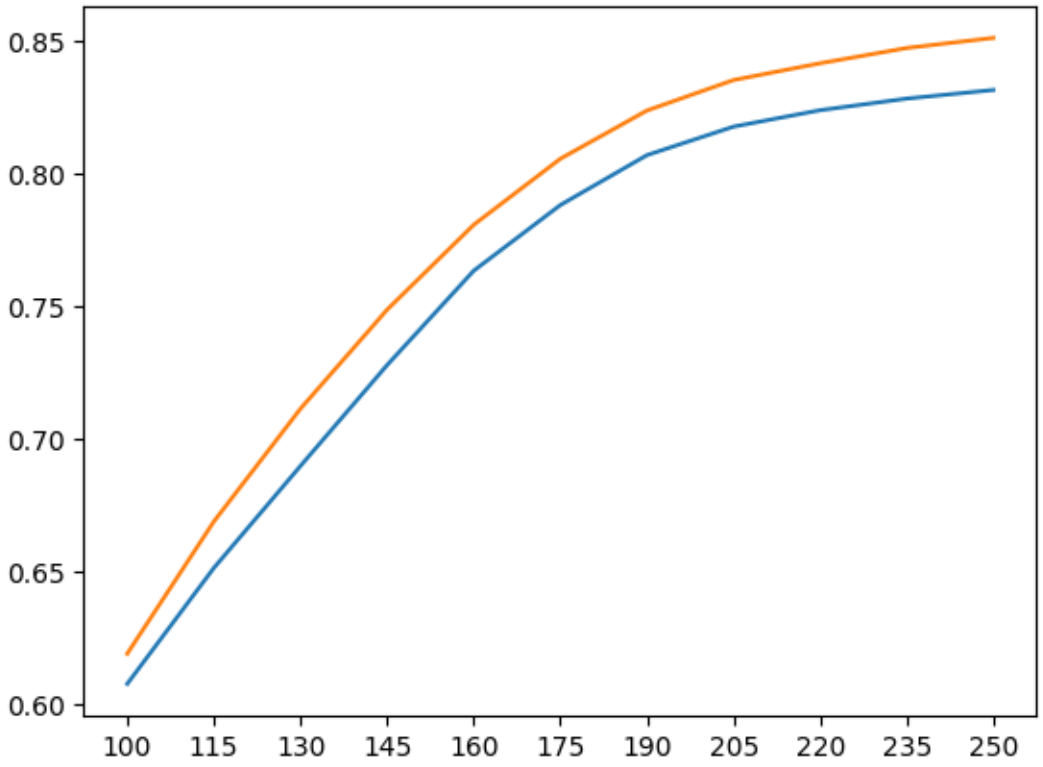
The plot shows that it would be the most ideal for this parameter to be greater than 250.

We further examine by plotting the performance of this model in the range $(100, 250)$.

[ ]:
```
# Setting the max_leaf_nodes range
max_leaf_nodes_list = [i for i in range(100, 251, 15)]
trs_list = list()
cvs_list = list()

for max_leaf_nodes in max_leaf_nodes_list:
  # Define model for each k
  dt_model = DecisionTreeClassifier(max_depth=2000, min_samples_leaf=3,
                       min_samples_split=500, max_leaf_nodes=max_leaf_nodes)
  dt_model.fit(X_train, y_train)

  # Calculate the cross validation score
  train_score = accuracy_score(y_train, dt_model.predict(X_train))
  cv_score = np.mean(cross_val_score(dt_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_leaf_nodes
fig = sns.lineplot(x=list(range(len(max_leaf_nodes_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(max_leaf_nodes_list))), y=trs_list)
fig.set_xticks(range(len(max_leaf_nodes_list)))
fig.set_xticklabels(max_leaf_nodes_list)
```

[ ]:
```
[Text(0, 0, '100'),
 Text(1, 0, '115'),
 Text(2, 0, '130'),
 Text(3, 0, '145'),
 Text(4, 0, '160'),
 Text(5, 0, '175'),
 Text(6, 0, '190'),
 Text(7, 0, '205'),
 Text(8, 0, '220'),
 Text(9, 0, '235'),
 Text(10, 0, '250')]
```



It is illustrated from the plot that this parameter should be near 250.

After all the plotting, we use GridSearchCV to find the best value for it.

```
[ ]: dict_param = {'max_leaf_nodes': [i for i in range(200, 251, 5)]}
best_max_leaf_nodes = GridSearchCV(DecisionTreeClassifier(max_depth=2000,␣
 ↪min_samples_leaf=3,
                        min_samples_split=500, max_leaf_nodes=max_leaf_nodes),␣
 ↪param_grid = dict_param, n_jobs = -1, cv = 10, scoring = 'accuracy')

best_max_leaf_nodes.fit(X_train, y_train)

print("Best max leaf nodes parameter for decision tree:", best_max_leaf_nodes.
 ↪best_params_)
data_best_max_leaf_nodes_y = best_max_leaf_nodes.predict(X_test)
print("Accuracy of that model:", accuracy_score(data_best_max_leaf_nodes_y,␣
 ↪y_test))
```

```
Best max leaf nodes parameter for decision tree: {'max_leaf_nodes': 250}
Accuracy of that model: 0.833
```

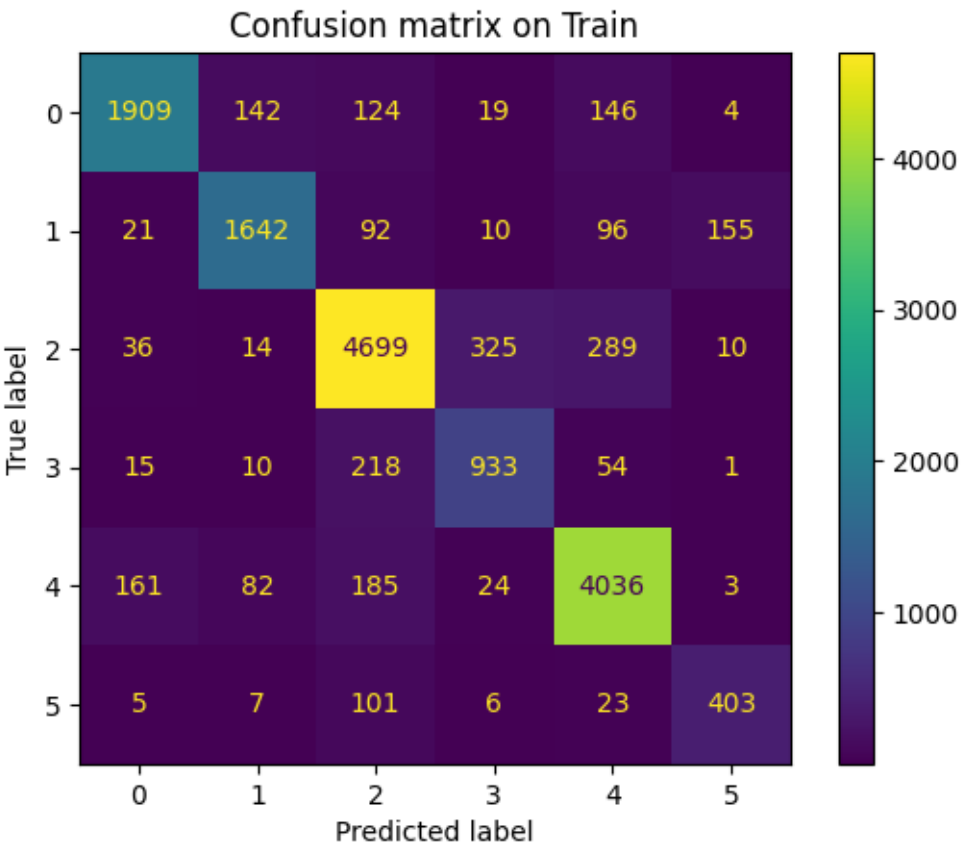
So, the best ***max_leaf_nodes*** parameter is 250.

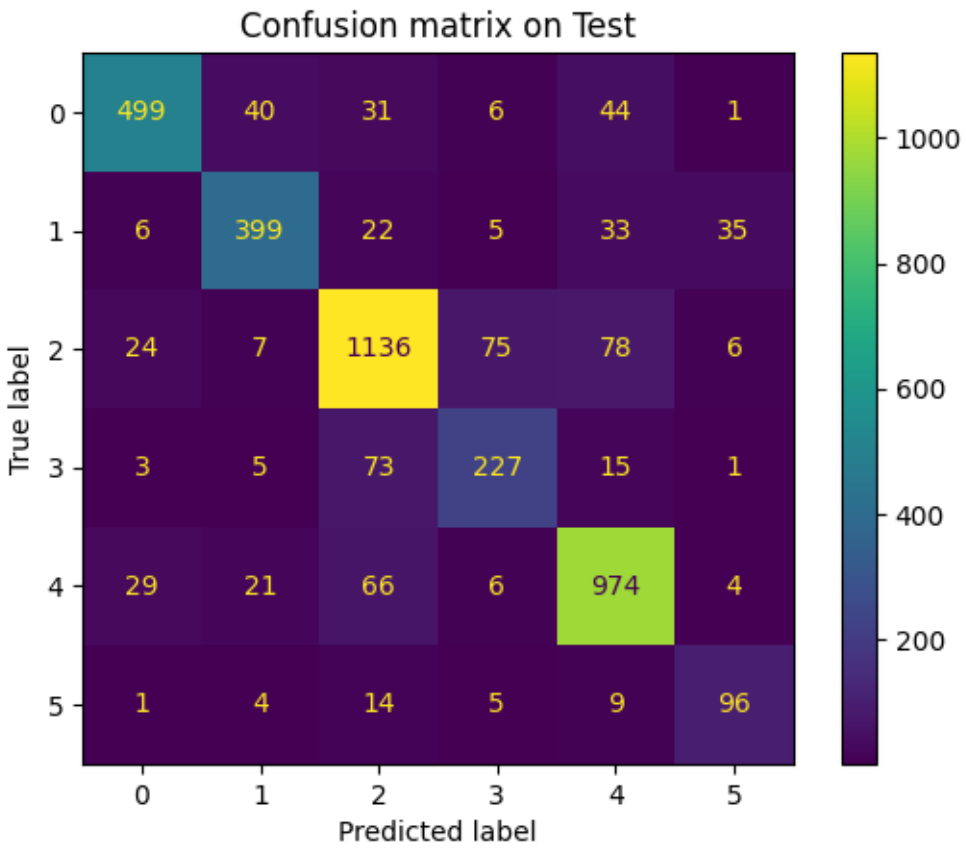
6 Conclusion

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

```
[ ]: best_dt_model = DecisionTreeClassifier(max_depth=2000, min_samples_leaf=3,
                        min_samples_split=500, max_leaf_nodes=250)
best_dt_model.fit(X_train, y_train)

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

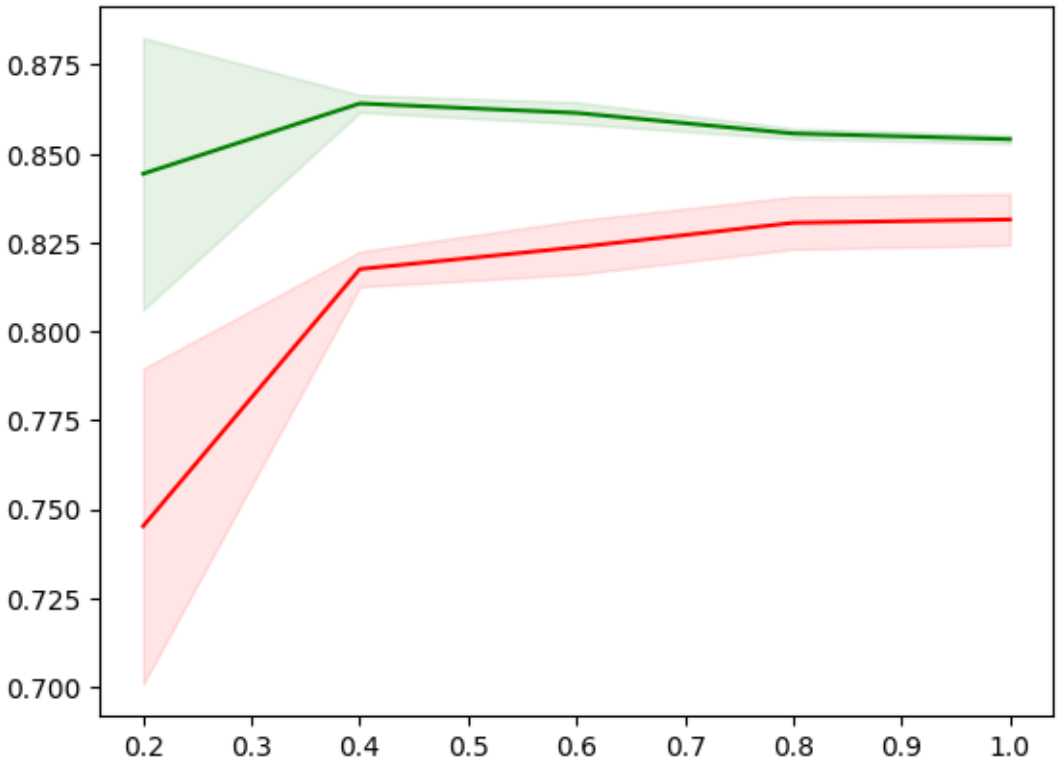
```
Score of on train are:
        - Accuracy score: 0.8514
        - Micro F1 score: 0.8514
        - Macro F1 score: 0.8159
Score of on test are:
        - Accuracy score: 0.8327
        - Micro F1 score: 0.8327
        - Macro F1 score: 0.7973
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
1909 142 124 19 146 4
21 1642 92 10 96 155
36 14 4699 325 289 10
15 10 218 933 54 1
161 82 185 24 4036 3
5 7 101 6 23 403
4000
3000
2000
1000



After that, we draw the learning curve of this Decision Tree model.

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



Finally, we export the model.

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

dump(best_dt_model, directory + "best_dt_model_tfidf.joblib")
```

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