Support Vector Machine (SVM) - TF-IDF_L1

May 4, 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 (may need to add more):

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import pandas as pd

     from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV, cross_val_score
     from sklearn.metrics import accuracy_score
     from joblib import dump, load

     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
```

Select dataset:

```
[ ]: X_train = X_train_tfidf_L1
     X_test = X_test_tfidf_L1
```

2 Basic training

We define and train a model with simple hyperparameter in which kernel is linear, $C = 1.0$, etc:

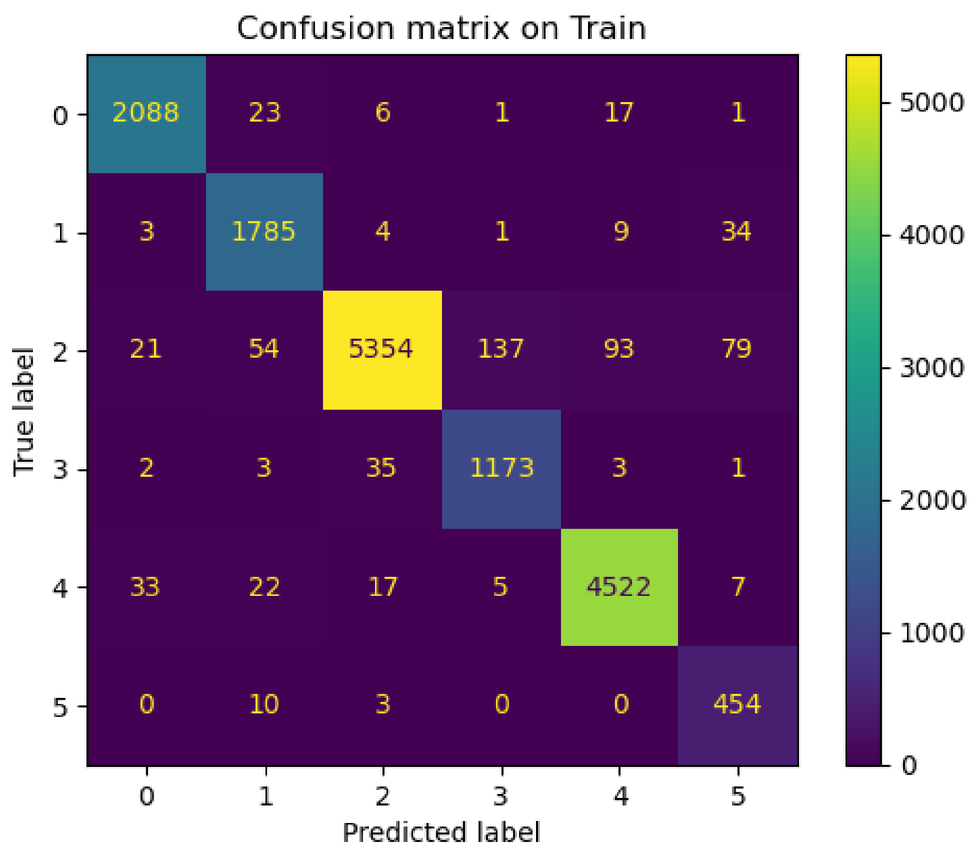
[ ]:
```
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
```

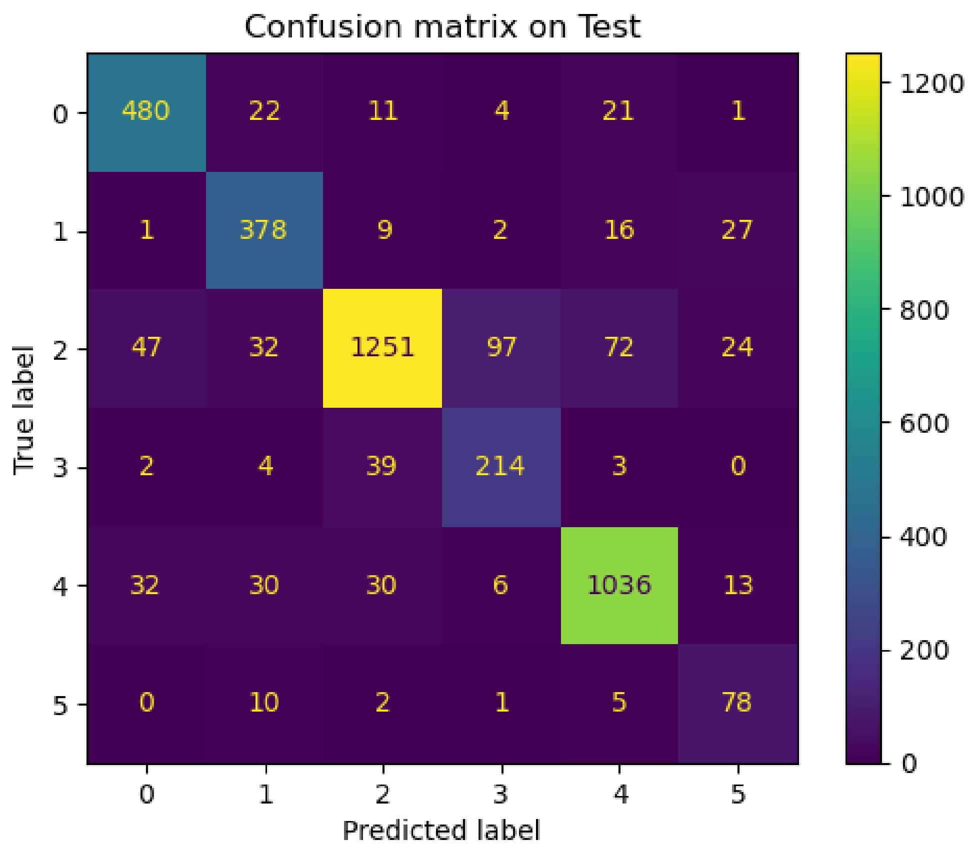
[ ]:
```
SVC(kernel='linear')
```

Evaluate model using preset function:

[ ]:
```
evaluate_model(svm_model, X_train, X_test, y_train, y_test,␣
    ↪include_training=True)
```

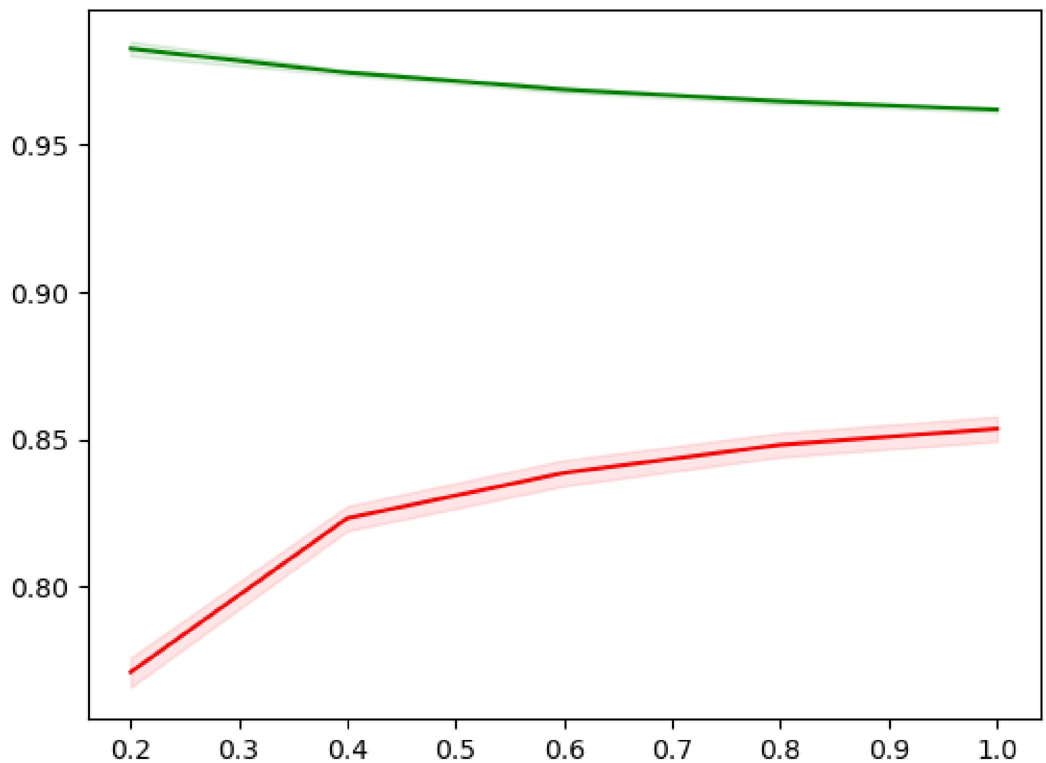
```
Score of on train are:
        - Accuracy score: 0.9610
        - Micro F1 score: 0.9610
        - Macro F1 score: 0.9442
Score of on test are:
        - Accuracy score: 0.8592
        - Micro F1 score: 0.8592
        - Macro F1 score: 0.8101
```





Draw learning curve using preset function:

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



3 Model selection

This section will be separated in 4 parts for 4 kernels instead of using GridSearchCV in order to get the better result

3.1 Linear function kernel

Formula:

$$k(x, z) = x^T z$$

First, we search in a big range from 0.001 to 100

[ ]:
```
C_list = [0.001, 0.01, 0.1, 1, 5, 10, 100]

# Define a list in order to store accuracy points
cvs_list = list()
trs_list = list()

for c in C_list:
    # Define model for each C
    svm_model = SVC(kernel='linear', C=c)
    svm_model.fit(X_train, y_train)
```

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

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

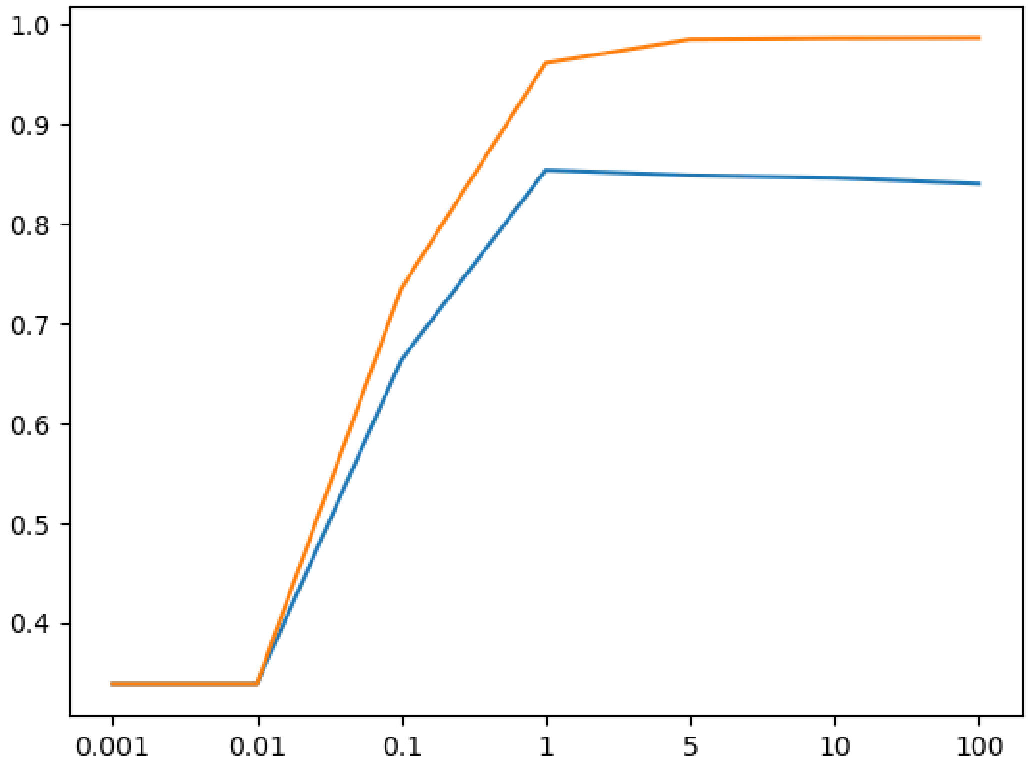
[ ]:
```python
# Print the result
print(C_list)
print(trs_list)
print(cvs_list)

# Draw the plot
fig = sns.lineplot(x=list(range(len(C_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(C_list))), y=trs_list)
fig.set_xticks(range(len(C_list)))
fig.set_xticklabels(C_list)
```

```
[0.001, 0.01, 0.1, 1, 5, 10, 100]
[0.3386875, 0.3386875, 0.735, 0.961, 0.9845625, 0.9855625, 0.9859375]
[0.3386875000000001, 0.3386875000000001, 0.66325, 0.8535625, 0.84825, 0.845875,
0.8399375000000001]
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[ ]:
```
[Text(0, 0, '0.001'),
 Text(1, 0, '0.01'),
 Text(2, 0, '0.1'),
 Text(3, 0, '1'),
 Text(4, 0, '5'),
 Text(5, 0, '10'),
```

```
Text(6, 0, '100')]
```



From the result of above section, we can see the good value of C is near the value 1.

Scope to $C = 1$:

```
[ ]: C_list = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75]

     # Define a list in order to store accuracy points
     cvs_list = list()
     trs_list = list()

     for c in C_list:
         # Define model for each C
         svm_model = SVC(kernel='linear', C=c)
         svm_model.fit(X_train, y_train)

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

         trs_list.append(train_score)
```

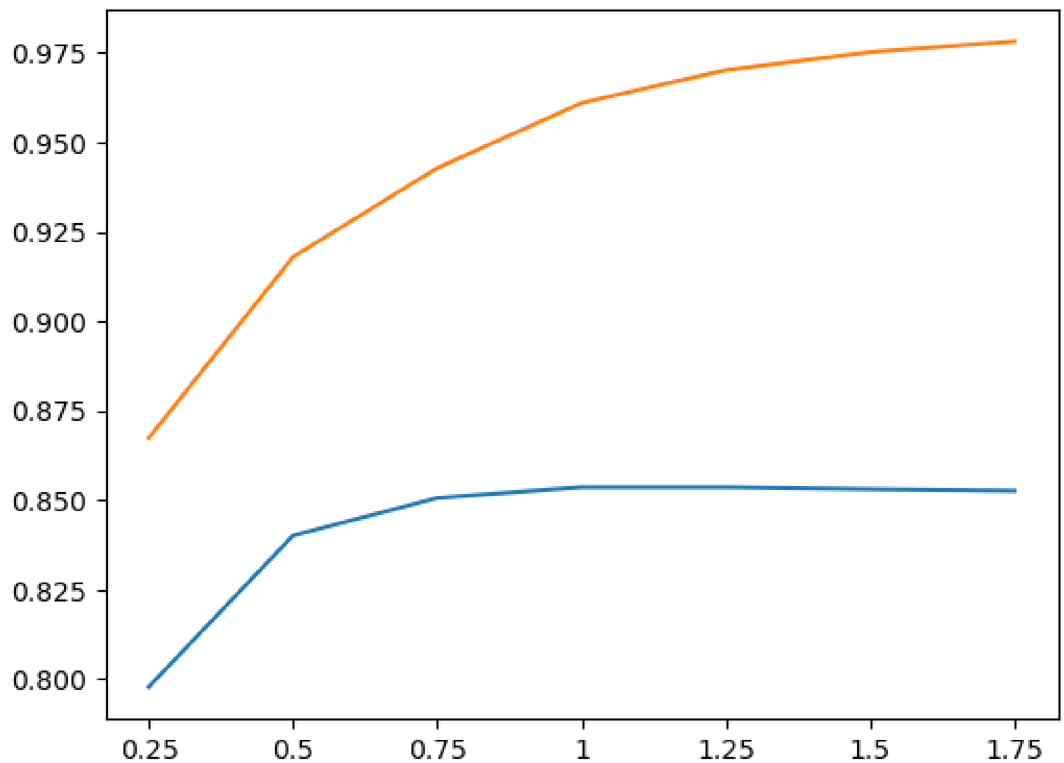
```
    cvs_list.append(cv_score)
```

[ ]:
```
# Print the result
print(C_list)
print(trs_list)
print(cvs_list)

# Draw the plot
fig = sns.lineplot(x=list(range(len(C_list))), y=cvs_list)
fig = sns.lineplot(x=list(range(len(C_list))), y=trs_list)
fig.set_xticks(range(len(C_list)))
fig.set_xticklabels(C_list)
```

```
[0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75]
[0.8673125, 0.9179375, 0.9428125, 0.961, 0.9701875, 0.9751875, 0.9781875]
[0.79775, 0.8400624999999999, 0.8505625, 0.8535625, 0.8535625, 0.853, 0.8525625]
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[ ]:
```
[Text(0, 0, '0.25'),
 Text(1, 0, '0.5'),
 Text(2, 0, '0.75'),
 Text(3, 0, '1'),
 Text(4, 0, '1.25'),
 Text(5, 0, '1.5'),
 Text(6, 0, '1.75')]
```

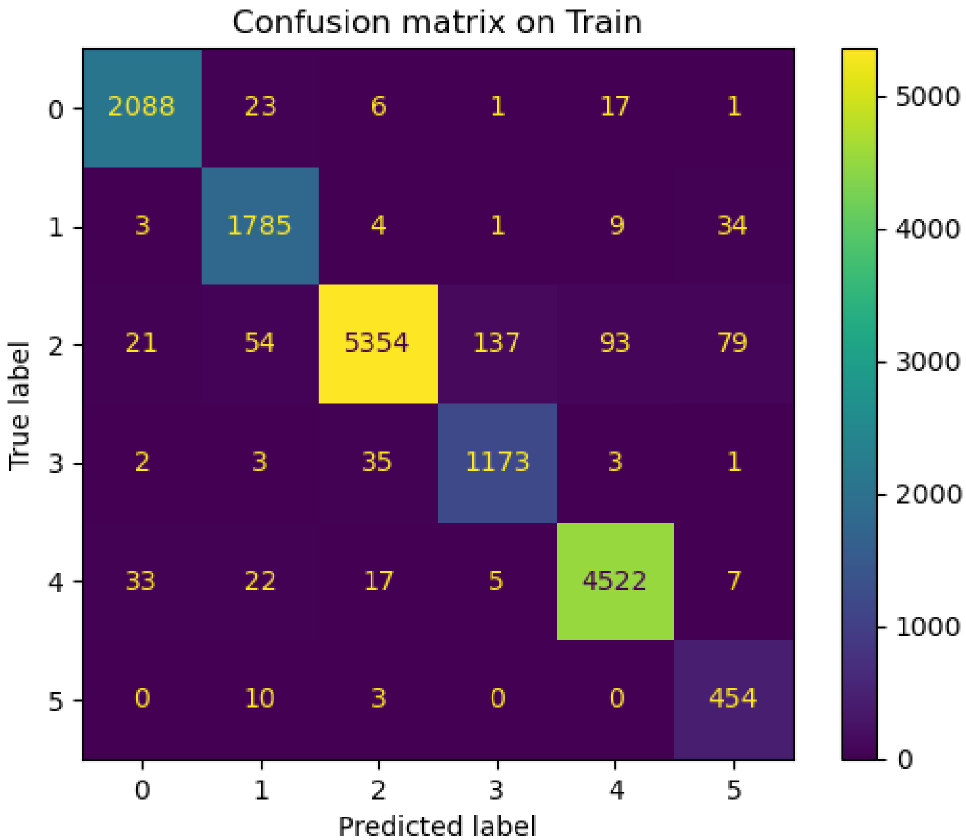


As the result, we can claim that $C = 1$ give a model with good accuracy and avoid overfitting. We will test the model again in test set.

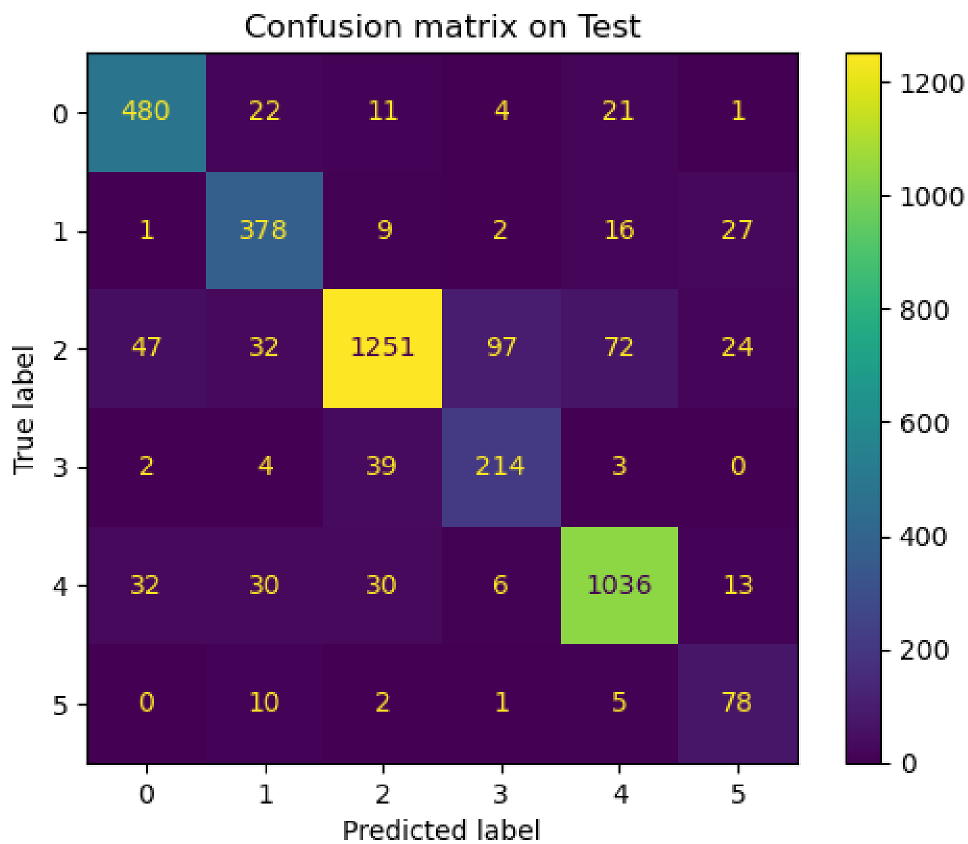
```
[ ]: best_svm_linear_model = SVC(kernel='linear', C=1)
```

```
[ ]: best_svm_linear_model.fit(X_train, y_train)
     evaluate_model(best_svm_linear_model, X_train, X_test, y_train, y_test,␣
      ↪include_training=True)
```

```
Score of on train are:
        - Accuracy score: 0.9610
        - Micro F1 score: 0.9610
        - Macro F1 score: 0.9442
Score of on test are:
        - Accuracy score: 0.8592
        - Micro F1 score: 0.8592
        - Macro F1 score: 0.8101
```



Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2088
23
6
1
17
1
3
1785
4
1
9
34
21
54
5354
137
93
79
2
3
35
1173
3
1
33
22
17
5
4522
7
0
10
3
0
0
454
5000
4000
3000
2000
1000
0



3.2 Radial basis function kernel

Formula:

$$k(x, z) = e^{-\gamma||x-z||_2^2}$$

First, we using grid search in a big domain.

[ ]:
```
dict_param = {
    'C' : np.asarray([0.001, 0.01, 0.1, 1, 10.0, 100]),
    'gamma': np.logspace(-3, 2, 6)
}

grid_search = GridSearchCV(SVC(kernel='rbf'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=8,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02]),
                         'gamma': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02])})
```

[ ]:
```
print('Best score: ', grid_search.best_score_, '\n')

print('Bad hyperparameter:')
df = pd.DataFrame(
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)
df = df[df['score'] < 0.8]
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)
```

```
Best score:  0.8520625000000001

Bad hyperparameter:
C 0.001
C 0.01
C 0.1
gamma 0.001
gamma 10.0
gamma 100.0
```

We fiter all the parameter that appear in all the bad model (validation accuracy < 0.80) * C = 0.001 * C = 0.01 * C = 0.1 * $\gamma = 0.001$ * $\gamma = 10$ * $\gamma = 100$

So that we can shrink the range of parameter

We repeat the algorithm again and again until there is no bad parameter to recieve the best model

[ ]:
```
dict_param = {
    'C' : np.linspace(1, 100, 10),
    'gamma': np.logspace(-2, 0, 10)
}

grid_search = GridSearchCV(SVC(kernel='rbf'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=8,
             param_grid={'C': array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,
78.,  89., 100.]),
                         'gamma': array([0.01      , 0.01668101, 0.02782559,
0.04641589, 0.07742637,
       0.12915497, 0.21544347, 0.35938137, 0.59948425, 1.        ])})
```

[ ]:
```python
df = pd.DataFrame(
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)

df = df[df['score'] < 0.8]
for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 100 // len(dict_param[param]):
      print(param, value)
```

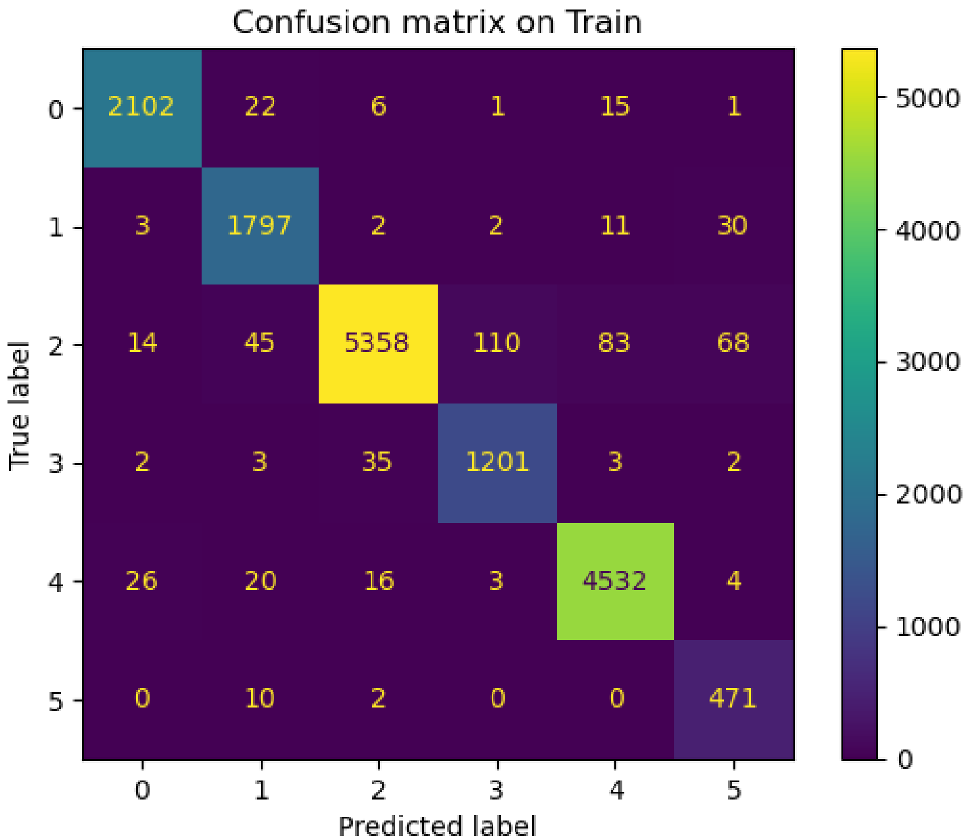
[ ]:
```python
print(grid_search.best_estimator_, grid_search.best_score_)
```

```
SVC(C=56.0, gamma=0.01) 0.8540000000000001
```

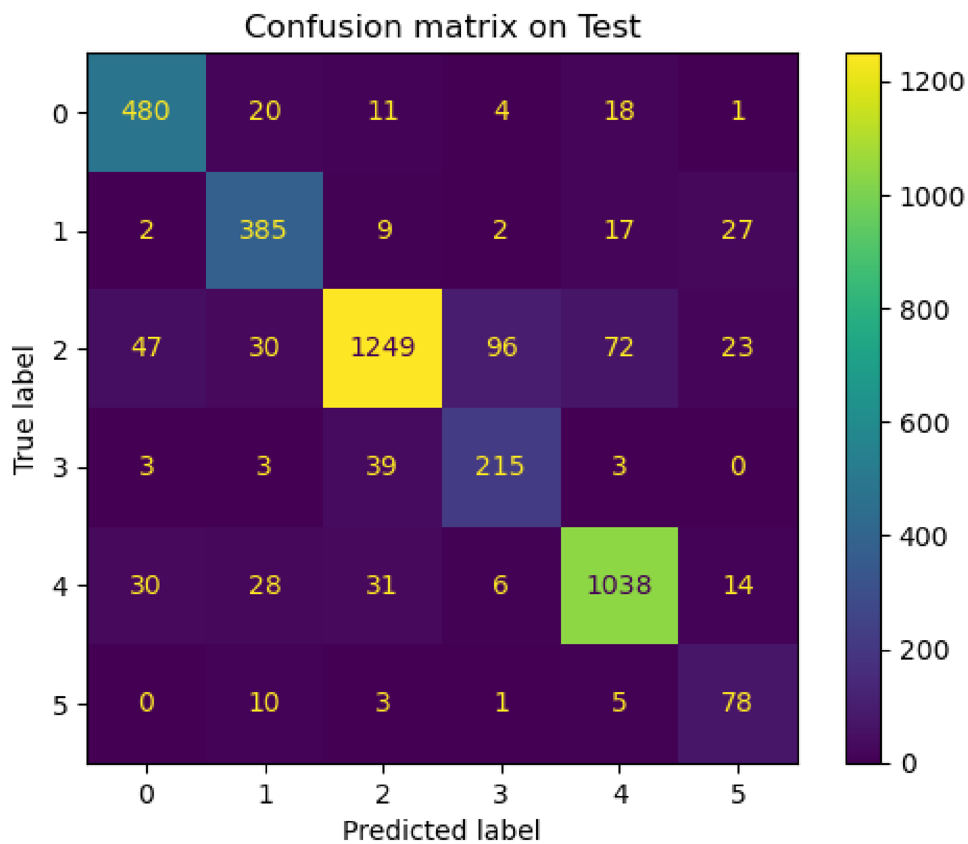
Evaluate the best rbf kernel SVM model:

[ ]:
```python
best_svm_rbf_model = SVC(C=56.0, gamma=0.01)
best_svm_rbf_model.fit(X_train, y_train)
evaluate_model(best_svm_rbf_model, X_train, X_test, y_train, y_test,␣
 ↪include_training=True)
```

```
Score of on train are:
        - Accuracy score: 0.9663
        - Micro F1 score: 0.9663
        - Macro F1 score: 0.9520
Score of on test are:
        - Accuracy score: 0.8612
        - Micro F1 score: 0.8612
        - Macro F1 score: 0.8123
```



Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2102
22
6
1
15
1
3
1797
2
2
11
30
14
45
5358
110
83
68
2
3
35
1201
3
2
26
20
16
3
4532
4
0
10
2
0
0
471
5000
4000
3000
2000
1000
0



3.3 Sigmoid function kernel

Formula:

$$k(x, z) = tanh(\gamma x^T z + r)$$

We use the same method in the above section to tuning this kernel

[ ]:
```
dict_param = {
    'C' : np.asarray([0.001, 0.01, 0.1, 1, 10.0, 100]),
    'gamma': np.asarray([0.001, 0.01, 0.1, 1, 10.0, 100]),
    'coef0': np.asarray([0.001, 0.01, 0.1, 1, 10.0, 100])
}

grid_search = GridSearchCV(SVC(kernel='sigmoid'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='sigmoid'), n_jobs=8,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02]),
                         'coef0': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
```

```
1.e+02]),
                         'gamma': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02])})
```

[ ]:
```python
print('Best score: ', grid_search.best_score_, '\n')

print('Bad hyperparameter:')
df = pd.DataFrame(
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    coef0 = [val['coef0'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)
df = df[df['score'] < 0.8]

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 216 // len(dict_param[param]):
      print(param, value)
```

```
Best score:  0.8536249999999999

Bad hyperparameter:
C 0.001
C 0.01
gamma 0.001
gamma 100.0
coef0 10.0
coef0 100.0
```

[ ]:
```python
dict_param = {
    'C' : np.logspace(0, 2, 4),
    'gamma': np.logspace(-2, 0, 4),
    'coef0': np.logspace(-3, -1, 4),
}

grid_search = GridSearchCV(SVC(kernel='sigmoid'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='sigmoid'), n_jobs=8,
             param_grid={'C': array([  1.        ,   4.64158883,  21.5443469 ,
100.        ]),
                         'coef0': array([0.001     , 0.00464159, 0.02154435, 0.1
]),
                         'gamma': array([0.01      , 0.04641589, 0.21544347, 1.
])})
```

```
[ ]: print('Best score:', grid_search.best_score_)
df = pd.DataFrame(
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    coef0 = [val['coef0'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)
df = df[df['score'] < 0.8]
print(len(df))
for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 64 // len(dict_param[param]):
      print(param, value)
```

```
Best score: 0.85375
32
```

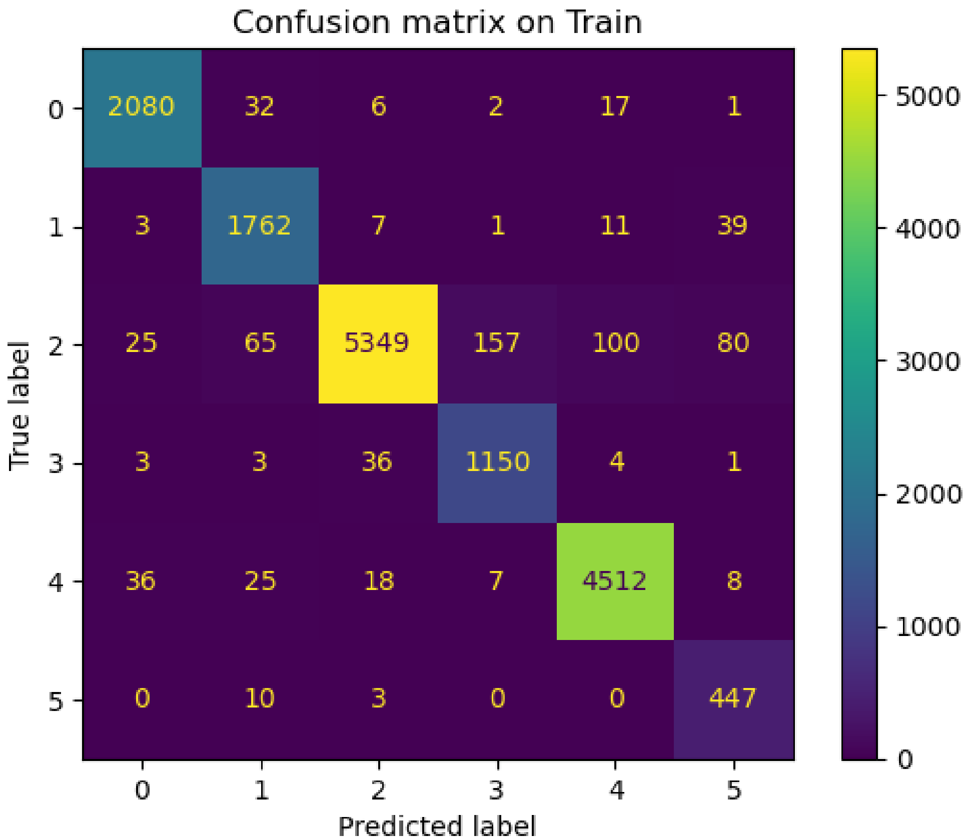
Since the parameters become unshrinkable. We will take the best value until now

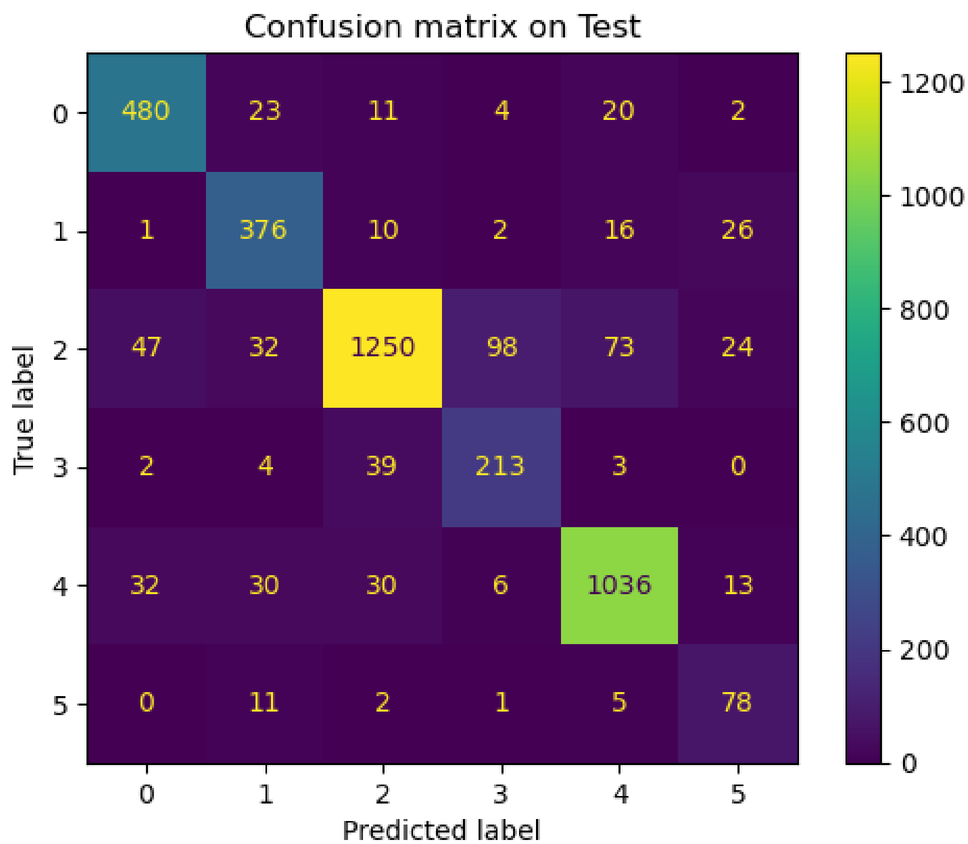
```
[ ]: print(grid_search.best_estimator_, grid_search.best_score_)
```

```
SVC(C=4.641588833612778, coef0=0.1, gamma=0.21544346900318834, kernel='sigmoid')
0.85375
```

```
[ ]: best_svm_sig_model = SVC(C=4.641588833612778, coef0=0.1, gamma=0.
  ↪21544346900318834, kernel='sigmoid')
best_svm_sig_model.fit(X_train, y_train)
evaluate_model(best_svm_sig_model, X_train, X_test, y_train, y_test,␣
  ↪include_training=True)
```

```
Score of on train are:
        - Accuracy score: 0.9563
        - Micro F1 score: 0.9563
        - Macro F1 score: 0.9379
Score of on test are:
        - Accuracy score: 0.8582
        - Micro F1 score: 0.8582
        - Macro F1 score: 0.8086
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2080
32
6
2
17
1
3
1762
7
1
11
39
25
65
5349
157
100
80
3
3
36
1150
4
1
36
25
18
7
4512
8
0
10
3
0
0
447
5000
4000
3000
2000
1000
0



3.4 Polynomial function kernel

Formula:

$$k(x, z) = (r + \gamma x^T z)^d$$

[ ]:
```
dict_param = {
    'C' : np.asarray([0.001, 0.01, 0.1, 1, 10.0, 100]),
    'gamma': np.asarray([0.001, 0.01, 0.1, 1]),
    'coef0': np.asarray([0.001, 0.01, 0.1, 1]),
    'degree': np.asarray([2, 3, 4])
}

grid_search = GridSearchCV(SVC(kernel='poly'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='poly'), n_jobs=8,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02]),
                         'coef0': array([0.001, 0.01 , 0.1  , 1.   ]),
                         'degree': array([2, 3, 4]),
```

```
                         'gamma': array([0.001, 0.01 , 0.1  , 1.   ])})
```

[ ]:
```
print('Best score: ', grid_search.best_score_, '\n')

print('Bad hyperparameter:')
df = pd.DataFrame(
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    coef0 = [val['coef0'] for val in grid_search.cv_results_['params']],
    degree = [val['degree'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)
df = df[df['score'] < 0.80]

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 288 // len(dict_param[param]):
      print(param, value)
```

```
Best score:  0.8518125000000001

Bad hyperparameter:
C 0.001
C 0.01
coef0 0.001
```

[ ]:
```
dict_param = {
    'C' : np.asarray([0.1, 1, 10.0, 100]),
    'gamma': np.asarray([0.001, 0.01, 0.1, 1]),
    'coef0': np.linspace(0.1, 1, 4),
    'degree': np.asarray([2, 3, 4])
}

grid_search = GridSearchCV(SVC(kernel='poly'), dict_param, cv = 5, n_jobs=8)
grid_search.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='poly'), n_jobs=8,
             param_grid={'C': array([  0.1,   1. ,  10. , 100. ]),
                         'coef0': array([0.1, 0.4, 0.7, 1. ]),
                         'degree': array([2, 3, 4]),
                         'gamma': array([0.001, 0.01 , 0.1  , 1.   ])})
```

[ ]:
```
print('Best score: ', grid_search.best_score_, '\n')

print('Bad hyperparameter:')
df = pd.DataFrame(
```

```
  dict(
    C = [val['C'] for val in grid_search.cv_results_['params']],
    gamma = [val['gamma'] for val in grid_search.cv_results_['params']],
    coef0 = [val['coef0'] for val in grid_search.cv_results_['params']],
    degree = [val['degree'] for val in grid_search.cv_results_['params']],
    score = grid_search.cv_results_['mean_test_score']
  )
)
df = df[df['score'] < 0.8]
print('Number of filtered models:', len(df))

for param in dict_param:
  for value in dict_param[param]:
    if len(df[df[param] == value]) == 192 // len(dict_param[param]):
      print(param, value)
```

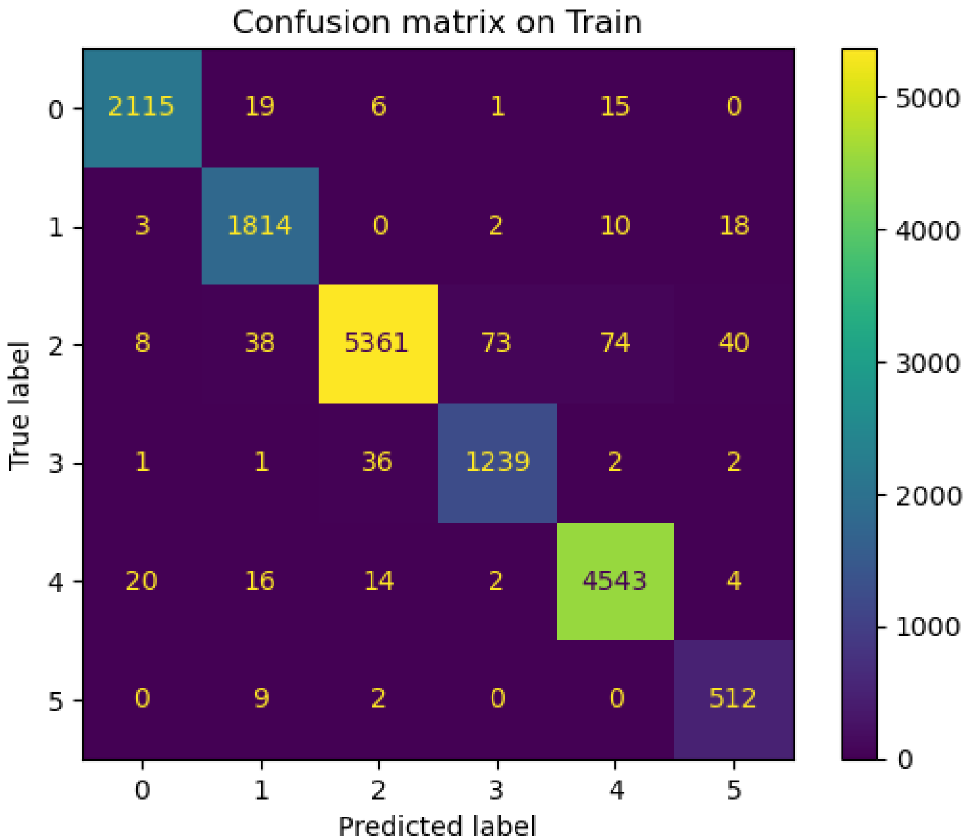
```
Best score:  0.853375

Bad hyperparameter:
Number of filtered models: 131
```

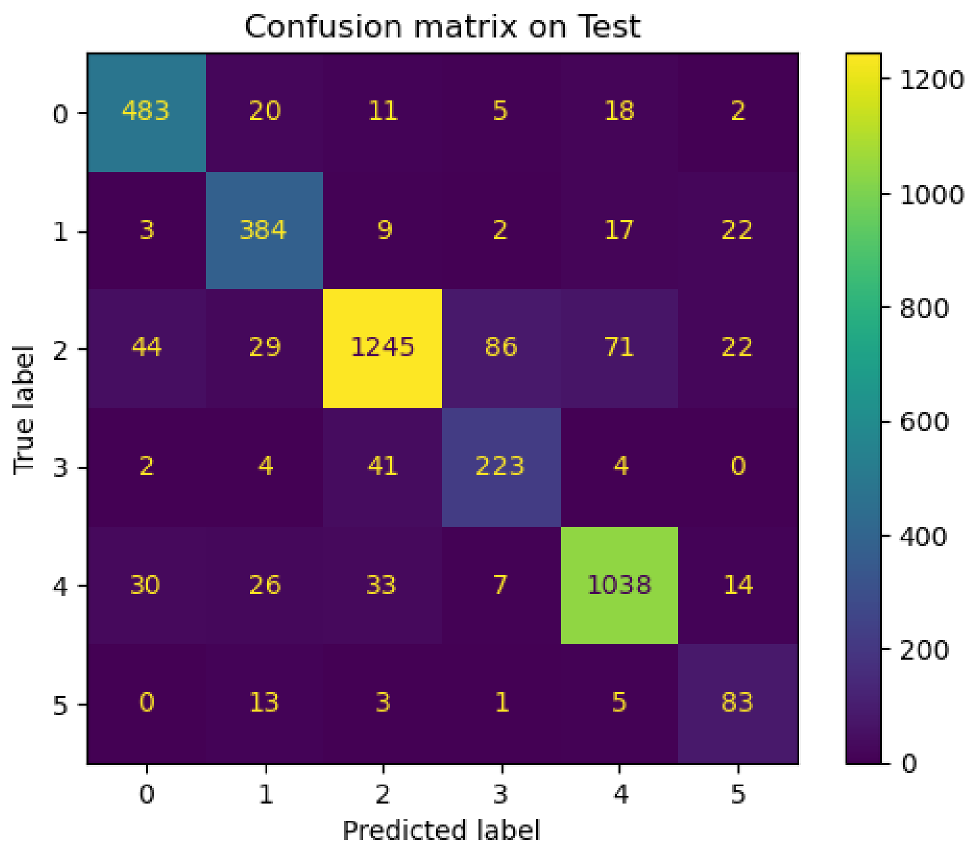
[ ]:
```
print(grid_search.best_estimator_, grid_search.best_score_)
```

```
SVC(C=100.0, coef0=0.7, degree=2, gamma=0.01, kernel='poly') 0.853375
```

[ ]:
```
best_svm_poly_model = SVC(C=100.0, coef0=0.7, degree=2, gamma=0.01,
 ↪kernel='poly')
best_svm_poly_model.fit(X_train, y_train)
evaluate_model(best_svm_poly_model, X_train, X_test, y_train, y_test,
 ↪include_training=True)
```

```
Score of on train are:
        - Accuracy score: 0.9740
        - Micro F1 score: 0.9740
        - Macro F1 score: 0.9657
Score of on test are:
        - Accuracy score: 0.8640
        - Micro F1 score: 0.8640
        - Macro F1 score: 0.8188
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2115
19
6
1
15
0
3
1814
0
2
10
18
8
38
5361
73
74
40
1
1
36
1239
2
2
20
16
14
2
4543
4
0
9
2
0
0
512
5000
4000
3000
2000
1000
0



4 Conclusion

All the kernels have almost the same result.

From the result, even though poly has a small better score compared to others, I choose rbf kernel to be the best one in this dataset.

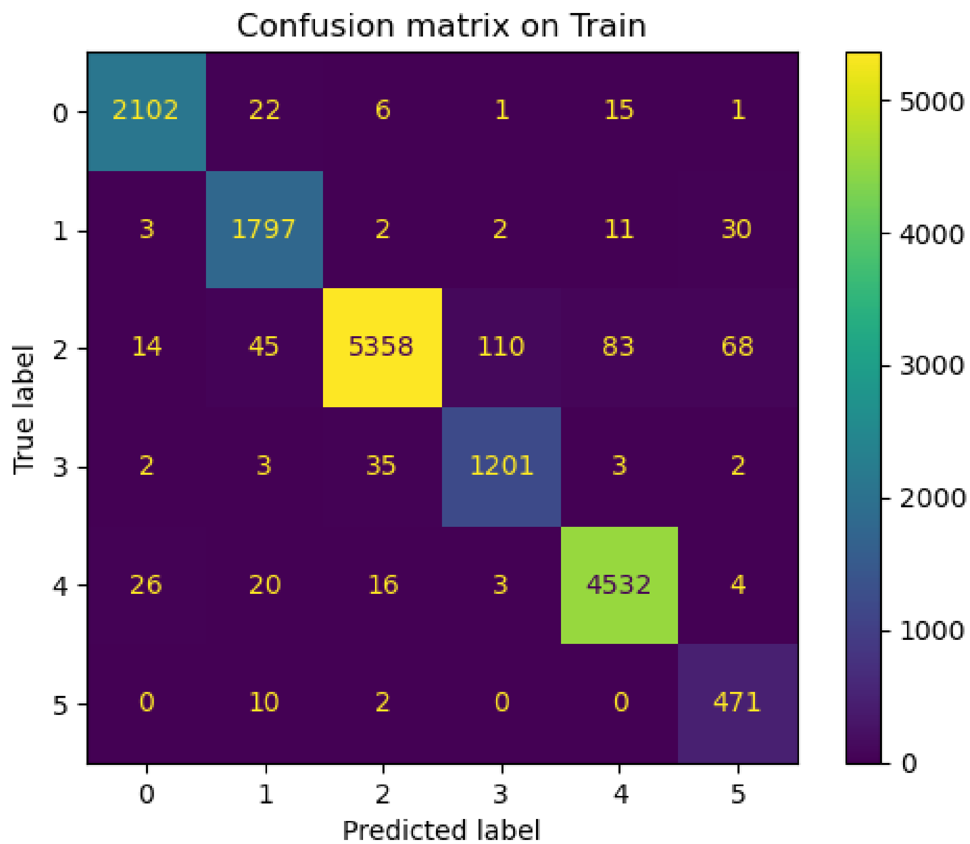
```
[ ]: best_svm_model = best_svm_rbf_model
```

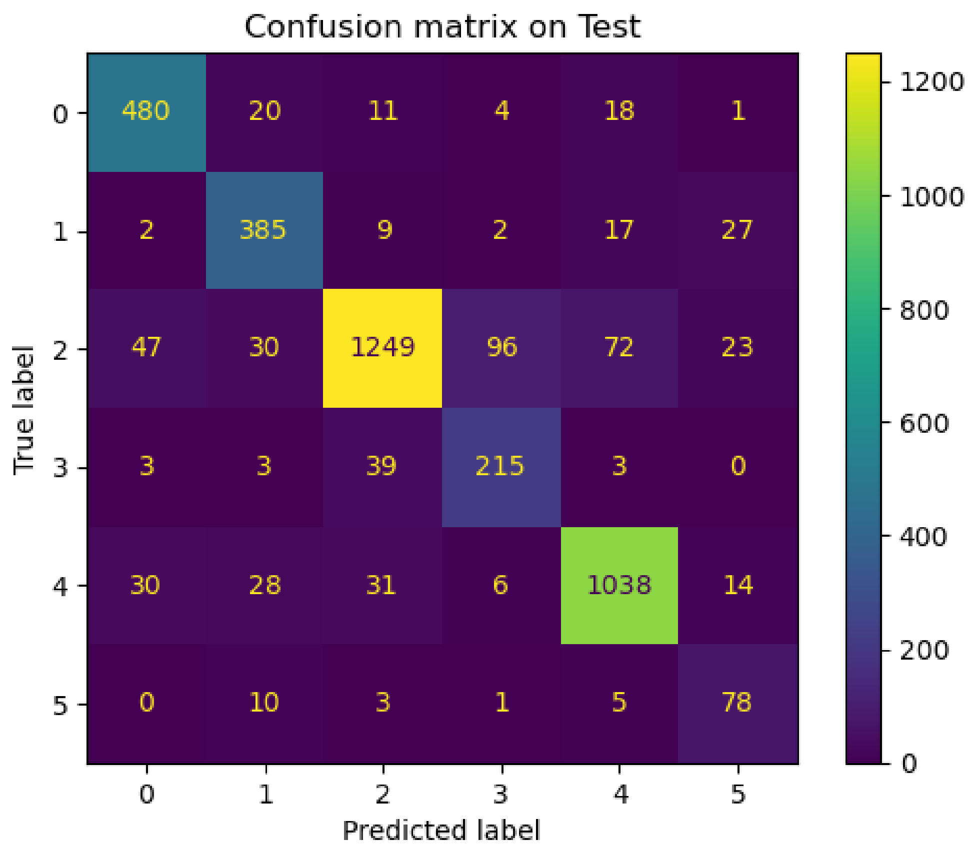
Evaluate the model:

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

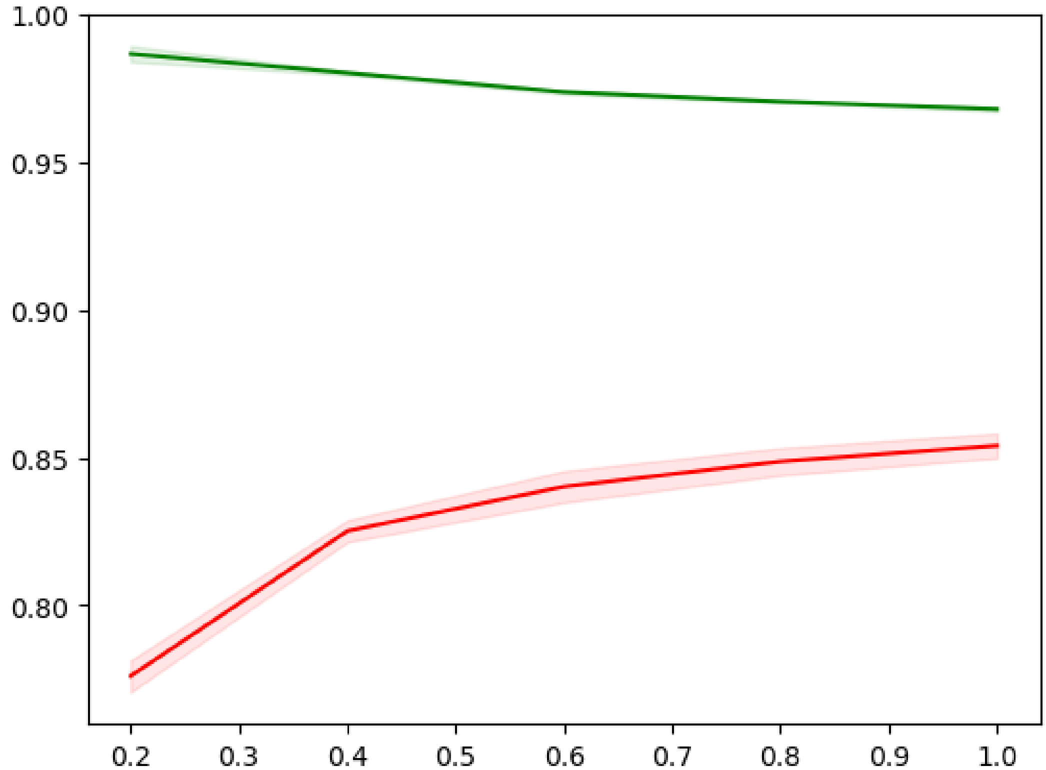
```
Score of on train are:
        - Accuracy score: 0.9663
        - Micro F1 score: 0.9663
        - Macro F1 score: 0.9520
Score of on test are:
        - Accuracy score: 0.8612
        - Micro F1 score: 0.8612
```

- Macro F1 score: 0.8123





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



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

dump(best_svm_model, directory + "best_svm_tfidf_l1_model.joblib")
```

[ ]: ['data/models/svm/best_svm_tfidf_l1_model.joblib']