Support Vector Machine (SVM) - TF-IDF

May 3, 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
X_test = X_test_tfidf
```

2 Basic training

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

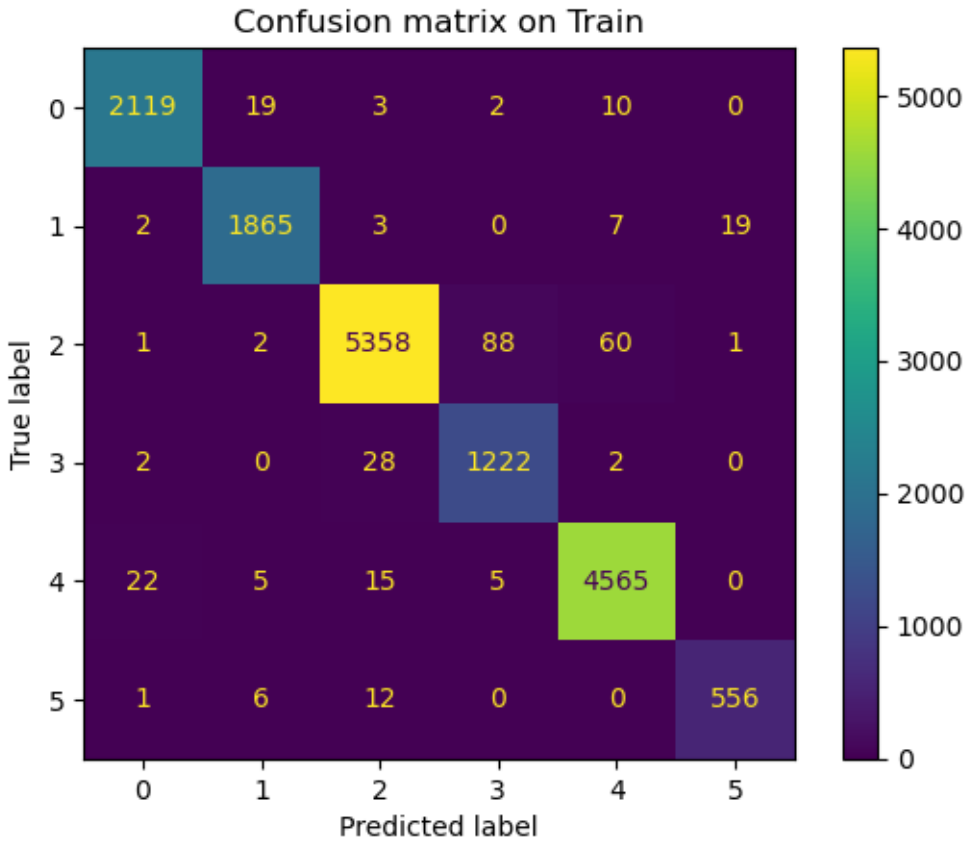
[ ]:
```
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
```

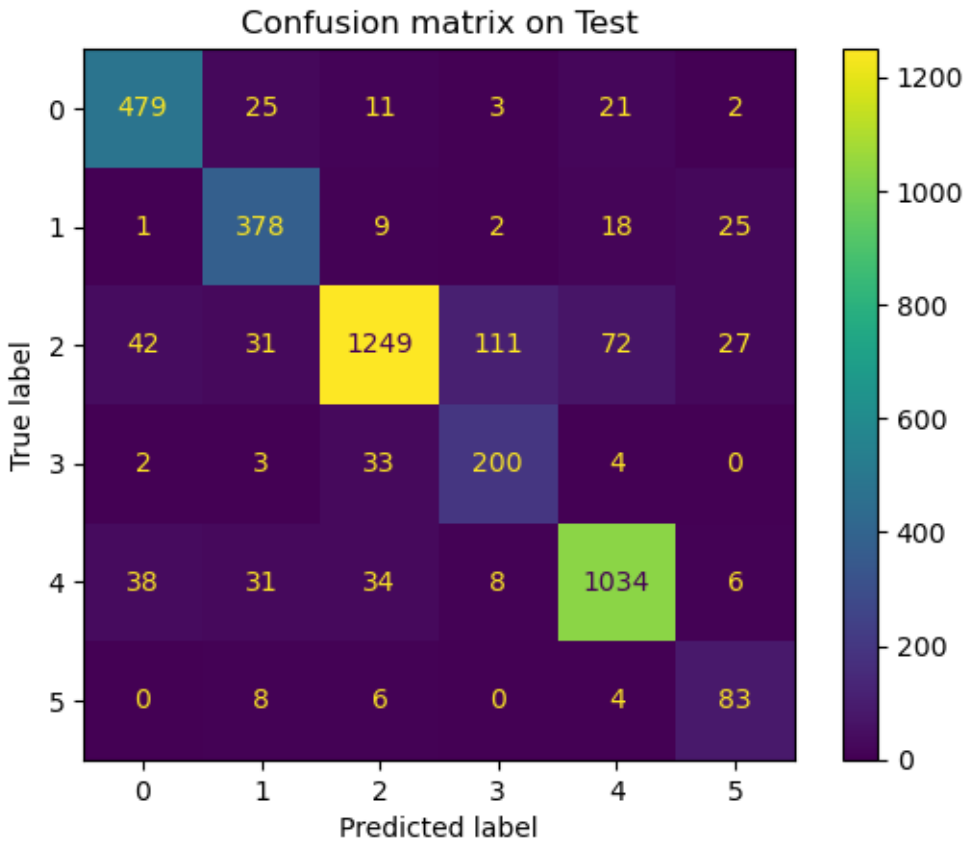
[ ]:
```
SVC(kernel='linear')
```

Evaluate model using preset function:

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

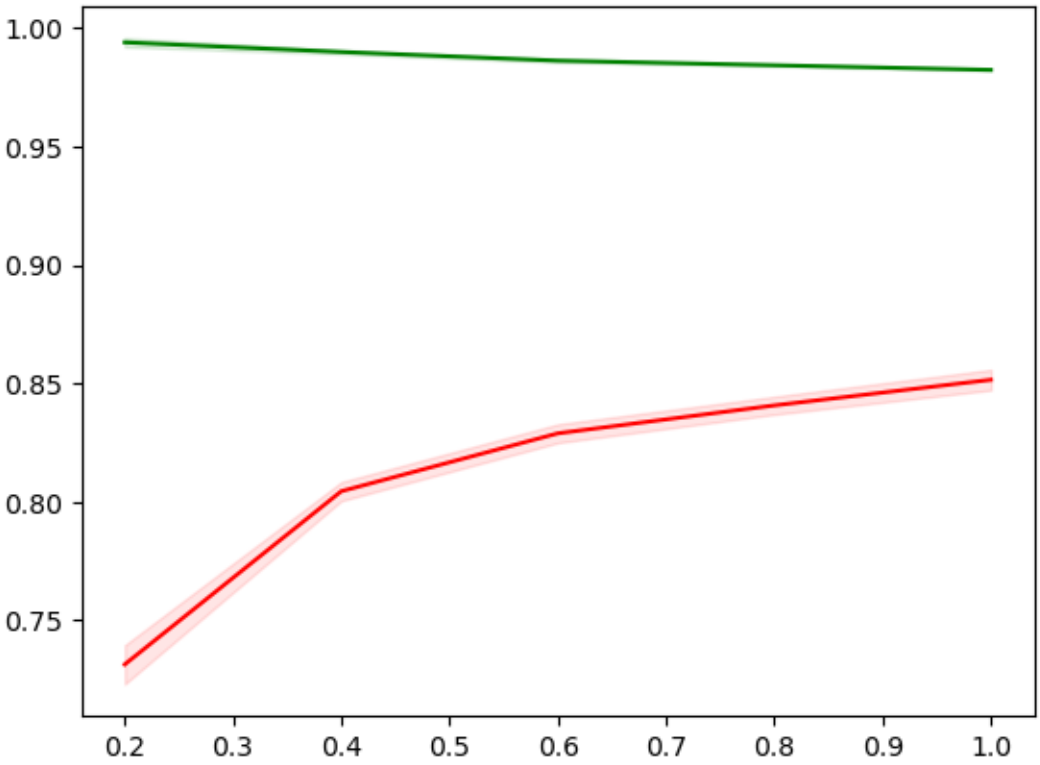
```
Score of on train are:
        - Accuracy score: 0.9803
        - Micro F1 score: 0.9803
        - Macro F1 score: 0.9754
Score of on test are:
        - Accuracy score: 0.8558
        - Micro F1 score: 0.8558
        - Macro F1 score: 0.8090
```





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)
```

```
    # 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)
```

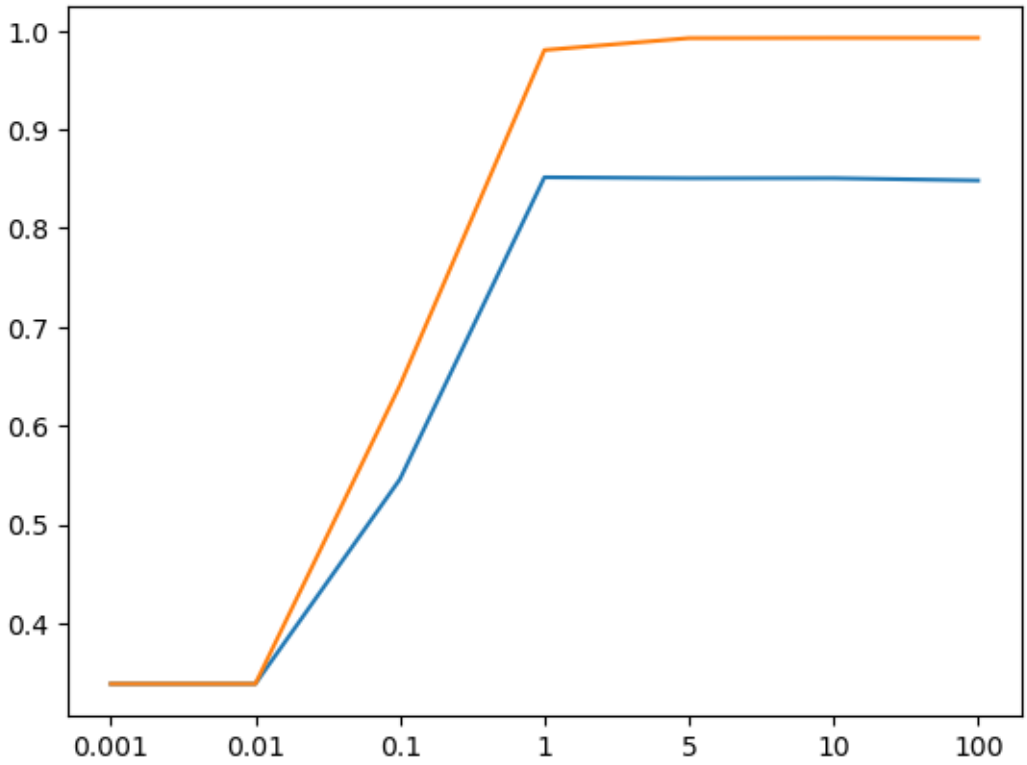
[ ]:
```
# Print the result
print(C_list)
print(trs_list)
print(cvs_list)

# 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)
```

```
[0.001, 0.01, 0.1, 1, 5, 10, 100]
[0.3386875, 0.3386875, 0.640875, 0.9803125, 0.992375, 0.992625, 0.992625]
[0.3386875000000001, 0.3386875000000001, 0.5458125, 0.8514999999999999,
0.8505625, 0.850625, 0.8483124999999999]
```

```
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)
```

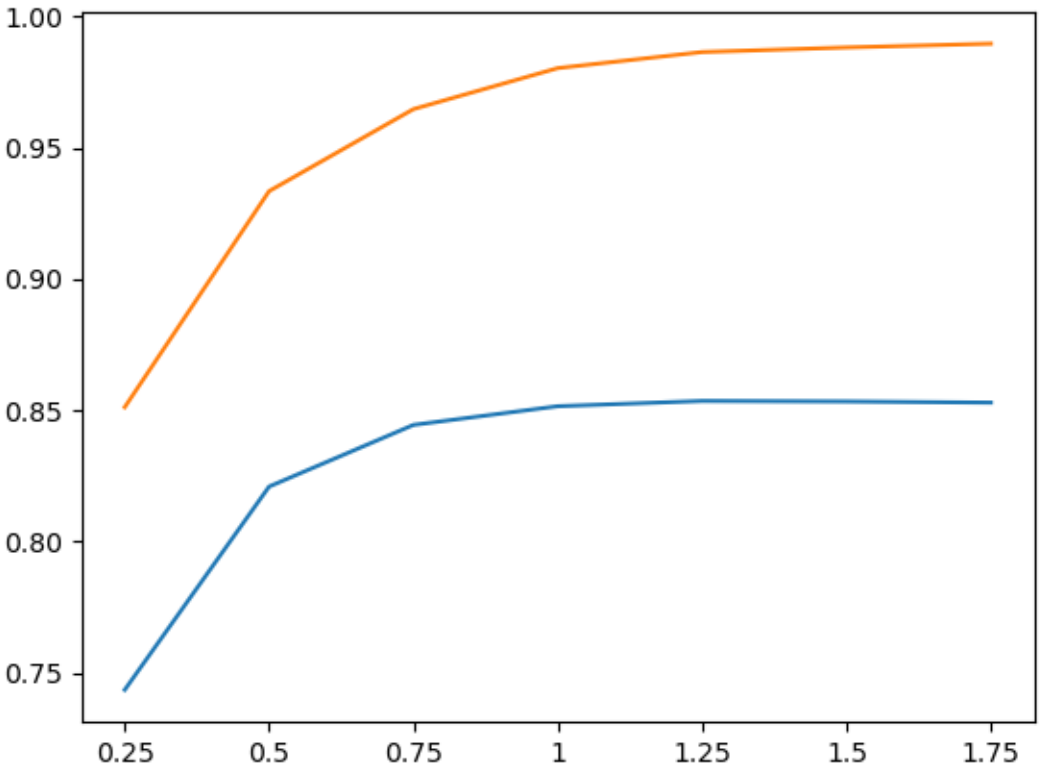
```
    cvs_list.append(cv_score)
```

[ ]:
```
# 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]
[0.851125, 0.9334375, 0.9646875, 0.9803125, 0.9864375, 0.9881875, 0.989625]
[0.7434375, 0.820875, 0.844375, 0.8514999999999999, 0.8535, 0.8533125, 0.852875]
```

```
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')]
```

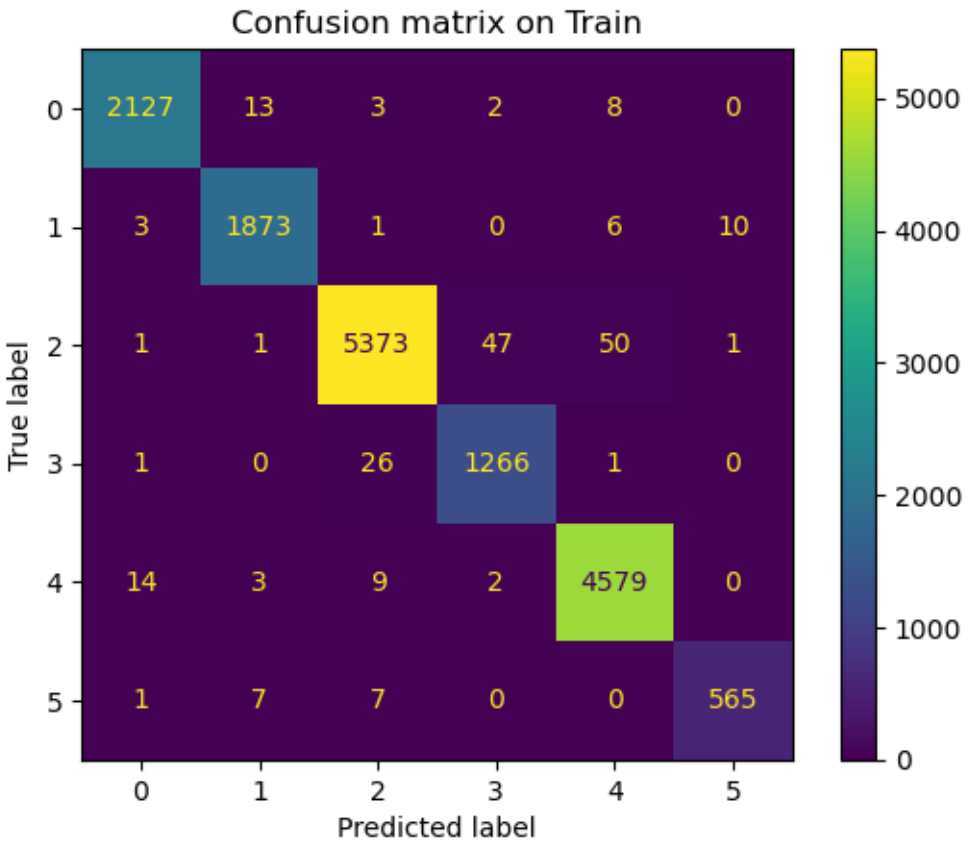


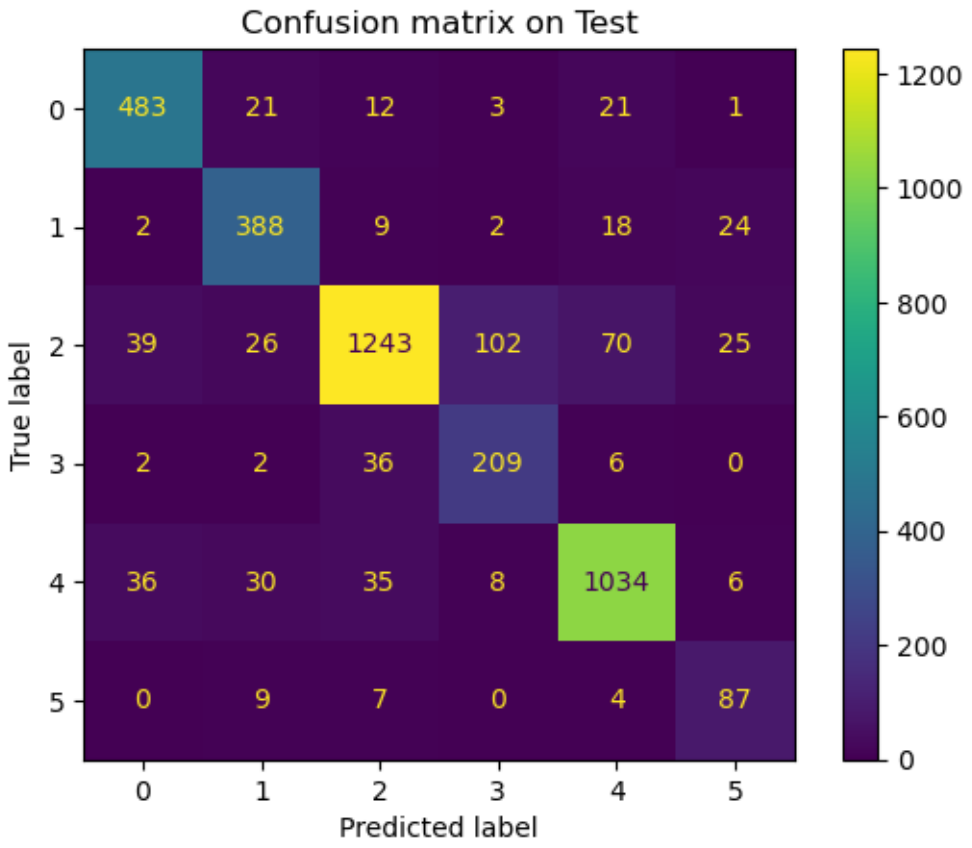
As the result, we can claim that $C = 1.25$ give a model with good accuracy and avoid overfitting. We will test the model again in test set.

[ ]:
```
best_svm_linear_model = SVC(kernel='linear', C=1.25)
```

[ ]:
```
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: 0.9864
        - Micro F1 score: 0.9864
        - Macro F1 score: 0.9836
Score of on test are:
        - Accuracy score: 0.8610
        - Micro F1 score: 0.8610
        - Macro F1 score: 0.8183
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2127 13 3 2 8 0
3 1873 1 0 6 10
1 1 5373 47 50 1
1 0 26 1266 1 0
14 3 9 2 4579 0
1 7 7 0 0 565
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:  0.8518125

Bad hyperparameter:
C 0.001
C 0.01
C 0.1
C 1.0
gamma 0.001
gamma 10.0
gamma 100.0
```

We fiter all the parameter that appear in all the bad model (validation accuracy < 0.85) * C $= 0.001$ * C $= 0.01$ * C $= 0.1$ * C $= 1$ * $\gamma = 0.001$ * $\gamma = 10$ * $\gamma = 100$

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(10, 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([ 10.,  20.,  30.,  40.,  50.,  60.,  70.,
80.,  90., 100.]),
                         'gamma': array([0.01      , 0.01668101, 0.02782559,
0.04641589, 0.07742637,
       0.12915497, 0.21544347, 0.35938137, 0.59948425, 1.        ])})
```

[ ]:
```
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'] < 0.80]
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)
```

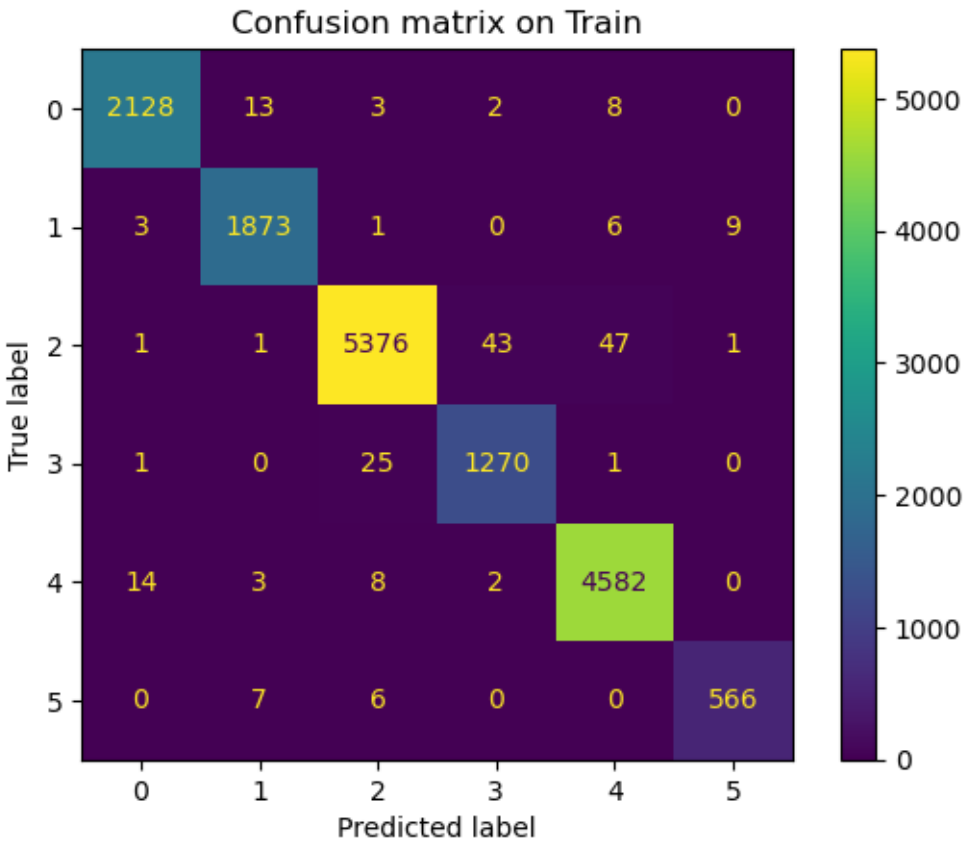
[ ]:
```
print(grid_search.best_estimator_, grid_search.best_score_)
```

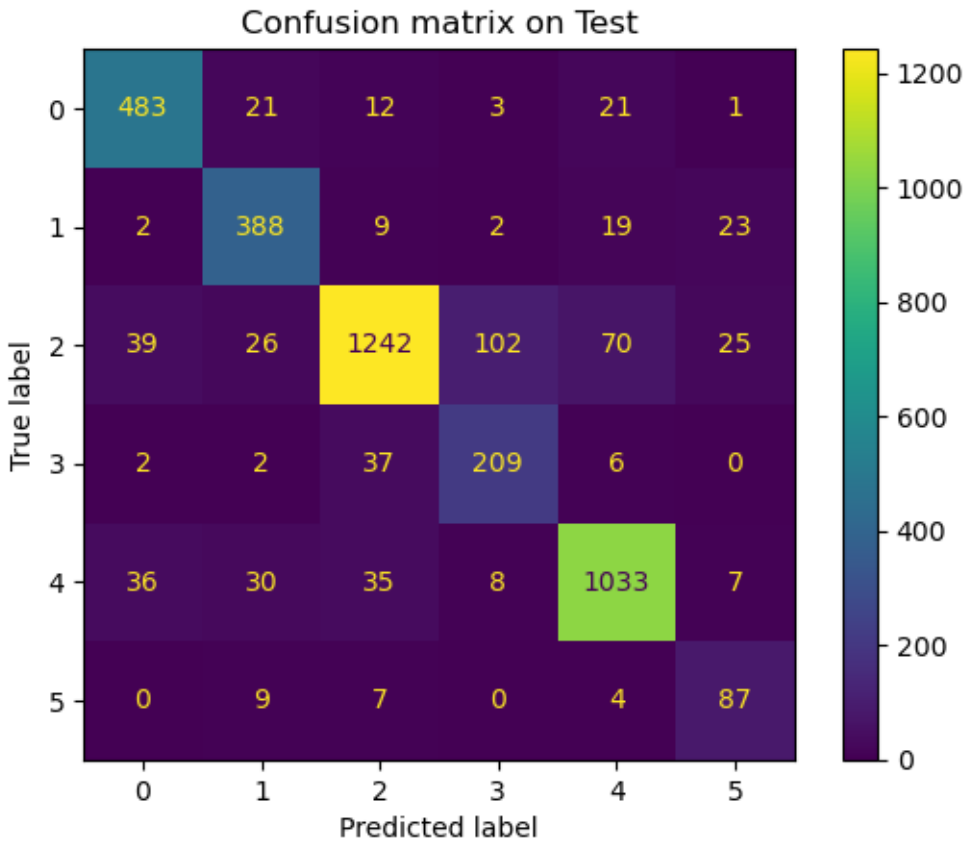
```
SVC(C=40.0, gamma=0.016681005372000592) 0.8535625
```

Evaluate the best rbf kernel SVM model:

[ ]:
```
best_svm_rbf_model = SVC(C=40.0, gamma=0.016681005372000592)
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: 0.9872
        - Micro F1 score: 0.9872
        - Macro F1 score: 0.9846
Score of on test are:
        - Accuracy score: 0.8605
        - Micro F1 score: 0.8605
        - Macro F1 score: 0.8179
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2128 13 3 2 8 0
3 1873 1 0 6 9
1 1 5376 43 47 1
1 0 25 1270 1 0
14 3 8 2 4582 0
0 7 6 0 0 566
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])})
```

[ ]:
```
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:  0.851625

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

[ ]:
```
dict_param = {
    'C' : np.linspace(0.1, 100, 5),
    'gamma': np.linspace(0.01, 10, 5),
    'coef0': np.linspace(0.001, 0.1, 5)
}

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([  0.1  ,  25.075,  50.05 ,  75.025, 100.
]),
                         'coef0': array([0.001  , 0.25075, 0.5005 , 0.75025, 1.
]),
                         'gamma': array([ 0.01  ,  2.5075,  5.005 ,  7.5025, 10.
])})
```

[ ]:
```
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]) == 125 // len(dict_param[param]):
      print(param, value)
```

```
Best score: 0.8514999999999999
106
C 25.075000000000003
gamma 2.5075
```

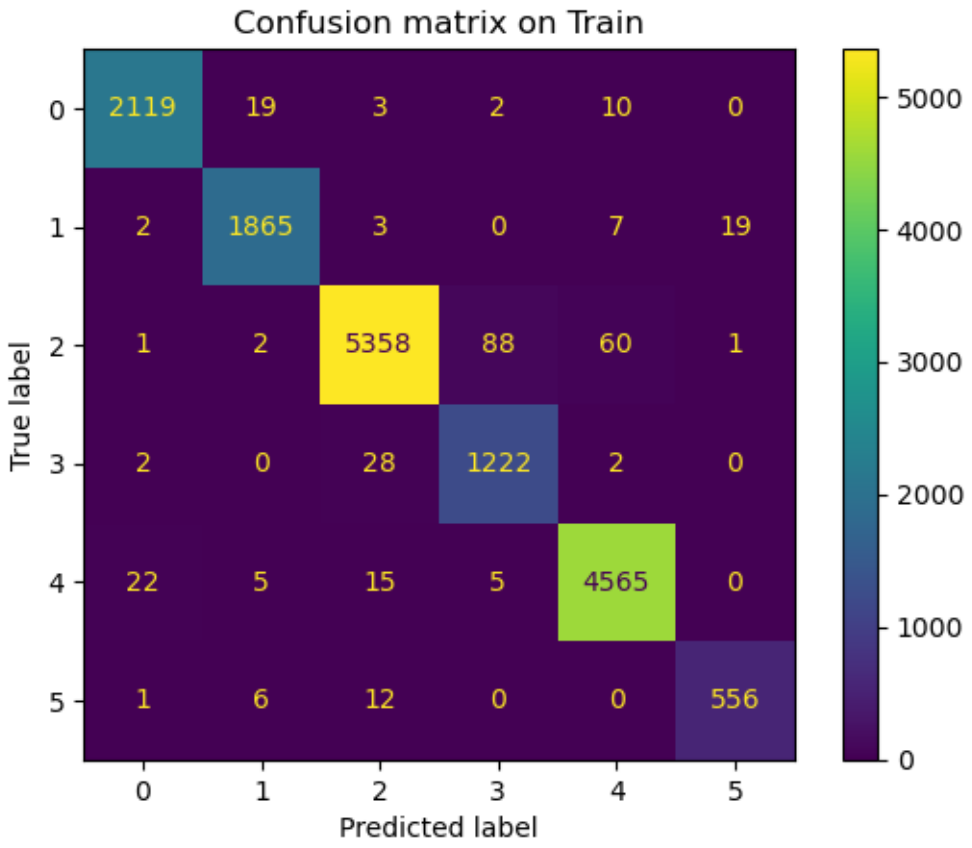
Since the parameters become unshrinkable. We will take the best value until now

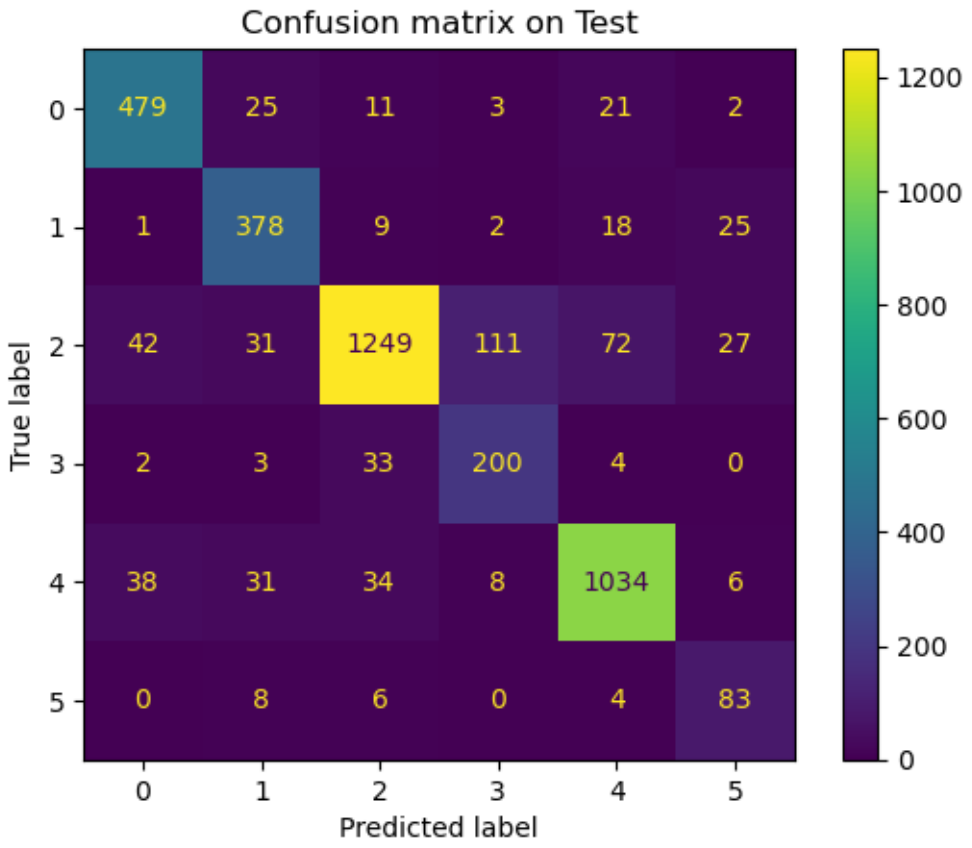
[ ]:
```
print(grid_search.best_estimator_, grid_search.best_score_)
```

```
SVC(C=100.0, coef0=0.001, gamma=0.01, kernel='sigmoid') 0.8514999999999999
```

[ ]:
```
best_svm_sig_model = SVC(C=100.0, coef0=0.001, gamma=0.01, 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: 0.9803
        - Micro F1 score: 0.9803
        - Macro F1 score: 0.9754
Score of on test are:
        - Accuracy score: 0.8558
        - Micro F1 score: 0.8558
        - Macro F1 score: 0.8090
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2119 19 3 2 10 0
2 1865 3 0 7 19
1 2 5358 88 60 1
2 0 28 1222 2 0
22 5 15 5 4565 0
1 6 12 0 0 556
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'] < 0.82]

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:  0.8518125

Bad hyperparameter:
C 0.001
C 0.01
C 0.1
gamma 0.001
coef0 0.001
coef0 0.01
```

[ ]:
```
dict_param = {
    'C' : np.linspace(1, 100, 4),
    'gamma': np.linspace(0.01, 1, 4),
    '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([  1.,  34.,  67., 100.]),
                         'coef0': array([0.1, 0.4, 0.7, 1. ]),
                         'degree': array([2, 3, 4]),
                         'gamma': array([0.01, 0.34, 0.67, 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)
```

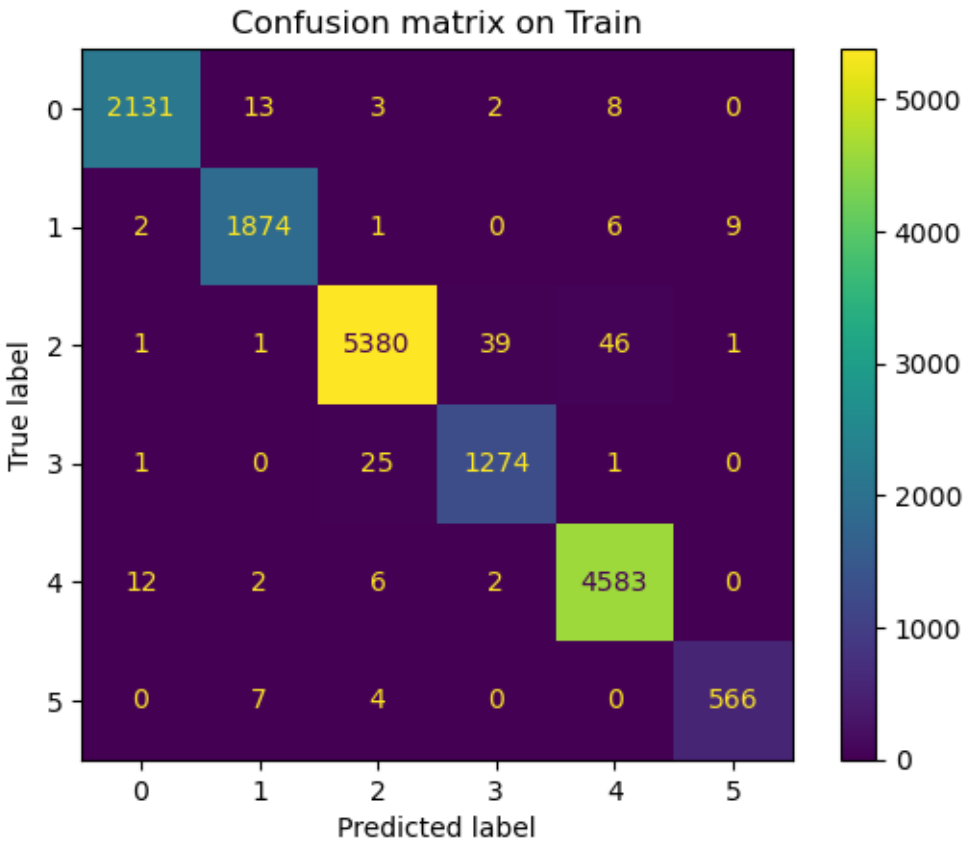
```
Best score:  0.8535

Bad hyperparameter:
Number of filtered models: 84
```

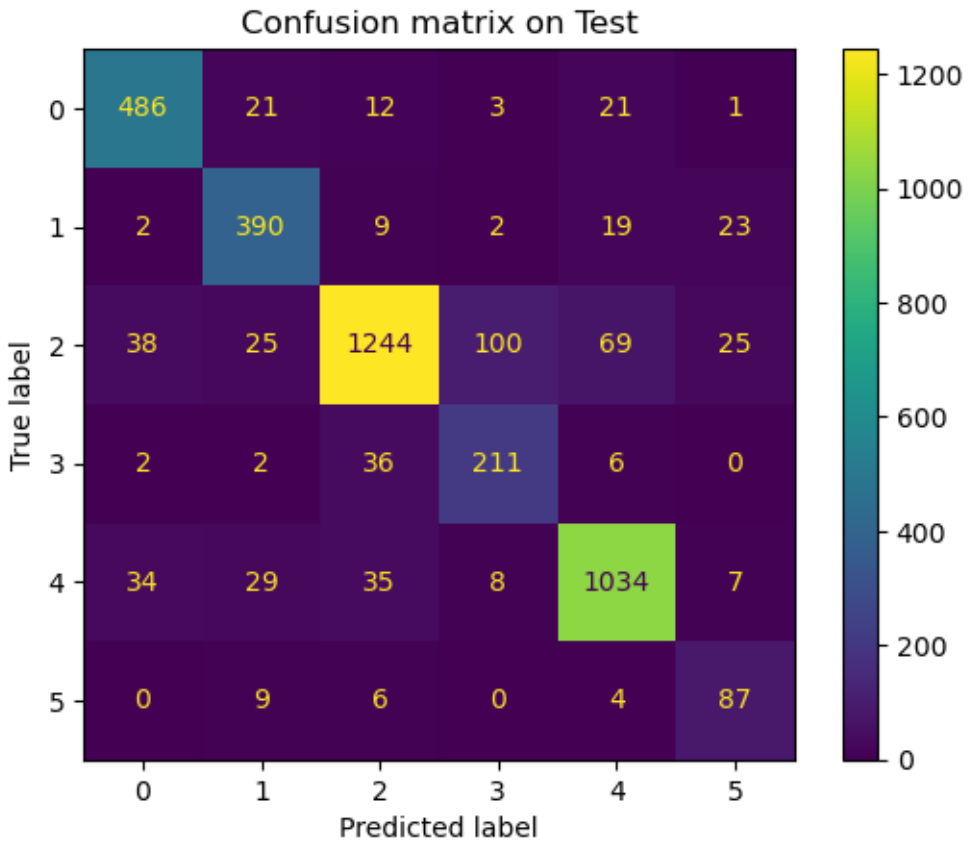
[ ]:
```
print(grid_search.best_estimator_, grid_search.best_score_)
```

```
SVC(C=100.0, coef0=0.7, degree=2, gamma=0.01, kernel='poly') 0.8535
```

[ ]:
```
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: 0.9880
        - Micro F1 score: 0.9880
        - Macro F1 score: 0.9856
Score of on test are:
        - Accuracy score: 0.8630
        - Micro F1 score: 0.8630
        - Macro F1 score: 0.8209
```


Confusion matrix on Train
True label
Predicted label
0
1
2
3
4
5
2131 13 3 2 8 0
2 1874 1 0 6 9
1 1 5380 39 46 1
1 0 25 1274 1 0
12 2 6 2 4583 0
0 7 4 0 0 566
5000
4000
3000
2000
1000
0



4 Conclusion

All the kernels have almost the same result. Sigmoid kernel is litte worse than the other

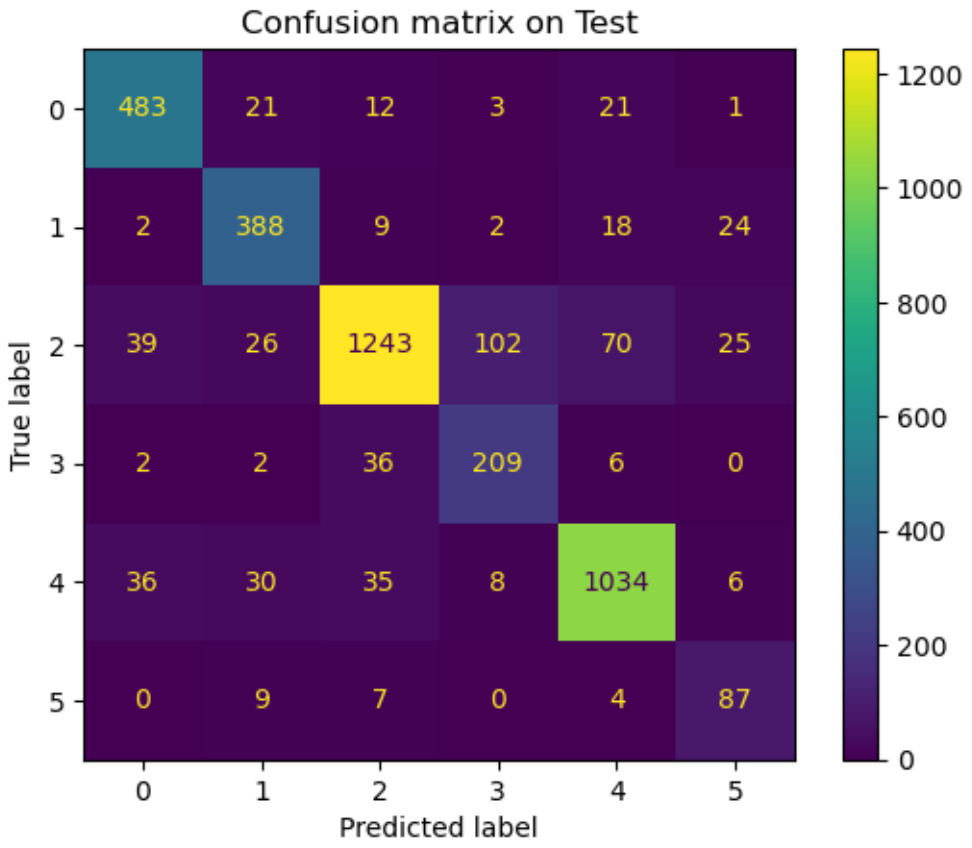
From the result, I choose linear kernel to be the best one in this dataset.

[ ]:
```
best_svm_model = best_svm_linear_model
```

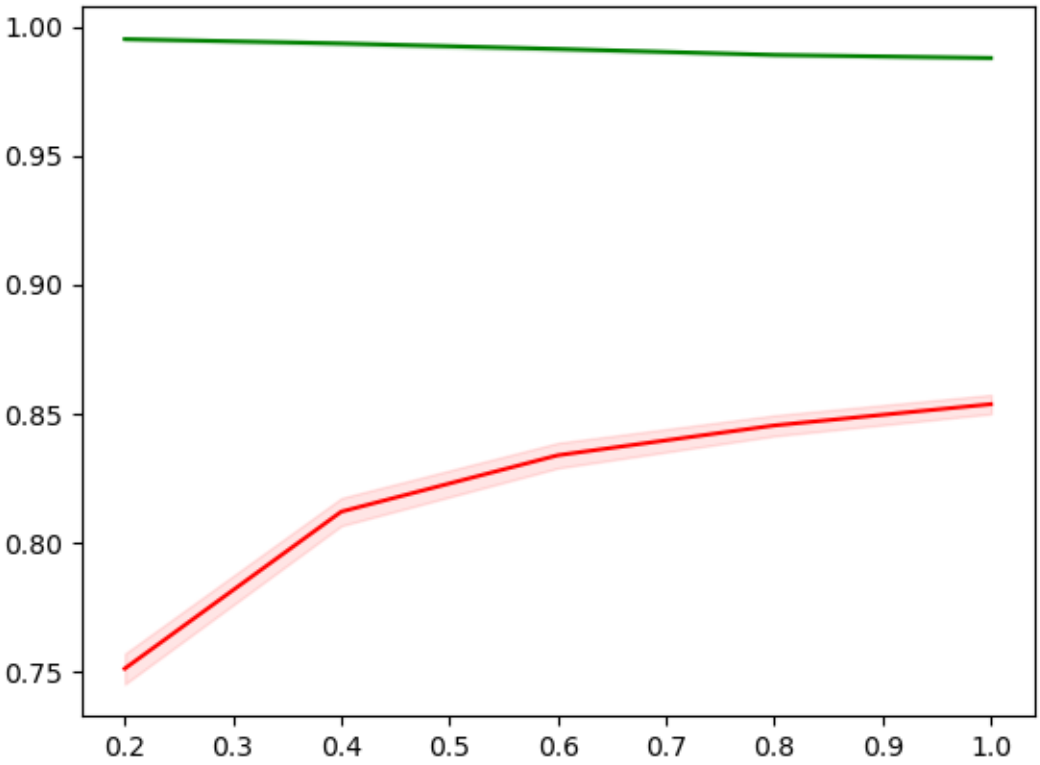
Evaluate the model:

[ ]:
```
evaluate_model(best_svm_model, X_train, X_test, y_train, y_test)
```

```
Score of on test are:
        - Accuracy score: 0.8610
        - Micro F1 score: 0.8610
        - Macro F1 score: 0.8183
```



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



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

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

[ ]:
```
['data/models/svm/best_svm_tfidf_model.joblib']
```