# SVM

December 14, 2020

[7]:
```
## Used train.csv and test.csv which are csv files that comprise of all the␣
 ↪training
# and testing data from the MNIST handwritten Dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import sklearn.preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn import metrics

# The training data has 42000 entries and 785 features
train_digits = pd.read_csv("train.csv")
```

[8]:
```
# Creating sections for training and testing data
# Creating training and test sets
# Splitting the data into train and test
A_matrix = train_digits.iloc[:, 1:]
Labels_A = train_digits.iloc[:, 0]

# Processing the images before analysis using scale function from sklearn.
 ↪preprocessing
# Scale is used to standardise the pixels(features) in the image. This needs to␣
 ↪be done because our dataset is sparse/widely soread out.
# Which can cause issues with our model being statistically skewed. Scaling the␣
 ↪data in this manner
# makes  the matrix entries have 0 mean and the standard variation of 1 in each␣
 ↪field.
A_matrix = sklearn.preprocessing.scale(A_matrix)

# Splitting the data with 42k datapoints into 15% training data and 5% testing␣
 ↪data
# The stratify property was used to ensure the data is split with even label␣
 ↪distribution
```

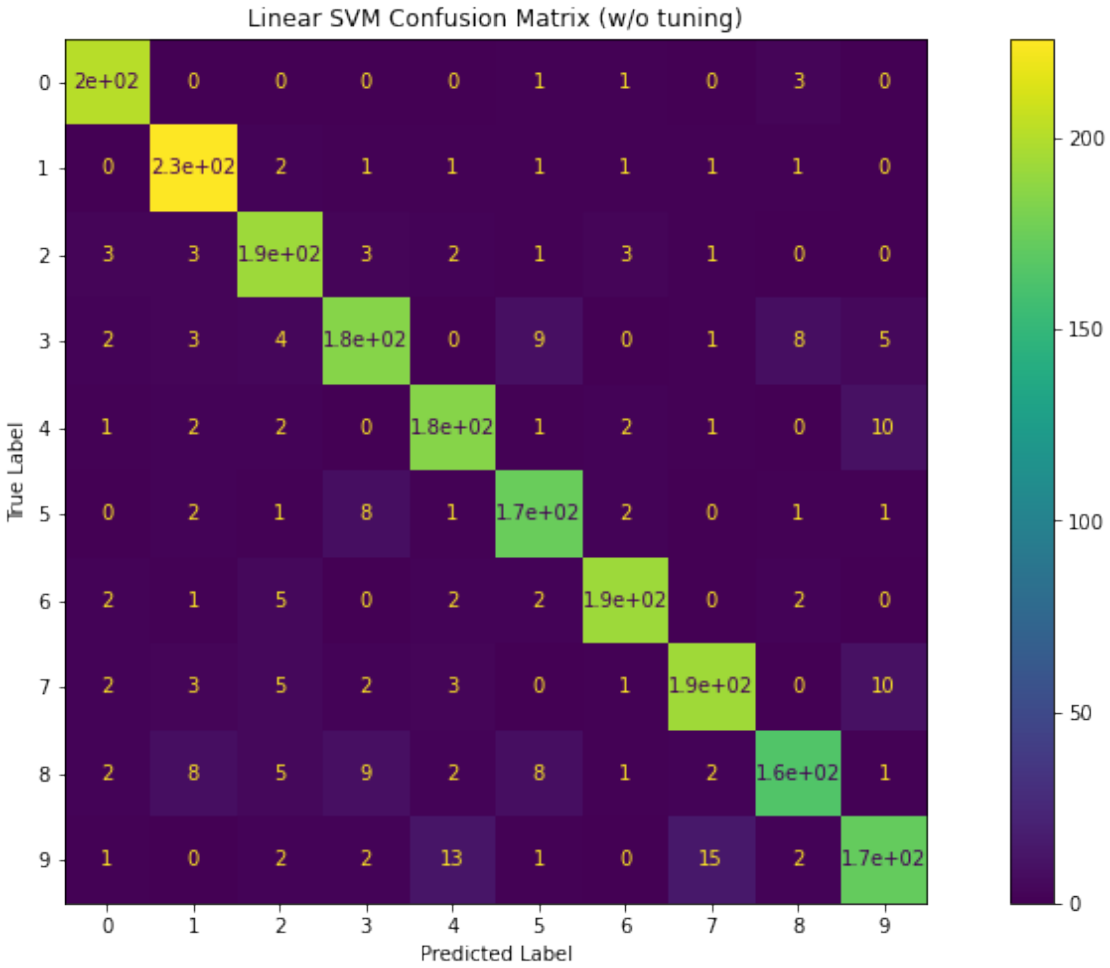
```
Train_A, Test_A, Train_y, Test_y = train_test_split(A_matrix, Labels_A,␣
↪test_size = 0.05, train_size=0.15, random_state=10, stratify=Labels_A)
```

[9]:
```
## In order to carry our Support vector classifications, I used the sklearn.svm.
↪svc function
## This function, as with any SVC has three main parameters which will be␣
↪altered
# 1)C: C is the tolerance/penalty for the error term 2)Gamma- controld the␣
↪kernel coeficient 3) kernel type
```

### 0.1 Building a Linear SVM

[10]:
```
#LINEAR SVM
Linear_SVM = svm.SVC(kernel='linear')
#Model Fitting
Linear_SVM.fit(Train_A, Train_y)
accuracy_predicting = svm_linear.predict(Test_A)
```

[11]:
```
#Creating a confusion matrix for visualising
confusion_Mat = metrics.plot_confusion_matrix(Linear_SVM, Test_A, Test_y)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.gcf().set_size_inches(15,8)
plt.title("Linear SVM Confusion Matrix (w/o tuning)")
plt.show()
```



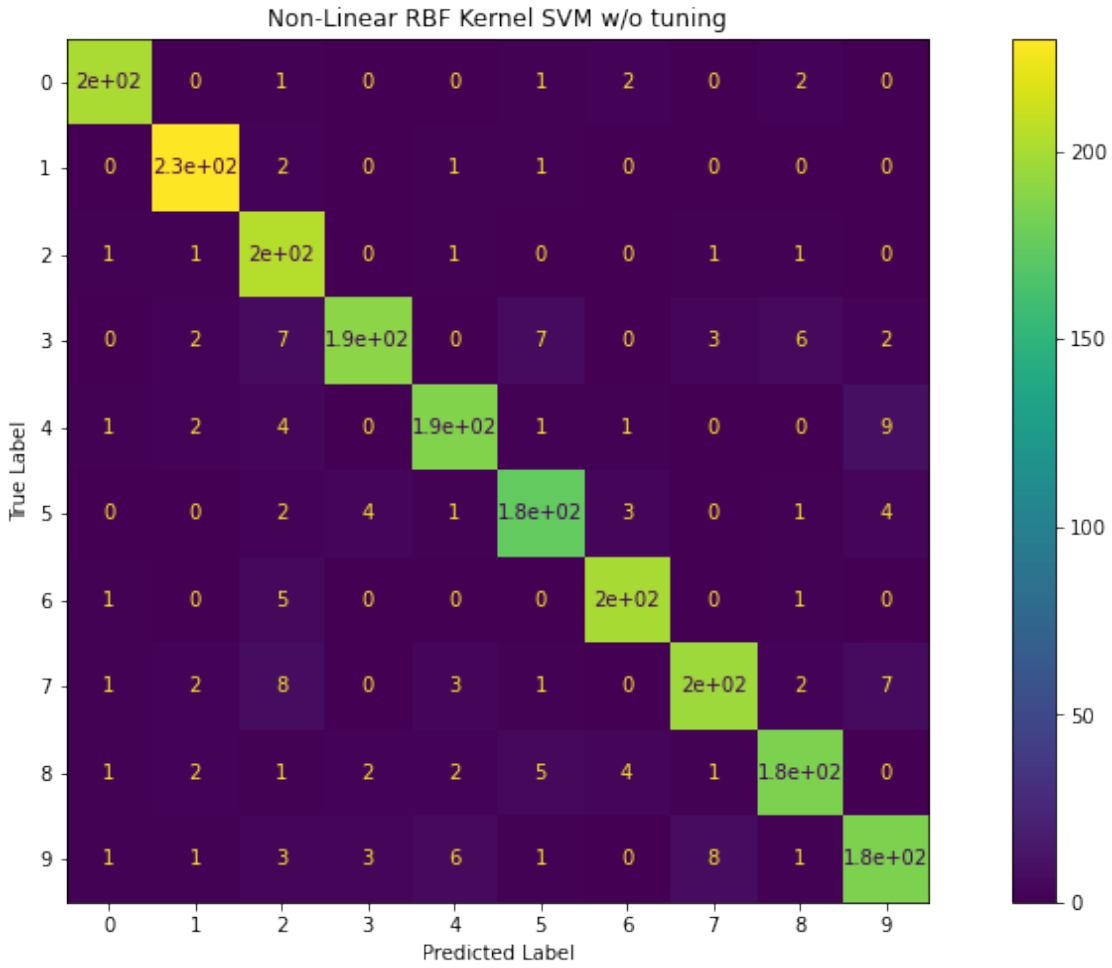

[12]:
```
#Linear SVM Prediction Accuracy
print(metrics.accuracy_score(y_true=Test_y, y_pred=accuracy_predicting))
```

[12]: 0.9

### 0.2 Non-Linear RBF Kernel SVM

[13]:
```
#NON-LINEAR RBF SVM
svm_rbf = svm.SVC(kernel='rbf')
# MODEL FITTING
svm_rbf.fit(Train_A, Train_y)
rbf_predictions = svm_rbf.predict(Test_A)

mat = metrics.plot_confusion_matrix(svm_rbf, Test_A, Test_y)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.gcf().set_size_inches(15,8)
plt.title("Non-Linear RBF Kernel SVM w/o tuning")
plt.show()
```




[14]:
```
#Non-Linear SVM Prediction Accuracy
metrics.accuracy_score(y_true=Test_y, y_pred=rbf_predictions)
```

[14]: 0.9295238095238095

### 0.2.1 Parameter Tuning for RBF Kernel SVM: grid search CV to tune the hyperparameters C and gamma.

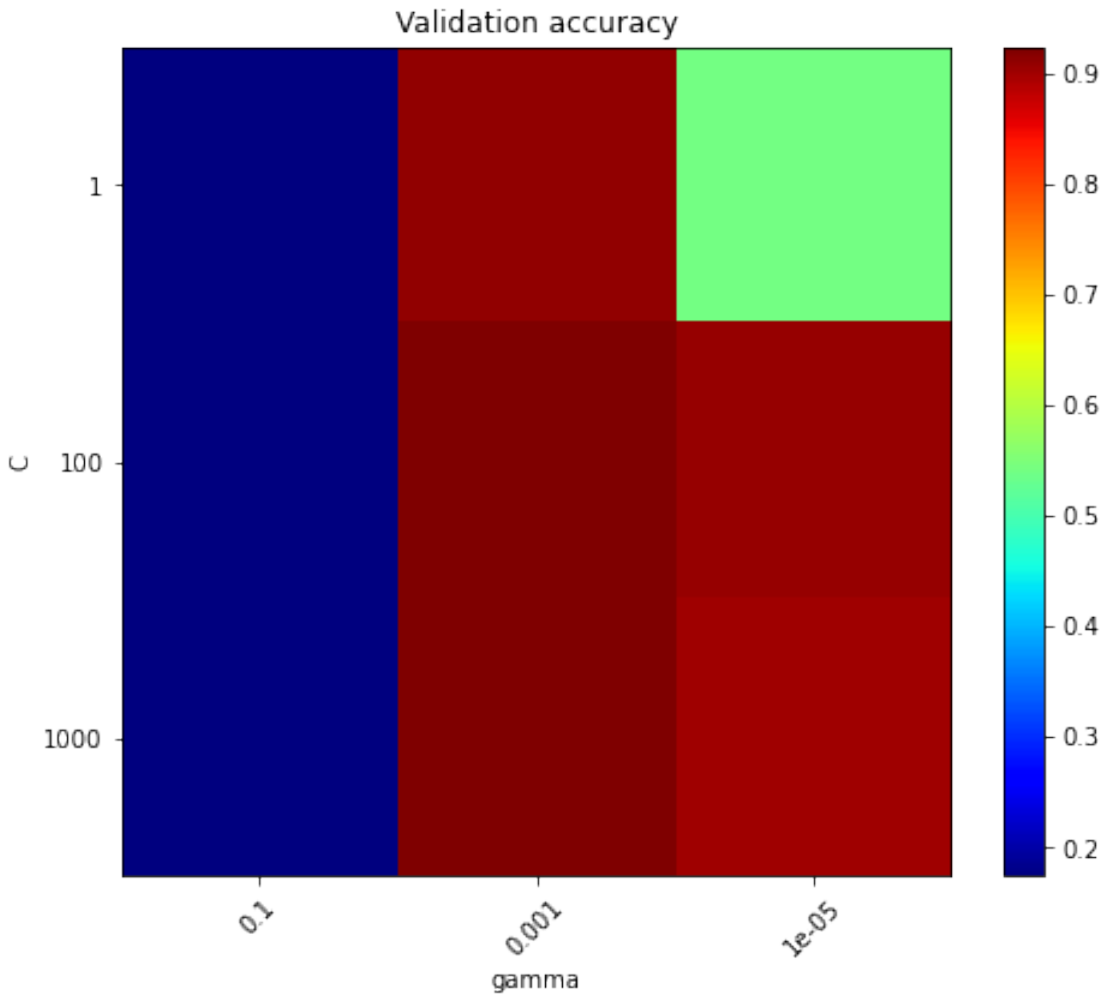
[15]:
```
SVM_Testing_Param = {'kernel':['rbf'], 'C':[1, 100, 1000], 'gamma': [1e-1,␣
↪1e-3, 1e-5]}
```

```
#Performing cross validation over different values of C and gamma to obtain␣
 ↪ideal values with
# highest rbf kernel SVM model accuracy
girdsearch_classf = GridSearchCV(estimator=svm.SVC(),cv = 2,␣
 ↪param_grid=SVM_Testing_Param, scoring='accuracy', n_jobs=1, verbose=2)
girdsearch_classf.fit(Train_A, Train_y)
#Determining the best parameters from teh gscv classification
params = girdsearch_classf.best_params_
mean_rating = girdsearch_classf.cv_results_['mean_test_score'].reshape(3, 3)
#Only using one kernel type to keep things easier and reduce computation time
rbfclassifier = girdsearch_classf.best_estimator_
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV] C=1, gamma=0.1, kernel=rbf …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] … C=1, gamma=0.1, kernel=rbf, total=  32.0s
[CV] C=1, gamma=0.1, kernel=rbf …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   32.0s remaining:    0.0s

[CV] … C=1, gamma=0.1, kernel=rbf, total=  31.7s
[CV] C=1, gamma=0.001, kernel=rbf …
[CV] … C=1, gamma=0.001, kernel=rbf, total=  13.1s
[CV] C=1, gamma=0.001, kernel=rbf …
[CV] … C=1, gamma=0.001, kernel=rbf, total=  12.8s
[CV] C=1, gamma=1e-05, kernel=rbf …
[CV] … C=1, gamma=1e-05, kernel=rbf, total=  30.2s
[CV] C=1, gamma=1e-05, kernel=rbf …
[CV] … C=1, gamma=1e-05, kernel=rbf, total=  30.2s
[CV] C=100, gamma=0.1, kernel=rbf …
[CV] … C=100, gamma=0.1, kernel=rbf, total=  31.9s
[CV] C=100, gamma=0.1, kernel=rbf …
[CV] … C=100, gamma=0.1, kernel=rbf, total=  31.8s
[CV] C=100, gamma=0.001, kernel=rbf …
[CV] … C=100, gamma=0.001, kernel=rbf, total=  11.5s
[CV] C=100, gamma=0.001, kernel=rbf …
[CV] … C=100, gamma=0.001, kernel=rbf, total=  11.2s
[CV] C=100, gamma=1e-05, kernel=rbf …
[CV] … C=100, gamma=1e-05, kernel=rbf, total=   9.7s
[CV] C=100, gamma=1e-05, kernel=rbf …
[CV] … C=100, gamma=1e-05, kernel=rbf, total=  10.2s
[CV] C=1000, gamma=0.1, kernel=rbf …
[CV] … C=1000, gamma=0.1, kernel=rbf, total=  31.8s
[CV] C=1000, gamma=0.1, kernel=rbf …
[CV] … C=1000, gamma=0.1, kernel=rbf, total=  33.0s
[CV] C=1000, gamma=0.001, kernel=rbf …
[CV] … C=1000, gamma=0.001, kernel=rbf, total=  11.6s
```

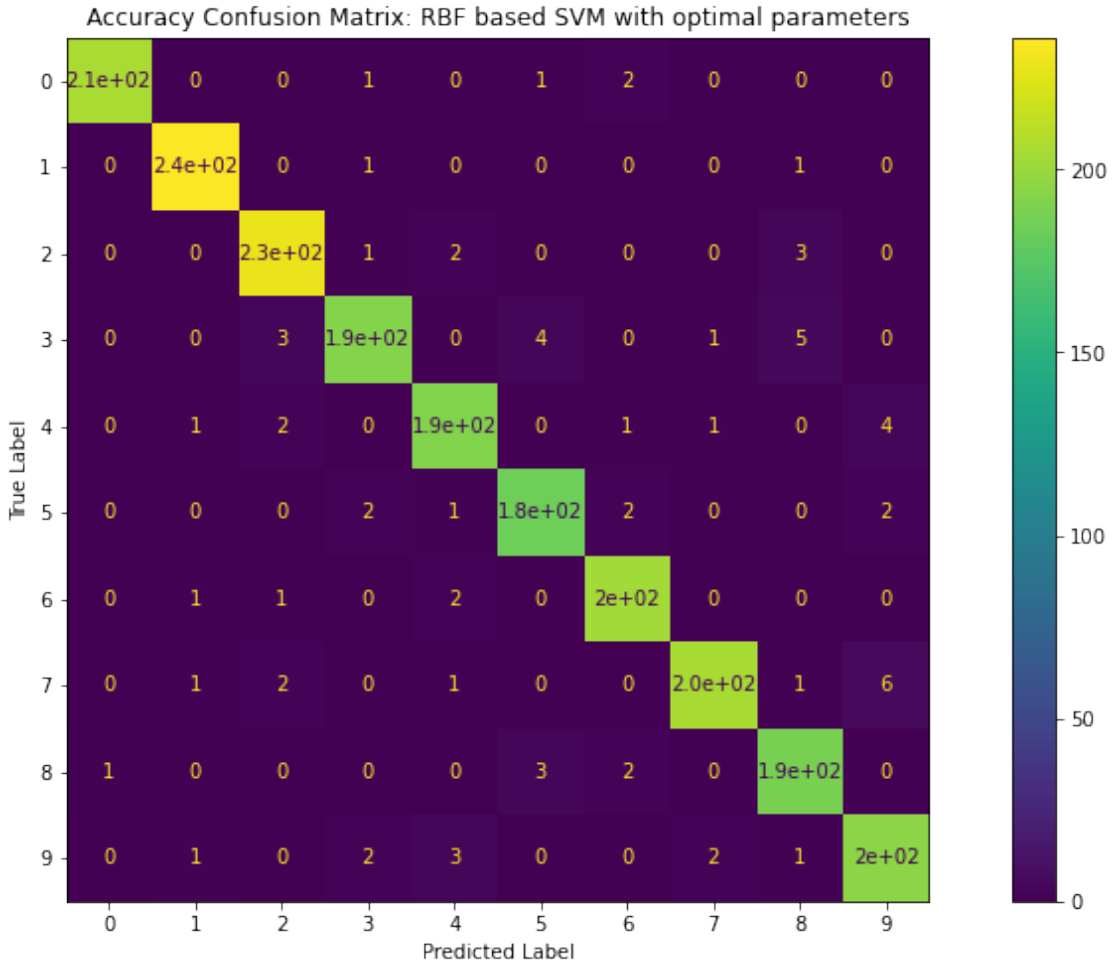
```
[CV] C=1000, gamma=0.001, kernel=rbf …
[CV] … C=1000, gamma=0.001, kernel=rbf, total=  11.2s
[CV] C=1000, gamma=1e-05, kernel=rbf …
[CV] … C=1000, gamma=1e-05, kernel=rbf, total=   7.8s
[CV] C=1000, gamma=1e-05, kernel=rbf …
[CV] … C=1000, gamma=1e-05, kernel=rbf, total=   8.5s

[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:  6.0min finished
```

[18]:
```
# Creating a heatmap for all the C and Gamma values we tested with to depict␣
 ↪testing accuracy pictorially
def heatmap_parameters(mean_rating, C_range, gamma_range):
    plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
    plt.title('Validation accuracy')
    plt.xticks(np.arange(3), 3)
    plt.yticks(np.arange(3), 3)
    plt.xlabel('gamma')
    plt.ylabel('C')
    plt.figure(figsize=(10, 8))
    plt.imshow(mean_rating, interpolation='nearest', cmap=plt.cm.jet)
    plt.show()

heatmap_parameters(grid_clsf.cv_results_['mean_test_score'].reshape(3,3), [1,␣
 ↪100, 1000], [1e-1, 1e-3, 1e-5])
#Printing the classification report for the cv conducted earlier
print("Classification report %s:\n%s\n"% (rbfclassifier, metrics.
 ↪classification_report(Test_y, rbfclassifier.predict(Test_A))))
#Printing the confusion matrix for the "optimal" parameters based SVM classifier
confusion_Mat = metrics.plot_confusion_matrix(svm_rbf, Test_A, rbfclassifier.
 ↪predict(Test_A))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.gcf().set_size_inches(15,8)
plt.title("Accuracy Confusion Matrix: RBF based SVM with optimal parameters")
plt.show()

#Accuracy of the optimal SVM RBF Based classifier
print("Accuracy={}".format(metrics.accuracy_score(Test_y, rbfclassifier.
 ↪predict(Test_A))))
```




```
Classification report for classifier SVC(C=100, break_ties=False,
cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       207
           1       0.95      0.97      0.96       234
           2       0.87      0.98      0.92       209
           3       0.95      0.90      0.92       217
           4       0.94      0.93      0.94       204
           5       0.93      0.94      0.93       190
           6       0.97      0.97      0.97       207
           7       0.93      0.91      0.92       220
           8       0.96      0.92      0.94       203
```

| | | | | |
|---|---|---|---|---|
| 9 | 0.90 | 0.88 | 0.89 | 209 |
| accuracy | | | 0.94 | 2100 |
| macro avg | 0.94 | 0.94 | 0.94 | 2100 |
| weighted avg | 0.94 | 0.94 | 0.94 | 2100 |




Accuracy=0.9352380952380952