School of Computer Engineering

Iran University of Science and Technology

Project Title:

MNIST Digit recognition using Support Vector Machine & Decision Tree

Course name:

Statistical Pattern Recognition

Instructor:

Dr. M. Analoui

Poorya MohammadiNasab

Fall 2021

Table of Contents

1. Introduction

Digit Character Recognition is an important concept in Artificial Intelligence and Computer Vision. The data of handwritten digits is fed to the computer and the required output is the digit that was given as the data. The problem with digit recognition arises because a digit can have various styles. Handwritten characters can have different flavors which is why it is difficult for a computer to understand unless the right approach is used. It is an important benchmark in the field of computer vision. It reduced the need of retyping and allowed quicker text-editing, digital searches, and saved physical storage. This technology is solving a plethora of problems like automated recognition of:

- **Zip Codes:** zip code consists of some digits and is one of the most important parts of a letter for it to be delivered to the correct location. Many years ago, the postman would read the zip code manually for delivery. However, this type of work is now automated by using optical character recognition (OCR).
- **Bank Cheques:** The most frequent use of OCR is to handle Cheques. A handwritten cheque is scanned, its contents converted into digital text, the signature verified and the cheque cleared in real time, all without human involvement.
- **Legal Records:** Judgments, filings, statements, wills and other legal documents, especially the printed ones, can be digitized, stored, and made searchable using the simplest of OCR readers.
- **Healthcare Records:** Having one's entire medical history on a searchable, digital store means that things like past illnesses and treatments, diagnostic tests, hospital records, insurance payments etc. can be made available in one unified place, rather than having to maintain unwieldy files of reports and X-rays. OCR can be used to scan these documents and create a digital database.

The aim of this project is to implement a classification algorithm to recognize handwritten digits (0- 9). This report presents our implementation of the methods that we have used in this project. We used support vector machine (SVM) with two different kernel functions and decision tree as out classifiers to solve this problem.

2. Dataset

We will be using MNIST Handwritten Digits dataset to train, test and evaluate the three classifiers. The dataset contains around 70000 samples of handwritten digits. Each sample consists of a grayscale image of size 28x28 (784 total pixels). The MNIST dataset has been partitioned into two parts. 60,000 images are used for training and the remaining 10,000 images are used for testing purposes. Different images of each digit from the MNIST dataset is shown in Fig. 1.

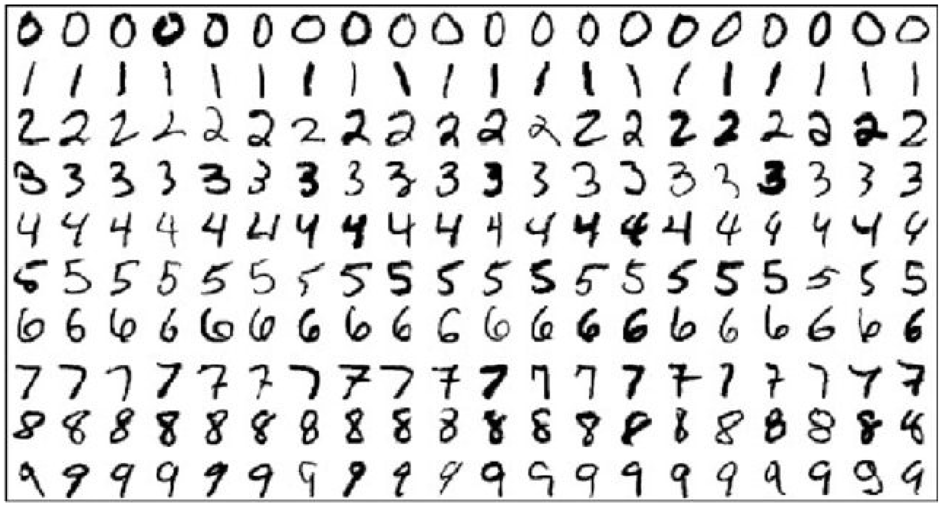


Figure 1: Samples images from the MNIST dataset.

3. Implementation

We implemented this project in three main steps. The first step is **preprocessing**, next we **trained** the data and the final step is **testing and evaluating** trained models and report the results. In this section we explain the source code of each part.

3. 1. Preprocessing

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. General info on this format is given. All the integers in the files are stored in the MSB first (high endian) format used by most non-Intel processors. Users of Intel processors and other low-endian machines must flip the bytes of the header. There are 4 files:

- train-images-idx3-ubyte: training set images
- train-labels-idx1-ubyte: training set labels
- t10k-images-idx3-ubyte: test set images
- t10k-labels-idx1-ubyte: test set labels

The IDX file format is a simple format for vectors and multidimensional matrices of various numerical types. The basic format is:

- magic number
- size in dimension 0
- size in dimension 1
- size in dimension 2
- ...
- size in dimension N
- data

The magic number is an integer (MSB first). The first 2 bytes are always 0. The third byte codes the type of the data:

- **0x08**: unsigned byte
- **0x09**: signed byte
- **0x0B**: short (2 bytes)
- **0x0C**: int (4 bytes)
- **0x0D**: float (4 bytes)
- **0x0E**: double (8 bytes)

The 4-th byte codes the number of dimensions of the vector/matrix: 1 for vectors, 2 for matrices....

```
[1]: import numpy as np
     import pandas as pd
```

```
[2]: # read and convert ubyte data to csv files
     def Read_Convert(imagefile,labelfile,outputfile,sample_size):
         """
             image file : ubyte image data
             label file : ubyte label data
             outputfile : name of output csv
             sample_size: number of samples in ubyte data
         """
         # open input image & label as binary format
         image_in = open(imagefile,mode='rb')
         label_in = open(labelfile,mode='rb')

         # open output file
         file_out = open(outputfile,mode='w')

         # read file headers
         image_in.read(16)
         label_in.read(8)

         images = []

         for i in range(sample_size):
             # add one label
             image = [ord(label_in.read(1))]

             for j in range(28*28):
                 # assign an image to each label
                 image.append(ord(image_in.read(1)))

             # append the image and its label to images
             images.append(image)

         # write images data to file
         for image in images:
             file_out.write(','.join(str(pix) for pix in image) + '\n')

         # close files
         image_in.close()
         label_in.close()
         file_out.close()
```

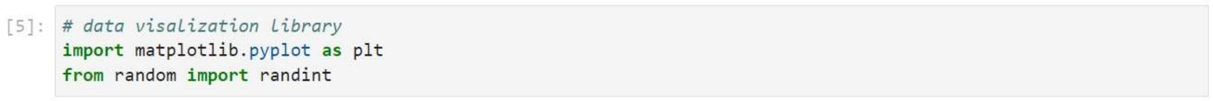
```
[4]: # Run Data Conversion
     print('Start!')

     Read_Convert(imagefile='Dataset/train-images.idx3-ubyte',labelfile='Dataset/train-labels.idx1-ubyte'
                 ,outputfile='Dataset/minist_train.csv',sample_size=60000)

     Read_Convert(imagefile='Dataset/t10k-images.idx3-ubyte' ,labelfile='Dataset/t10k-labels.idx1-ubyte'
                 ,outputfile='Dataset/minist_test.csv' ,sample_size=10000)

     print('Finish!')

     Start!
     Finish!
```

After running this code, we had two .csv files in dataset directory. The next step is to add headers to each dataset. If we didn't do this task, Pandas considers the first row of data as headers. In addition we have to set a suitable name for label column.

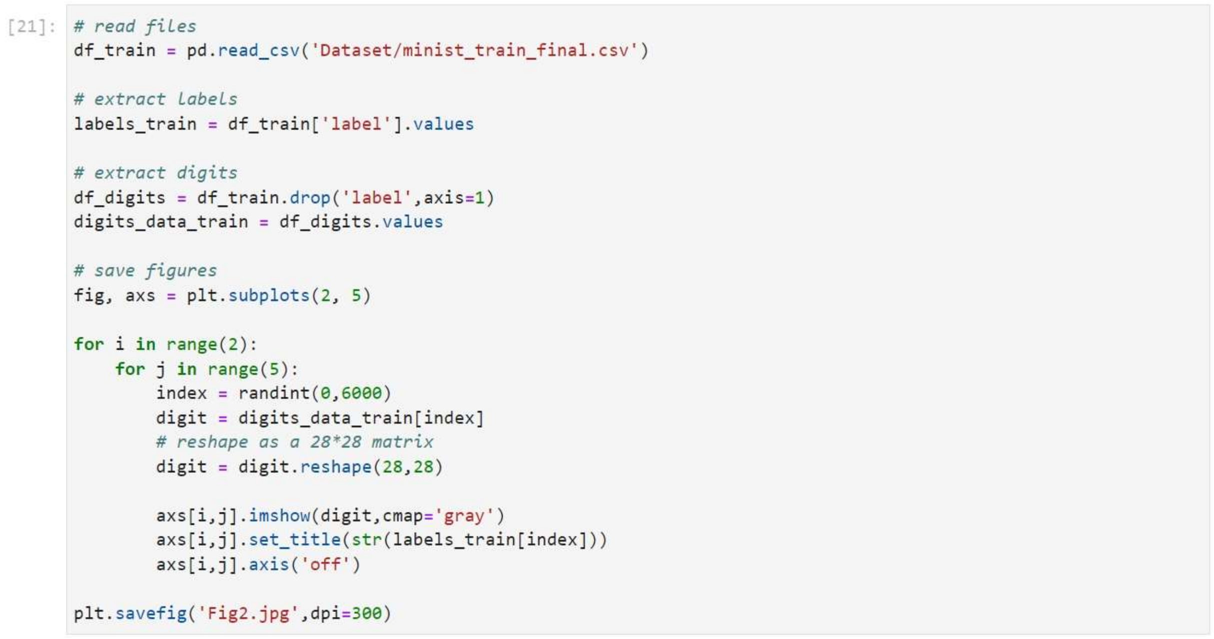
```
[5]: def Add_label(input_file, Output_file):

         col_names = ['label']

         # coln_names = ['label', '1', '2', ..., '783','784']
         for i in range(1,785):
             col_names.append(str(i))

         # read csv file to check and set the correct name for label column
         df = pd.read_csv(input_file, header=None)

         df.columns = col_names

         # write new df to csv
         df.to_csv(Output_filee, index=False)
```

```
[6]: # run add_label function for each .csv file

     Add_label(input_file='Dataset/minist_train.csv', Output_file='Dataset/minist_train_final.csv')
     Add_label(input_file='Dataset/minist_test.csv', Output_file='Dataset/minist_test_final.csv')
```

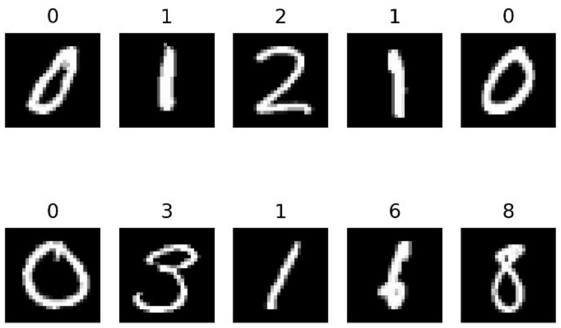
After preparing the datasets, now we want to look into it and see what the data are actually look like. So we wrote some lines of code.



```
[5]: # data visalization library
     import matplotlib.pyplot as plt
     from random import randint
```



```
[21]: # read files
      df_train = pd.read_csv('Dataset/minist_train_final.csv')

      # extract labels
      labels_train = df_train['label'].values

      # extract digits
      df_digits = df_train.drop('label',axis=1)
      digits_data_train = df_digits.values

      # save figures
      fig, axs = plt.subplots(2, 5)

      for i in range(2):
          for j in range(5):
              index = randint(0,6000)
              digit = digits_data_train[index]
              # reshape as a 28*28 matrix
              digit = digit.reshape(28,28)

              axs[i,j].imshow(digit,cmap='gray')
              axs[i,j].set_title(str(labels_train[index]))
              axs[i,j].axis('off')

      plt.savefig('Fig2.jpg',dpi=300)
```



3. 2. Dimension Reduction

Principal Component Analysis (PCA) is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data. PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

Steps for PCA algorithm:

- **Getting the dataset:** Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
- **Representing data into a structure:** Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
- **Standardizing the data:** In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance. If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
- **Calculating the Covariance of Z:** To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
- **Calculating the Eigen Values and Eigen Vectors:** Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
- **Sorting the Eigen Vectors:** In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.
- **Calculating the new features Or Principal Components:** Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.
- **Remove less or unimportant features from the new dataset:** The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

[3]:
```python
from sklearn import decomposition
```

[4]:
```python
def reduce_dimension(data_in,data_out,num_of_features):
    # read data
    df = pd.read_csv(data_in)

    # extract labels
    labels = df['label']

    # extract digits
    df.drop('label',axis=1,inplace=True)

    # apply PCA
    pca = decomposition.PCA()
    pca.n_components = num_of_features
    pca_data = pca.fit_transform(df)
    pca_data = pd.DataFrame(pca_data)

    # add labels to reduced training dataset
    out_df = pd.concat([labels,pca_data],axis=1)

    # write new dataset to file
    out_df.to_csv(data_out,index=False)
```

[5]:
```python
# run dimension reduction
print('start!')
reduce_dimension(data_in='Dataset/minist_test_final.csv',data_out='Dataset/minist_test_100.csv',100)

reduce_dimension(data_in='Dataset/minist_train_final.csv',data_out='Dataset/minist_train_100.csv',100)

reduce_dimension(data_in='Dataset/minist_train_final.csv',data_out='Dataset/minist_train_40.csv',40)

reduce_dimension(data_in='Dataset/minist_test_final.csv',data_out='Dataset/minist_test_40.csv',40)
print('Finish!')
```

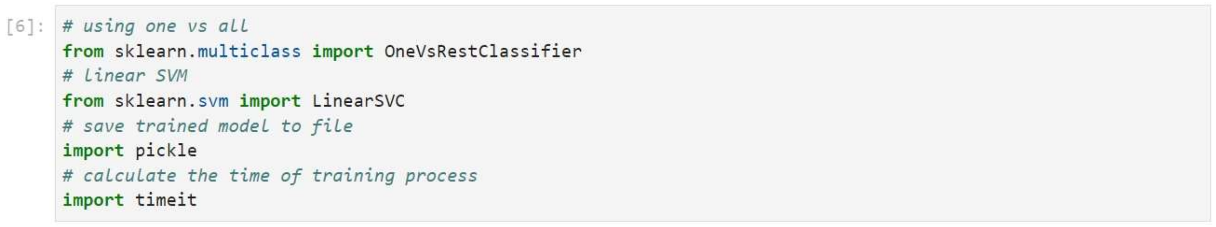
```
start!
Finish!
```

3. 3. Training SVM with Linear kernel

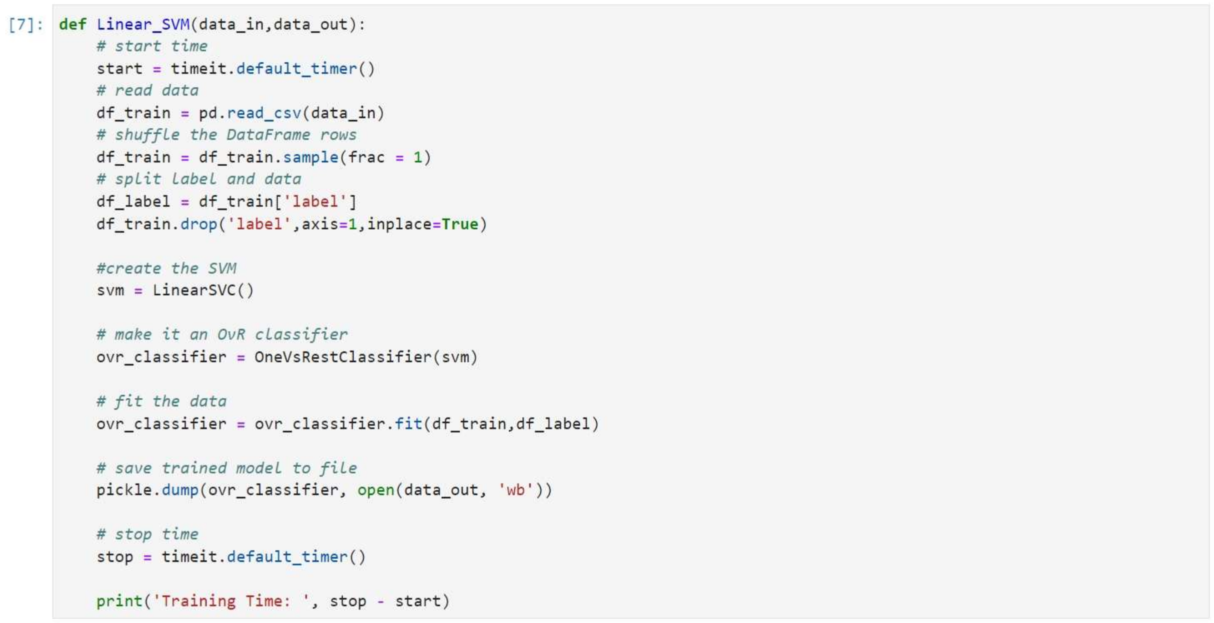
Linear Kernel is used when the data is linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a large number of Features in a particular Data Set. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature. So we mostly use Linear Kernel in Text Classification.

Advantages of using Linear Kernel:

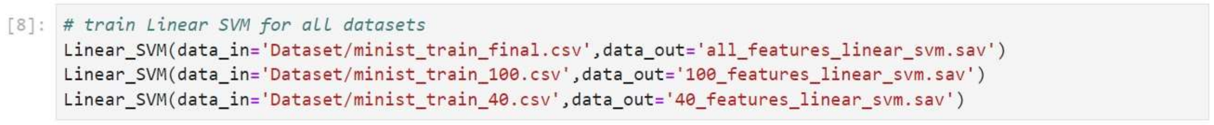
- Training a SVM with a Linear Kernel is faster than with any other Kernel.
- When training a SVM with a Linear Kernel, only the optimization of the C Regularization parameter is required. On the other hand, when training with other kernels, there is a need to optimize the γ parameter which means that performing a grid search will usually take more time.



```
[6]: # using one vs all
     from sklearn.multiclass import OneVsRestClassifier
     # linear SVM
     from sklearn.svm import LinearSVC
     # save trained model to file
     import pickle
     # calculate the time of training process
     import timeit
```



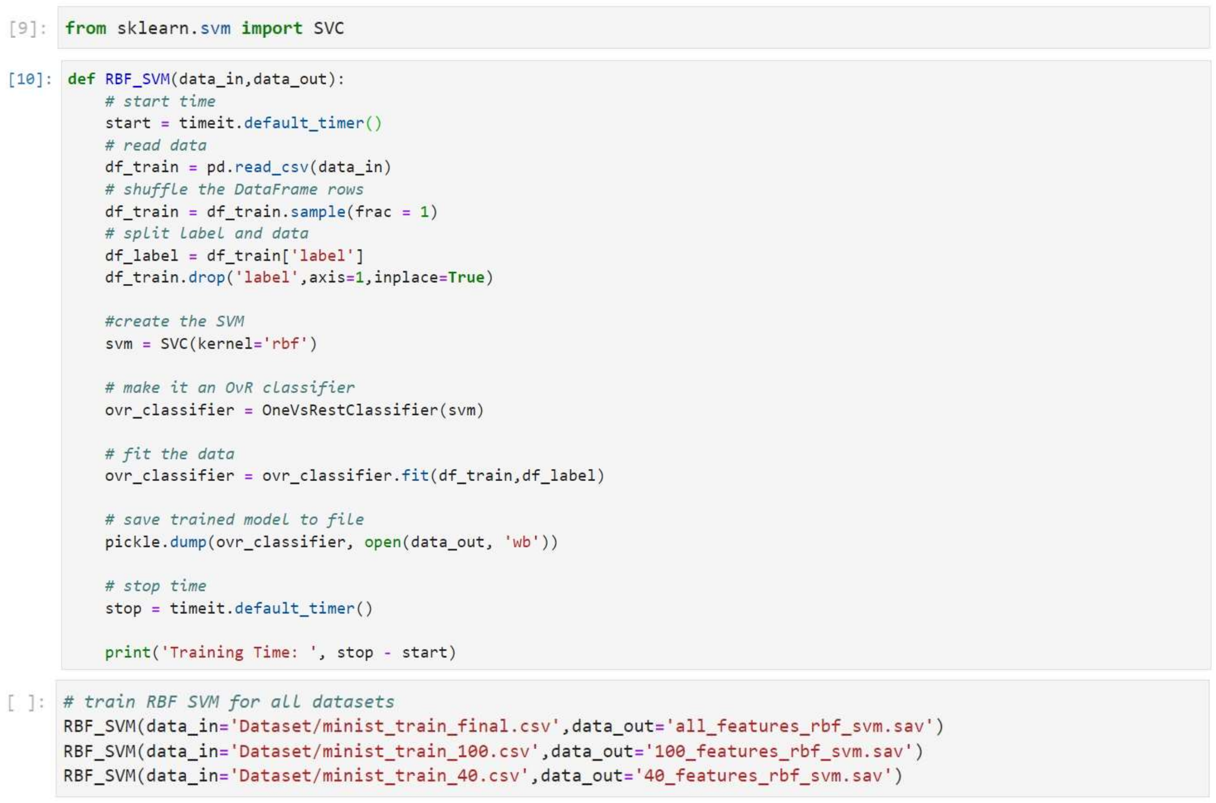
```
[7]: def Linear_SVM(data_in,data_out):
         # start time
         start = timeit.default_timer()
         # read data
         df_train = pd.read_csv(data_in)
         # shuffle the DataFrame rows
         df_train = df_train.sample(frac = 1)
         # split label and data
         df_label = df_train['label']
         df_train.drop('label',axis=1,inplace=True)

         #create the SVM
         svm = LinearSVC()

         # make it an OvR classifier
         ovr_classifier = OneVsRestClassifier(svm)

         # fit the data
         ovr_classifier = ovr_classifier.fit(df_train,df_label)

         # save trained model to file
         pickle.dump(ovr_classifier, open(data_out, 'wb'))

         # stop time
         stop = timeit.default_timer()

         print('Training Time: ', stop - start)
```



```
[8]: # train Linear SVM for all datasets
     Linear_SVM(data_in='Dataset/minist_train_final.csv',data_out='all_features_linear_svm.sav')
     Linear_SVM(data_in='Dataset/minist_train_100.csv',data_out='100_features_linear_svm.sav')
     Linear_SVM(data_in='Dataset/minist_train_40.csv',data_out='40_features_linear_svm.sav')
```

3. 4. Training SVM with RBF kernel

Radial Basis Kernel (RBF) is a kernel function that is used in machine learning to find a non-linear classifier or regression line. The main motive of the kernel is to do calculations in any d-dimensional space where $d > 1$, so that we can get a quadratic, cubic or any polynomial equation of large degree for our classification/regression line. Since Radial basis kernel uses exponent and as we know the expansion of e^x gives a polynomial equation of infinite power, so using this kernel, we make our regression/classification line infinitely powerful too.



```
[9]:  from sklearn.svm import SVC
```

```
[10]: def RBF_SVM(data_in,data_out):
          # start time
          start = timeit.default_timer()
          # read data
          df_train = pd.read_csv(data_in)
          # shuffle the DataFrame rows
          df_train = df_train.sample(frac = 1)
          # split label and data
          df_label = df_train['label']
          df_train.drop('label',axis=1,inplace=True)

          #create the SVM
          svm = SVC(kernel='rbf')

          # make it an OvR classifier
          ovr_classifier = OneVsRestClassifier(svm)

          # fit the data
          ovr_classifier = ovr_classifier.fit(df_train,df_label)

          # save trained model to file
          pickle.dump(ovr_classifier, open(data_out, 'wb'))

          # stop time
          stop = timeit.default_timer()

          print('Training Time: ', stop - start)
```

```
[ ]:  # train RBF SVM for all datasets
      RBF_SVM(data_in='Dataset/minist_train_final.csv',data_out='all_features_rbf_svm.sav')
      RBF_SVM(data_in='Dataset/minist_train_100.csv',data_out='100_features_rbf_svm.sav')
      RBF_SVM(data_in='Dataset/minist_train_40.csv',data_out='40_features_rbf_svm.sav')
```

3. 5. Training Decision Tree

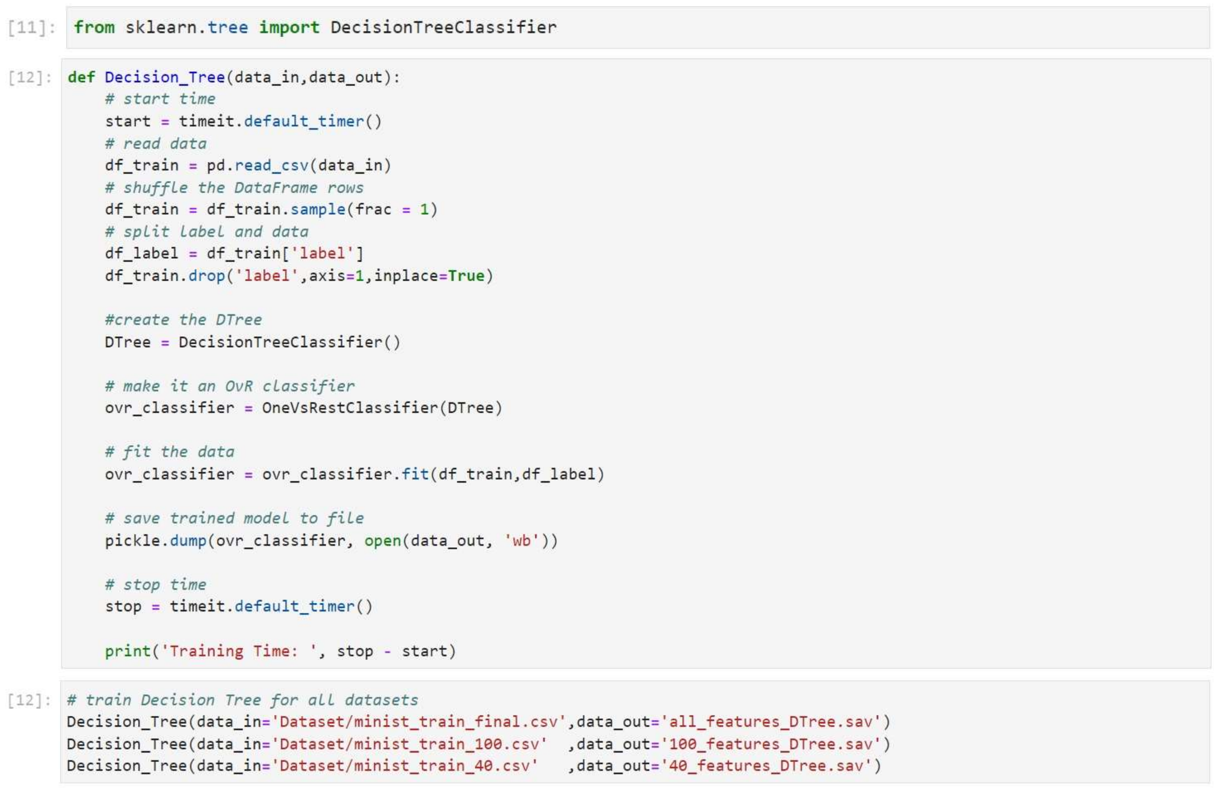
Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, and then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the sub-tree rooted at the new node.

The **strengths** of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

The **weaknesses** of decision tree methods:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.



[11]:
```
from sklearn.tree import DecisionTreeClassifier
```

[12]:
```
def Decision_Tree(data_in,data_out):
    # start time
    start = timeit.default_timer()
    # read data
    df_train = pd.read_csv(data_in)
    # shuffle the DataFrame rows
    df_train = df_train.sample(frac = 1)
    # split label and data
    df_label = df_train['label']
    df_train.drop('label',axis=1,inplace=True)

    #create the DTree
    DTree = DecisionTreeClassifier()

    # make it an OvR classifier
    ovr_classifier = OneVsRestClassifier(DTree)

    # fit the data
    ovr_classifier = ovr_classifier.fit(df_train,df_label)

    # save trained model to file
    pickle.dump(ovr_classifier, open(data_out, 'wb'))

    # stop time
    stop = timeit.default_timer()

    print('Training Time: ', stop - start)
```

[12]:
```
# train Decision Tree for all datasets
Decision_Tree(data_in='Dataset/minist_train_final.csv',data_out='all_features_DTree.sav')
Decision_Tree(data_in='Dataset/minist_train_100.csv'  ,data_out='100_features_DTree.sav')
Decision_Tree(data_in='Dataset/minist_train_40.csv'   ,data_out='40_features_DTree.sav')
```

3. 6. Evaluation metrics

To evaluate trained models, we implemented a function to measure the performance of each model with different metrics. In this project we used **Confusion Matrix**, **Precision**, **Recall**, **F1 score, Receiver Operating Characteristic (ROC)**, and **Area Under Curve (AUC)**.

- **Confusion matrix:** A much better way to evaluate the performance of a classifier is to look at the confusion matrix. The general idea is to count the number of times instances of class 'A' are classified as class 'B'. For example, to know the number of times the classifier confused images of 5s with 3s, you would look in the 5th row and 3rd column of the confusion matrix. Each row in a confusion matrix represents an actual class, while each column represents a predicted class.
- **Precision:** Precision is a metric that quantifies the number of correct positive predictions made. Precision, therefore, calculates the accuracy for the minority class. It is calculated as the ratio of correctly predicted positive examples divided by the total number of positive examples that were predicted. If we want to calculate precision for multi-class classification, precision is calculated as the sum of true positives across all classes divided by the sum of true positives and false positives across all classes.

$$Precision = \frac{True\ Positive}{True\ Positive + \ False\ Positive}$$

- **Recall:** It is also known as **Sensitivity**. The recall is the measure of our model correctly identifying True Positives. Thus, for all the patients who actually have heart disease, recall tells us how many we correctly identified as having a heart disease. Mathematically:

$$Recall = \frac{True\ Positive}{Positive}$$

- **F1 Score:** Understanding Accuracy made us realize, we need a tradeoff between Precision and Recall. We first need to decide which is more important for our classification problem. For example, in a specific application we can consider that achieving a high recall is more important than getting a high precision. For some other models, it is desirable to have a high precision. In such cases, we use something called F1-score. F1-score is the Harmonic mean of the Precision and Recall:

$$F1 = \frac{2\ \times Precision\ \times Recall}{Precision + Recall}$$

- **ROC Curve:** we can calculate more values from the confusion matrix.
  - **False Positive Rate (FPR):** It is the ratio of the False Positives to the Actual number of Negatives.
  - **True Negative Rate (TNR) or the Specificity:** It is the ratio of the True Negatives and the Actual Number of Negatives.

  We can also visualize Precision and Recall using ROC curves. ROC Curve is the plot between the TPR (y-axis) and FPR (x-axis).

- **AUC Interpretation:** The area with the curve and the axes as the boundaries is called the Area Under Curve (AUC). It is this area which is considered as a metric of a good model. With this metric ranging from 0 to 1, we should aim for a high value of AUC. Models with a high AUC are called as models with good skill.

[11]:
```
from sklearn.metrics import confusion_matrix,precision_score,recall_score
from sklearn.metrics import f1_score,roc_curve,auc,roc_auc_score
import seaborn as sns
```

[12]:
```
def evaluate(model_name,test_file,output_name):

    # open trained model file
    trained_file = open(model_name,'rb')
    # Load trained file
    trained_model = pickle.load(trained_file)

    #Load test dataset
    Test = pd.read_csv(test_file)
    # split X and Y
    Y_test = Test['label']
    X_test = Test.drop('label',axis=1)

    # Confusion Matrix
    predicted = trained_model.predict(X_test)
    ConfMatrix = confusion_matrix(Y_test,predicted)
    # save matrix to file
    Tmp_name = output_name + '_confusion_matrix.csv'
    pd.DataFrame(ConfMatrix).to_csv(Tmp_name)
    # plot data
    plt.figure(figsize = (10,7))
    Tmp_name = output_name + '_confusion.jpg'
    sns.heatmap(ConfMatrix,annot=True,cbar=False,fmt='g',linewidths=0.5,cmap='PuBu').figure.savefig(Tmp_name,dpi=400)

    # Precision
    Precision = precision_score(Y_test,predicted,average="macro")
    print('Precision: ',Precision)
    # Recall
    Recall = recall_score(Y_test,predicted,average="macro")
    print('Recall: ',Recall)
    # f1
    F1 = f1_score(Y_test,predicted,average="macro")
    print('F1: ',F1)
    # write measures in a text file
    Tmp_name = output_name + '_measure.txt'
    outfile = open(Tmp_name,'w')
    outfile.write('Precision: ')
    outfile.write(str(Precision))
    outfile.write('\n')
    outfile.write('Recall: ')
    outfile.write(str(Recall))
    outfile.write('\n')
    outfile.write('F1: ')
    outfile.write(str(F1))
    outfile.close()

    #Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc_s = dict()
    n_classes = 10

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(Y_test, predicted,i)
        roc_auc_s[i] = auc(fpr[i], tpr[i])

    # plot ROC and AUC
    fig, axs = plt.subplots(2, 5)
    fig.set_figheight(9)
    fig.set_figwidth(16)

    for i in range(2):
        for j in range(5):
            axs[i,j].plot(fpr[i*5+j],tpr[i*5+j],label='C {0} (area = %0.2f)'.format(i*5+j) %roc_auc_s[i*5+j])
            axs[i,j].set_title(str('C{0} (area = %0.2f)'.format(i*5+j) %roc_auc_s[i*5+j]))
            axs[i,j].plot([0, 1], [0, 1], 'k--')

            if(i != 1):
                axs[i,j].get_xaxis().set_ticks([])
            if(j != 0):
                axs[i,j].get_yaxis().set_ticks([])

    Tmp_name = output_name + '_ROC.jpg'
    fig.savefig(Tmp_name,dpi=450)
```

```
[13]: # run evaluation for all 9 model

      # Linear SVM
      evaluate('Models/all_features_linear_svm.sav','Dataset/minist_test_final.csv','data/minist_linear_svm_all')
      evaluate('Models/100_features_linear_svm.sav','Dataset/minist_test_100.csv','data/minist_linear_svm_100')
      evaluate('Models/40_features_linear_svm.sav','Dataset/minist_test_40.csv','data/minist_linear_svm_40')

      # RBF SVM
      evaluate('Models/all_features_rbf_svm.sav','Dataset/minist_test_final.csv','data/minist_rbf_svm_all')
      evaluate('Models/100_features_rbf_svm.sav','Dataset/minist_test_100.csv','data/minist_rbf_svm_100')
      evaluate('Models/40_features_rbf_svm.sav','Dataset/minist_test_40.csv','data/minist_rbf_svm_40')

      # Decision Tree
      evaluate('Models/all_features_DTree.sav','Dataset/minist_test_final.csv','data/minist_Dtree_all')
      evaluate('Models/100_features_DTree.sav','Dataset/minist_test_100.csv','data/minist_Dtree_100')
      evaluate('Models/40_features_DTree.sav','Dataset/minist_test_40.csv','data/minist_Dtree_40')
```

4. Discussion

Now after implementation of the project we have to explain and discuss about numbers and figures. We trained 9 different models and now it is the time to compare them in different aspects and metrics.

4. 1. Comparison of Precision, Recall, and F1 measure

Table 1 is an overall look to all trained models in three different metrics. The SVM model with RBF kernel improves all three metrics significantly more than all the other models. It means that the SVM with RBF kernel has a suitable complexity for this problem. Figures 2 to 4 show these three metrics for each trained model.

Table 1. Overall comparison with 3 metrics

	number of features	Precision	Recall	F1
Linear SVM	40	0.8235	0.8285	0.8285
	100	0.9028	0.9028	0.9028
	all features	0.8741	0.8680	0.8678
RBF SVM	40	0.8825	0.9027	0.8995
	100	0.9822	0.9723	0.9821
	all features	0.9793	0.9792	0.9792
Decision Tree	40	0.8206	0.8035	0.8068
	100	0.7782	0.7752	0.7774
	all features	0.8903	0.8525	0.8620

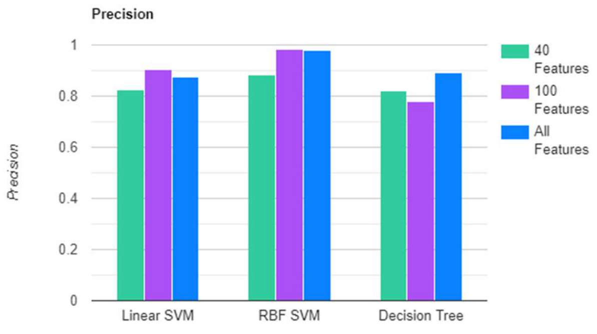


Figure 2: Precision plot

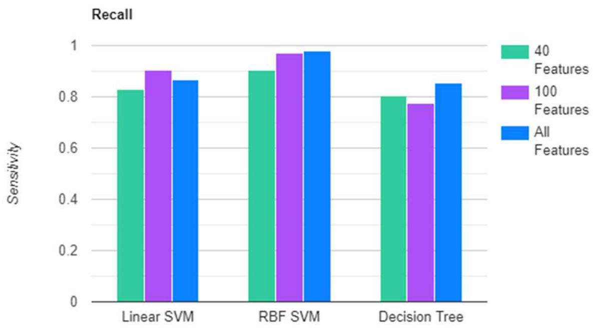


Figure 3: Recall plot

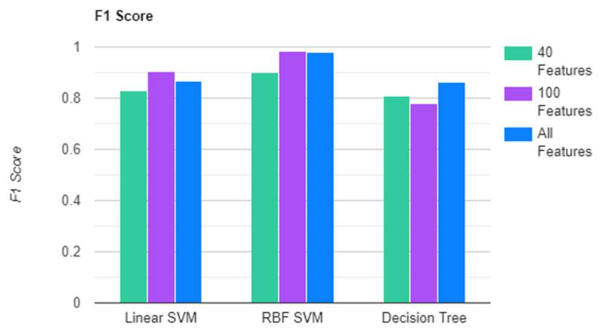


Figure 4: F1 plot

4. 2. Comparison of Training time

In training step, we capture the time just before and after the training process. Figure 5 illustrates the time for each model in this project. It is clear that SVM with RBF kernel has the longest training time especially in case of the dataset with all features.

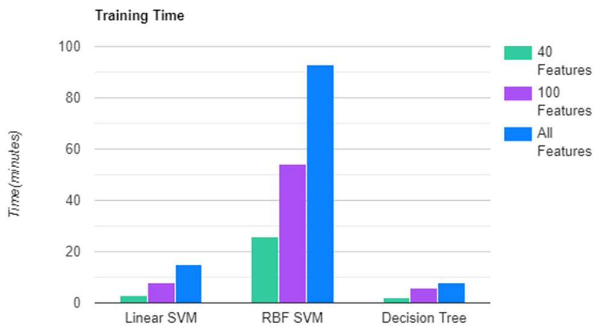
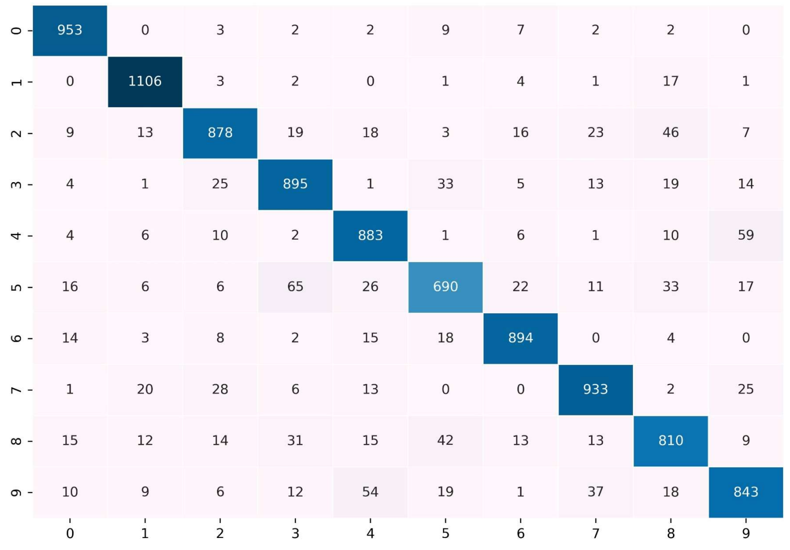


Figure 5: Training Time plot

4. 3. Confusion Matrix

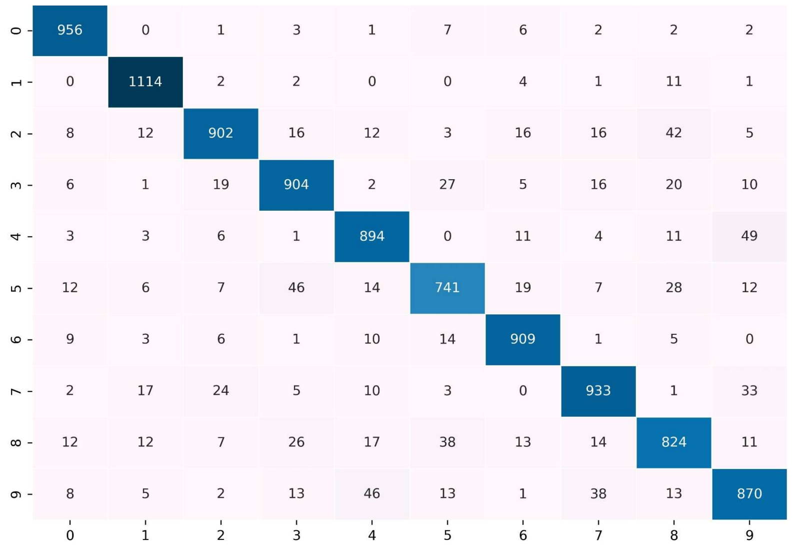
As we mention in section 3.6, confusion matrix contains a lot of information about the whole training process. We plot Confusion Matrix of each model. So we can count true positives and false positives for all classes. Generally, in a Confusion Matrix the numbers in diameter are precision and all other indexes are misclassification.

- **Confusion matrix of Linear SVM with 40 features**



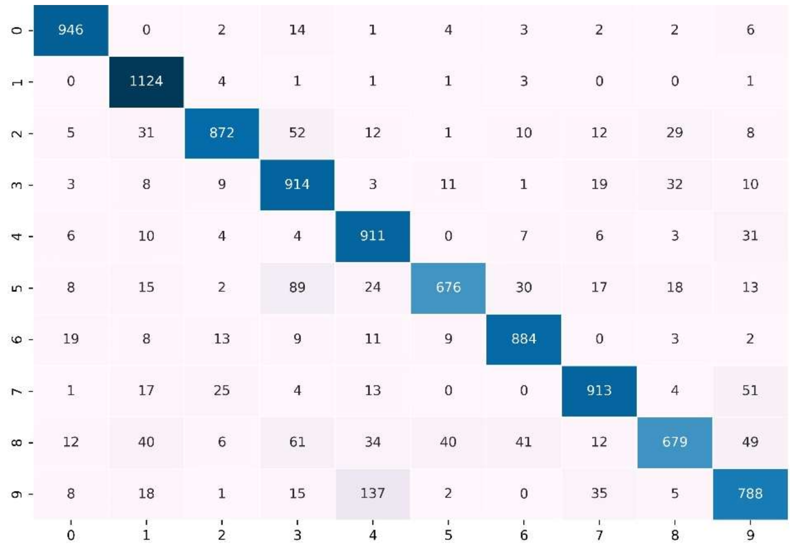
	0	1	2	3	4	5	6	7	8	9
0	953	0	3	2	2	9	7	2	2	0
1	0	1106	3	2	0	1	4	1	17	1
2	9	13	878	19	18	3	16	23	46	7
3	4	1	25	895	1	33	5	13	19	14
4	4	6	10	2	883	1	6	1	10	59
5	16	6	6	65	26	690	22	11	33	17
6	14	3	8	2	15	18	894	0	4	0
7	1	20	28	6	13	0	0	933	2	25
8	15	12	14	31	15	42	13	13	810	9
9	10	9	6	12	54	19	1	37	18	843

- **Confusion matrix of Linear SVM with 100 features**



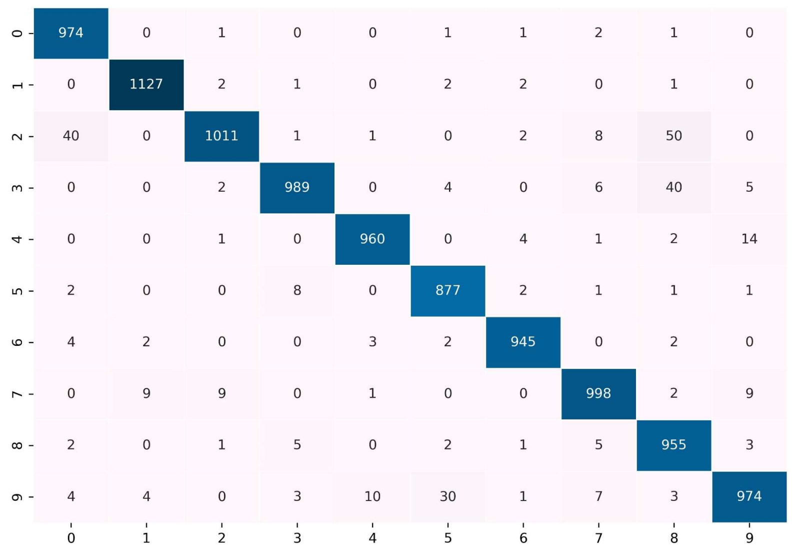
	0	1	2	3	4	5	6	7	8	9
0	956	0	1	3	1	7	6	2	2	2
1	0	1114	2	2	0	0	4	1	11	1
2	8	12	902	16	12	3	16	16	42	5
3	6	1	19	904	2	27	5	16	20	10
4	3	3	6	1	894	0	11	4	11	49
5	12	6	7	46	14	741	19	7	28	12
6	9	3	6	1	10	14	909	1	5	0
7	2	17	24	5	10	3	0	933	1	33
8	12	12	7	26	17	38	13	14	824	11
9	8	5	2	13	46	13	1	38	13	870

- **Confusion matrix of Linear SVM with all features**



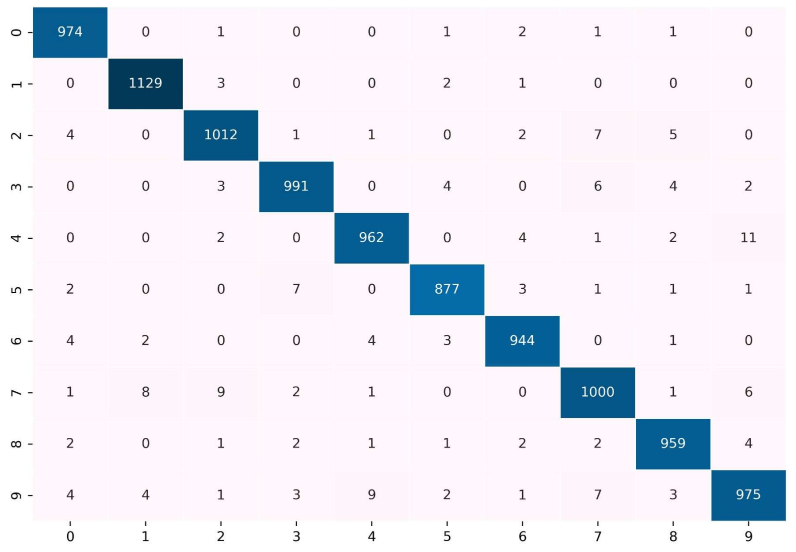
	0	1	2	3	4	5	6	7	8	9
0	946	0	2	14	1	4	3	2	2	6
1	0	1124	4	1	1	1	3	0	0	1
2	5	31	872	52	12	1	10	12	29	8
3	3	8	9	914	3	11	1	19	32	10
4	6	10	4	4	911	0	7	6	3	31
5	8	15	2	89	24	676	30	17	18	13
6	19	8	13	9	11	9	884	0	3	2
7	1	17	25	4	13	0	0	913	4	51
8	12	40	6	61	34	40	41	12	679	49
9	8	18	1	15	137	2	0	35	5	788

- **Confusion matrix of RBF SVM with 40 features**



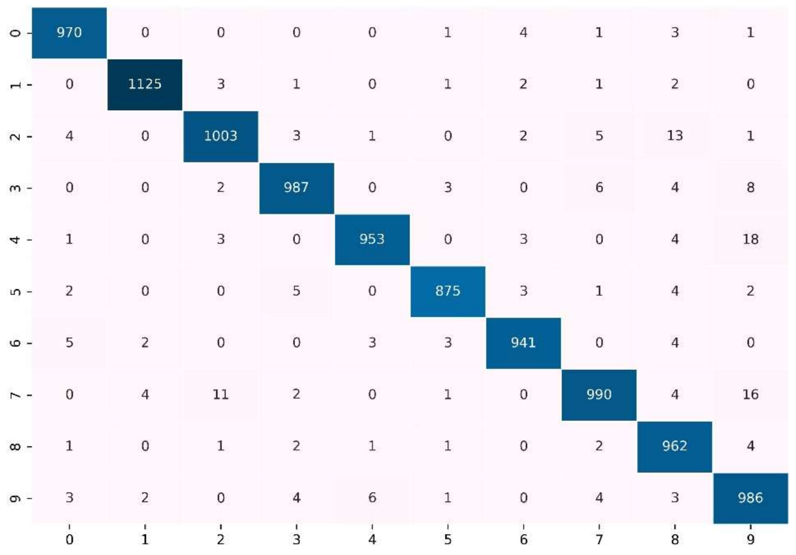
	0	1	2	3	4	5	6	7	8	9
0	974	0	1	0	0	1	1	2	1	0
1	0	1127	2	1	0	2	2	0	1	0
2	40	0	1011	1	1	0	2	8	50	0
3	0	0	2	989	0	4	0	6	40	5
4	0	0	1	0	960	0	4	1	2	14
5	2	0	0	8	0	877	2	1	1	1
6	4	2	0	0	3	2	945	0	2	0
7	0	9	9	0	1	0	0	998	2	9
8	2	0	1	5	0	2	1	5	955	3
9	4	4	0	3	10	30	1	7	3	974

- **Confusion matrix of RBF SVM with 100 features**



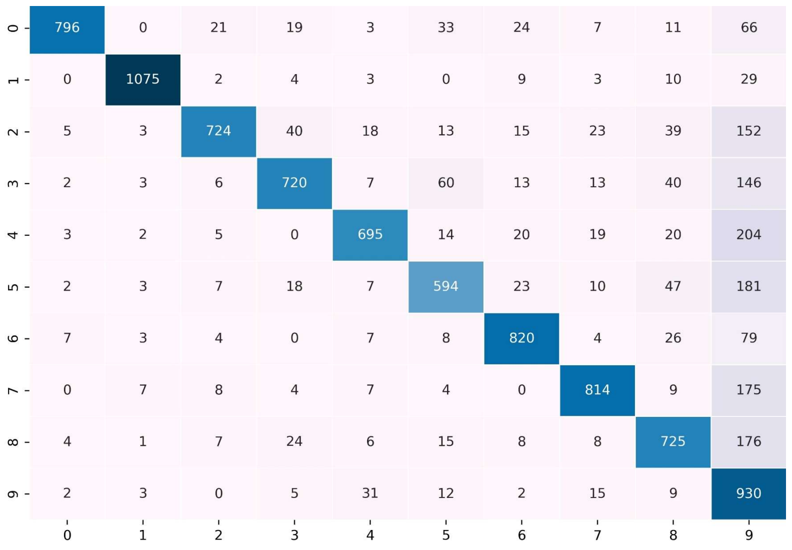
	0	1	2	3	4	5	6	7	8	9
0	974	0	1	0	0	1	2	1	1	0
1	0	1129	3	0	0	2	1	0	0	0
2	4	0	1012	1	1	0	2	7	5	0
3	0	0	3	991	0	4	0	6	4	2
4	0	0	2	0	962	0	4	1	2	11
5	2	0	0	7	0	877	3	1	1	1
6	4	2	0	0	4	3	944	0	1	0
7	1	8	9	2	1	0	0	1000	1	6
8	2	0	1	2	1	1	2	2	959	4
9	4	4	1	3	9	2	1	7	3	975

- **Confusion matrix of RBF SVM with all features**



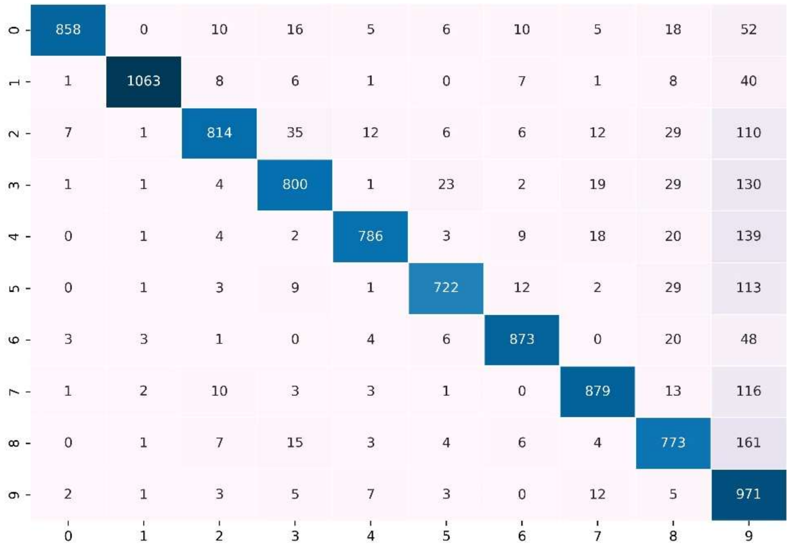
	0	1	2	3	4	5	6	7	8	9
0	970	0	0	0	0	1	4	1	3	1
1	0	1125	3	1	0	1	2	1	2	0
2	4	0	1003	3	1	0	2	5	13	1
3	0	0	2	987	0	3	0	6	4	8
4	1	0	3	0	953	0	3	0	4	18
5	2	0	0	5	0	875	3	1	4	2
6	5	2	0	0	3	3	941	0	4	0
7	0	4	11	2	0	1	0	990	4	16
8	1	0	1	2	1	1	0	2	962	4
9	3	2	0	4	6	1	0	4	3	986

- **Confusion matrix of Decision Tree with 40 features**



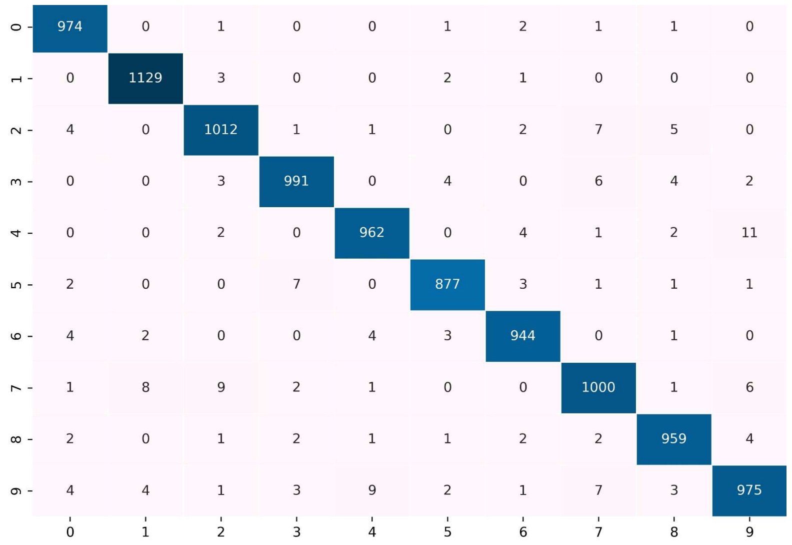
	0	1	2	3	4	5	6	7	8	9
0	796	0	21	19	3	33	24	7	11	66
1	0	1075	2	4	3	0	9	3	10	29
2	5	3	724	40	18	13	15	23	39	152
3	2	3	6	720	7	60	13	13	40	146
4	3	2	5	0	695	14	20	19	20	204
5	2	3	7	18	7	594	23	10	47	181
6	7	3	4	0	7	8	820	4	26	79
7	0	7	8	4	7	4	0	814	9	175
8	4	1	7	24	6	15	8	8	725	176
9	2	3	0	5	31	12	2	15	9	930

- **Confusion matrix of Decision Tree with 100 features**



	0	1	2	3	4	5	6	7	8	9
0	858	0	10	16	5	6	10	5	18	52
1	1	1063	8	6	1	0	7	1	8	40
2	7	1	814	35	12	6	6	12	29	110
3	1	1	4	800	1	23	2	19	29	130
4	0	1	4	2	786	3	9	18	20	139
5	0	1	3	9	1	722	12	2	29	113
6	3	3	1	0	4	6	873	0	20	48
7	1	2	10	3	3	1	0	879	13	116
8	0	1	7	15	3	4	6	4	773	161
9	2	1	3	5	7	3	0	12	5	971

- **Confusion matrix of Decision Tree with all features**

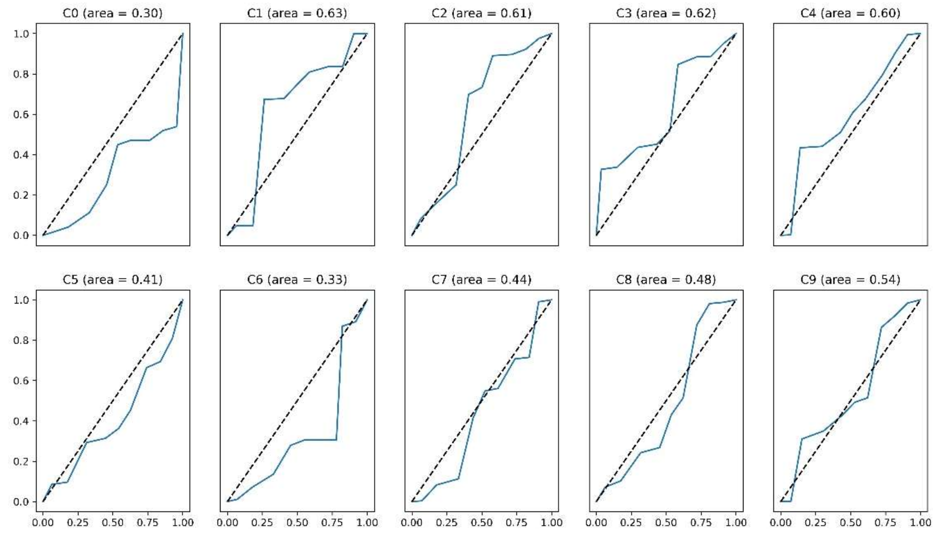


	0	1	2	3	4	5	6	7	8	9
0	974	0	1	0	0	1	2	1	1	0
1	0	1129	3	0	0	2	1	0	0	0
2	4	0	1012	1	1	0	2	7	5	0
3	0	0	3	991	0	4	0	6	4	2
4	0	0	2	0	962	0	4	1	2	11
5	2	0	0	7	0	877	3	1	1	1
6	4	2	0	0	4	3	944	0	1	0
7	1	8	9	2	1	0	0	1000	1	6
8	2	0	1	2	1	1	2	2	959	4
9	4	4	1	3	9	2	1	7	3	975

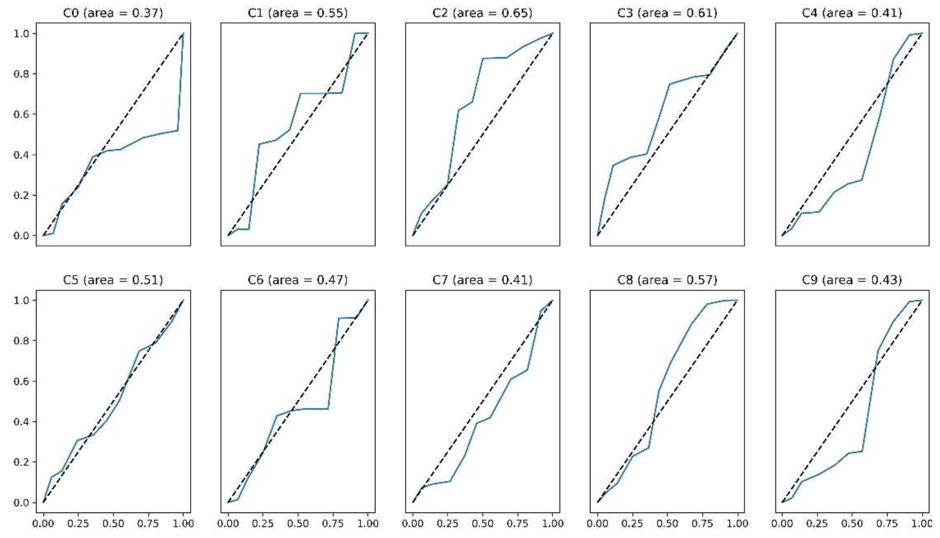
4. 4. ROC curve and AUC value

To compare the models in more details, we plot ROC curve for each model and for each class separately. The AUC value set as title of each subplot.

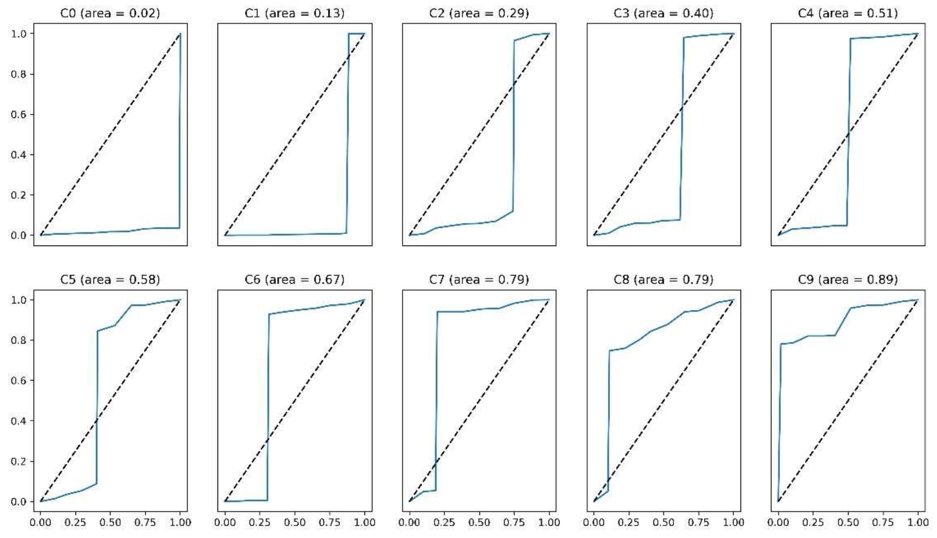
- **ROC curve of Linear SVM with 40 features**



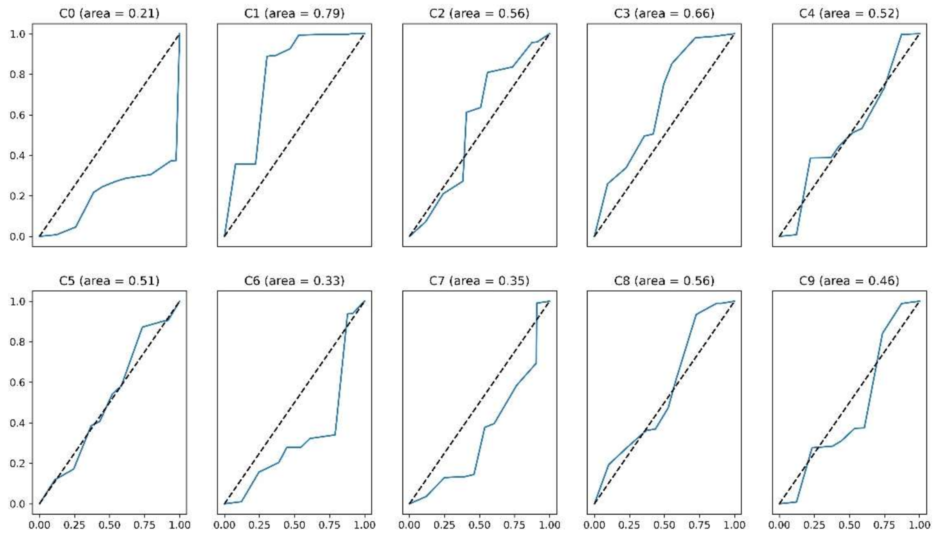
- **ROC curve of Linear SVM with 100 features**



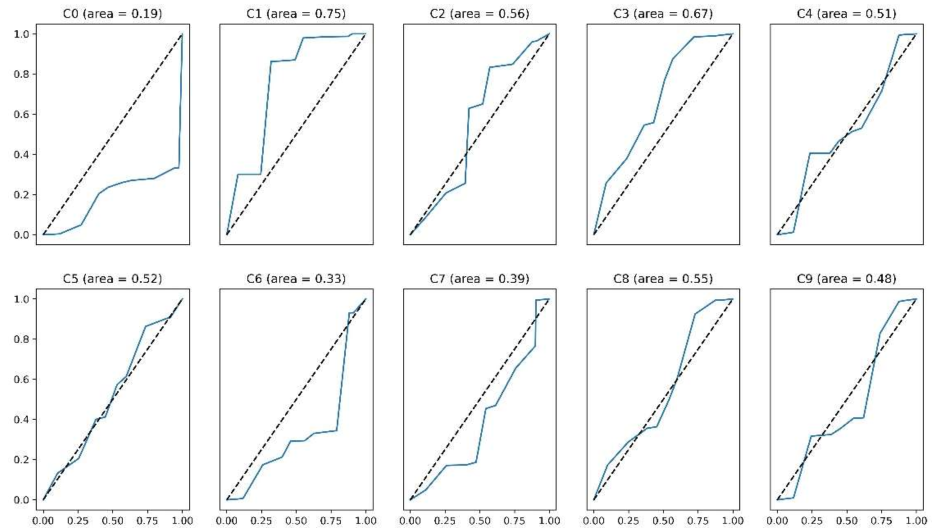
- **ROC curve of Linear SVM with all features**



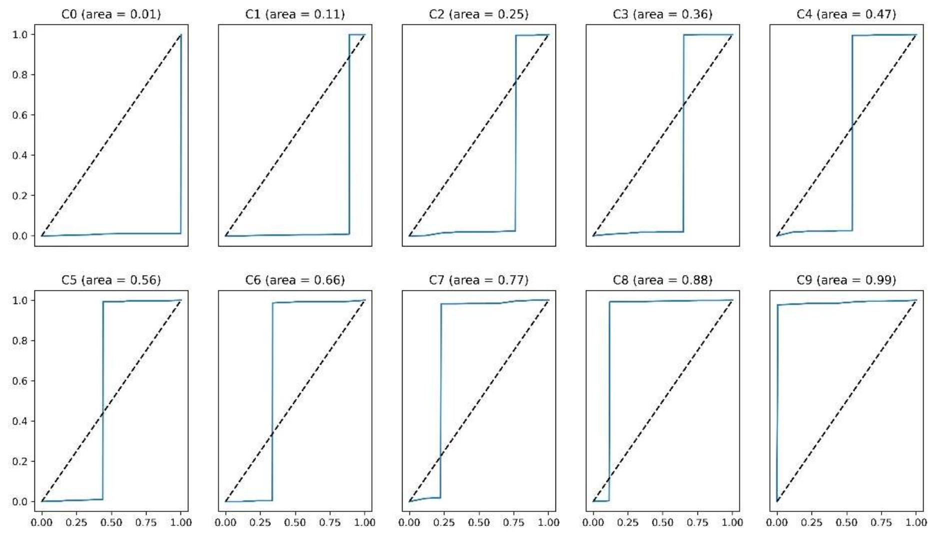
- **ROC curve of RBF SVM with 40 features**



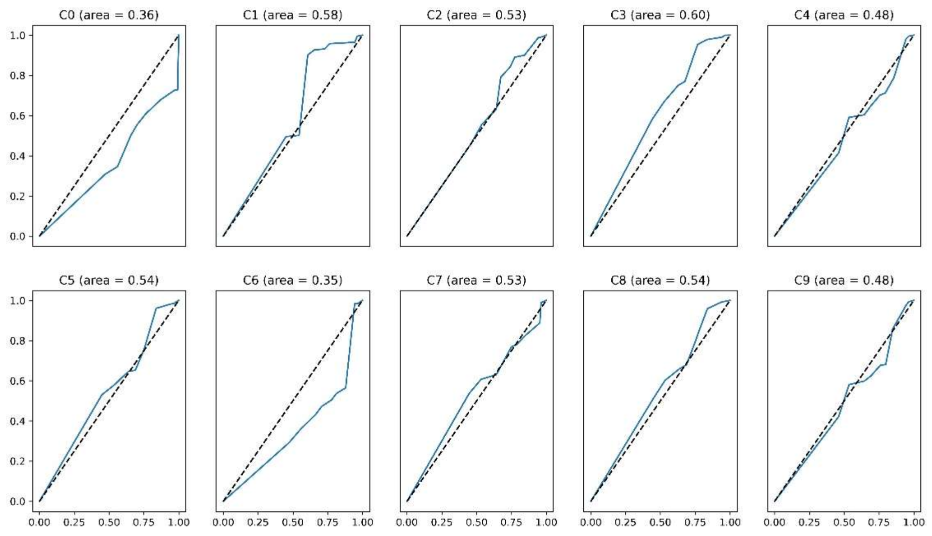
- **ROC curve of RBF SVM with 100 features**



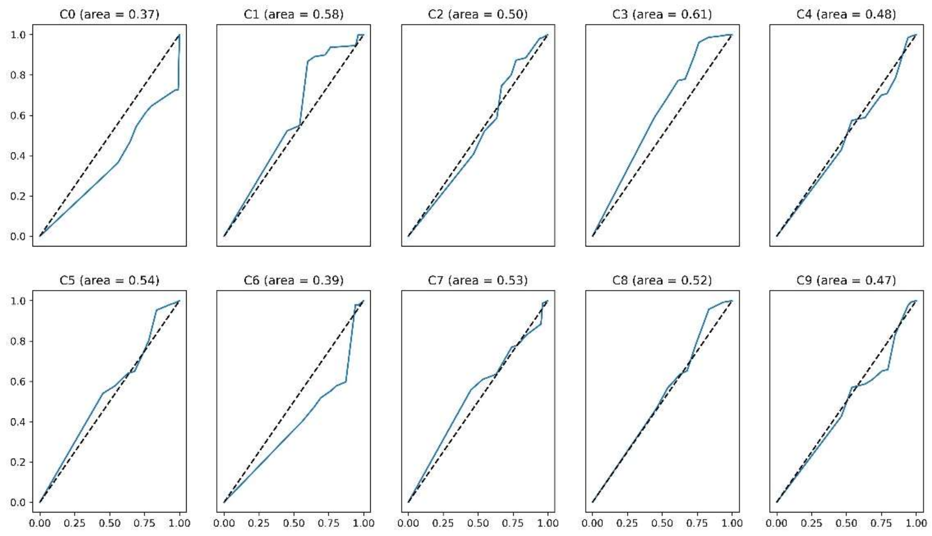
- **ROC curve of RBF SVM with all features**



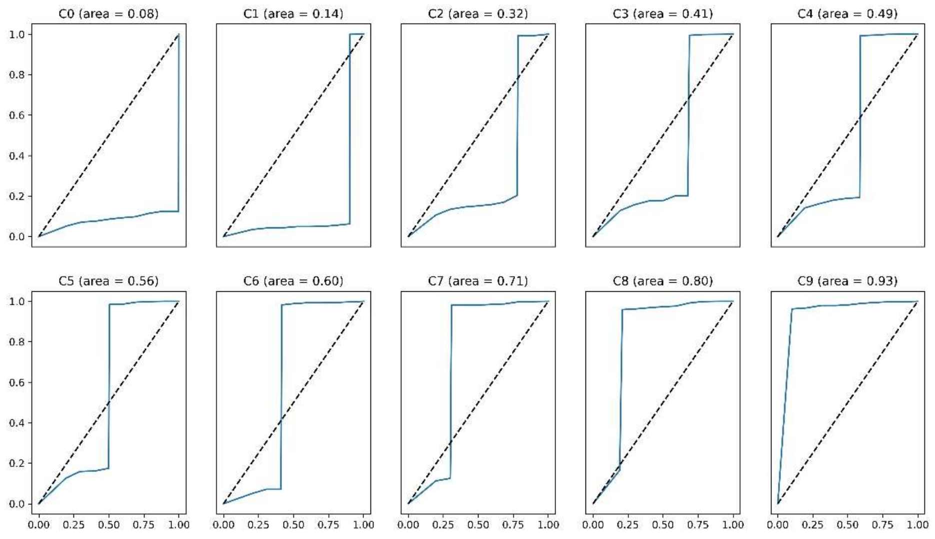
- **ROC curve of Decision Tree with 40 features**



- **ROC curve of Decision Tree with 100 features**

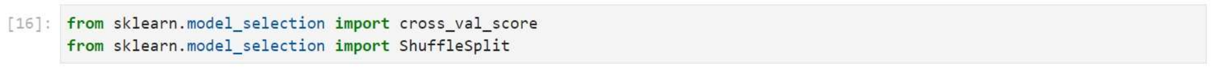


- **ROC curve of Decision Tree with all features**

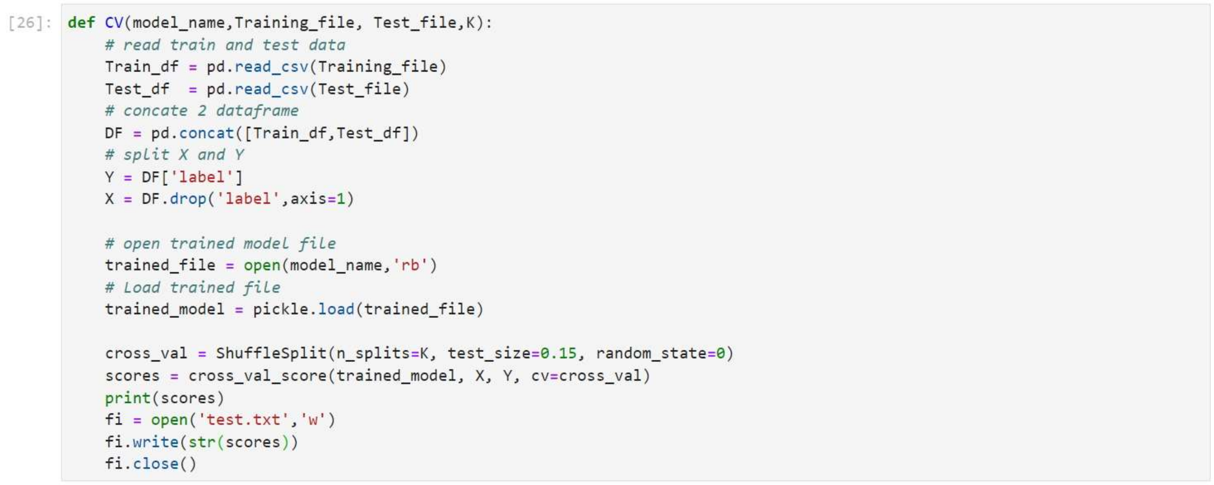


5. Bounce Section (k-fold cross validation)

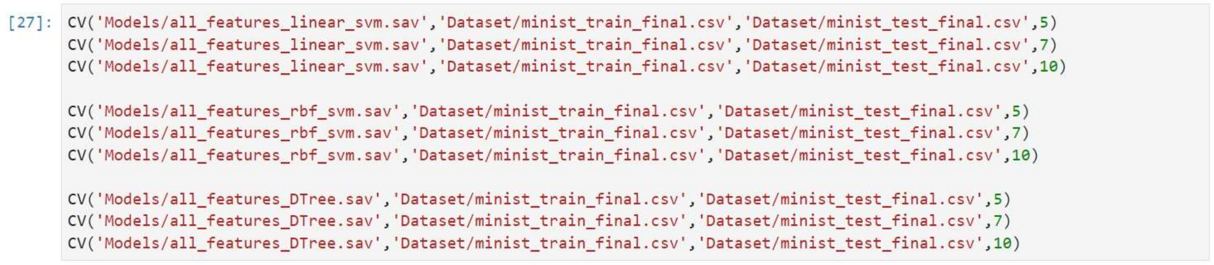
In this section, we chose the best models from previous sections and try to run k-fold cross validation on them. According to the results, datasets with all available features had the best results in all three kind of models (linear SVM, RBF SVM, and Decision Tree).



```
[16]: from sklearn.model_selection import cross_val_score
      from sklearn.model_selection import ShuffleSplit
```



```
[26]: def CV(model_name,Training_file, Test_file,K):
          # read train and test data
          Train_df = pd.read_csv(Training_file)
          Test_df  = pd.read_csv(Test_file)
          # concate 2 dataframe
          DF = pd.concat([Train_df,Test_df])
          # split X and Y
          Y = DF['label']
          X = DF.drop('label',axis=1)

          # open trained model file
          trained_file = open(model_name,'rb')
          # Load trained file
          trained_model = pickle.load(trained_file)

          cross_val = ShuffleSplit(n_splits=K, test_size=0.15, random_state=0)
          scores = cross_val_score(trained_model, X, Y, cv=cross_val)
          print(scores)
          fi = open('test.txt','w')
          fi.write(str(scores))
          fi.close()
```



```
[27]: CV('Models/all_features_linear_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',5)
      CV('Models/all_features_linear_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',7)
      CV('Models/all_features_linear_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',10)

      CV('Models/all_features_rbf_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',5)
      CV('Models/all_features_rbf_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',7)
      CV('Models/all_features_rbf_svm.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',10)

      CV('Models/all_features_DTree.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',5)
      CV('Models/all_features_DTree.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',7)
      CV('Models/all_features_DTree.sav','Dataset/minist_train_final.csv','Dataset/minist_test_final.csv',10)
```

References

1) Shamim, S. M., Miah, M. B. A., Angona Sarker, M. R., & Al Jobair, A. (2018). Handwritten digit recognition using machine learning algorithms. Global Journal Of Computer Science And Technology.

2) Dixit, S., Bharath, M., Amith, Y., Goutham, M. L., Ayappa, K., & Harshitha, D. (2018). Optical recognition of digital characters using machine learning. International Journal of Research Studies in Computer Science and Engineering, 5(1), 9-16.

3) LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

4) Guillevic, D., & Suen, C. Y. (1995, August). Cursive script recognition applied to the processing of bank cheques. In Proceedings of 3rd International Conference on Document Analysis and Recognition (Vol. 1, pp. 11-14). IEEE.

5) www.javatpoint.com/principal-component-analysis

6) www.geeksforgeeks.org/radial-basis-function-kernel-machine-learning/

7) www.geeksforgeeks.org/creating-linear-kernel-svm-in-python/

8) www.geeksforgeeks.org/decision-tree/

9) www.geeksforgeeks.org/confusion-matrix-machine-learning/

10) www.machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/

11) www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/