Assignment on Neural Networks № 2

Christos Chrysikos THMMY AEM 9432, Aristotle University of Thessaloniki 22/12/2022

Linear Kernel

Basic training Algorithm Support Vector Machine for the classification of images from CIFAR-10 data set using a linear kernel and different values for c parameter.

Listing 1: The code used for the implementaion of the test – Testing the effectiveness of Linear Kernel.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import cifar10
%matplotlib inline
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import warnings
from sklearn.exceptions import DataConversionWarning

warnings.filterwarnings(action='ignore', category=DataConversionWarning)

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
classesName = ["plane", "car", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

x_train = x_train.astype(np.float)
x_test = x_test.astype(np.float)

#reshaping data into a vector and normalizing it(-1 to 1)

x_train = np.reshape(x_train, (x_train.shape[0], -1))
y_train = np.reshape(y_train, (y_train.shape[0], -1))
x_test = np.reshape(x_test, (x_test.shape[0], -1))
y_test = np.reshape(y_test, (y_test.shape[0], -1))

x_train=((x_train/255)*2)-1
x_test = ((x_test/255)*2)-1


```

```
#Working on a smaller dataset(3000 samples for training and 2000 testing samples)

x_train=x_train[:3000,:]
y_train=y_train[:3000,:]
x_test = x_test[:2000,:]
y_test = y_test[:2000,:]

y_test = np.array(np.reshape(y_test,2000))

#function for plotting images

def plt_img(x, ax):
    n_row = 32
    n_col = 32
    n_colour=3
    x_new = x.reshape((n_row,n_col,n_colour))

    #reshaping images to their initial dimensions in order to represent them

    ax.imshow(x_new)

def linear_svm(c,krnl):
  clf = SVC(C=c, kernel= krnl)
  start = time.time()
  clf.fit(x_train, y_train)
  stop = time.time()
  training_time = stop - start
  y_pred_train = clf.predict(x_train)
  y_pred_test = clf.predict(x_test)
  test_accuracy = accuracy_score(y_test, y_pred_test)
  train_accuracy = accuracy_score(y_train, y_pred_train)
  print("For c = "+str(c)+" accuracy on testing set is: "+str(test_accuracy)+" , accuracy on training set is: "+str(train_accuracy)+" and training time is equal to: "+str(training_time) +" seconds")

#plot 4 false predicted images and 4 right predicted ones for c=0.1

  if c == 0.1:
    print("Some image predictions for c = 0.1: ")

    n=4
    err = np.where(y_pred_test != y_test)[0]
    right = np.where(y_pred_test == y_test)[0]

    f, axarr = plt.subplots(2,n, figsize=(20,8))
```

```
74
75      for i in range (n):
76        e_i = err[i]
77        r_i = right[i]
78
79        plt_img(x_test[e_i,:], axarr[0,i])
80        title1 = 'true={0:s} est={1:s}'.format(classesName[y_test[e_i].↩
              astype(int)], classesName[y_pred_test[e_i].astype(int)])
81        axarr[0,i].set_title(title1)
82
83        plt_img(x_test[r_i,:], axarr[1,i])
84        title2 = 'true={0:s} est={1:s}'.format(classesName[y_test[r_i].↩
              astype(int)], classesName[y_pred_test[r_i].astype(int)])
85        axarr[1,i].set_title(title2)
86
87  #Running SVM for different c values
88
89  c_svm_linear = [0.0001,0.001,0.01,0.1,1,10,100]
90
91  for c in c_svm_linear:
92    linear_svm(c, "linear")
```

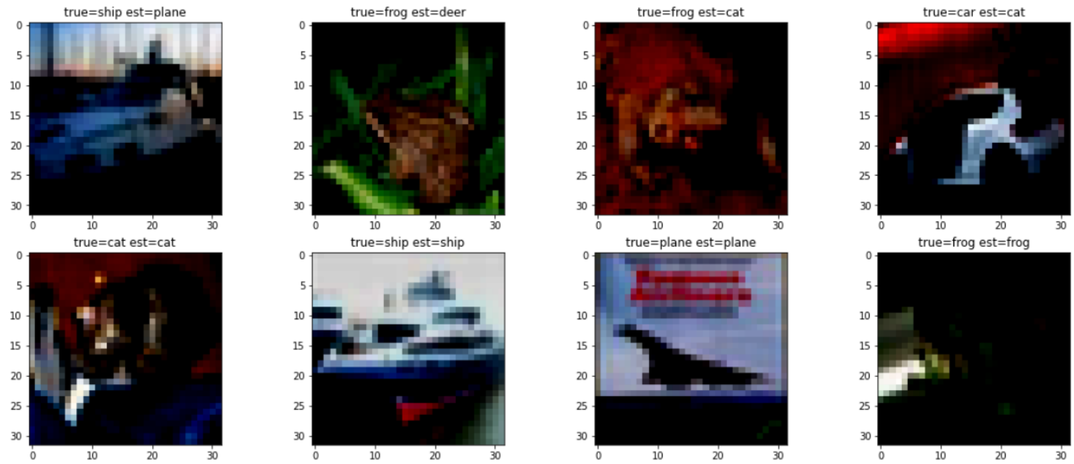
Listing 2: The following results are printed from the console.

```
1  Kernel Used linear
2  For c = 0.0001 accuracy on testing set is: 0.3185 , accuracy on training ↩
       set is: 0.3546666666666667 and training time is equal to: ↩
       15.688592672348022 seconds
3  For c = 0.001 accuracy on testing set is: 0.3595 , accuracy on training ↩
       set is: 0.4846666666666667 and training time is equal to: ↩
       12.517436265945435 seconds
4  For c = 0.01 accuracy on testing set is: 0.3405 , accuracy on training set↩
        is: 0.7093333333333334 and training time is equal to: ↩
       11.771913528442383 seconds
5  For c = 0.1 accuracy on testing set is: 0.303 , accuracy on training set ↩
       is: 0.9896666666666667 and training time is equal to: ↩
       14.323830842971802 seconds
6  For c = 1 accuracy on testing set is: 0.299 , accuracy on training set is:↩
        1.0 and training time is equal to: 14.685425996780396 seconds
7  For c = 10 accuracy on testing set is: 0.299 , accuracy on training set is↩
       : 1.0 and training time is equal to: 14.69365668296814 seconds
8  For c = 100 accuracy on testing set is: 0.299 , accuracy on training set ↩
       is: 1.0 and training time is equal to: 15.113477945327759 seconds
```

Following the results from Linear Model . . .
Above they are given the results for accuracy on training and testing set, time of completion as well as the examples of valid and non valid classification.

While we increase the c parameter, we achieve better accuracy on the training set, while this is not the case for the testing set. For c => 0.1 we observe the over-fitting phenomenon cause our model becomes almost perfectly accurate on our training set without managing to increase the accuracy on the testing data(accuracy < 30%) to an acceptable level.

```
For c = 0.0001 accuracy on testing set is: 0.3185 , accuracy on training set is: 0.3546666666666667 and training time is equal to: 15.688592672348022 seconds
For c = 0.001 accuracy on testing set is: 0.3595 , accuracy on training set is: 0.4846666666666667 and training time is equal to: 12.517436265945435 seconds
For c = 0.01 accuracy on testing set is: 0.3405 , accuracy on training set is: 0.7093333333333334 and training time is equal to: 11.771913528442383 seconds
For c = 0.1 accuracy on testing set is: 0.303 , accuracy on training set is: 0.9896666666666667 and training time is equal to: 14.323830842971802 seconds
Some image predictions for c = 0.1:
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
For c = 1 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 14.685425996780396 seconds
For c = 10 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 14.69365668296814 seconds
For c = 100 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 15.113477945327759 seconds
```

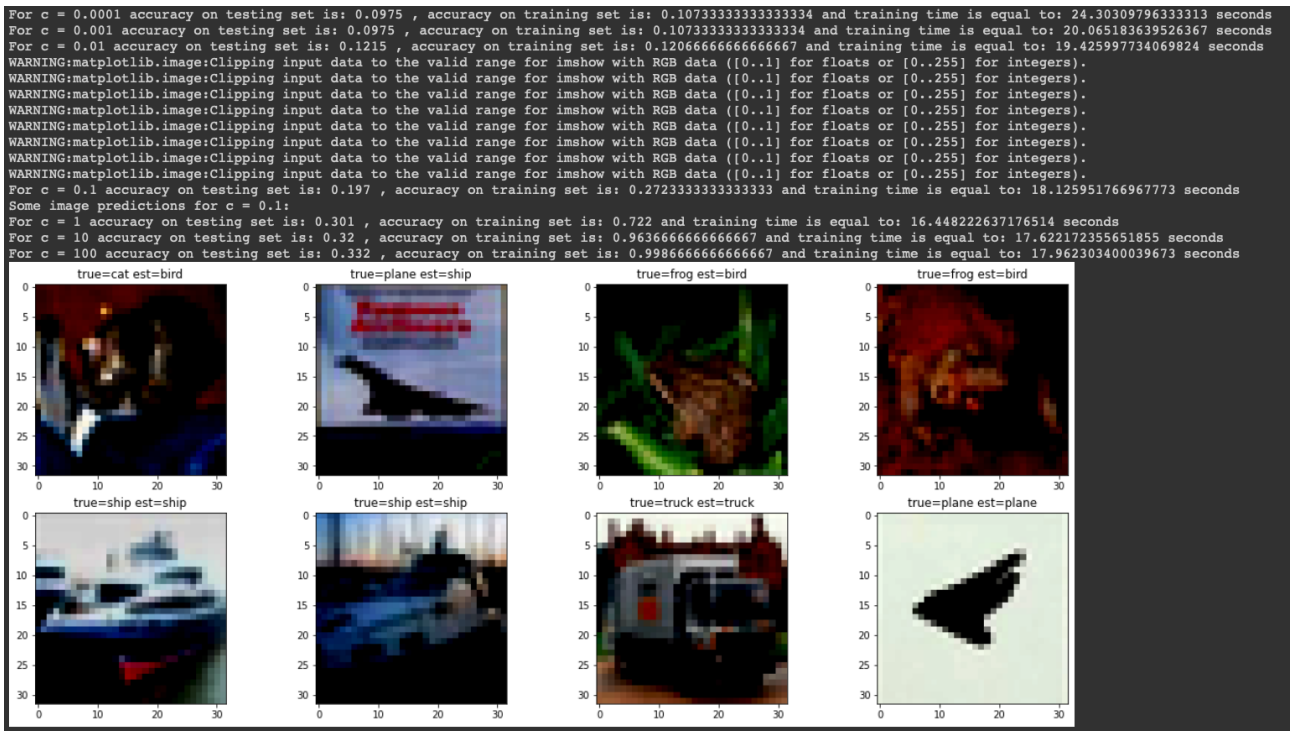


Polynomial Kernel

We repeat the same exact process but this time we are going to use the polynomial kernel. At line 86 we are going to change linear to **poly** and results are as shown bellow.

Listing 3: The following results are printed from the console.

```
1 Kernel Used poly
2 For c = 0.0001 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 20.302833557128906 seconds
3 For c = 0.001 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 20.478830337524414 seconds
4 For c = 0.01 accuracy on testing set is: 0.1215 , accuracy on training set is: 0.12066666666666667 and training time is equal to: 20.83187246322632 seconds
5 For c = 0.1 accuracy on testing set is: 0.197 , accuracy on training set is: 0.2723333333333333 and training time is equal to: 19.34217071533203 seconds
6 Some image predictions for c = 0.1:
7 For c = 1 accuracy on testing set is: 0.301 , accuracy on training set is: 0.722 and training time is equal to: 16.925872325897217 seconds
8 For c = 10 accuracy on testing set is: 0.32 , accuracy on training set is: 0.9636666666666667 and training time is equal to: 18.560601234436035 seconds
9 For c = 100 accuracy on testing set is: 0.332 , accuracy on training set is: 0.9986666666666667 and training time is equal to: 18.44046425819397 seconds
```



As well as the previous case the results remain the same for c >= 1. For the lower values of c the model is not "good" enough fro both training and testing data-sets the lower the c value, the worst the model gets. An other noticeable difference is the increase of training time, which makes us understand that the model with polynomial kernel isn't effective enough in comparison with the linear model.

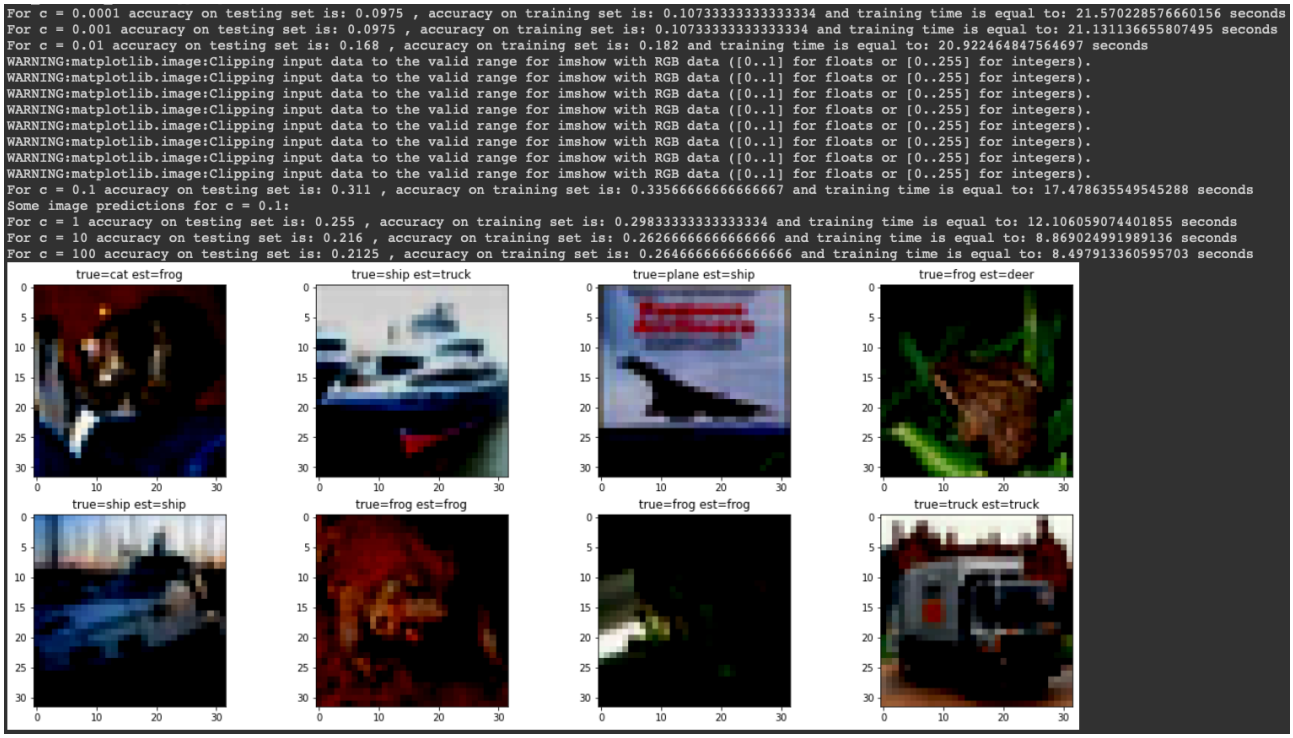
Sigmoid Kernel

We repeat the same exact process but this time we are going to use the polynomial kernel. At line 86 we are going to change linear to **sigmoid** and results are as shown bellow.

Listing 4: The following results are printed from the console.

```
1 Kernel Used sigmoid
2 For c = 0.0001 accuracy on testing set is: 0.3185 , accuracy on training ↩
    set is: 0.3546666666666667 and training time is equal to: ↩
    15.688592672348022 seconds
3 For c = 0.001 accuracy on testing set is: 0.3595 , accuracy on training ↩
    set is: 0.4846666666666667 and training time is equal to: ↩
    12.517436265945435 seconds
4 For c = 0.01 accuracy on testing set is: 0.3405 , accuracy on training set↩
     is: 0.7093333333333334 and training time is equal to: ↩
    11.771913528442383 seconds
5 For c = 0.1 accuracy on testing set is: 0.303 , accuracy on training set ↩
    is: 0.9896666666666667 and training time is equal to: ↩
    14.323830842971802 seconds
```

```
For c = 1 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 14.685425996780396 seconds
For c = 10 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 14.69365668296814 seconds
For c = 100 accuracy on testing set is: 0.299 , accuracy on training set is: 1.0 and training time is equal to: 15.113477945327759 seconds
```



In this case we see an increase in completion time exactly as we saw with the previous ("poly") kernel we used. Although while we increase the value of c the completion time decreases, especially for c>=1. As for the accuracy the results are not in the acceptable range because the testing accuracy does not overcome the 30'% barrier in any case.

RBF Kernel

At last we are going to use the RBF kernel but in this case we are going to change the value of gamma parameter trying to get different decision limits. Bellow are presented the differences that took place to code.

Listing 5: Changes made to the original code – Testing the effectiveness of RBF Kernel.

```
def rbf_svm(c,krnl,g):
  clf = SVC(C=c, kernel= krnl, gamma=g)
  start = time.time()
  clf.fit(x_train, y_train)
  stop = time.time()
```

```
  training_time = stop - start
  y_pred_train = clf.predict(x_train)
  y_pred_test = clf.predict(x_test)
  test_accuracy = accuracy_score(y_test, y_pred_test)
  train_accuracy = accuracy_score(y_train, y_pred_train)
  print("For c = "+str(c)+" and gamma = "+str(g)+" accuracy on testing set is: "+str(test_accuracy)+" , accuracy on training set is: "+str(train_accuracy)+" and training time is equal to: "+str(training_time) +" seconds")
```

Listing 6: Changes made to the original code – Testing the effectiveness of RBF Kernel.

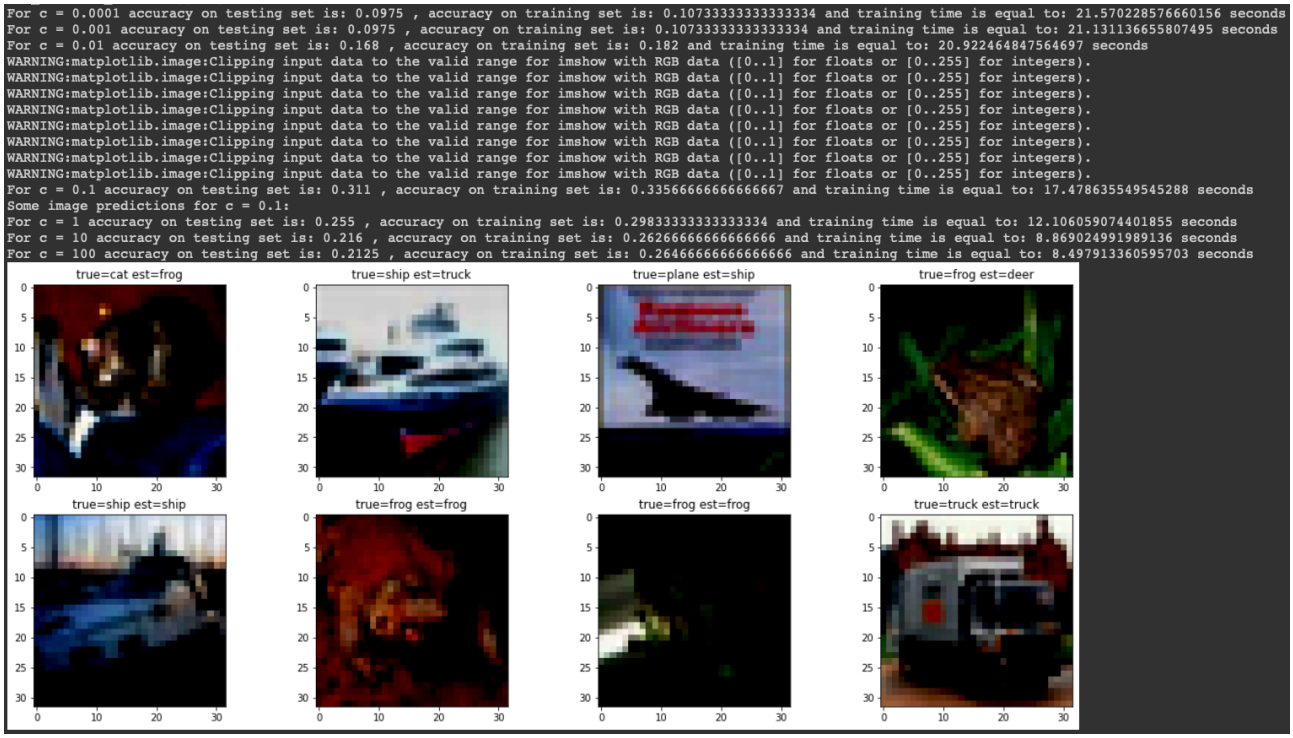
```
#running rbf svm for different c and gamma values
c_svm_rbf = [0.001, 0.1, 1, 10, 100]
gamma_values = [0.01,0.1,1,10,100]
for c in c_svm_rbf:
  for gamma in gamma_values:

    rbf_svm(c, 'rbf', gamma)
```

Listing 7: The following results are printed from the console.

```
Kernel Used rbf
For c = 0.001 and gamma = 0.01 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 27.51534652709961 seconds
For c = 0.001 and gamma = 0.1 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 25.443522691726685 seconds
For c = 0.001 and gamma = 1 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 21.723928213119507 seconds
For c = 0.001 and gamma = 10 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 22.977734327316284 seconds
For c = 0.001 and gamma = 100 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 20.535745859146118 seconds
For c = 0.1 and gamma = 0.01 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 22.791850328445435 seconds
For c = 0.1 and gamma = 0.1 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 20.06384515762329 seconds
```

```
For c = 0.1 and gamma = 1 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 28.352800130844116 seconds
For c = 0.1 and gamma = 10 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 22.631453275680542 seconds
For c = 0.1 and gamma = 100 accuracy on testing set is: 0.0975 , accuracy on training set is: 0.10733333333333334 and training time is equal to: 25.593669891357422 seconds
For c = 1 and gamma = 0.01 accuracy on testing set is: 0.267 , accuracy on training set is: 0.9993333333333333 and training time is equal to: 21.571255207061768 seconds
For c = 1 and gamma = 0.1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.46774196624756 seconds
For c = 1 and gamma = 1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.21293616294861 seconds
For c = 1 and gamma = 10 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 25.77965474128723 seconds
For c = 1 and gamma = 100 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 22.604223012924194 seconds
For c = 10 and gamma = 0.01 accuracy on testing set is: 0.28 , accuracy on training set is: 1.0 and training time is equal to: 23.440980911254883 seconds
For c = 10 and gamma = 0.1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.933592081069946 seconds
For c = 10 and gamma = 1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 25.909340858459473 seconds
For c = 10 and gamma = 10 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.636533498764038 seconds
For c = 10 and gamma = 100 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.968435525894165 seconds
For c = 100 and gamma = 0.01 accuracy on testing set is: 0.28 , accuracy on training set is: 1.0 and training time is equal to: 23.487674236297607 seconds
For c = 100 and gamma = 0.1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to: 23.905787706375122 seconds
For c = 100 and gamma = 1 accuracy on testing set is: 0.0975 , accuracy on training set is: 1.0 and training time is equal to:
```

```
    22.712003469467163 seconds
25  For c = 100 and gamma = 10 accuracy on testing set is: 0.0975 , accuracy ↩
    on training set is: 1.0 and training time is equal to: ↩
    24.44967246055603 seconds
26  For c = 100 and gamma = 100 accuracy on testing set is: 0.0975 , accuracy ↩
    on training set is: 1.0 and training time is equal to: ↩
    23.114515781402588 seconds
```



The training time here is even more increased than the two previous variations. All the combinations of gamma values when c<=1 result in a low percentage of correct predictions both in training and test set. When c > 1 we can see the over-fitting phenomenon because the accuracy of the training set is close to perfect but the accuracy on the training set is very low (approximately 1%).

Comparison

Listing 8: Comparing the results for SVM and Nearest Centroid and KNN.

```
For Nearest Centroid :
Accuracy: 0.82

For K Nearest Neighbor = 1 :
Accuracy: 0.97

For SVM :
Accuracy: ~0.3
```

Comparing the accuracy of the 3 different models (1-, NCC, SVM) it is obvious that the SVM is less effective with a difference in accuracy of about 50%. Even though the results make the SVM model seem as a lesser choice this is not the general case. The SVM's are highly dependant to the size of the training set and they can be very accurate when combined with a suitable training set size.