**Introduction to Machine Learning**

# Homework Solution - 4

*Instructor:* Prof Chinmay Hegde *TA:* Devansh Bisla, Maryam Majzoubi

**Problem 1**

In class, we discussed how to represent XOR-like functions using quadratic features, since standard linear classifiers (such as perceptrons) are insufficient for this task. However, here we show that XOR-like functions can indeed be simulated using *multi-layer* networks of perceptrons. This example shows a glimpse of the expressive power of "deep neural networks": merely increasing the depth from 1 to 2 layers can help reproduce nonlinear decision boundaries.

a. Consider a standard two-variable XOR function, where we have 2-dimensional inputs $x_1, x_2 = \pm 1$, and output $y = x_1 (XOR) x_2 = \begin{cases} -1 & \text{if} \quad x_1 = x_2 \\ 1 & \text{otherwise} \end{cases}$.

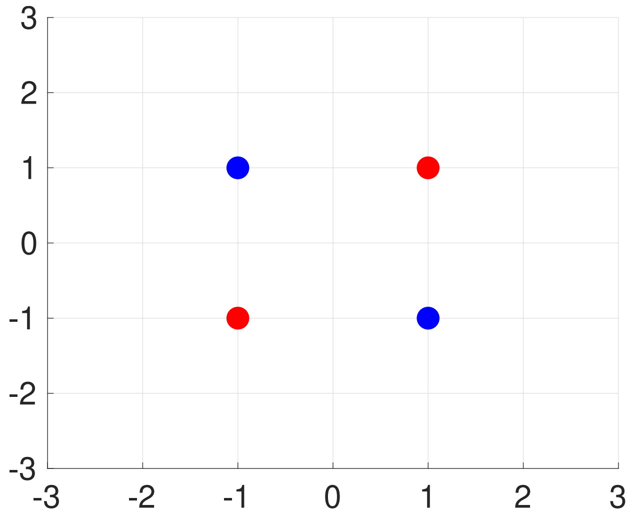
Geometrically argue why a single perceptron cannot be used to simulate the above function.

b. Graphically depict, and write down the equation for, the optimal decision region for the following logical functions: (i) $x_1 (AND)(NOT(x_2))$ (ii) $(NOT(x_1))(AND)x_2$ (iii) $x_1 (OR) x_2$ Make note of the weights learned corresponding to the optimal decision boundary for each function.

c. Using the above information, simulate a multi-layer perceptron *network* for the XOR operation with the learned weights from Part (b).

***(Solution)***

(a) The truth table of the XOR function with its graph is shown below

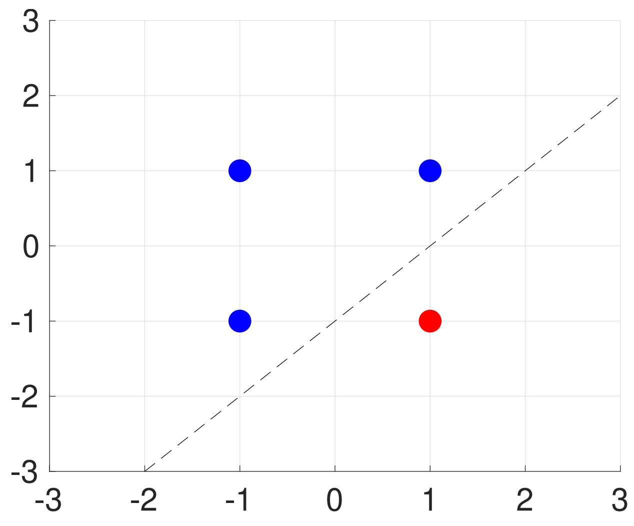


| $x_1$ | $x_2$ | y |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 1 | 1 |
| 1 | -1 | 1 |
| 1 | 1 | -1 |

A single perceptron only generates linear separator. It is observed form the figure that linear separator cannot solve the classification problem.

(b)

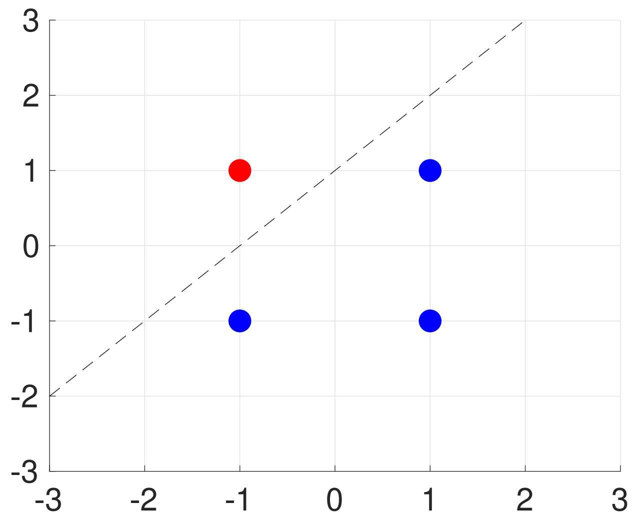
- $y = x_1(AND)(NOT(x_2))$. The truth table and corresponding graph is shown below:



| $x_1$ | $x_2$ | y |
|---|---|---|
| 1 | 1 | -1 |
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| -1 | -1 | -1 |

  It is clear from the graph that the linear separator can be easily defined as $x_1 - x_2 - 1 = 0$.

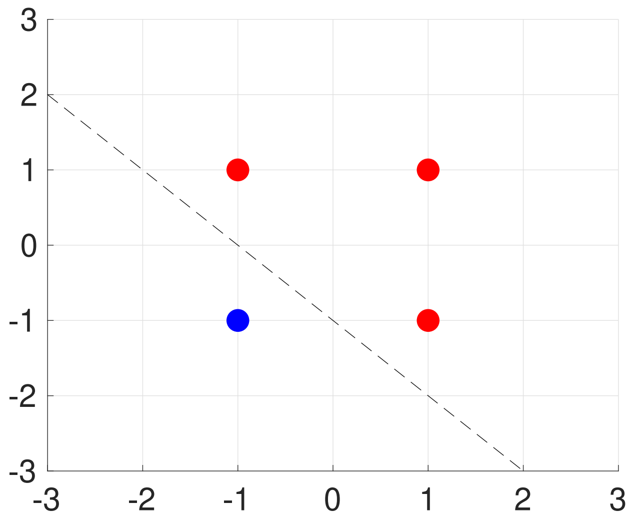
- $y = (NOT(x_1))(AND)x_2$. The truth table and corresponding graph is shown below:



| $x_1$ | $x_2$ | y |
|---|---|---|
| 1 | 1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | 1 |
| -1 | -1 | -1 |

  It is clear from the graph that the linear separator can be easily defined as $x_1 - x_2 + 1 = 0$.

- $y = x_1(OR)x_2$. The truth table and corresponding graph is shown below:



| $x_1$ | $x_2$ | y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -1 | 1 |
| -1 | 1 | 1 |
| -1 | -1 | -1 |

It is clear from the graph that the linear separator can be easily defined as $x_1 - x_2 + 1 = 0$.

(c) $y = XOR(x_1, x_2)$ can be easily written as $y = AND(NOT(AND(x_1, x_2)), OR(x_1, x_2))$. Thus, using multi-layer perceptron we can solve the XOR problem.

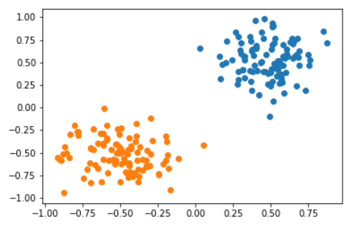
**Problem 2**
In this problem, we will implement the Perceptron algorithm discussed in class on synthetic training data.

a. Suppose that the data dimension $d$ equals 2. Generate two clusters of data points with 100 points each, by sampling from Gaussian distributions centered at $(0.5, 0.5)$ and $(-0.5, -0.5)$. Choose the variance of the Gaussian to be small enough so that the data points are sufficiently well separated. Plot the data points on the 2D plane to confirm that this is the case.

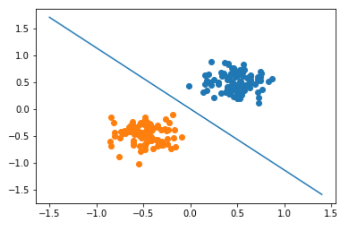
b. Implement the Perceptron algorithm as discussed in class. Choose the initial weights to be zero and the maximum number of epochs as $T = 100$, and the learning rate $\alpha = 1$. How quickly does your implementation converge?

c. Now, repeat the above experiment with a second synthetic dataset; this time, increase the variance (radius) of the two Gaussians such that the generated data points from different classes now overlap. What happens to the behavior of the algorithm? Does it converge? Show the classification regions obtained at the end of $T$ epochs.

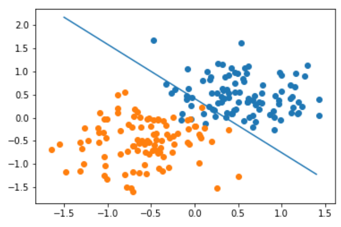
***(Solution)***



a



b



c

When the two clusters are linearly separable the convergence is pretty fast, however for the case they overlap the perceptron algorithm does not converge as it is observed in the final classification regions in part c. Thanks to Dimitrios Chaikalis for the following Python solution:

```
import numpy as np
import matplotlib.pyplot as plt
import math

d = 2
means1 = np.array([0.5,0.5])
# 0.03 as an example gives separated clusters, try 0.2 for intertwined
cov = 0.03*np.eye(d)
x1 = np.random.multivariate_normal(means1,cov,100)
Y1 = np.ones(len(x1))
means2 = np.array([-0.5,-0.5])
x2 = np.random.multivariate_normal(means2,cov,100)
Y2 = -1*np.ones(len(x2))

X = np.r_[x1,x2]
Xb = np.ones(200)
X = np.c_[X, Xb]
Y = np.r_[Y1,Y2]
```

```
# plt.scatter(x1[:,0],x1[:,1])
# plt.scatter(x2[:,0],x2[:,1])

w = np.zeros((1,3))
for count in range(100):
    flag = 0
    for num in range(len(X)):
        pred = np.dot(w,X[num,:])
        if (pred*Y[num] <= 0) :
            w = w + 1*Y[num]*X[num,:]
        flag = 1
    if (flag==0):
        print(count)
        break

time = np.arange(-1.5,1.5,0.1)
line = -(w[0,0]/w[0,1])*time - w[0,2]/w[0,1]
plt.plot(time,line)
plt.scatter(x1[:,0],x1[:,1])
plt.scatter(x2[:,0],x2[:,1])
plt.savefig('2.png')
plt.title('Classification')
```

**Problem 3**
In Lectures 2 and 5, we derived a closed form expression for solving linear and ridge regression problems. This is great for finding linear behavior in data; however, if the data is non-linear, just as in the classification case, we have to resort to the *kernel trick*, i.e., replace all dot products in the data space with kernel inner products. In this problem, we theoretically derive kernel regression. Suppose we are given training data $\{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$ where each response $y_i$ is a scalar, and each data point $x_i$ is a vector in $d$ dimensions.

a. Assume that all data have been mapped into a higher dimensional space using the feature mapping $x \mapsto \phi(x)$, write down an expression for the squared error function using a linear predictor $w$ in the high-dimensional space.

b. Let $\Phi$ be the matrix with $n$ rows, where row $i$ consists of the feature mapping $\phi(x_i)$. Write down a closed form expression for the optimal linear predictor $w$ as a function of $\Phi$ and $y$.

c. For a new query data point $z$, the predicted value is given by $f(z) = \langle w, \phi(z) \rangle$. Plug in the closed form expression for $w$ from the previous sub-problem to get an expression for $f(z)$.

d. Suppose you are given black-box access to a kernel dot product function $K$ where $K(x, x') = \langle \phi(x), \phi(x') \rangle$. Mathematically show that all the calculations in (b) and (c) can be performed by invoking the kernel dot product alone *without explicitly writing down $\phi(x)$ ever*. You may want to use the Sherman-Morrison-Woodbury identity for matrices:

$$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = AB^T (BAB^T + C)^{-1}.$$

***(Solution)***

a.

$$MSE = \frac{1}{2} \sum_{i=1}^{n} (y_i - \langle w, \phi(x_i) \rangle)^2 + \frac{1}{2} \lambda ||w||_2^2$$

b. We can re-write the above using matrix representations:

$$MSE = \frac{1}{2} ||Y - \Phi w||_2^2 + \frac{1}{2} \lambda ||w||_2^2$$

by taking derivative wrt to $w$ and set it to zero, the optimal $w^*$ is:

$$w^* = (\Phi^T \Phi + \lambda I_d)^{-1} \Phi^T Y = \Phi^T (\Phi \Phi^T + \lambda I_n)^{-1} Y$$

c.

$$f(z) = \langle w, \phi(z) \rangle = w^{*T} \phi(z) = Y^T (\Phi \Phi^T + \lambda I_n)^{-1} \Phi \phi(z)$$

d. Let $K_z$ and $K_{nn}$ as follows:

$$K_z = \Phi \phi(z) = \begin{bmatrix} \langle \phi(x_1), \phi(z) \rangle \\ \vdots \\ \langle \phi(x_n), \phi(z) \rangle \end{bmatrix} = \begin{bmatrix} K(x_1, z) \\ \vdots \\ K(x_n, z) \end{bmatrix}$$

and

$$K_{nn} = \Phi \Phi^T = \begin{bmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_1) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{bmatrix} = \begin{bmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{bmatrix}$$

Hence, we can re-write $f(z)$ using only kernel dot products:

$$f(z) = {w^*}^T \phi(z) = Y^T(\Phi\Phi^T + \lambda I_n)^{-1}\Phi\phi(z) = Y^T(K_{nn} + \lambda I_n)^{-1}K_z$$

**Problem 4**
The *Fashion MNIST* dataset is a database of (low-resolution) clothing images that is similar to MNIST but somewhat more challenging. You can load it in Colab using the Python code below.

a. Load the dataset and display 10 representative images from each class.

b. Implement the following classification methods: k-NN, logistic regression, and support vector machines (with linear and rbf kernels). You can use sklearn.

b. Report best possible test-error performances by tuning hyperparameters in each of your methods.

c. Report train- and test-running time of each of your methods in the form of a table, and comment on the relative tradeoffs across the different methods.

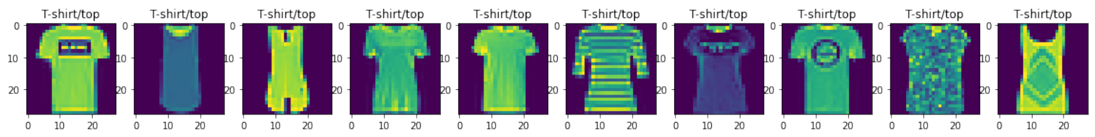
***(Solution)*** Solution by Aditya Ashtekar.

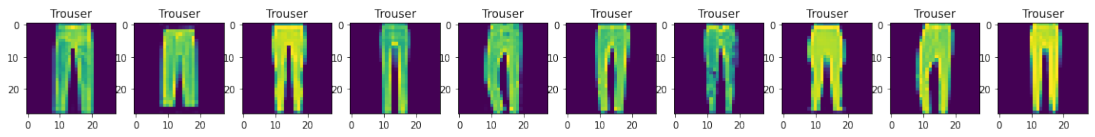
# ML_ass4_q4

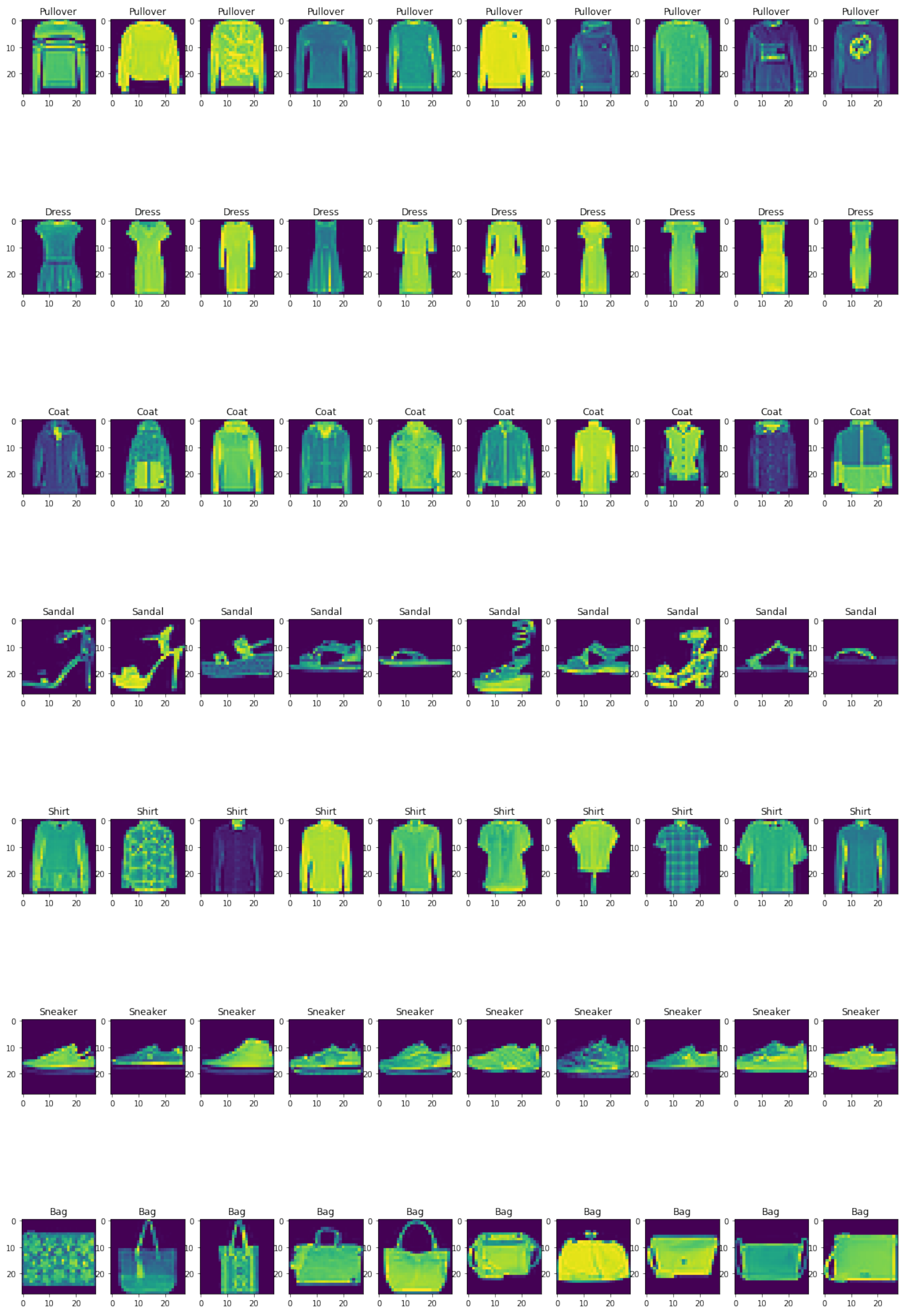
April 8, 2020

[0]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
import tensorflow as tf
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels),(test_images, test_labels) = fashion_mnist.
 ↪load_data()
```

[19]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal',␣
 ↪'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(20,8))
for i in range(0,9):
  index = 0
  for (image, label) in zip(train_images,train_labels):
    if(i == label):
      index += 1
      plt.subplot(1,10,index)
      plt.imshow(image)
      plt.title(class_names[label])
    if(index == 10):
      break
  plt.figure(figsize=(20,8))
```






Pullover
Pullover
Pullover
Pullover
Pullover
Pullover
Pullover
Pullover
Pullover
Pullover
Dress
Dress
Dress
Dress
Dress
Dress
Dress
Dress
Dress
Dress
Coat
Coat
Coat
Coat
Coat
Coat
Coat
Coat
Coat
Coat
Sandal
Sandal
Sandal
Sandal
Sandal
Sandal
Sandal
Sandal
Sandal
Sandal
Shirt
Shirt
Shirt
Shirt
Shirt
Shirt
Shirt
Shirt
Shirt
Shirt
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Sneaker
Bag
Bag
Bag
Bag
Bag
Bag
Bag
Bag
Bag
Bag
0
10
20

```
<Figure size 1440x576 with 0 Axes>
```

```
[0]: from sklearn.preprocessing import StandardScaler
     from sklearn.linear_model import LogisticRegression

     scaler = StandardScaler()
     train_images = train_images.reshape(60000,784)
     test_images = test_images.reshape(10000,784)
     train_images = scaler.fit_transform(train_images)
     test_images = scaler.fit_transform(test_images)
```

```
[12]: logisticReg = LogisticRegression(penalty='l2',tol=0.1,solver='saga',C=0.001)
      train_time_lr = time.time()
      logisticReg.fit(train_images,train_labels)
      train_time_lr = time.time() - train_time_lr
      score_lr = logisticReg.score(test_images,test_labels)
      test_time_lr = time.time()
      predictions_lr = logisticReg.predict(test_images)
      test_time_lr = time.time() - test_time_lr
      print("Score is",score_lr)
      print("Train Time is",train_time_lr)
      print("Test Time is",test_time_lr)
```

```
Score is 0.8247
Train Time is 13.293883562088013
Test Time is 0.026405811309814453
```

```
[13]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier(n_neighbors=3)
      train_time_knn = time.time()
      knn.fit(train_images,train_labels)
      train_time_knn = time.time() - train_time_knn
      score_knn = knn.score(test_images,test_labels)
      test_time_knn = time.time()
      predictions_knn = logisticReg.predict(test_images)
      test_time_knn = time.time() - test_time_knn
      print("Score is",score_knn)
      print("Train Time is",train_time_knn)
      print("Test Time is",test_time_knn)
```

```
Score is 0.8499
Train Time is 14.780428409576416
Test Time is 0.026642799377441406
```

[14]:
```python
from sklearn.svm import SVC
svm_lin = SVC(gamma='auto',kernel='linear', max_iter=50000)
train_time_svmlin = time.time()
svm_lin.fit(train_images,train_labels)
train_time_svmlin = time.time() - train_time_svmlin
score_svmlin = svm_lin.score(test_images,test_labels)
test_time_svmlin = time.time()
predictions_svmlin = logisticReg.predict(test_images)
test_time_svmlin = time.time() - test_time_svmlin
print("Score is",score_svmlin)
print("Train Time is",train_time_svmlin)
print("Test Time is",test_time_svmlin)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:231:
ConvergenceWarning: Solver terminated early (max_iter=50000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

```
Score is 0.7357
Train Time is 1086.3073859214783
Test Time is 0.025488376617431 64
```

[15]:
```python
from sklearn.svm import SVC
svm_rbf = SVC(gamma='auto',kernel='rbf', max_iter=50000)
train_time_rbf = time.time()
svm_rbf.fit(train_images,train_labels)
train_time_rbf = time.time() - train_time_rbf
score_rbf = svm_rbf.score(test_images,test_labels)
test_time_rbf = time.time()
predictions_rbf = logisticReg.predict(test_images)
test_time_rbf = time.time() - test_time_rbf
print("Score is",score_rbf)
print("Train Time is",train_time_rbf)
print("Test Time is",test_time_rbf)
```

```
Score is 0.8836
Train Time is 892.761557340622
Test Time is 0.06520485877990723
```

[16]:
```python
print("Score for Logistic Regression is    ",score_lr)
print("Score for K Nearest Neighbors is    ",score_knn)
print("Score for SVM with Linear Kernel is ",score_svmlin)
print("Score for SVM with RBF Kernel is    ",score_rbf)
```

```
Score for Logistic Regression is     0.8247
Score for K Nearest Neighbors is     0.8499
Score for SVM with Linear Kernel is  0.7357
Score for SVM with RBF Kernel is     0.8836
```

```
[17]: print("Algorithm               Train Time(s)  Test Time(s)")
      print("Logistic Regression   ",round(train_time_lr,2),"         ␣
       ↪",round(test_time_lr,3))
      print("K Nearest Neighbours  ",round(train_time_knn,2),"        ␣
       ↪",round(test_time_knn,3))
      print("SVM Linear Kernel     ",round(train_time_svmlin,2),"     ␣
       ↪",round(test_time_svmlin,3))
      print("SVM RBF Kernel        ",round(train_time_rbf,2),"        ␣
       ↪",round(test_time_rbf,3))
```

```
Algorithm               Train Time(s)  Test Time(s)
Logistic Regression     13.29           0.026
K Nearest Neighbours    14.78           0.027
SVM Linear Kernel       1086.31         0.025
SVM RBF Kernel          892.76         0.065
```

We can see that SVM with RBF kernel gives the highest score. That is because this algorithm is best for treating a dataset with such high dimension rather than others. Although, the training time is one of the highest and testing time is the highest. This is the trade-off we see between time taken and accuracy of prediction.