IE406 Machine Learning

Lab Assignment - 7

Group 28

201801015: Shantanu Tyagi
201801076: Shivani Nandani
201801407: Pratvi Shah
201801408: Arkaprabha Baerjee

Question 1

Cars24 is the most popular website of used vehicles for sale, yet it's very difficult to collect all of them in the same place. Among all cities, data from 5 major cities which include Hyderabad, New Delhi, Mumbai, Bangalore, and Chennai is collected. Develop an algorithm for predict price of car.

Answer

For this dataset we performed preliminary analysis on the data and cleaned the data for making it a good fit for model creation. After that we standardized the data and implemented multiple models to find an algorithm which predicts price of car accurately.

Code

```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
#Read data
df = pd.read_csv('Cars24.csv', index_col=0)
df.head()
df.dropna(inplace=True)

print(len(np.unique(df['Car Brand'])))
print(len(np.unique(df['Model'].astype(str))))
print(len(np.unique(df['Model Year'])))
print(len(np.unique(df['Location'])))
print(len(np.unique(df['Fuel'])))
print(len(np.unique(df['Gear'].astype(str))))

# Categorical to numeric
def getVecForm(vocab, df):
    for i in range(len(vocab)):
        df.replace(vocab[i], i, inplace=True)
    return df

df['Fuel'] = getVecForm(np.unique(df['Fuel']), df['Fuel'])
df['Car Brand'] = getVecForm(np.unique(df['Car Brand']), df['Car Brand'])
df['Model'] = getVecForm(np.unique(df['Model'].astype(str)), df['Model'])
df['Location'] = getVecForm(np.unique(df['Location']), df['Location'])
df['Gear'] = getVecForm(np.unique(df['Gear'].astype(str)), df['Gear'])
print(df)
```

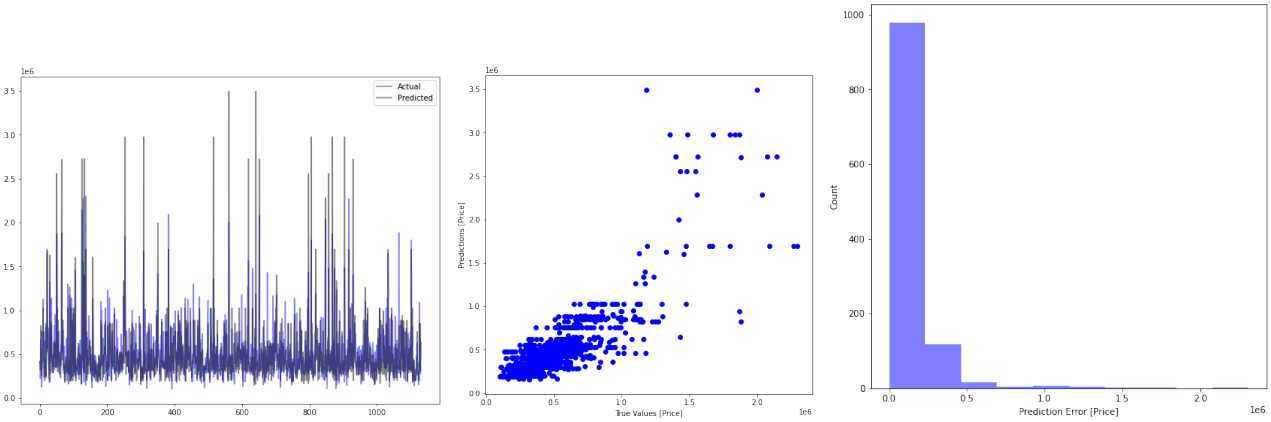
```
39
40 #Final data
41 X = df.drop(['Price'], axis=1).to_numpy()
42 y = df['Price'].to_numpy()
43
44 #Standardize
45 scaler = StandardScaler()
46 X = scaler.fit_transform(X)
47
48 # Logistic Regression
49 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
50 reg = LogisticRegression(random_state=0, solver='liblinear', max_iter=1000).fit(X_train,
       y_train)
51 y_pred = reg.predict(X_test)
52 reg.score(X_test, y_test)
53 print(mean_squared_error(y_test, y_pred, squared=False))
54 plt.figure(figsize=[10,8])
55 plt.plot(y_test, 'b', alpha=0.5, label='Actual')
56 plt.plot(y_pred, 'k', alpha=0.5, label='Predicted')
57 plt.legend()
58 plt.show()
59 plt.figure(figsize=[8,8])
60 plt.scatter(y_test, y_pred, color='b')
61 plt.xlabel('True Values [Price]')
62 plt.ylabel('Predictions [Price]')
63 plt.show()
64 error = abs(y_test-y_pred)
65 plt.figure(figsize=[8,8])
66 plt.hist(error, bins = 10, color='b', alpha=0.5)
67 plt.xlabel("Prediction Error [Price]")
68 plt.ylabel("Count")
69 plt.show()
70
71 # PCA followed by Logistic Regression
72 pca = PCA(n_components=X.shape[1])
73 pca.fit_transform(X)
74 ex_var_ratio = pca.explained_variance_ratio_
75
76 plt.figure(figsize=[8,8])
77 plt.plot(ex_var_ratio, color='b', linestyle='-', marker='x')
78 plt.show()
79 pca = PCA(n_components=3)
80 X_new = pca.fit_transform(X)
81 print(np.sum(pca.explained_variance_ratio_))
82 X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=42)
83
84 reg = LogisticRegression(max_iter=1000).fit(X_train, y_train)
85 y_pred = reg.predict(X_test)
86 reg.score(X_test, y_test)
87 print(mean_squared_error(y_test, y_pred, squared=False))
88 plt.figure(figsize=[8,6])
89 plt.plot(y_test, 'b', alpha=0.5, label='Actual')
90 plt.plot(y_pred, 'k', alpha=0.5, label='Predicted')
91 plt.legend()
92 plt.show()
93 plt.figure(figsize=[8,8])
94 plt.scatter(y_test, y_pred, color='b')
95 plt.xlabel('True Values [Price]')
96 plt.ylabel('Predictions [Price]')
97 plt.show()
98 error = abs(y_test-y_pred)
99 plt.figure(figsize=[8,8])
100 plt.hist(error, bins = 10, color='b', alpha=0.5)
101 plt.xlabel("Prediction Error [Price]")
102 plt.ylabel("Count")
103 plt.show()
104
105 # Stochastic Gradient Descent
106 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
107 n_samples, n_features = X_train.shape[0], X_train.shape[1]
108 reg = make_pipeline(StandardScaler(),
109                     SGDRegressor(max_iter=1000, tol=1e-3))
110 reg.fit(X_train, y_train)
111 y_pred = reg.predict(X_test)
112 reg.score(X_test, y_test)
```

```
plt.figure(figsize=[10,8])
plt.plot(y_test, 'b', alpha=0.5, label='Actual')
plt.plot(y_pred, 'k', alpha=0.5, label='Predicted')
plt.legend()
plt.show()

plt.figure(figsize=[8,8])
plt.scatter(y_test, y_pred, color='b')
plt.xlabel('True Values [Price]')
plt.ylabel('Predictions [Price]')
plt.show()

error = abs(y_test-y_pred)
plt.figure(figsize=[8,8])
plt.hist(error, bins = 10, color='b', alpha=0.5)
plt.xlabel("Prediction Error [Price]")
plt.ylabel("Count")
plt.show()

# Neural Network
def build_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(256, activation='relu', input_shape=[X_train.shape[1]]),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(1)
    ])

    optimizer = tf.keras.optimizers.Adam()


    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mae', 'mse'])
    return model

class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

model = build_model()
history = model.fit(X_train,
                    y_train,
                    epochs=1000, validation_split = 0.2, verbose=0,
                    callbacks=[PrintDot()])

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

def plot_history():
    plt.figure(figsize=[10,8])
    plt.xlabel('Epoch')
    plt.ylabel('Mean Square Error')
    plt.plot(hist.epoch[1:], hist.loss[1:], label='Train Error')
    plt.plot(hist.epoch[1:], hist.val_loss[1:], label = 'Val Error')
    plt.legend()

plot_history()
mse, _, _ = model.evaluate(X_test, y_test)
rmse = np.sqrt(mse)
print('Root Mean Square Error on test set: {}'.format(round(rmse, 3)))
y_pred = model.predict(X_test).flatten()
print(mean_squared_error(y_test, y_pred, squared=False))
plt.figure(figsize=[10,8])
plt.plot(y_test, 'b', alpha=0.5, label='Actual')
plt.plot(y_pred, 'k', alpha=0.5, label='Predicted')
plt.legend()
plt.show()

```

```
plt.figure(figsize=[8,8])
plt.scatter(y_test, y_pred, color='b')
plt.xlabel('True Values [Price]')
plt.ylabel('Predictions [Price]')
plt.show()

error = abs(y_test - y_pred)
plt.figure(figsize=[8,8])
plt.hist(error, bins = 10, color='b', alpha=0.5)
plt.xlabel("Prediction Error [Price]")
plt.ylabel("Count")
plt.show()
```

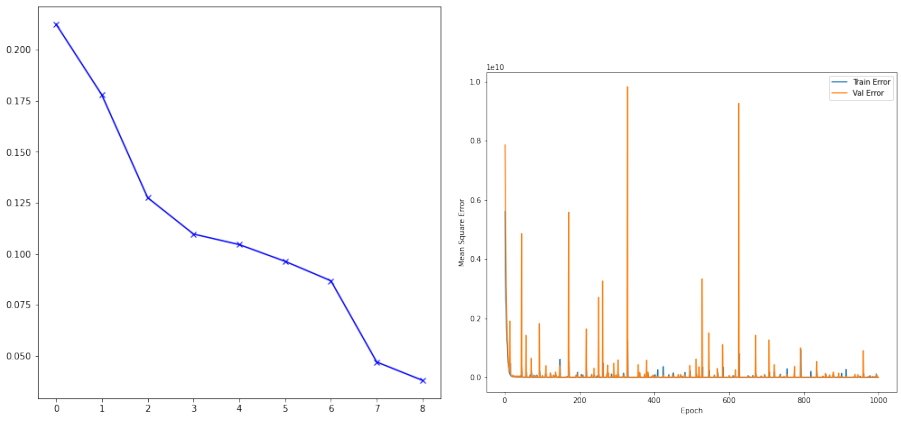
Listing 1: Question 1

Result



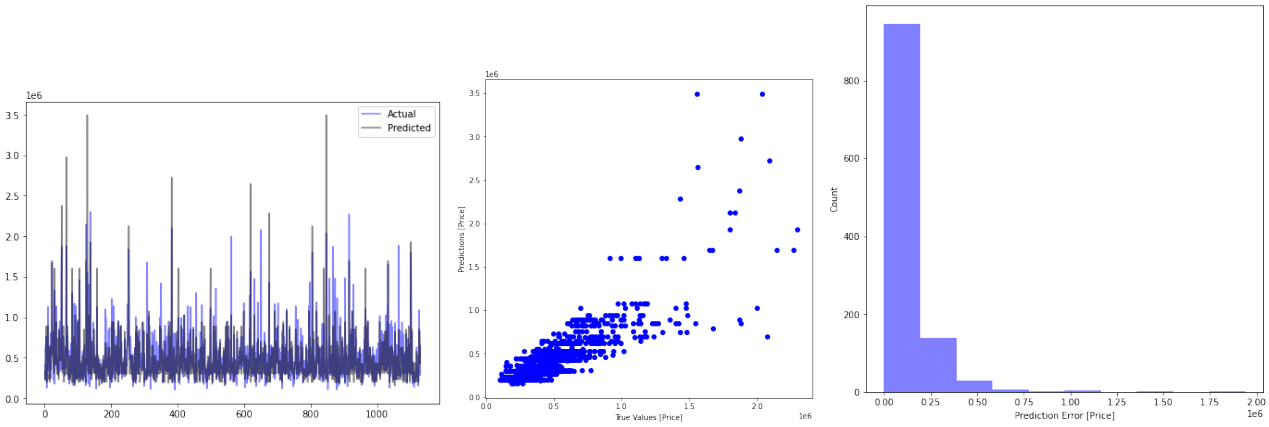
(a) (b) (c)

Figure 1: Logistic Regression



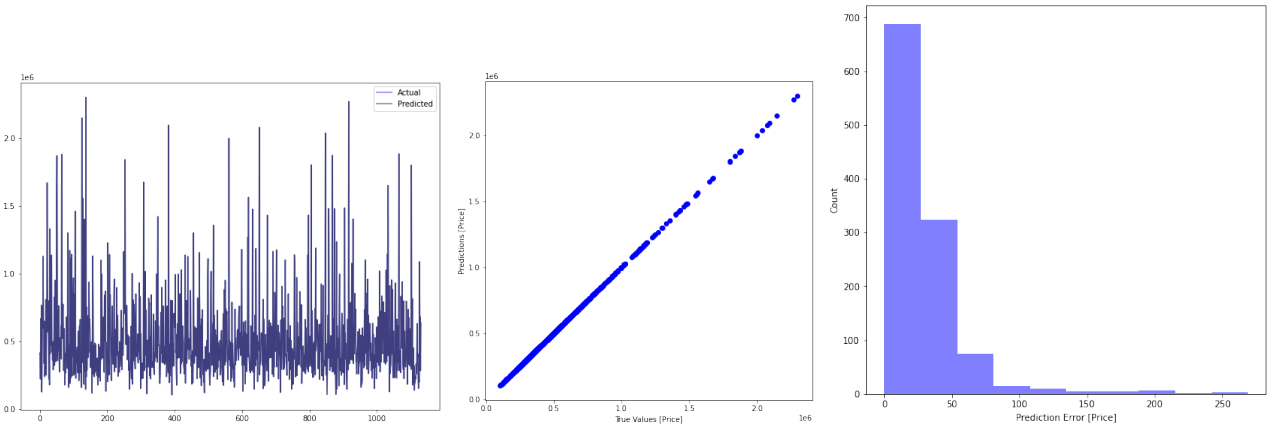
(a) (b)

Figure 2: (a) Elbow curve for PCA (b) MSE v/s Epoch (Model loss plot)



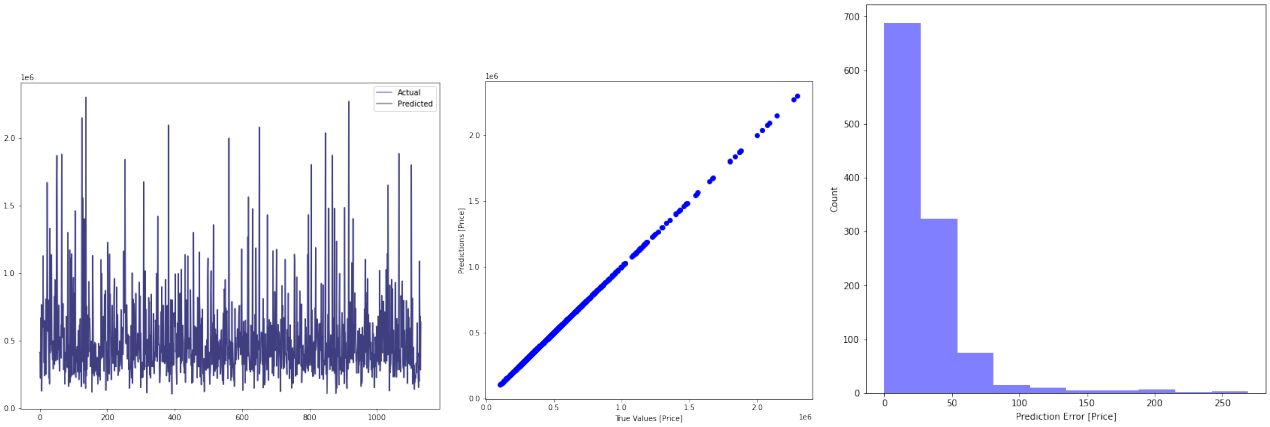
(a) (b) (c)

Figure 3: Logistic Regression post PCA



(a) (b) (c)

Figure 4: Stochastic Gradient Descent



(a) (b) (c)

Figure 5: Neural Network

Observation/ Justification

We can make following observations from above:

1. From Figure.2(a), we can see that we do not get a clear cutoff for number of components that best describes the price and this can be due to the inherent categorical nature of majority of the features.

2. Figure.2(b) shows the Training and Validation set error for each epoch. It can be observed that, on an average, the error for Validation set for each epoch has a decreasing trend. This shows that the model is not over-fitting the data.

3. We could clearly observe from the plot (c) in the above figures that the error in prediction of car price is highest for Logistic regression model with and without PCA.

4. Due to the transformation from categorical to numeric data mean squared error after performing PCA is(= 189427.88) lesser than the original Logistic Regression model(= 223357.67).

5. From plot (b) in the above figures we can see that Stochastic GD and Neural Network models perform much better than original Logistic Regression model evident from the linear line whose slope is almost 1.

Question 2

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. That means one image contains 784 pixel and pixel-value is an integer between 0 and 255. Make a classification model to classify the product.

Answer

The following classification models have been used :

1. Logistic Regression
2. Linear Discriminant Analysis
3. SVM
   (a) Linear Kernel
   (b) Polynomial Kernel
4. Neural Network architecture with 512 neurons in dense layer and 10 nodes in output layer. The Dense layer uses relu activation function and the final layer employs the softmax function. An adam optimizer has been used along categorical cross-entropy loss function.

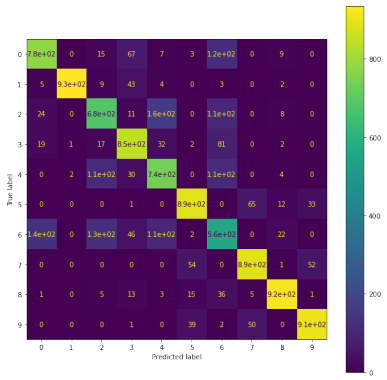
Code

```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn import svm
import tensorflow as tf
import keras as keras

# Load data
training_images = pd.read_csv('fashion-mnist_train.csv')
training_labels = training_images['label']
training_images = training_images[training_images.columns[1:]]
test_images = pd.read_csv('fashion-mnist_test.csv')
test_labels = test_images['label']
test_images = test_images[test_images.columns[1:]]
training_images=StandardScaler().fit_transform(training_images)
test_images=StandardScaler().fit_transform(test_images)

# NN architecture with 512 neurons and 10 output nodes.
# Dense layer uses relu activation function and the final layer employs the softmax function.
    An adam optimizer has been used.
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),tf.keras.layers.Dense(512,
    activation=tf.nn.relu), tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
model.compile(optimizer = tf.optimizers.Adam(),
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=10)

model.evaluate(test_images, test_labels)
print(model.metrics_names)

# SVM
def plot_confusion_matrix(title,y_test,y_pred):
```

```
    confusionMatrix = confusion_matrix(y_test,y_pred)
    sns.heatmap(confusionMatrix,annot=True, fmt='.4g')
    plt.ylabel('True',fontsize=15)
    plt.xlabel('Predicted',fontsize=15)
    plt.title(title,fontsize=15)
    plt.show()
    print(classification_report(y_test,y_pred))


def svm_func(x_train, x_test, y_train, y_test):
    k = ['linear', 'poly']

    for i in range(len(k)):
        clf = svm.SVC(kernel=k[i])
        clf.fit(x_train, y_train)
        y_pred = clf.predict(x_test)
        plot_confusion_matrix(k[i]+' kernel', y_test, y_pred)

svm_func(training_images, test_images, training_labels, test_labels)

# Logistic regression
clf = LogisticRegression(max_iter=10000).fit(training_images, training_labels)
accuracy_inbuilt = accuracy_score(test_labels, clf.predict(test_images))

cm = confusion_matrix(test_labels, pred_data)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
print(classification_report(test_labels,pred_data))

# LDA
clf = LinearDiscriminantAnalysis()
clf.fit(training_images, training_labels)
pred_data = clf.predict(test_images)

cm = confusion_matrix(test_labels, pred_data)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
print(classification_report(test_labels,pred_data))
```

Listing 2: Question 2

Result

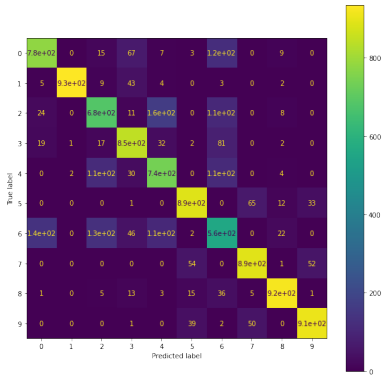


(a)

	precision	recall	f1-score	support
0	0.81	0.78	0.79	1000
1	1.00	0.93	0.96	1000
2	0.70	0.68	0.69	1000
3	0.80	0.85	0.82	1000
4	0.70	0.74	0.72	1000
5	0.89	0.89	0.89	1000
6	0.55	0.56	0.55	1000
7	0.88	0.89	0.89	1000
8	0.94	0.92	0.93	1000
9	0.91	0.91	0.91	1000
accuracy			0.81	10000
macro avg	0.82	0.82	0.82	10000
weighted avg	0.82	0.81	0.82	10000

(b)

Figure 6: Logistic Regression

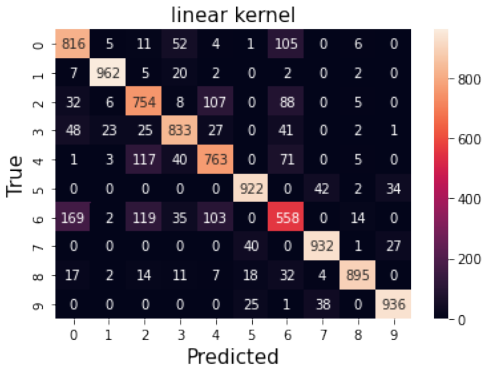


(a)

	precision	recall	f1-score	support
0	0.81	0.78	0.79	1000
1	1.00	0.93	0.96	1000
2	0.70	0.68	0.69	1000
3	0.80	0.85	0.82	1000
4	0.70	0.74	0.72	1000
5	0.89	0.89	0.89	1000
6	0.55	0.56	0.55	1000
7	0.88	0.89	0.89	1000
8	0.94	0.92	0.93	1000
9	0.91	0.91	0.91	1000
accuracy			0.81	10000
macro avg	0.82	0.82	0.82	10000
weighted avg	0.82	0.81	0.82	10000

(b)

Figure 7: Linear Discriminant Analysis

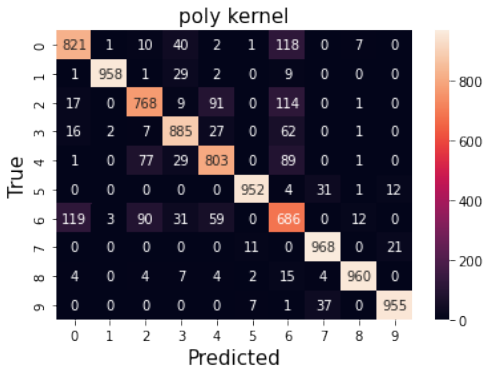


(a)

	precision	recall	f1-score	support
0	0.75	0.82	0.78	1000
1	0.96	0.96	0.96	1000
2	0.72	0.75	0.74	1000
3	0.83	0.83	0.83	1000
4	0.75	0.76	0.76	1000
5	0.92	0.92	0.92	1000
6	0.62	0.56	0.59	1000
7	0.92	0.93	0.92	1000
8	0.96	0.90	0.93	1000
9	0.94	0.94	0.94	1000
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

(b)

Figure 8: SVM Linear kernel



(a)

	precision	recall	f1-score	support
0	0.84	0.82	0.83	1000
1	0.99	0.96	0.98	1000
2	0.80	0.77	0.78	1000
3	0.86	0.89	0.87	1000
4	0.81	0.80	0.81	1000
5	0.98	0.95	0.97	1000
6	0.62	0.69	0.65	1000
7	0.93	0.97	0.95	1000
8	0.98	0.96	0.97	1000
9	0.97	0.95	0.96	1000
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

(b)

Figure 9: SVM Polynomial kernel

Model	Accuracy (Testing Dataset)
Logistic Regression	0.81
LDA	0.81
Neural Network	0.8721
SVM(linear kernel)	0.84
SVM(poly kernel)	0.88

Table 1: accuracy

Observation/ Justification

We could draw the following observations:

- From the above results we can see that polynomial kernel for support vector machine performs the best out of all other methods.
- The Neural network achieves a saturation point for its accuracy within 10 epochs.
- Logistic Regression and LDA perform the worst among all models chosen.
- Shirt (Number 6) has often been confused with T-shirt/top (Number 0), Pullover (Number 2) and Coat (Number 4).
- Coat (Number 4) has also often been confused with a pullover (Number 2).