1 CSCC11 - Introduction to Machine Learning, Fall 2022, Assignment 1

1.1 Authors

Shawn Santhoshgeorge (1006094673)
Anaqi Amir Razif (1005813880)

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import statistics
from scipy import stats
```

[2]:
```
#TO-DO
"""
Read the csv file into a DataFrame - df
"""
df = pd.read_csv('Admission_Predict.csv')
```

[3]:
```
"""
Print the DataFrame
"""
df
```

[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
..	...	...	...	...	...	...	...	...	...
395	396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	333	117	4	5.0	4.0	9.66	1	0.95

[400 rows x 9 columns]

[4]:
```
#TO-DO
"""
Print the length of the DataFrame.
Print the column names of the DataFrame.
"""

print("Length of df: ", len(df))
print("Column Names of df: ", list(df.columns))
```

```
Length of df:  400
Column Names of df:  ['Serial No.', 'GRE Score', 'TOEFL Score', 'University
Rating', 'SOP', 'LOR ', 'CGPA', 'Research', 'Chance of Admit ']
```

[5]:
```
#TO-DO
"""
Define an “X” array that would hold our independent features for regression purposes.
Define a "Y" array that would hold our target variable.

Print the shape of both the arrays.
"""

X = df[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research']]
Y = df['Chance of Admit ']

print("Shape of X: ", X.shape)
print("Shape of Y: ", Y.shape)
```

```
Shape of X:  (400, 7)
Shape of Y:  (400,)
```

1.2 Split the data

[6]:
```
#TO-DO
"""
Split the dataset into train dataset and test dataset.
Set the random state to any number in order to maintain consistency while generating random numbers over several runs.
"""
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, random_state=69)
```

1.3 Linear Regression

[7]:
```
#TO-DO
def find_optimal_parameters(x, y):
    """ Compute closed form solution for linear regression!
    Optimal weight w* in linear regression is given by w* = (X^T X)^(-1) X^T Y

    Args:
    - x (ndarray (Shape: (N, D))): A NxD matrix corresponding to the inputs.
    - y (ndarray (Shape: (N, 1))): A N-column vector corresponding to the outputs given the inputs.

    Output:
    - w (ndarray (Shape: (D+1, 1))): A (D+1)x1 column vector corresponding to the bias and weights of the linear model.
    """
    # Pad 1's for the bias term, Why? Used for the bias
    pad_x = np.hstack((np.ones((x.shape[0], 1)), x))

    # Note that we could use pseudoinverse here instead: np.linalg.pinv
    # @ is alias for matmul
    p1 = np.linalg.pinv(np.matrix.transpose(pad_x) @ pad_x) # (X^T X)^(-1)
    p2 = np.matrix.transpose(pad_x) @ y # X^T Y
    w = p1 @ p2
    return w
```

1.3.1 Train linear regression model using training data

[8]:
```
#TO-DO
def get_pred_Y(trained_w, X_pred):
    """ Return predicted Y
    Args:
    - trained_w (ndarray (Shape: (D+1, 1))): A (D+1)x1 column vector containing linear regression weights.
    - X_pred (ndarray (Shape: (N, D))): A NxD matrix corresponding to the prediction inputs.

    Output:
    - pred_Y (ndarray (Shape: (N, 1))): A Nx1 column vector corresponding to the predicted outputs.
    """
    # Pad 1's for the bias term
    pad_x = np.hstack((np.ones((X_pred.shape[0], 1)), X_pred))

    pred_Y = pad_x @ trained_w
    return pred_Y
```

Define these metrics and discuss why one would be preferred over the other ?

The Mean Absolute Error (MAE) is defined as the following $\text{MAE} = \frac{\sum_{i=1}^{N} |y_i - f(x_i)|}{N}$ and the Mean Squared Error (MSE) is defined as the following $\text{MSE} = \frac{\sum_{i=1}^{N} (y_i - f(x_i))^2}{N}$. The MAE is the average absolute error between the actual and predicted values and the MSE is the average squared error between the actual and predicted values. They both can be used to get an overall performance of the model compared to the dataset but, MSE is preferred over MAE since it helps to point out large errors to a greater extent since it squares the error value.

[9]:
```
#TO-DO
def get_mae(Y_truth, Y_pred):
    """ Return Mean absolute error
    Args:
    - Y_truth (ndarray (Shape: (N, 1))): A Nx1 column vector corresponding to the actual outputs.
    - Y_pred (ndarray (Shape: (N, 1))): A Nx1 column vector corresponding to the predicted outputs.

    Output:
    - MAE (ndarray (Shape: (1,))).
    """

    'check if both inputs are of the same shape'
    assert Y_truth.shape == Y_pred.shape, f"Number of Actual should equal the Number of Predicted Outputs, but {Y_truth.shape} !=␣
↪{Y_pred.shape}"

    return np.mean(np.absolute(Y_truth - Y_pred))

def get_mse(Y_truth, Y_pred):
    """ Return Mean squared error
    Args:
    - Y_truth (ndarray (Shape: (N, 1))): A Nx1 column vector corresponding to the actual outputs.
    - Y_pred (ndarray (Shape: (N, 1))): A Nx1 column vector corresponding to the predicted outputs.

    Output:
    - MSE (ndarray (Shape: (1,))).
    """

    'check if both inputs are of the same shape'
    assert Y_truth.shape == Y_pred.shape, f"Number of Actual should equal the Number of Predicted Outputs, but {Y_truth.shape} !=␣
↪{Y_pred.shape}"

    return np.mean(np.square(Y_truth - Y_pred))
```

1.3.2 Get predictions on train data

[10]:
```
w_optimal = find_optimal_parameters(X_train, Y_train)
print(w_optimal)
```

```
[-1.13911877  0.00129245  0.00299385  0.00304951  0.00153358  0.01990276
  0.12032817  0.03328811]
```

[11]:
```
pred_Y = get_pred_Y(w_optimal, X_train)
print('Train Error (MSE): ', get_mse(Y_train.to_numpy(), pred_Y))
print('Train Error (MAE): ', get_mae(Y_train.to_numpy(), pred_Y))
```

```
Train Error (MSE):  0.004147808502232658
Train Error (MAE):  0.045728954899761275
```

1.3.3 Get predictions and performance on test data

[12]:
```
pred_Y = get_pred_Y(w_optimal, X_test)
print('Test Error (MSE): ', get_mse(Y_test.to_numpy(), pred_Y))
print('Test Error (MAE): ', get_mae(Y_test.to_numpy(), pred_Y))
```

```
Test Error (MSE):  0.003748193147899049
Test Error (MAE):  0.04226151047842842
```

Report the corresponding MAE and MSE values

The Train Error for MAE and MSE is approximately as follows:

MSE	MAE
0.004147808502232658	0.045728954899761275

The Test Error for MAE and MSE is approximately as follows:

MSE	MAE
0.003748193147899049	0.04226151047842842

1.4 Silouette Coefficient

[13]:
```
## TO-DO
n_silhouette = []

kmeans_kwargs= {
    "init":"k-means++",
    "n_init":30,
    "max_iter":250,
    "random_state":2
}

"""
Perform the following steps:

1. Loop over the various possible K values you wish to test
2. Initialize a K means object.
3. Fit the training data on the K means object.
4. Use the silhouette score method available from the sklearn metrics.
5. Append the score to the silhouetter_coefficients list.
6. Display the the silhouette coefficient associated with each value of K.
"""

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    cluster_labels = kmeans.fit_predict(X_train, Y_train)
    silhouette_avg = silhouette_score(X_train, cluster_labels)
    n_silhouette.append(silhouette_avg)
    print(f"For K = {k}. The Silhouette Score is: {silhouette_avg}")
```

```
For K = 2. The Silhouette Score is: 0.523697003020886
For K = 3. The Silhouette Score is: 0.46196799620849327
For K = 4. The Silhouette Score is: 0.46609629756368104
For K = 5. The Silhouette Score is: 0.4114277091098109
For K = 6. The Silhouette Score is: 0.40338552975455844
For K = 7. The Silhouette Score is: 0.3847943002802233
For K = 8. The Silhouette Score is: 0.3439602276947788
For K = 9. The Silhouette Score is: 0.3374920312014737
For K = 10. The Silhouette Score is: 0.32371773649768576
```

For values of K $\in$ [2, 10]. Which value would be the most appropriate ?

From above we can see that the highest value resulting from the Silhouette coefficient analysis is approximately 0.5237 for K = 2. So the most appropriate value would be K = 2.

2 K Means

[14]:
```
#TO-DO
# Set the number of clusters based on the silhouette coefficient analysis
N_CLUSTERS = 2

kmeans = KMeans(
    init="k-means++",
    n_clusters=N_CLUSTERS , #Input the value you configured using the Silhouette coefficient analysis.
    n_init=30,
    max_iter=250,
    random_state=2
)

#TO-DO
# Fit to the training data
kmeans.fit(X_train.to_numpy(), Y_train.to_numpy())

#TO-DO
# Add the features and the training data you used to the variable below.
training_df_clustered = X_train.assign(cluster=kmeans.labels_)

#TO-DO
# Predict clusters for the training data
train_cluster = kmeans.predict(X_train.to_numpy())

#TO-DO
# Add the target and predicted clusters to the training DataFrame
training_df_clustered['cluster'] = train_cluster

X_train_clusters_df = []
for i in range(N_CLUSTERS):
    X_train_clusters_df.append(training_df_clustered[training_df_clustered['cluster']==i])
```

3 Building Linear Regression for our clusters

[15]:
```
from sklearn.linear_model import LinearRegression

"""
The number of clusters would be defined by the outcome of the silhouetter coefficient
Set up the model of Linear Regression by exploring the different parameters: https://scikit-learn.org/stable/modules/generated/
↪sklearn.linear_model.LinearRegression.html
train_clusters_df is a dataframe that contains both the true cluster values and the predicted cluster values. Feel free to change␣
↪the variable name to something else if you have been following a different naming convention.
"""

obj_cluster = []

for i in range(N_CLUSTERS):
    #TO-DO
    # Initialize a Linear Regression object.
    reg_model = LinearRegression()
    #Get the specific X_train values according to their predicted clusters.
    X_clustered_data = X_train_clusters_df[i].drop(columns=['cluster'])
    #Get the specific Y_train values according to their predicted clusters.
    Y_clustered_data = Y_train[X_clustered_data.index]
    obj_cluster.append(reg_model.fit(X_clustered_data.to_numpy(), Y_clustered_data.to_numpy()))
```

[16]:
```
def predict_value(x_test, kmeans, cluster_linear):
  """
  Input:
  x_test is the test value that you wish to predict on.
  kmeans is the kmeans object that you have finalized to predict on the test dataset.
  cluster_linear is the list of fitted models on different clusters.

  Return:
  linear_pred - linear_pred will be type list with prediction values
  clusters - clusters_pred will be the prediction of clusters using k means.

  Follow these steps:
  1. Predict clusters using K means object on the test data.
  2. Predict regression values using Linear Regression list.
  3. return both the predictions.

  """
  clusters = []
  linear_pred = []
  for index, row in x_test.iterrows():
    value = [row]
    cluster_label = int(kmeans.predict(value))
    chance_to_admit = float(cluster_linear[cluster_label].predict(value))
    linear_pred.append(chance_to_admit)
  return np.asarray(linear_pred, dtype=float)
```

4 Final Steps

[17]:
```
#Apply the clustering-based linear regression to the test set.
Y_svr_k_means_pred = predict_value(X_test, kmeans, obj_cluster)
```

[18]:
```
print('Test Error (MSE): ', get_mse(Y_test.to_numpy(), Y_svr_k_means_pred))
print('Test Error (MAE): ', get_mae(Y_test.to_numpy(), Y_svr_k_means_pred))
```

```
Test Error (MSE):  0.003594171024942518
Test Error (MAE):  0.04166689615330025
```

Report the corresponding MAE and MSE values The Test Error for MAE and MSE is as follows:

MSE	MAE
0.003594171024942518	0.04166689615330026

In the previous model, we assumed that all of the students belonged to one group, but after the Silhouette Coefficient Analysis we noticed that there are 2 clusters. After splitting then using Kmeans++, and then implementing Linear Regression based on the 2 clusters we see that the MSE and MAE have both drastically improved from the previous value.

Provide a brief discussion regarding the factors that might have contributed to this result Most master programs are split into 2 types, one for those interested in Research and one for those who would like to take more advanced courses. So it can be said that there are 2 types of applicants with different interests. Thus we can see the 2 distinct clusters from the Silhouette Coefficient Analysis. Comparing the Linear Regression Coefficients of each Cluster will be very clear.

Cluster	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	-0.0001575616593	0.004520914248	0.008651619805	0.014163505258	0.00825716335	0.10094909980	0.06810396677
1	0.001835465193	0.003454110966	-0.0037986638654	-0.004936007238	0.027089841781	0.13311695689	0.01719038474

As we can see that there are many differences between the Linear Regression Coefficients for each cluster.
Cluster 0: Favours TOFEL Score, University Rating, SOP, Research
Cluster 1: Favours GRE Score, LOR, CGPA

Looking at what each cluster favours we can conclude that what each cluster represents
Cluster 0: Students who have done Research with a Strong Statment of Purpose - **Research Oriented Masters Programs**
Cluster 1: Students who have done well in Academic Courses with Strong Letter of Recommendation - **Course Based Masters Programs**