DATA MINING HOMEWORK 1 – HEM REGMI (Y40015550)

To run the program, we need to install all the packages in: environment.yml file. The details to download and run project is given in: https://github.com/hregmi77/DataMiningHW1

Problem 1: It asks us to fill in the data where fields are missing. We have different options for imputing as the strategy. Following procedures are used to fill in data:

a. First read the csv data using pandas.
b. Then find out which columns are missing values, they are put as "NaN" values during csv read so we can look for invalid or missing values in the column to figure out.
c. Found out following features are missing values: ["Car", "BuildingArea", "Year Built", "Council Area"].
d. We use SimpleImputer from scikit-learn package to fill the missing values. Since the data contains the number as well as strings, we used the "most frequent" strategy, even though there are other strategies like mean, median, constant also. Additionally, there are other imputers like KNImputer, IterativeImputer etc.
e. Since SimpleImputer is flexible for number and string both with "most frequent" strategy, I used that strategy in my implementation.

```
original_data = pd.read_csv('melb_data.csv')
# Columns are: ["Suburb", "Address", "Room", "Type", "Price", "Method",
#               "SellerG", "Date", "Distance", "PostCode", "Bedroom2", "Bedroom", "Car",
#               "Landsize", "BuildingArea", "Year Built", "CouncilArea",
#               "Lattitude", "Longitude", "Regionname", "PropertyCount"]
y = original_data.iloc[:,16].values
y = np.expand_dims(y, axis=-1)
mycommuter = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
original_data.iloc[:,16] = mycommuter.fit_transform(y)
y = original_data.iloc[:,15].values
y = np.expand_dims(y, axis=-1)
original_data.iloc[:,15] = mycommuter.fit_transform(y)
y = original_data.iloc[:,14].values
y = np.expand_dims(y, axis=-1)
original_data.iloc[:,14] = mycommuter.fit_transform(y)

y = original_data.iloc[:,12].values
y = np.expand_dims(y, axis=-1)
original_data.iloc[:,12] = mycommuter.fit_transform(y)

processed_data = original_data
```

Problem 2: Replacing Categorical and Nominal Values with One-Hot encoding. Following process is followed to achieve that result.

a. After we fill the missing values, we need to encode the categorical and nominal features to make it suitable for classification.

b. Following are the features which has categorical data and needed to convert to one-hot code: ['Suburb', 'Type', 'Method', 'SellerG', 'CouncilArea', 'Regionname']
c. We dropped the features ["Address", "Date"] since it is difficult to encode them.
d. These features are encoded by using one-hot encoder from the category_encoders package.
e. We also sorted the price values and divided them equally into five classes from ["Top Value", "High Value", "medium value", "low value", "bottom value"] into 0 to 4 values.
f. Now we have features and classes that can be used in learning and prediction. After encoding, we have total 643 features.

```
# Dropping "Address" and "Date" From Features
original_data = original_data.sort_values(by=['Price'])
x = original_data.drop(['Address', 'Date', 'Price'], axis=1)
# Converting Price in to 5 classes: 0 to 4
y = original_data['Price']
interval = y.shape[0] // 5
y[0:interval] = 0
y[interval:2*interval] = 1
y[2*interval:3*interval] = 2
y[3*interval:4*interval] = 3
y[4*interval:5*interval] = 4

ce_onehot_code = ce.OneHotEncoder(cols=['Suburb', 'Type', 'Method', 'SellerG', 'CouncilArea', 'Regionname'])
x = ce_onehot_code.fit_transform(x, y)
# print(x.shape[1])
newdataframe = x.join(y)
# print(newdataframe.shape[1])
#newdataframe.to_csv("Encoded_Data.csv", encoding='utf-8', index=False)
Encoded_data = newdataframe
```

Problem 3:

In this problem, we used the scikit-learn beside weka to find the optimal value of K and corresponding test accuracy. We followed the following process to achieve that.

a. The encoder features are first normalized using MinMaxScalar from the scikit-learn package. I also tried other StandardScalar, but MinMaxScalar performed better.
b. We then divide the dataset into 75% train, 10% validation, 15% test using train_test_split() from scikit-learn package. We first divided data into train 85% and test 15%. Then, that 85% of train data is further divided into train and validation to achieve 75% train, 10% validation, and 15% test.
c. We used the KNeighborsClassifier from scikit-learn with value of K ranging from 5 to 10 to find the accuracy in validation dataset. We found that K=9 gives the best validation accuracy of 57.72% .
d. We selected K=9, to form classifier and trained on the train data and tested with test data and found accuracy of 56.84%.

Code:

```
############################ Solution of Q.N.3 ################################################
original_data = Encoded_data
classes = np.array(original_data['Price'])
features = original_data.drop('Price', axis=1)
feature_name = list(features.columns)
features = np.array(features)
min_max_scaler = MinMaxScaler()
features = min_max_scaler.fit_transform(features)

# Splitting data between train, validation and test
# (Train + Validation: 85%, Test: 15%)
X_train, X_test, y_train, y_test = train_test_split(features, classes, test_size=0.15, random_state=0)
# Now separation between train and validation: 75% and 10% --> 88%, 12%
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.12, random_state=0)

# Implementing KNN and use Validation Data to get optimal depth from 5 - 10
k_values = [5, 6, 7, 8, 9, 10]
optimal_k = 5
prev_accuracy = 0
ValAcc = []
```

```
for k_value in k_values:
    clf = KNeighborsClassifier(n_neighbors=k_value)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_val)
    Total = y_pred.shape[0]
    count = 0
    for idx in range(Total):
        if y_pred[idx] == y_val[idx]:
            count = count + 1
    accuracy = (count/Total)*100
    ValAcc.append([k_value, accuracy])
    if prev_accuracy < accuracy:
        prev_accuracy = accuracy
        optimal_k = k_value
# Use the optimal depth to form Random Forest Classifier and Test on Test Data
clf = KNeighborsClassifier(n_neighbors=optimal_k)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
Total = y_pred.shape[0]
count = 0
for idx in range(Total):
    if y_pred[idx] == y_test[idx]:
        count = count + 1
accuracy = (count/Total)*100
# Print Results

df_k = pd.DataFrame(ValAcc, columns=['K Value', 'Accuracy'])
print('Validation with Different K Values')
print(df_k)
print('Optimal K Found: '+ str(optimal_k))
print('Testing Accuracy with optimal K is', str(accuracy) + ' %')
#############################################################################################
```

Result:

```
   K Value   Accuracy
0        5  55.122655
1        6  54.834055
2        7  55.916306
3        8  56.349206
4        9  57.720058
5       10  57.287157
Optimal K Found: 9
Testing Accuracy with optimal K is 56.84830633284241 %
```

Problem 4:

For RandomForest for the encoded data. We followed the similar process. In this case we selected different depth of tree as the parameter to select the best performing tree on validation data. We followed the following process to get the result.

a. The encoder features are first normalized using MinMaxScalar from the scikit-learn package. I also tried other StandardScalar, but MinMaxScalar performed better.
b. We then divide the dataset into 75% train, 10% validation, 15% test using train_test_split() from scikit-learn package. We first divided data into train 85% and test 15%. Then, that 85% of train data is further divided into train and validation to achieve 75% train, 10% validation, and 15% test.
c. We used the RandomForestClassifier from scikit-learn ensemble with value of depth ranging from 5 to 10 to find the accuracy in validation dataset. We found that depth = 10 gives the best validation accuracy of 59.45% .
d. We selected depth = 10, to form classifier and trained on the train data and tested with test data and found accuracy of 60.53%.

Code:

```
######################### Solution of Q.N.4 ##############################################
original_data = Encoded_data
classes = np.array(original_data['Price'])
features = original_data.drop('Price', axis=1)
feature_name = list(features.columns)
features = np.array(features)
min_max_scaler = MinMaxScaler()
features = min_max_scaler.fit_transform(features)

# Splitting data between train, validation and test
# (Train + Validation: 85%, Test: 15%)
X_train, X_test, y_train, y_test = train_test_split(features, classes, test_size=0.15, random_state=0)
# Now separation between train and validation: 75% and 10% --> 88%, 12%
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.12, random_state=0)

# Implementing Random Forest Classifier and use Validation Data to get optimal depth from 5 - 10
depth_values = [5, 6, 7, 8, 9, 10]
optimal_depth = 5
prev_accuracy = 0
ValAcc = []
```

```
for depth_value in depth_values:
    clf = RandomForestClassifier(max_depth=depth_value, random_state=0, verbose=0)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_val)
    Total = y_pred.shape[0]
    count = 0
    for idx in range(Total):
        if y_pred[idx] == y_val[idx]:
            count = count + 1
    accuracy = (count/Total)*100
    ValAcc.append([depth_value, accuracy])
    if prev_accuracy < accuracy:
        prev_accuracy = accuracy
        optimal_depth = depth_value
# Use the optimal depth to form Random Forest Classifier and Test on Test Data
clf = RandomForestClassifier(max_depth=optimal_depth, random_state=0, verbose=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
Total = y_pred.shape[0]
count = 0
for idx in range(Total):
    if y_pred[idx] == y_test[idx]:
        count = count + 1
accuracy = (count/Total)*100
# Print Results

df = pd.DataFrame(ValAcc, columns=['Depth', 'Accuracy'])
print('Validation with Different Depth Value')
print(df)
print('Optimal Depth Found: '+ str(optimal_depth))
print('Testing Accuracy with optimal depth is', str(accuracy) + ' %')
```

Result:

```
   Depth   Accuracy
0      5  52.020202
1      6  54.401154
2      7  54.906205
3      8  56.349206
4      9  58.585859
5     10  59.451659
Optimal Depth Found: 10
Testing Accuracy with optimal depth is 60.53019145802651 %
```