Machine Learning Lab – 20ISL68A

Program 7 - Implement a program in python to illustrate the Bias Variance Trade-off in a machine learning model.

Step 1: Importing Libraries

```
import numpy as np # linear algebra
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
```

Step 2: Loading the Dataset

```
data_file_path = 'diabetes.csv'
data_df = pd.read_csv(data_file_path)
data_df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Step 3: Considering required features

```
y = data_df["Outcome"].values
x = data_df.drop(["Outcome"],axis=1)
```

Step 4: Dividing data into training and testing data

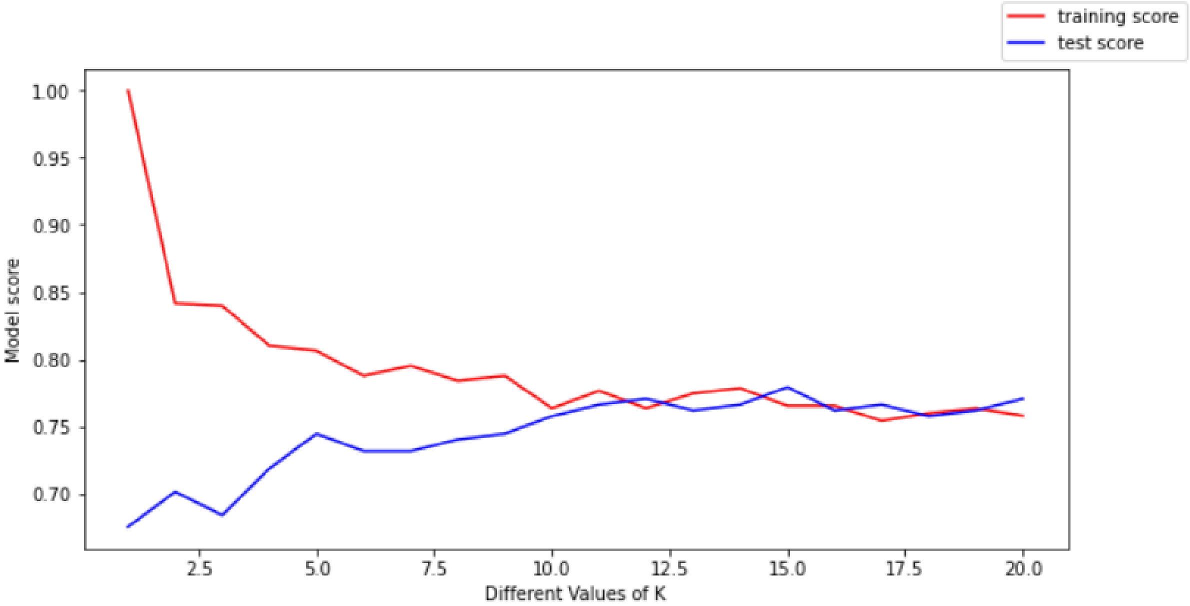
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
data_df = ss.fit_transform(data_df)

#Divide into training and test data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3) # 70% training and 30% test
```

Step 5: Function for Classifier

```
train_score = []
test_score = []
k_vals = []

for k in range(1, 21):
    k_vals.append(k)
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)

    tr_score = knn.score(X_train, y_train)
    train_score.append(tr_score)

    te_score = knn.score(X_test, y_test)
    test_score.append(te_score)
```

Step 7: Plotting the graph

```
plt.figure(figsize=(10,5))
plt.xlabel('Different Values of K')
plt.ylabel('Model score')
plt.plot(k_vals, train_score, color = 'r', label = "training score")
plt.plot(k_vals, test_score, color = 'b', label = 'test score')
plt.legend(bbox_to_anchor=(1, 1),
           bbox_transform=plt.gcf().transFigure)
plt.show()
```



Step 8: Displaying the score

```
knn = KNeighborsClassifier(n_neighbors = 14)

#Fit the model
knn.fit(X_train,y_train)

#get the score
knn.score(X_test,y_test)
```

```
0.7662337662337663
```

Machine Learning Lab – 20ISL68A

Program 8 - Implement and demonstrate the Association Rule Mining using Apriori Algorithm.

Step 1: Importing Libraries

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Step 2: Creating the Dataset

```
dataset = [
    ["Apple", "Mango", "Grapes"],
    ["Banana", "Mango", "Pineapple"],
    ["Apple", "Banana", "Mango", "Pineapple"],
    ["Banana", "Pineapple"],
    ["Apple", "Mango", "Pineapple"],
]
```

Step 3: Printing Dataset

```
print(dataset)
```

```
[['Apple', 'Mango', 'Grapes'], ['Banana', 'Mango', 'Pineapple'], ['Apple', 'Banana', 'Mango', 'Pineapple'], ['Banana', 'Pineapp
le'], ['Apple', 'Mango', 'Pineapple']]
```

Step 4: Converting the Dataset to Boolean

```
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array, columns=te.columns_)
print(df)
```

```
   Apple  Banana  Grapes  Mango  Pineapple
0   True   False    True   True      False
1  False    True   False   True       True
2   True    True   False   True       True
3  False    True   False  False       True
4   True   False   False   True       True
```

Step 5: Creating Frequent Itemset

```
frequent_itemsets_ap = apriori(df, min_support=0.3, use_colnames=True)
print(frequent_itemsets_ap)
```

```
    support                     itemsets
0       0.6                      (Apple)
1       0.6                     (Banana)
2       0.8                      (Mango)
3       0.8                  (Pineapple)
4       0.6               (Apple, Mango)
5       0.4           (Apple, Pineapple)
6       0.4              (Mango, Banana)
7       0.6          (Pineapple, Banana)
8       0.6           (Mango, Pineapple)
9       0.4    (Apple, Mango, Pineapple)
10      0.4   (Mango, Pineapple, Banana)
```

Step 6: Printing the Rules

```
rules_ap = association_rules(frequent_itemsets_ap, metric="confidence", min_threshold=0.61)
print(rules_ap)
```

```
            antecedents          consequents
0               (Apple)              (Mango)
1               (Mango)              (Apple)
2               (Apple)          (Pineapple)
3              (Banana)              (Mango)
4           (Pineapple)             (Banana)
5              (Banana)          (Pineapple)
6               (Mango)          (Pineapple)
7           (Pineapple)              (Mango)
8        (Apple, Mango)          (Pineapple)
9    (Apple, Pineapple)              (Mango)
10   (Mango, Pineapple)              (Apple)
11              (Apple)   (Mango, Pineapple)
12   (Mango, Pineapple)             (Banana)
13      (Mango, Banana)          (Pineapple)
14  (Pineapple, Banana)              (Mango)
15             (Banana)   (Mango, Pineapple)
```

Machine Learning Lab – 20ISL68A

Program 9 - Implement and demonstrate the Association Rule Mining using FP-Growth Algorithm.

Step 1: Importing Libraries

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules
```

Step 2: Creating the Dataset

```
dataset = [
    ["Apple", "Mango", "Grapes"],
    ["Banana", "Mango", "Pineapple"],
    ["Apple", "Banana", "Mango", "Pineapple"],
    ["Banana", "Pineapple"],
    ["Apple", "Mango", "Pineapple"],
]
```

Step 3: Printing Dataset

```
print(dataset)
```

```
[['Apple', 'Mango', 'Grapes'], ['Banana', 'Mango', 'Pineapple'], ['Apple', 'Banana', 'Mango', 'Pineapple'], ['Banana', 'Pineapp
le'], ['Apple', 'Mango', 'Pineapple']]
```

Step 4: Converting the Dataset to Boolean

```
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array, columns=te.columns_)
print(df)
```

```
   Apple  Banana  Grapes  Mango  Pineapple
0   True   False    True   True      False
1  False    True   False   True       True
2   True    True   False   True       True
3  False    True   False  False       True
4   True   False   False   True       True
```

Step 5: Creating Frequent Itemset

```
frequent_itemsets_fp=fpgrowth(df, min_support=0.3, use_colnames=True)
print(frequent_itemsets_fp)
```

```
    support                     itemsets
0       0.8                      (Mango)
1       0.6                      (Apple)
2       0.8                  (Pineapple)
3       0.6                     (Banana)
4       0.6           (Mango, Pineapple)
5       0.6               (Apple, Mango)
6       0.4           (Apple, Pineapple)
7       0.4    (Apple, Mango, Pineapple)
8       0.6          (Pineapple, Banana)
9       0.4              (Mango, Banana)
10      0.4   (Mango, Pineapple, Banana)
```

Step 6: Printing the Rules

```
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=0.61)
print(rules_fp)
```

```
            antecedents         consequents
0               (Mango)         (Pineapple)
1           (Pineapple)             (Mango)
2               (Apple)             (Mango)
3               (Mango)             (Apple)
4               (Apple)         (Pineapple)
5        (Apple, Mango)         (Pineapple)
6    (Apple, Pineapple)             (Mango)
7    (Mango, Pineapple)             (Apple)
8               (Apple)  (Mango, Pineapple)
9           (Pineapple)            (Banana)
10             (Banana)         (Pineapple)
11             (Banana)             (Mango)
12   (Mango, Pineapple)            (Banana)
13      (Mango, Banana)         (Pineapple)
14  (Pineapple, Banana)             (Mango)
15             (Banana)  (Mango, Pineapple)
```

Machine Learning Lab – 20ISL68A

Program 10 - Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Step 1: Importing Libraries

```
import numpy as np
```

Step 2: Initializing training data and label data

```
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
```

Step 3: Normalizing inputs

```
X = X/np.amax(X,axis=0)
y = y/100
```

Step 4: Sigmoid Function

```
def sigmoid (x):
    return 1/(1 + np.exp(-x))
```

Step 5: Derivative of Sigmoid Function

```
def derivatives_sigmoid(x):
    return x * (1 - x)
```

Step 6: Variable Initialization

```
epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

Step 7: Weight and Bias Initialization

```
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

Step 8: Forward Propagation

```
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
```

Step 9: Backward Propagation

```
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr
```

Step 10: Output

```
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.79479351]
 [0.77374732]
 [0.79381395]]
```

Machine Learning Lab - 20ISL68A

Program 11 - Build a Convolutional Neural Network and test the same using appropriate data sets.

Step 1: Importing Libraries

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
```

Step 2: Loading and reshaping data

```
(X_train,y_train) , (X_test,y_test)=mnist.load_data()

X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
X_test = X_test.reshape((X_test.shape[0],X_test.shape[1],X_test.shape[2],1))
```

Step 3: Normalizing pixel values

```
X_train=X_train/255
X_test=X_test/255
```

Step 4: Defining model

```
model=Sequential()
```

Step 5: Adding Convolution layer → Pooling layer → Fully connected layer→ Output layer

```
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
```

```
model.add(MaxPool2D(2,2))
```

```
model.add(Flatten())
model.add(Dense(100,activation='relu'))
```

```
model.add(Dense(10,activation='softmax'))
```

Step 6: Compiling & Fitting the model

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.fit(X_train,y_train,epochs=10)
```

Expected Output

```
Epoch 1/10
1875/1875 [==============================] - 39s 20ms/step - loss: 0.1525 - accuracy: 0.9543
Epoch 2/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0535 - accuracy: 0.9844
Epoch 3/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0351 - accuracy: 0.9891
Epoch 4/10
1875/1875 [==============================] - 37s 20ms/step - loss: 0.0231 - accuracy: 0.9925
Epoch 5/10
1875/1875 [==============================] - 37s 20ms/step - loss: 0.0157 - accuracy: 0.9951
Epoch 6/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0115 - accuracy: 0.9963
Epoch 7/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0092 - accuracy: 0.9970
Epoch 8/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0072 - accuracy: 0.9976
Epoch 9/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0051 - accuracy: 0.9984
Epoch 10/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0053 - accuracy: 0.9984
<keras.callbacks.History at 0x7f6b04ab0210>
```

Machine Learning Lab – 20ISL68A

Program 12 - Implement Q learning algorithm.

Step 1: Importing Libraries

```
import numpy as np
```

Step 2: Initialize Parameters

```
gamma=0.75
alpha =0.9
```

Step 3: Define the states

```
location_to_state={
    'L1':0,
    'L2':1,
    'L3':2,
    'L4':3,
    'L5':4,
    'L6':5,
    'L7':6,
    'L8':7,
    'L9':8,
}
```

Step 4: Define the actions

```
actions=[0,1,2,3,4,5,6,7,8]
```

Step 5: Define the rewards

```
rewards =np.array([[0,1,0,0,0,0,0,0,0],
    [1,0,1,0,0,0,0,0,0],
    [0,1,0,0,0,1,0,0,0],
    [0,0,0,0,0,0,1,0,0],
    [0,1,0,0,0,0,0,1,0],
    [0,0,1,0,0,0,0,0,0],
    [0,0,0,1,0,0,0,1,0],
    [0,0,0,0,1,0,1,0,1],
    [0,0,0,0,0,0,0,1,0]])
```

Step 6: Map indices to the location

```
state_to_location=dict((state,location) for location,state in location_to_state.items())
```

Step 7: Q-Learning Implementation

```
def get_optimal_route(start_location,end_location):
       rewards_new = np.copy(rewards) # Copy reward matrix to new matrix
       # Getting end state corresponding to the ending location
       ending_state = location_to_state[end_location]
       #automatically set the priority of ending state to the highest
       rewards_new[ending_state,ending_state]=999

        #------ Q-Learning Algorithm---------
       Q=np.array(np.zeros([9,9]))  #intializing Q-Values
       for i in range(1000):        # Q-Learning processing
           current_state = np.random.randint(0,9)  #
           playable_actions =[]  # Traversing neighbor locations
           for j in range(9): # iterate new rewards matrix aand get actions>0
               if rewards_new[current_state,j]>0:
                   playable_actions.append(j)
           #Pick on action randomly from the list of playable actions that leads to next state
           next_state =np.random.choice(playable_actions)
           #Compute temporal difference
           TD= rewards_new[current_state,next_state] + gamma*Q[next_state, np.argmax(Q[next_state,])]-Q[current_state,next_state]
           Q[current_state,next_state]+=alpha*TD    #Update Q-Value using Bellman equation

       #Intialize the optimal route with the starting location
       route = [start_location]
       next_location = start_location #intialise with the starting location value
       while(next_location!=end_location):
               starting_state = location_to_state[start_location] #Fetch the starting state
               next_state = np.argmax(Q[starting_state,])         # Fetch highest Q-Value
               next_location =state_to_location[next_state]       #Getting next state letter
               route.append(next_location)
               start_location =next_location                      #Update starting location
       return route
```

Step 7: Optimal Route

```
print(get_optimal_route('L9','L1'))
```

```
['L9', 'L8', 'L5', 'L2', 'L1']
```