ACADEMIC YEAR : 2024-25 (EVEN)

Machine Learning Lab – 22ISL61

1. **Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint ai in h
   If the constraint ai is satisfied by x Then
   do nothing
   Else replace ai in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Training Examples:

Example		AirTemp	Humidity		Water		
1		Warm	Normal		Warm		
2		Warm	High		Warm		
3		Cold	High		Warm		
4		Warm	High		Cool		

Program:

```
import csv

num_attributes = 6 a = []
print("\n The Given Training Data Set \n")

with open('enjoysport.csv', 'r') as csvfile: reader = csv.reader(csvfile)
        for row in reader: a.append
                (row) print(row)

print("\n The initial value of hypothesis: ") hypothesis = ['0'] *
num_attributes print(hypothesis)


for j in range(0,num_attributes): hypothesis[j] =
                a[0][j];
print("\n Find S: Finding a Maximally Specific Hypothesis\n") for i in range(0,len(a)):
        if a[i][num_attributes]=='yes':
                for j in range(0,num_attributes): if
                        a[i][j]!=hypothesis[j]:
                                hypothesis[j]='?' else :
                                hypothesis[j]= a[i][j]
        print(" For Training instance No:{0} the hypothesis is ".format(i),hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

Data Set:

sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
The Given Training Data Set
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
Find S: Finding a Maximally Specific Hypothesis For Training Example
No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No:1 the hypothesis is ['sunny', 'warm', '?', 'strong', 'warm',
'same']

For Training Example No:2 the hypothesis is 'sunny', 'warm', '?', 'strong',
```

'warm', 'same']

For Training Example No:3 the hypothesis is 'sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples:

['sunny', 'warm', '?', 'strong', '?', '?']

Program 2 – For or a given set of training data examples stored in a .CSV file, implement and demonstrate the Document classifier using Naiive Bayes.

Step 1: Importing Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

Step 2: Understanding Datasets

data = pd.read_excel(r "C:\dataset.xlsx", name = ['Message','Label'])
print("Dataset:\n", data)

```
Dataset:
                                  Message Label
0                This is an amazing place   pos
1      I feel very good about these beers   pos
2                    This is my best work   pos
3                    What an awesome view   pos
4           I do not like this restaurant   neg
5                I am tired of this stuff   neg
6                  I can't deal with this   neg
7                    He is my sworn enemy   neg
8                     My boss is horrible   neg
9                This is an awesome place   pos
10  I do not like the taste of this juice   neg
11                         I love to dance   pos
12      I am sick and tired of this place   neg
13                   What a great holiday   pos
14         That is a bad locality to stay   neg
15         We will have good fun tomorrow   pos
16       I went to my enemy's house today   neg
```

Step 3: Printing Dimensions of data set

```
print('The dimensions of the dataset',data.shape)
```

Step 4: Converting the labels to numerical data

```
data['Labelnum']=data.Label.map({'pos':1,'neg':0})
x=data.Message
y=data.Labelnum
print(x)
print(y)
```

```
0                 This is an amazing place
1        I feel very good about these beers
2                     This is my best work
3                     What an awesome view
4             I do not like this restaurant
5                  I am tired of this stuff
6                    I can't deal with this
7                     He is my sworn enemy
8                      My boss is horrible
9                 This is an awesome place
10    I do not like the taste of this juice
11                           I love to dance
12        I am sick and tired of this place
13                    What a great holiday
14           That is a bad locality to stay
15           We will have good fun tomorrow
16         I went to my enemy's house today
Name: Message, dtype: object
0     1
1     1
2     1
3     1
4     0
5     0
6     0
7     0
8     0
9     1
10    0
11    1
12    0
13    1
14    0
15    1
16    0
Name: Labelnum, dtype: int64
```

Step 5: Converting the message to numerical data

```
vectorizer=TfidfVectorizer()
data=vectorizer.fit_transform(x)
```

Step 6: Displaying TFIDF features

```
print("\n The TFIDF features of Dataset:\n")
df=pd.DataFrame(data.toarray(), columns=vectorizer.get_feature_names())
df.head()
```

The TFIDF features of Dataset:

	about	am	amazing	an	and	awesome	bad	beers	best	boss	...	today	tomorrow	very	view	we	went	what	will	with
0	0.000000	0.0	0.589549	0.461737	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.0
1	0.416578	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.416578	0.000000	0.0	...	0.0	0.0	0.416578	0.000000	0.0	0.0	0.000000	0.0	0.0
2	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.562609	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.0
3	0.000000	0.0	0.000000	0.442107	0.0	0.492899	0.0	0.000000	0.000000	0.0	...	0.0	0.0	0.000000	0.564485	0.0	0.0	0.492899	0.0	0.0
4	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.0	0.0

5 rows × 55 columns

Step 7: Dividing dataset into training and testing data

```
print("\n Train Test Split:\n")
xtrain,xtest,ytrain,ytest=train_test_split(data,y,test_size=0.3,random_state=2)
print('\n The total number of Training Data:',ytrain.shape)
print('\n The total number of Test Data:',ytest.shape)
```

```
 Train Test Split:


 The total number of Training Data: (11,)

 The total number of Test Data: (6,)
```

Step 8: Training Naïve Bayes classifier on training data

```
clf= MultinomialNB().fit(xtrain, ytrain)
predicted = clf.predict(xtest)

#printing accuracy, Confusion matrix, Precision and Recall
print("\n Accuracy of the classifier is:", metrics.accuracy_score(ytest,predicted))
print("\nConfision Matrix is:", metrics.confusion_matrix(ytest,predicted))
print("\nClassification Report:", metrics.classification_report(ytest,predicted))
print("\nThe value of Precision :", metrics.precision_score(ytest,predicted))
print("\nThe value of Recall:", metrics.recall_score(ytest,predicted))
```

```
 Accuracy of the classifier is: 0.6666666666666666

Confision Matrix is: [[3 0]
 [2 1]]

Classification Report:               precision    recall  f1-score   support

           0       0.60      1.00      0.75         3
           1       1.00      0.33      0.50         3

    accuracy                           0.67         6
   macro avg       0.80      0.67      0.62         6
weighted avg       0.80      0.67      0.62         6


The value of Precision : 1.0

The value of Recall: 0.3333333333333333
```

Description

- Naïve Bayes methods are a set of supervised learning algorithms
- Applications Real time prediction, Multiclass prediction
- 3 types of Naïve Bayes Gaussian, Multinomial, Bernoulli
- Tfidf Vectorizer Term Frequency Inverse Document Frequency

Program 3 - Develop a program to demonstrate the working of the decision tree based CHAID algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Step 1: Importing Libraries

```
from chefboost import Chefboost as cb
```

Step 2: Loading the Dataset

```
import pandas as pd
data = pd.read_csv(r"C:\Users\kvsuv\OneDrive\Desktop\dataset3.csv")
```

Step 3: Understanding the Dataset

```
data.head()
```

	outlook	temperature	humidity	wind	Decision
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes

Step 4: Passing the Data Frame

```
config = {"algorithm": "CHAID"}
tree = cb.fit(data, config)
```

```
[INFO]:  6 CPU cores will be allocated in parallel running
CHAID  tree is going to be built...
------------------------
finished in  1.4622445106506348  seconds
------------------------
Evaluate  train set
------------------------
Accuracy:  100.0 % on  14  instances
Labels:  ['no ' 'yes ' 'no']
Confusion matrix:  [[4, 0, 0], [0, 9, 0], [0, 0, 1]]
Decision  no   => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
Decision  yes   => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
Decision  no  => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
```

Step 5: Prediction using instance – Passing Index

```
test_instance = data.iloc[2]
test_instance
```

```
outlook        overcast
temperature         hot
humidity           high
wind               weak
Decision           yes
Name: 2, dtype: object
```

Step 6: Prediction using instance

```
cb.predict(tree,test_instance)
```

```
'yes '
```

Step 7: Prediction using data values

```
moduleName = "outputs/rules/rules"
tree = cb.restoreTree(moduleName)
prediction = tree.findDecision(['sunny', 'hot', 'high', 'weak'])
prediction
```

```
'no '
```

Step 8: Decision Rule

```
df = cb.feature_importance("outputs/rules/rules.py")
df
```

```
Decision rule:  outputs/rules/rules.py
```

	feature	importance
0	outlook	0.6393
3	wind	0.1814
2	humidity	0.1337
1	temperature	0.0456

Built Decision Tree

```
def findDecision(obj): #obj[0]: outlook, obj[1]: temperature, obj[2]: humidity, obj[3]: wind
    # {"feature": "outlook", "instances": 14, "metric_value": 9.2664, "depth": 1}
    if obj[0] == 'sunny':
        # {"feature": "humidity", "instances": 5, "metric_value": 4.4495, "depth": 2}
        if obj[2] == 'high':
            return 'no '
        elif obj[2] == 'normal':
            return 'yes '
        else: return 'yes '
    elif obj[0] == 'rain':
        # {"feature": "wind", "instances": 5, "metric_value": 5.633, "depth": 2}
        if obj[3] == 'weak':
            return 'yes '
        elif obj[3] == 'strong':
            # {"feature": "temperature", "instances": 2, "metric_value": 2.8284, "depth": 3}
            if obj[1] == 'cool':
                return 'no '
            elif obj[1] == 'mild':
                return 'no'
            else: return 'no'
        else: return 'no '
    elif obj[0] == 'overcast':
        return 'yes '
    else: return 'yes '
```

Program 4 - Develop a program to demonstrate the working of the decision tree based CART algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Step 1: Importing Libraries

```
from chefboost import Chefboost as cb
```

Step 2: Loading the Dataset

```
import pandas as pd
data = pd.read_csv(r"C:\Users\kvsuv\OneDrive\Desktop\dataset4.csv")
```

Step 3: Understanding the Dataset

```
data.head()
```

	outlook	temperature	humidity	wind	Decision
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes

Step 4: Passing the Data Frame

```
config = {"algorithm": "CART"}
tree = cb.fit(data, config)
```

```
[INFO]:  6 CPU cores will be allocated in parallel running
CART  tree is going to be built...
------------------------
finished in  1.4239444732666016  seconds
------------------------
Evaluate  train set
------------------------
Accuracy:  100.0 % on  14  instances
Labels:  ['no ' 'yes ' 'no']
Confusion matrix:  [[4, 0, 0], [0, 9, 0], [0, 0, 1]]
Decision  no   => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
Decision  yes   => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
Decision  no  => Accuray:  100.0 %, Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
```

Step 5: Prediction using instance – Passing Index

```
test_instance = data.iloc[2]
test_instance
```

```
outlook        overcast
temperature         hot
humidity           high
wind               weak
Decision           yes
Name: 2, dtype: object
```

Step 6: Prediction using instance

```
cb.predict(tree,test_instance)
```

```
'yes '
```

Step 7: Prediction using data values

```
moduleName = "outputs/rules/rules"
tree = cb.restoreTree(moduleName)
prediction = tree.findDecision(['sunny', 'hot', 'high', 'weak'])
prediction
```

```
'no '
```

Step 8: Decision Rule

```
df = cb.feature_importance("outputs/rules/rules.py")
df
```

```
Decision rule:  outputs/rules/rules.py
```

	feature	importance
0	outlook	0.8077
3	wind	0.1923
1	temperature	0.0000
2	humidity	0.0000

Built Decision Tree

```
def findDecision(obj): #obj[0]: outlook, obj[1]: temperature, obj[2]: humidity, obj[3]: wind
    # {"feature": "outlook", "instances": 14, "metric_value": 0.3714, "depth": 1}
    if obj[0] == 'sunny':
        # {"feature": "humidity", "instances": 5, "metric_value": 0.0, "depth": 2}
        if obj[2] == 'high':
            return 'no '
        elif obj[2] == 'normal':
            return 'yes '
        else: return 'yes '
    elif obj[0] == 'rain':
        # {"feature": "wind", "instances": 5, "metric_value": 0.2, "depth": 2}
        if obj[3] == 'weak':
            return 'yes '
        elif obj[3] == 'strong':
            # {"feature": "temperature", "instances": 2, "metric_value": 0.0, "depth": 3}
            if obj[1] == 'cool':
                return 'no '
            elif obj[1] == 'mild':
                return 'no'
            else: return 'no'
        else: return 'no '
    elif obj[0] == 'overcast':
        return 'yes '
    else: return 'yes '
```

Program 5 - Develop a program to demonstrate the working of the Gradient Descent algorithm. Use an appropriate data set for building the model and apply this knowledge to predict a value for a test case.

Step 1: Importing Libraries

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

Step 2: Generating Random Values for x

```
x=np.arange(10)
x
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

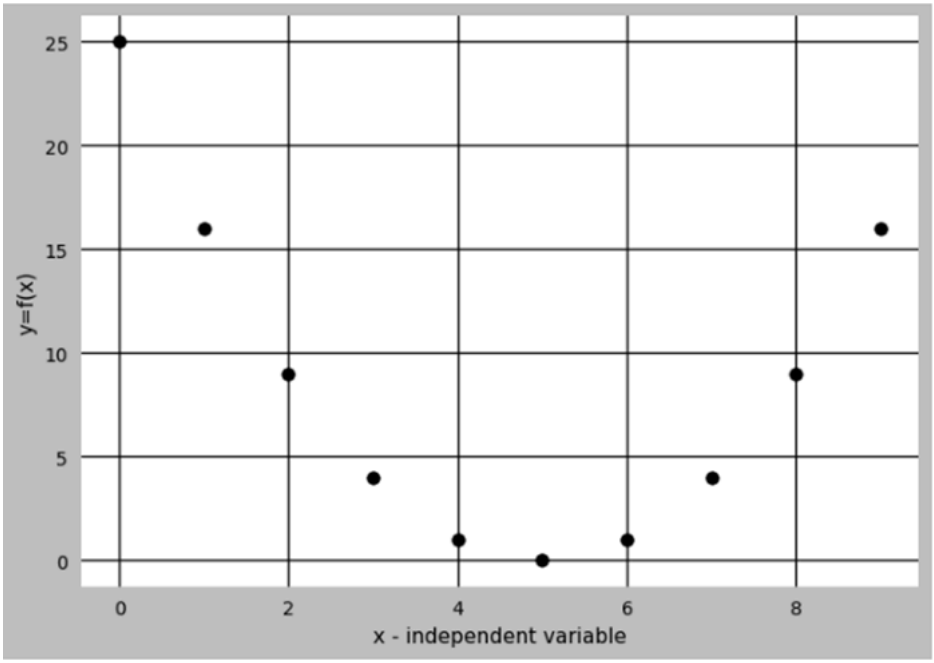
Step 3: Calculating Function Values for y

```
y=(x-5)**2
y
```

```
array([25, 16,  9,  4,  1,  0,  1,  4,  9, 16])
```

Step 4: Plotting Graph

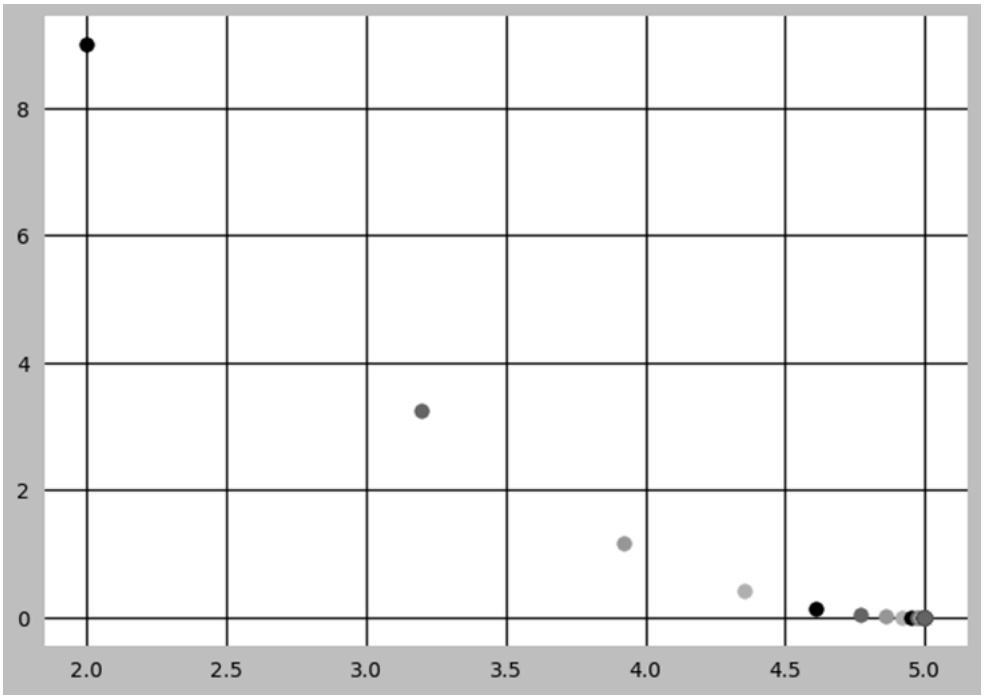
```
plt.style.use("grayscale")
plt.scatter(x,y)
plt.xlabel("x - independent variable")
plt.ylabel("y=f(x)")
plt.show()
```



<u>Step 5</u>: Gradient Descent Function

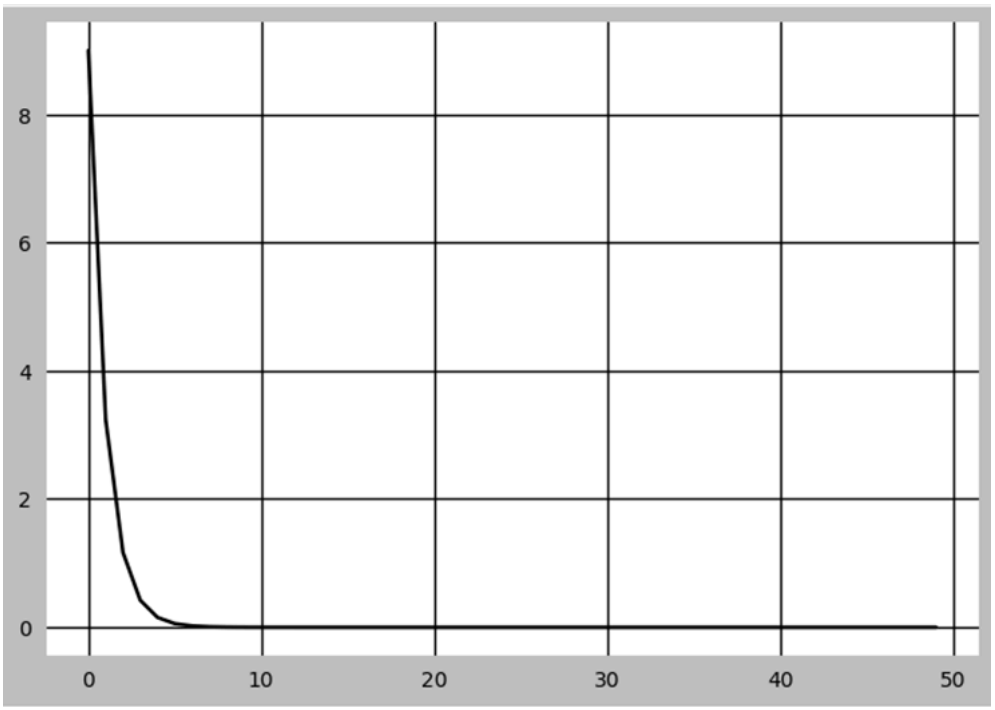
```
x=0
lr=0.2
error=[]

for i in range(50):
    grad = 2*(x-5)
    x = x-lr*grad
    y = (x-5)**2
    error.append(y)
    plt.scatter(x,y)
    print(x)
```

```
2.0
3.2
3.92
4.352
4.6112
4.76672
4.860032
4.9160192
4.94961152
4.969766912
4.9818601472
4.98911608832
4.993469652992
4.9960817917952
4.9976490750771205
4.998589445046273
4.999153667027763
4.999492200216658
4.999695320129995
4.999817192077997
4.999890315246798
4.999934189148079
4.999960513488848
4.999976308093308
4.999985784855985
4.999991470913591
4.999994882548155
4.999996929528892
4.999998157717336
```



<u>Step 6</u>: Plotting Error

```
plt.plot(error)
plt.show()
```



Program 6 - Develop a program to construct a Support Vector Machine considering a Sample Dataset.

Step 1: Importing Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

Step 2: Loading the Dataset

```
iris = datasets.load_iris()
```

Step 3: Considering only first two features (Sepal Length and Sepal Width)

```
X = iris.data[:, :2]
y = iris.target
```

Step 4: Creating instance of SVM

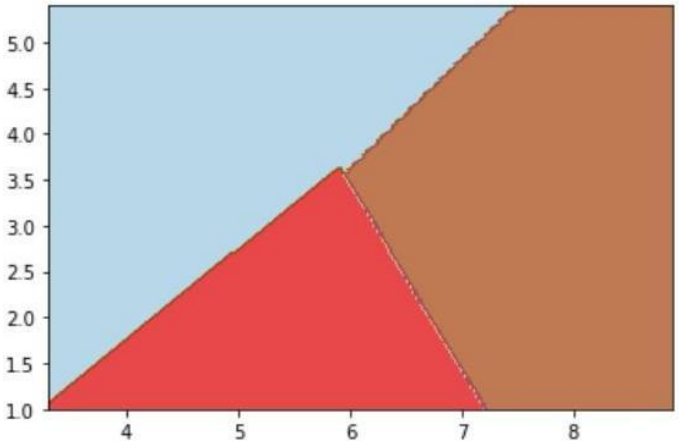
```
svc = svm.SVC(kernel='linear').fit(X, y)
```

Step 5: Creating a mesh to plot

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

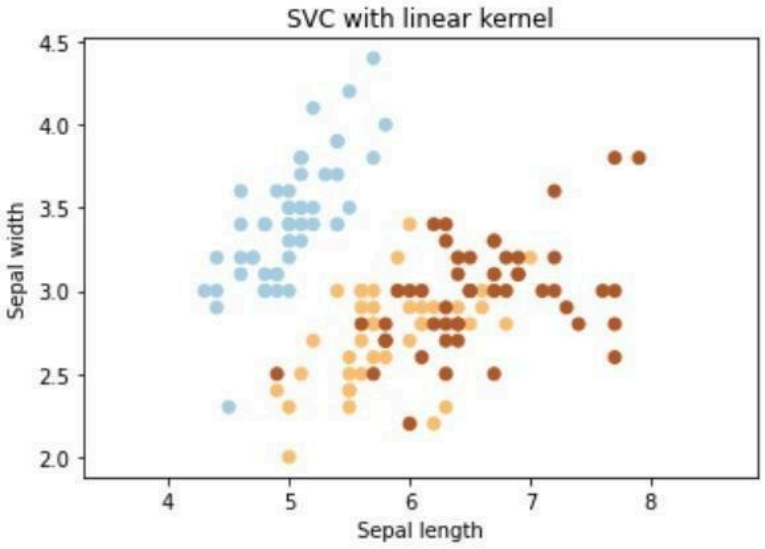
Step 6: Plotting the graph

```
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```



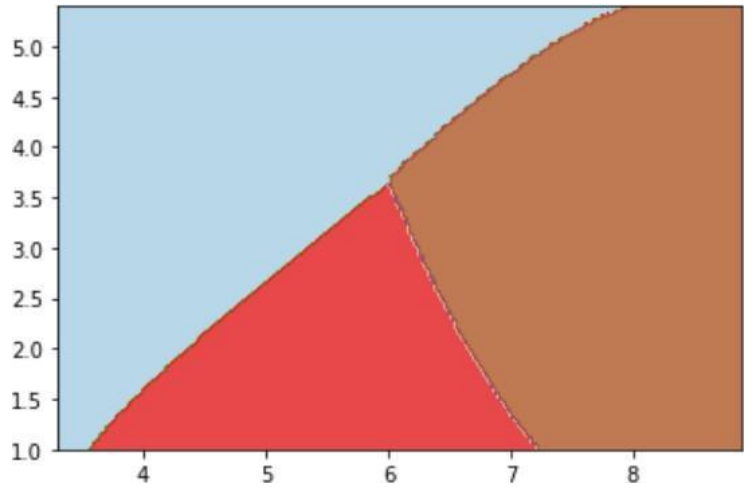
Step 7: Plotting the data values

```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```

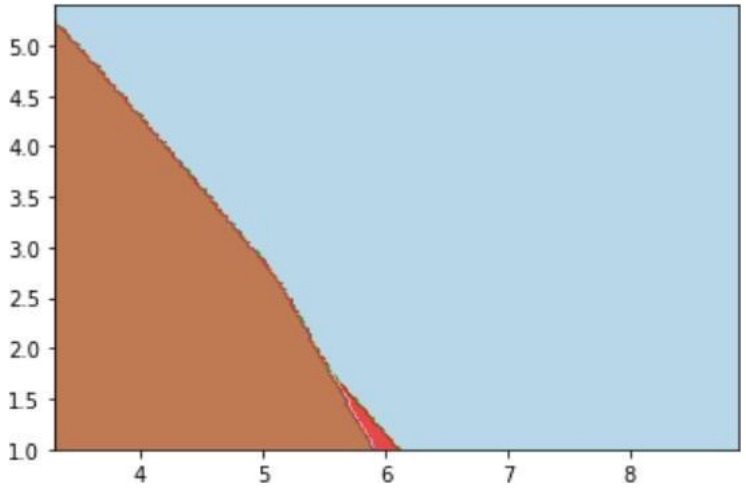


Different types of Kernels

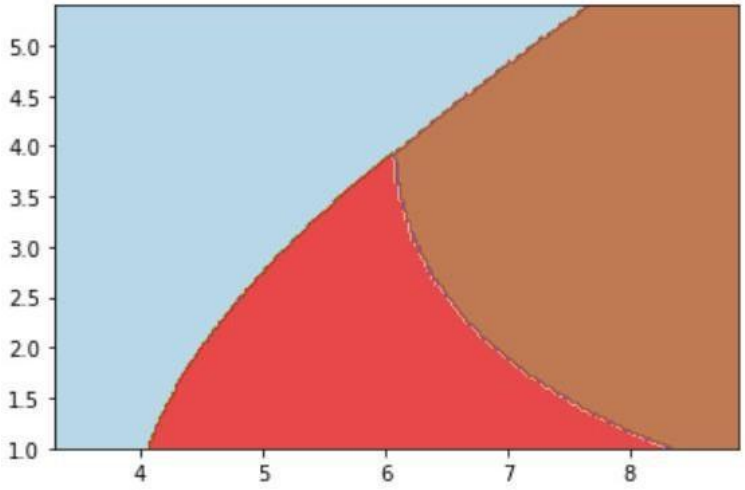
rbf



sigmoid



poly



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

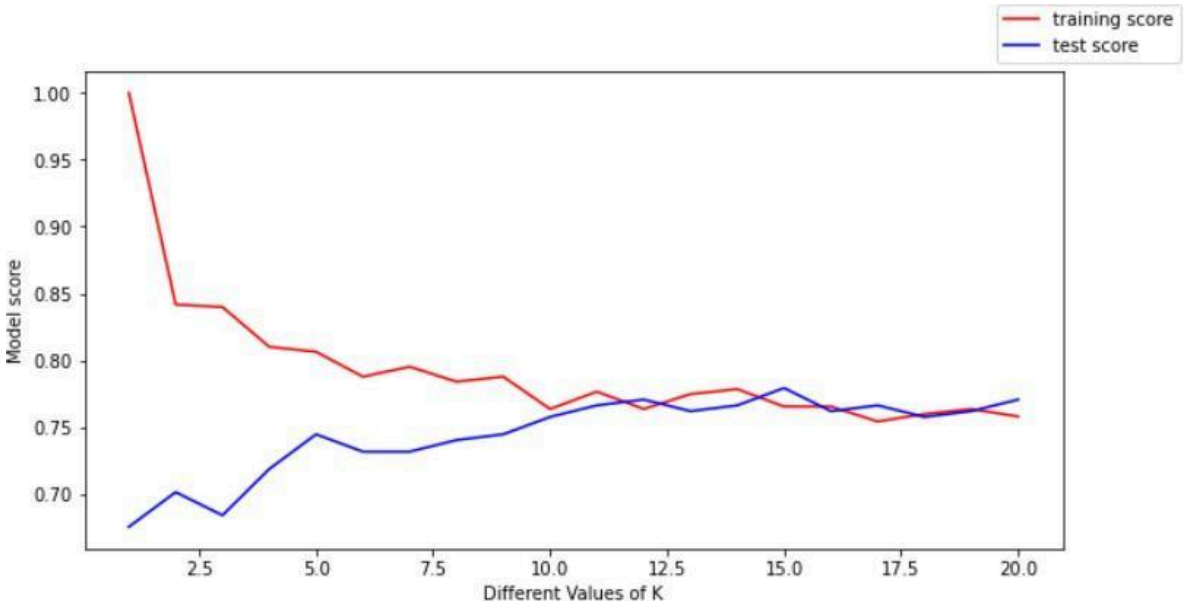
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
data_df = ss.fit_transform(data_df)

#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
```

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)
```

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)
```

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]]
```

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>
```

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']
```