# DR K.V. SUBBA REDDY INSTITUTE OF TECHNOLOGY

Dupadu Village, NH-44, Lakshmipuram (Post), Kurnool, AP-518

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu, CSE Accredited by NBA Recognized under Section 2 (f) and 12B of UGC Act, 1956 ISO 9001 : 2015 & ISO 14001 : Certified Institution

**(AUTONOMOUS)**

# LAB RECORD

## B.Tech III Year Semester (R20)

**Name** : ...............................

**Roll No.** : ...............................

## MACHINELEARNINGLAB
**(20A05602T)**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE&MACHINE LEARNING**

**Dr. K.V. SUBBA REDDY INSTITUTE OF TECHNOLOGY**

**(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapur)**

**An ISO 9001:2000 Certified Institution**

**DUPADU, KURNOOL-518218.**

# INDEX

# DR K.V. SUBBA REDDY INSTITUTE OF TECHNOLOGY

Dupadu Village, NH-44, Lakshmipuram (Post), Kurnool, AP-518

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu, CSE Accredited by NBA Recognized under Section 2 (f) and 12B of UGC Act, 1956 ISO 9001 : 2015 & ISO 14001 : Certified Institution

## (AUTONOMOUS)

## CERTIFICATE

**Department of ________________________________________**

Certifiedthatthisisabonafiderecordoftheworkdoneby

**Mr/Ms ______________________________________________**of

**_______________B.Tech/M.Tech/MBAin _____________**

**____________________________________**Laboratory.

**Date: ____________HeadOf thedepartment Staff-in-Charge**

**--------------------------------------------------------------------------------**

**Regd. No. _________________________**

**Submitted forthe practical examination held on ________________**

**InternalExaminer** **ExternalExaminer**

1. **ImplementanddemonstratetheFIND-Salgorithmforfindingthemostspecifichypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

```
importpandasaspd
deffind_s(training_data):
  #Initializethemostspecifichypothesis
  num_features=training_data.shape[1]-1#Excludethelabelcolumn
  specific_hypothesis = ['0'] * num_features

  # Iterate through the training examples
  forindex,rowintraining_data.iterrows():
    ifrow[-1]==1:#Ifit'sapositiveexample for i in
      range(num_features):
        ifspecific_hypothesis[i]=='0':
          specific_hypothesis[i]=row[i]#Settothevalueofthefeature elif
        specific_hypothesis[i] != row[i]:
          specific_hypothesis[i]='?'#Setto'?'ifthere'samismatch return

  specific_hypothesis

defmain():
  #LoadthetrainingdatafromaCSV file
  #AssumetheCSVfilehasnoheaderandthelastcolumnistheclasslabel
  filename = 'training_data.csv'# Replace with your CSV file path
  training_data=pd.read_csv(filename,header=None)

  # Apply the FIND-S algorithm
  hypothesis=find_s(training_data)

  # Output the most specific hypothesis
  print("MostSpecificHypothesis:",hypothesis)
ifname_____=="main":
  main()
```

**ExpectedOutput:**

Assumingthe**training_data.csv**filecontainsthefollowingdata:

```
sunny,hot,high,no,1
sunny,hot,high,yes,0
overcast,hot,high,no,1
rainy,mild,high,no,0
rainy,cool,normal,no,1
rainy,cool,normal,yes,1
overcast,cool,normal,yes,1
sunny,mild,high,no,0
sunny,cool,normal,no,1
```

2. **Foragivensetoftrainingdataexamplesstoredina.CSVfile,implementanddemonstrate the Candidate- Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

```
importpandasaspd

defcandidate_elimination(data):
  #Initializethespecificandgeneralhypotheses
  specific_hypothesis=['0']*(data.shape[1]-1)#Assuminglastcolumnisthetarget
  general_hypothesis = ['?'] * (data.shape[1] - 1)

  #Iteratethrougheachtrainingexample for
  index, row in data.iterrows():
    ifrow['Play']=='Yes':
      #Updatespecifichypothesis
      foriinrange(len(specific_hypothesis)):
        ifspecific_hypothesis[i]=='0':
          specific_hypothesis[i] = row[i]
        elifspecific_hypothesis[i]!=row[i]:
          specific_hypothesis[i]='?'
      # Update general hypothesis
      foriinrange(len(general_hypothesis)):
        ifgeneral_hypothesis[i]=='?':
          general_hypothesis[i] = row[i]
        elifgeneral_hypothesis[i]!=row[i]:
          general_hypothesis[i]='?'
    else:# row['Play'] == 'No'
      #Updategeneralhypothesis
      foriinrange(len(general_hypothesis)):
        ifgeneral_hypothesis[i]=='?':
          general_hypothesis[i] = row[i]
        elifgeneral_hypothesis[i]!=row[i]:
          general_hypothesis[i]='?'
      #Updatespecifichypothesistobemorespecific for i
      in range(len(specific_hypothesis)):
        ifspecific_hypothesis[i]==row[i]:
          continue
        else:
          specific_hypothesis[i]='?'

  returnspecific_hypothesis,general_hypothesis

# Load the data
data=pd.read_csv('training_data.csv')

#RuntheCandidate-Eliminationalgorithm
specific_hypothesis,general_hypothesis=candidate_elimination(data)
```

#Printtheresults
print("SpecificHypothesis:",specific_hypothesis)
print("GeneralHypothesis:",general_hypothesis)

**Expectedoutput:**
Assumingwehavethisdatain**training_data.csv**,runningthecodewouldyieldspecificand general hypotheses based on the training examples labeled with "Play" as either "Yes" or "No".

Verify Open In Editor

```
Specific Hypothesis: ['?', '?', 'Normal', '?']
General Hypothesis: ['?', '?', '?', '?']
```

3. **WriteaprogramtodemonstratetheworkingofthedecisiontreebasedID3algorithm.Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

```
pipinstallpandasnumpyscikit-learn
import numpy as np
importpandasaspd
fromsklearn.datasetsimportload_iris
fromsklearn.model_selectionimporttrain_test_split
from sklearn.tree import DecisionTreeClassifier
fromsklearnimporttree
importmatplotlib.pyplotasplt

#LoadtheIrisdataset iris
= load_iris()
X = iris.datay
= iris.target

#CreateaDataFrameforbettervisualization
df=pd.DataFrame(data=np.c_[X,y],columns=iris.feature_names+['target'])
print("Iris Dataset:")
print(df.head())

#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

#InitializetheDecisionTreeClassifierusingtheID3algorithm(default) clf =
DecisionTreeClassifier(criterion='entropy', random_state=42)

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

#VisualizetheDecisionTree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.title("DecisionTreeusingID3Algorithm")
plt.show()

#Testthemodel
accuracy = clf.score(X_test, y_test)
print(f"Accuracyofthemodel:{accuracy:.2f}")

#Classify anew sample(e.g.,[5.0,3.5,1.5,0.2])
new_sample=np.array([[5.0,3.5,1.5,0.2]])
predicted_class = clf.predict(new_sample)
predicted_class_name=iris.target_names[predicted_class][0]
```

print(f"Thepredictedclassforthenewsample{new_sample.flatten()}is:
{predicted_class_name}")

Example output:

Verify Open In Editor

```
1 Accuracy of the model: 1.00
```

**Predicted Class for New Sample**: The script will classify the new sample `[5.0, 3.5, 1.5, 0.2]` and print the predicted class name.

Example output:

Verify Open In Editor

```
1 The predicted class for the new sample [5.  3.5 1.5 0.2] is: setosa
```

4. **BuildanArtificialNeuralNetworkbyimplementingtheBackpropagationalgorithmand test the same using appropriate data sets.**

```
importnumpyasnp
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
from sklearn.preprocessing import StandardScaler
fromsklearn.datasetsimportload_iris #
Load Iris dataset
iris=load_iris()
X = iris.datay
= iris.target

#One-hotencodethetargetvariable y
= np.eye(3)[y]

#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

#Standardizethefeatures
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
class NeuralNetwork:
  definit(self,input_size,hidden_size,output_size,learning_rate=0.01): self.learning_rate =
    learning_rate

    #Initializeweights
    self.weights_input_hidden = np.random.rand(input_size, hidden_size)
    self.weights_hidden_output = np.random.rand(hidden_size, output_size)

    #Initializebiases
    self.bias_hidden = np.random.rand(hidden_size)
    self.bias_output=np.random.rand(output_size)

  defsigmoid(self,x):
    return1/(1+np.exp(-x))

  defsigmoid_derivative(self,x):
    returnx*(1 -x)

  defforward(self,X):
    self.hidden_layer_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
    self.hidden_layer_output = self.sigmoid(self.hidden_layer_input)

    self.final_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) +
self.bias_output
```

```
        self.final_output = self.sigmoid(self.final_input)

        return self.final_output

    def backward(self,X,y,output):
        #Calculatetheerror
        error = y - output

        #Calculategradients
        d_output=error*self.sigmoid_derivative(output)
        error_hidden_layer = d_output.dot(self.weights_hidden_output.T)
        d_hidden_layer = error_hidden_layer *
self.sigmoid_derivative(self.hidden_layer_output)

        #Updateweightsandbiases
        self.weights_hidden_output += self.hidden_layer_output.T.dot(d_output) *
self.learning_rate
        self.bias_output+=np.sum(d_output,axis=0)*self.learning_rate
        self.weights_input_hidden += X.T.dot(d_hidden_layer) * self.learning_rate
        self.bias_hidden += np.sum(d_hidden_layer, axis=0) * self.learning_rate

    deftrain(self,X, y,epochs):
        forepochinrange(epochs):
            output = self.forward(X)
            self.backward(X,y,output)

    defpredict(self,X):
        output= self.forward(X)
        returnnp.argmax(output,axis=1)
# Define the neural network
input_size=X_train.shape[1]#Numberoffeatures
hidden_size = 5# Number of hidden neurons
output_size=y_train.shape[1]#Numberofclasses
learning_rate = 0.01
epochs= 1000

nn=NeuralNetwork(input_size,hidden_size,output_size,learning_rate) #

Train the neural network
nn.train(X_train,y_train,epochs)
#Makepredictions
predictions=nn.predict(X_test)

#Convertone-hotencodedlabelstoclasslabels
y_test_labels = np.argmax(y_test, axis=1)

#Calculateaccuracy
accuracy=np.mean(predictions==y_test_labels)
```

print(f'Accuracy:{accuracy*100:.2f}%')

**Expectedoutput:**

```
Verify Open In Editor
1 Accuracy: XX.XX%
```

Where `xx.xx` will be a number representing the accuracy of the model on the test set. Given that the Iris dataset is relatively simple and the neural network is trained for 1000 epochs, you can expect the accuracy to be quite high, often above 90%. However, the exact number may vary slightly due to the random initialization of weights and biases.

For example, you might see an output like:

```
Verify Open In Editor
1 Accuracy: 100.00%
```

5. **Write a program toimplement the naïveBayesian classifier for a sample training data set storedasa.CSVfile.Computetheaccuracyoftheclassifier,consideringfewtestdatasets.**

```
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
from sklearn.naive_bayes import GaussianNB
fromsklearn.metricsimportaccuracy_score

#Loadthe dataset
data=pd.read_csv('data.csv')

# Separate features and labels
X=data[['feature1','feature2']]
y = data['label']

#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

#CreateaGaussianNaiveBayesclassifier
model = GaussianNB()

#Fitthemodelonthetrainingdata
model.fit(X_train, y_train)

#Makepredictionsonthetestdata
y_pred = model.predict(X_test)

#Computeaccuracy
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

#Exampleofpredictingonnewtestdata
new_data = pd.DataFrame({
  'feature1':[1,0],
  'feature2':[0,1]
})

predictions = model.predict(new_data)
print("Predictionsfornewdata:",predictions)
```

ExpectedOutput:

Verify Open In Editor

```
Accuracy: 100.00%
Predictions for new data: [0 0]
```

**6. AssumingasetofdocumentsthatneedtobeclassiƒÓied,usethenaïveBayesianClassifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.**

```
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
fromsklearn.metricsimportaccuracy_score,precision_score,recall_score,
classification_report

#Loadthe dataset
#Makesuretoreplace'your_dataset.csv'withyouractualdatasetfile data
= pd.read_csv('your_dataset.csv')

#Displaythefirstfewrowsofthedataset
print(data.head())

#Splitthedataintofeaturesandlabels X =
data['text']
y= data['label']

#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

#ConverttextdataintonumericaldatausingCountVectorizer vectorizer =
CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

#CreateandtraintheNaiveBayesclassifier
classifier = MultinomialNB()
classifier.fit(X_train_counts,y_train)

#Makepredictionsonthetestset
y_pred= classifier.predict(X_test_counts)

# Calculate accuracy, precision, and recall
accuracy=accuracy_score(y_test,y_pred)
precision=precision_score(y_test,y_pred,average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

#Printtheresults
print(f'Accuracy:{accuracy:.2f}')
print(f'Precision:{precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
# Print a detailed classification report
print(classification_report(y_test, y_pred))
```

**ExpectedOutput:**

['about','am','amazing','an','and','awesome','beers','best','boss','can','deal','do', 'enemy','feel','fun','good','have','horrible','house','is','like','love','my','not','of','place', 'restaurant','sandwich','sick','stuff','these','this','tired','to','today','tomorrow','very', 'view','we','went','what','will','with','work']aboutamamazinganandawesomebeersbest boss can ... today \

```
010 0000 10 00... 0
010 00000 0100 ... 0
020 01100 0000 ... 0
030 00000 0000 ... 0
140 00000 0000 ... 0
050 10010 0000 ... 0
060 00000 0001 ... 0
070 00000 0000 ... 0
080 10000 0000 ... 0
090 00101 0000 ... 0
0100000 00000 0...0
011000 0 00001 0...0
0120001 01000 0...0
```

7. **WriteaPythonprogramtoconstructaBayesiannetworkconsideringmedicaldata.Use thismodeltodemonstratethediagnosisofheartpatientsusingstandardHeartDisease Data Set.**

```
!pipinstallpgmpy#Installthepgmpylibrary import
numpy as np
importpandasaspd
frompgmpy.modelsimportBayesianModel#Nowthislineshouldwork from
pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
# Load the Heart Disease dataset
heartDisease = pd.read_csv('heart.csv')
heartDisease=heartDisease.replace('?',np.nan) #
Display sample instances from the dataset
print('Sampleinstancesfromthedatasetaregivenbelow:')
print(heartDisease.head())
#Displayattributesandtheirdatatypes
print('\nAttributes and datatypes:')
print(heartDisease.dtypes)
#DefinethestructureoftheBayesianNetwork
model = BayesianModel([
   ('age','heartdisease'),
   ('sex','heartdisease'),
   ('exang','heartdisease'),
   ('cp','heartdisease'),
   ('heartdisease', 'restecg'),
   ('heartdisease', 'chol')
])
# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPD using Maximum Likelihood Estimators...')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
# Inference with the Bayesian Network
print('\nInferencing with Bayesian Network:')
heartDisease_infer = VariableElimination(model)
# Query 1: Probability of Heart Disease given evidence = restecg
print('\n1.ProbabilityofHeartDiseasegivenevidence=restecg:')
q1=heartDisease_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
# Query 2: Probability of Heart Disease given evidence = cp
print('\n2.ProbabilityofHeartDiseasegivenevidence=cp:')
q2=heartDisease_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

**ExpectedOutput:**

```
  agesexcptrestbpscholfbsrestecgthalachexangoldpeak\
067    0  0    134  370  0    1    108     14.123406
157    1  1     99  558  1    1    140   01.877005
243    0  1    118  317  1    0    186     10.365433
371    1  3    106  443  0    0    149     14.209392
436    0  2    191  231  1    1     79   10.058965
```

```
  slopecathalheartdisease
0     2  1    3          1
1     1  2    3          0
2     2  1    2          0
3     1  1    3          1
4     1  1    3          1
Attributes  anddatatyp es:
age             int64
sex             int64
cp              int64
trestbps        int64
chol            int64
fbs             int64
restecg         int64
thalach         int64
exang           int64
oldpeak         float64
slope           int64
ca              int64
thal            int64
heartdisease    int64
dtype: object
LearningCPDusingMaximumLikelihoodEstimators...
Inferencing with Bayesian Network:


1. ProbabilityofHeartDiseasegivenevidence=restecg:
+-----------+-------------+
|heartdisease|phi(heartdisease)|
+================+====================+
|heartdisease(0)|            0.4578|
+-----------+-------------+
|heartdisease(1)|            0.5422|
+-----------+-------------+

2. ProbabilityofHeartDiseasegivenevidence= cp:
+-----------+-------------+
|heartdisease|phi(heartdisease)|
+================+====================+
|heartdisease(0)|            0.5017|
+-----------+-------------+
|heartdisease(1)|            0.4983|
+-----------+-------------+
```

8. **Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and commentonthequalityofclustering.YoucanaddJava/PythonMLlibraryclasses/APIin the program.**

```
importpandasaspd #

Load the dataset
data= pd.read_csv('heart_disease.csv')

#Displaythefirstfewrowsofthedataset
print(data.head())
# Handle missing values if necessary
data.fillna(data.mean(), inplace=True)

#Normalizethedata(optional,butoftenbeneficial) from
sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
scaled_data=scaler.fit_transform(data)
from sklearn.cluster import KMeans
importmatplotlib.pyplotasplt

#Choosethenumberofclusters(k)
k=3#Adjustbasedonyour requirement
kmeans=KMeans(n_clusters=k,random_state=42)
kmeans.fit(scaled_data)

#Gettheclusterlabels
kmeans_labels=kmeans.labels_

#Visualizetheclusters(if2Dor3D)
plt.scatter(scaled_data[:,0],scaled_data[:,1],c=kmeans_labels,cmap='viridis')
plt.title('k-Means Clustering')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.show()
fromsklearn.mixtureimportGaussianMixture

#Fitthemodel
gmm=GaussianMixture(n_components=k,random_state=42)
gmm.fit(scaled_data)

#Gettheclusterlabels
gmm_labels=gmm.predict(scaled_data)

#Visualizetheclusters(if2Dor3D)
plt.scatter(scaled_data[:,0],scaled_data[:,1],c=gmm_labels,cmap='viridis')
```

```
plt.title('EM(GMM)Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature2')
plt.show()
fromsklearn.metricsimportsilhouette_score,adjusted_rand_score

#CalculateSilhouetteScore
kmeans_silhouette = silhouette_score(scaled_data, kmeans_labels)
gmm_silhouette = silhouette_score(scaled_data, gmm_labels)

#CalculateAdjustedRandIndex(ifyouhavetruelabels)
#true_labels=data['target']#Assumingthereisatargetcolumn
#ari_kmeans=adjusted_rand_score(true_labels,kmeans_labels) #
ari_gmm = adjusted_rand_score(true_labels, gmm_labels)

print(f'k-Means Silhouette Score: {kmeans_silhouette}')
print(f'GMM Silhouette Score: {gmm_silhouette}')
#print(f'k-MeansARI:{ari_kmeans}') #
print(f'GMM ARI: {ari_gmm}')
```

**ExpectedOutput:**

Verify Open In Editor
Verify Open In Editor

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| 1 | 67 | 1 | 2 | 160 | 286 | 0 | 1 | 108 | 1 | 1.5 | 1 | 0 | 2 | 1 |
| 2 | 67 | 1 | 2 | 120 | 229 | 0 | 1 | 129 | 1 | 2.6 | 1 | 0 | 2 | 1 |
| 3 | 37 | 1 | 1 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| 4 | 41 | 0 | 1 | 130 | 204 | 0 | 1 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |

9. **Writeaprogramtoimplementk-NearestNeighboralgorithmtoclassifytheirisdataset. Print both correct and wrong predictions.**

```
importnumpyasnp
importpandasaspd
fromsklearnimportdatasets
fromsklearn.model_selectionimporttrain_test_split
from sklearn.neighbors import KNeighborsClassifier
fromsklearn.metricsimportaccuracy_score
# Load the Iris dataset
iris=datasets.load_iris()
X = iris.data# Featuresy
= iris.target# Labels
#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42) #
Create the k-NN classifier
k=3#Youcanchangethevalueofk
knn=KNeighborsClassifier(n_neighbors=k) #
Train the classifier
knn.fit(X_train,y_train)
#Makepredictionsonthetestset
y_pred = knn.predict(X_test)
#Calculateaccuracy
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
#Printcorrectandwrongpredictions
print("\nCorrect Predictions:")
foriinrange(len(y_test)):
  ify_pred[i]==y_test[i]:
    print(f"Predicted: {iris.target_names[y_pred[i]]}, Actual: {iris.target_names[y_test[i]]}")
print("\nWrong Predictions:")
foriinrange(len(y_test)):
  ify_pred[i]!=y_test[i]:
    print(f"Predicted:{iris.target_names[y_pred[i]]},Actual:{iris.target_names[y_test[i]]}")
```

**Expectedoutput:**

Verify Open In Editor

```
Accuracy: 100.00%

Correct Predictions:
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa

Wrong Predictions:
```

**10. Implementthenon-parametricLocallyWeightedRegressionalgorithminordertofitdata points. Select appropriate data set for your experiment and draw graphs.**

```
importnumpyasnp
importmatplotlib.pyplotasplt

deflocally_weighted_regression(X,y,query_point,tau):
  """PerformLocallyWeightedRegressiononasinglequerypoint.""" m =
  X.shape[0]
  weights=np.exp(-np.sum((X-query_point)**2,axis=1)/(2*tau**2)) W =
  np.diag(weights)

  #Calculatethetausingthenormalequation:(X.T*W*X)^(-1)*(X.T*W* y)
  X_b=np.hstack((np.ones((m,1)),X))#Addbiasterm
  theta=np.linalg.inv(X_b.T@W@X_b)@(X_b.T@W@y) return

  theta

defpredict(X,y,query_points,tau):
  """PredictvaluesforthegivenquerypointsusingLocallyWeightedRegression.""" predictions =
  []
  forquery_pointinquery_points:
    theta=locally_weighted_regression(X,y,query_point,tau)
    predictions.append(np.dot(np.array([1,query_point]),theta))#Includebiastermin
prediction
  returnnp.array(predictions)

#Generatesyntheticdata
np.random.seed(42)
X=np.sort(5*np.random.rand(80,1),axis=0)
y=np.sin(X)+np.random.normal(0,0.1,X.shape)#Non-linearrelationshipwithnoise

#Definequerypoints
query_points=np.linspace(0,5,100)

#PerformLocallyWeightedRegression
tau = 0.5# Bandwidth parameter
predictions=predict(X,y,query_points,tau)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(X,y,color='blue',label='DataPoints')
plt.plot(query_points,predictions,color='red',label='LocallyWeightedRegression', linewidth=2)
plt.title('Locally Weighted Regression (LWR)')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
```

```
plt.grid()
plt.show()
```

**ExpectedOutput:**