CP4252 Machine Learning lab manual

Machine Learning (M. Kumarasamy College of Engineering)

solamalai
COLLEGE OF ENGINEERING
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
An ISO 9001: 2015 Certified Institution
S.V.Raja Nagar, Veerapanjan, Sivagangai Road , Madurai - 625020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Name : _____

Reg.No : _____ Semester : _____

Degree : _____ Branch : _____

Subject : _____ Subject Code: _____

solamalai
COLLEGE OF ENGINEERING
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
An ISO 9001: 2015 Certified Institution
S.V.Raja Nagar, Veerapanjan, Sivagangai Road , Madurai - 625020

BONAFIDECERTIFICATE

Name of the Student: _____

Register No.

This is to certify that this is a bonafide record of the workdone by the above student with RollNo._____of_____Semester B.E Degree in _____ in the Laboratory _____

during the academic year 2022 – 2023.

Staff-In-Charge **Head of the Dept.**

Date:

Submitted for the Practical Examination held on _____

Internal Examiner **External Examiner**

CS3461- OPERATING SYSTEMS LABORATORY

INDEX

EX.NO.	DATE	NAME OF THE EXPERIMENT	MARKS	SIGN

PROGRAM 1:

IMPLEMENT NAIVE BAYES THEOREM TO CLASSIFY THE ENGLISH TEXT:

AIM:

To Implement naive bayes theorem to classify the English text

DESCRIPTION:

The challenge of text classification is to attach labels to bodies of text, e.g., tax document, medical form, etc. based on the text itself. For example, think of your spam folder in your email. How does your email provider know that a particular message is spam or “ham” (not spam)? We’ll take a look at one natural language processing technique for text classification called Naive Bayes.

SOURCE CODE:

```
import pandas as pd

fromsklearn.model_selection import train_test_split

fromsklearn.feature_extraction.text import CountVectorizer

fromsklearn.naive_bayes import MultinomialNB

fromsklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score


msg = pd.read_csv('document.csv', names=['message', 'label'])

print("Total Instances of Dataset: ", msg.shape[0])

msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message

y = msg.labelnum

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)

count_v = CountVectorizer()


Xtrain_dm = count_v.fit_transform(Xtrain)

Xtest_dm = count_v.transform(Xtest)

df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
```

```
clf = MultinomialNB()

clf.fit(Xtrain_dm, ytrain)

pred = clf.predict(Xtest_dm)

print('Accuracy Metrics:')

print('Accuracy: ', accuracy_score(ytest, pred)) print('Recall: ', recall_score(ytest, pred)) print('Precision: ',

precision_score(ytest, pred))

print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

document.csv:

```
I love this sandwich,pos This

is an amazingplace,pos

I feel very good about these beers,pos

This is my best work,pos

What an awesome view,pos

I do not like this restaurant,neg

I am tired of this stuff,neg

I can't deal with this,neg He

is my sworn enemy,neg My

boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this juice,neg

I love to dance,pos

I am sick and tired of this place,neg

What a great holiday,pos

That is a bad locality to stay,neg

We will have good fun tomorrow,pos I
```

went to my enemy's house today,neg

OUTPUT:

Total Instances of Dataset: 18

Accuracy Metrics: Accuracy: 0.6

Recall: 0.6666666666666666

Precision: 0.6666666666666666

Confusion Matrix:

[[1 1]

[1 2]]

VIVA QUESTIONS & ANSWERS:

1. How Naive Bayes algorithm works?

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Whether	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	Yes
Rainy	Yes
Rainy	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table

Whether	No	Yes
Overcast		4
Sunny	2	3
Rainy	3	2
Total	5	9

Likelihood Table 1

Whether	No	Yes		
Overcast		4	=4/14	0.29
Sunny	2	3	=5/14	0.36
Rainy	3	2	=5/14	0.36
Total	5	9		
	=5/14	=9/14		
	0.36	0.64		

Likelihood Table 2

Whether	No	Yes	Posterior Probability for No	Posterior Probability for Yes
Overcast		4	0/5=0	4/9=0.44
Sunny	2	3	2/5=0.4	3/9=0.33
Rainy	3	2	3/5=0.6	2/9=0.22
Total	5	9		

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem:: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

2. Applications of Naive Bayes Algorithms:

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.

- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments).

- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not.

PROGRAM 2:

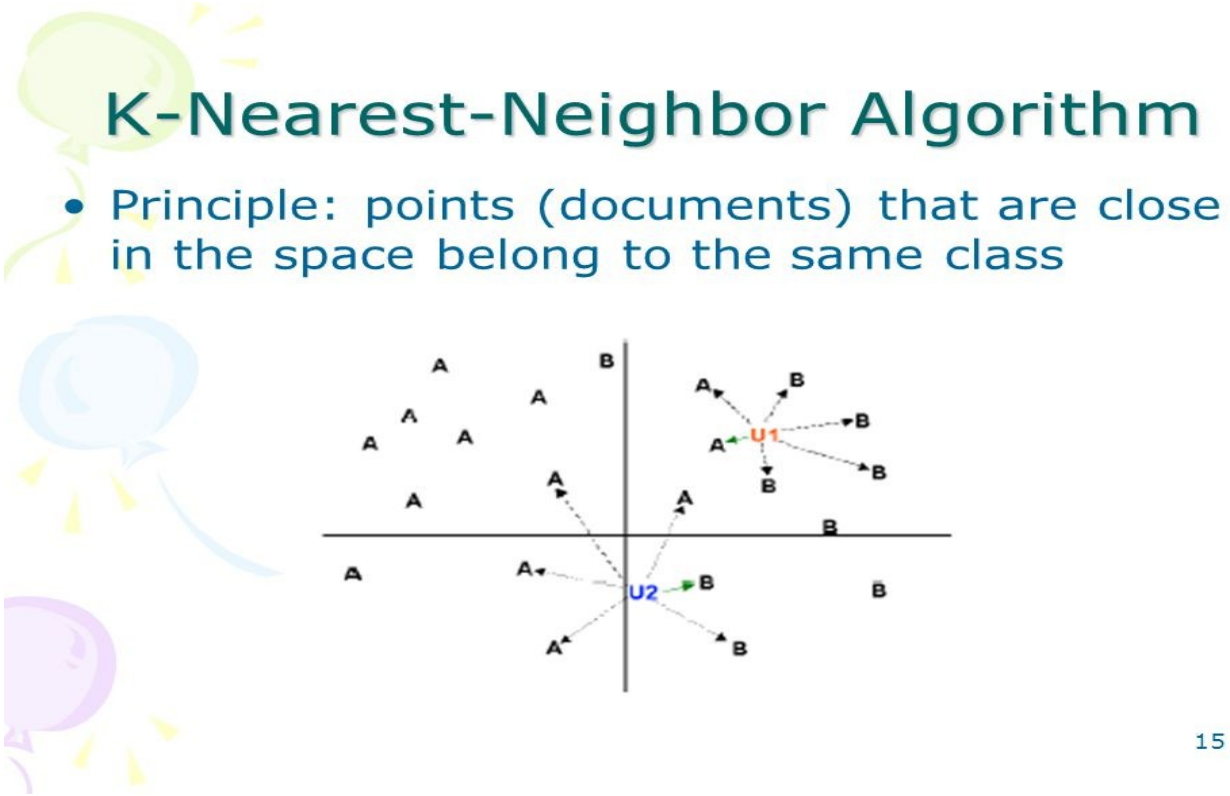
TO IMPLEMENT CLASSIFICATION WITH K-NEAREST NEIGHBORS

AIM:

Write a program to implement classification with K-Nearest Neighbors

DESCRIPTION:

Write a program to implement classification with K-Nearest Neighbors. In this question, you will use the scikit-learn's KNN classifier to classify real vs. fake news headlines. The aim of this question is for you to read the scikit-learn API and get comfortable with training /validation splits. Use California Housing Datasets.



Distance Metrics

Minkowsky:

$$D(x, y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(x, y) = \sum_{i=1}^{m} |x_i - y_i|$$

Canberra:

$$D(x, y) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(x, y) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(x, y) = (x - y)^T Q (x - y) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(x, y) = [\det V]^{1/m} (x - y)^T V^{-1} (x - y)$$

V is the covariance matrix of $A_1 \ldots A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(x, y) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set

Chi-square:

$$D(x, y) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_x} - \frac{y_i}{size_y} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector x.

Kendall's Rank Correlation:

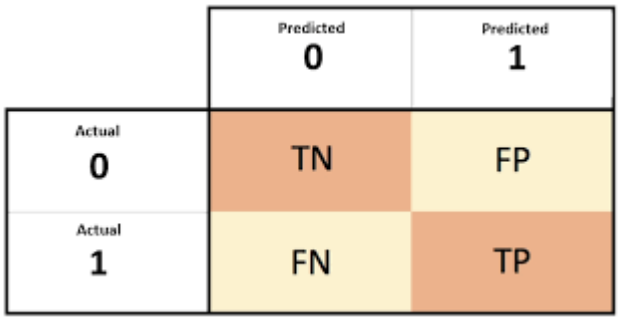
$$D(x, y) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \text{sign}(x_i - x_j) \text{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

K-Nearest-Neighbour Algorithm:

L-Load the data
M-Initialize the value of k
N-For getting the predicted class, iterate from 1 to total number of training data points

1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
2. Sort the calculated distances in ascending order based on distance values
3. Get top k rows from the sorted array.
4. Get the most frequent class of these rows i.e Get the labels of the selected K entries.
5. Return the predicted class. If regression, return the mean of the K labels 🎬 If classification, return the mode of the K labels.



Confusion matrix:

Note;

- Class 1 : Positive
- Class 2 : Negative

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative. (Also known as a "Type II error.".
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive. (Also known as a "Type I error.").

$$\textbf{ACCURACY} = \frac{\textbf{TP+TN}}{\textbf{TP+TN+FP+FN}}$$

$$\textbf{RECALL} = \frac{\textbf{TP}}{\textbf{TP+FN}}$$

$$\textbf{PRECISION} = \frac{\textbf{TP}}{\textbf{TP+FP}}$$

$$\textbf{F-MEASURE} = \frac{\textbf{2*RECALL*PRECISION}}{\textbf{RECALL+PRECISIO N}}$$

EXAMPLE:

PREDICTED: NO	**PREDICTED: YES**	
TN=50	FP=10	60
FN=5	TP=100	105
55	110	

n=165

ACTUAL = NO

ACTUAL = YES

Accuracy: Overall, how often is the classifier correct?
(TP+TN)/total = (100+50)/165 = 0.91

Misclassification Rate: Overall, how often is it wrong?
(FP+FN)/total = (10+5)/165 = 0.09
equivalent to 1 minus Accuracy also known as "Error Rate".

True Positive Rate: When it's actually yes, how often does it predict yes?
TP/actual yes = 100/105 = 0.95
also known as "Sensitivity" or "Recall".

False Positive Rate: When it's actually no, how often does it predict yes?
FP/actual no = 10/60 = 0.17

True Negative Rate: When it's actually no, how often does it predict no?
TN/actual no = 50/60 = 0.83
equivalent to 1 minus False Positive Rate also known as "Specificity".

Precision: When it predicts yes, how often is it correct?
TP/predicted yes = 100/110 = 0.91.

Prevalence: How often does the yes condition actually occur in our sample?
actual yes/total = 105/165 = 0.64

Source Code:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

import pandas as pd

dataset=pd.read_csv("iris.csv")

X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.25)

classifier=KNeighborsClassifier(n_neighbors=8,p=3,metric='euclidean')
```

```
classifier.fit(X_train,y_train)

#predict the test resuts
 y_pred=classifier.predict(X_test)

cm=confusion_matrix(y_test,y_pred)
print('Confusion matrix is as follows\n',cm)
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
print(" correct predicition",accuracy_score(y_test,y_pred))
print(" worng predicition",(1-accuracy_score(y_test,y_pred)))
```

Output :

```
Confusion matrix is as follows
[[13 0 0]
[ 0 15 1]
[ 0 0 9]]
```

Accuracy Metrics:

	Precision	Recall	F1-score	Support
Iris-setosa	1.00	1.00	1.00	13
Iris-versicolor	1.00	0.94	0.97	16
Iris-virginica	0.90	1.00	0.95	9
Avg / total	0.98	0.97	0.97	38

Correct Predicition: 0.9736842105263158
Wrong Predicition: 0.02631578947368418

PROGRAM 3:

IMPLEMENT A LINEAR REGRESSION WITH A REAL DATASET

AIM:

To Implement Linear Regression with a real datasets and Experiment with different features in building a model. Tune the model's hyper parameters.

DESCRIPTION:

1. Define business object.
2. Make sense of the data from a high level.
    - data types (number, text, object, etc.)
    - continuous/discrete
    - basic stats (min, max, std, median, etc.) using boxplot
    - frequency via histogram
    - scales and distributions of different features
3. Create the training and test sets using proper sampling methods, e.g., random vs. Stratified.
4. Correlation analysis (pair-wise and attribute combinations).
5. Data cleaning (missing data, outliers, data errors).
6. Data transformation via pipelines (categorical text to number using one hot encoding, feature scaling via normalization/standardization, feature combinations).
7. Train and cross validate different models and select the most promising one (Linear Regression, Decision Tree, and Random Forest were tried in this tutorial).
8. Fine tune the model using trying different combinations of hyper parameters.
9. Evaluate the model with best estimators in the test set.
10. Launch, monitor, and refresh the model and system.

SOURCE CODE:

```
#  This Python 3 environment comes with many helpful analytic libraries installed.

# It is defined by the kaggle/python docker image: https://github.com/kaggle/ docker-python.

import numpy as np

import pandas as pd

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns
```

```
# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input
 Directory

import os

print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['anscombe.csv', 'housing.csv']
```

```
# loading data

data_path = "../input/housing.csv"

housing = pd.read_csv(data_path)

# see the basic info

housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>

RangeIndex: 20640 entries, 0 to 20639

Data columns (total 10 columns):

longitude            20640 non-null float64

latitude             20640 non-null float64

housing_median_age    20640 non-null float64

total_rooms           20640 non-null float64

total_bedrooms        20433 non-null float64

population            20640 non-null float64

households            20640 non-null float64

median_income         20640 non-null float64

median_house_value    20640 non-null float64

ocean_proximity       20640 non-null object
```

dtypes: float64(9), object(1)

memory usage: 1.6+ MB

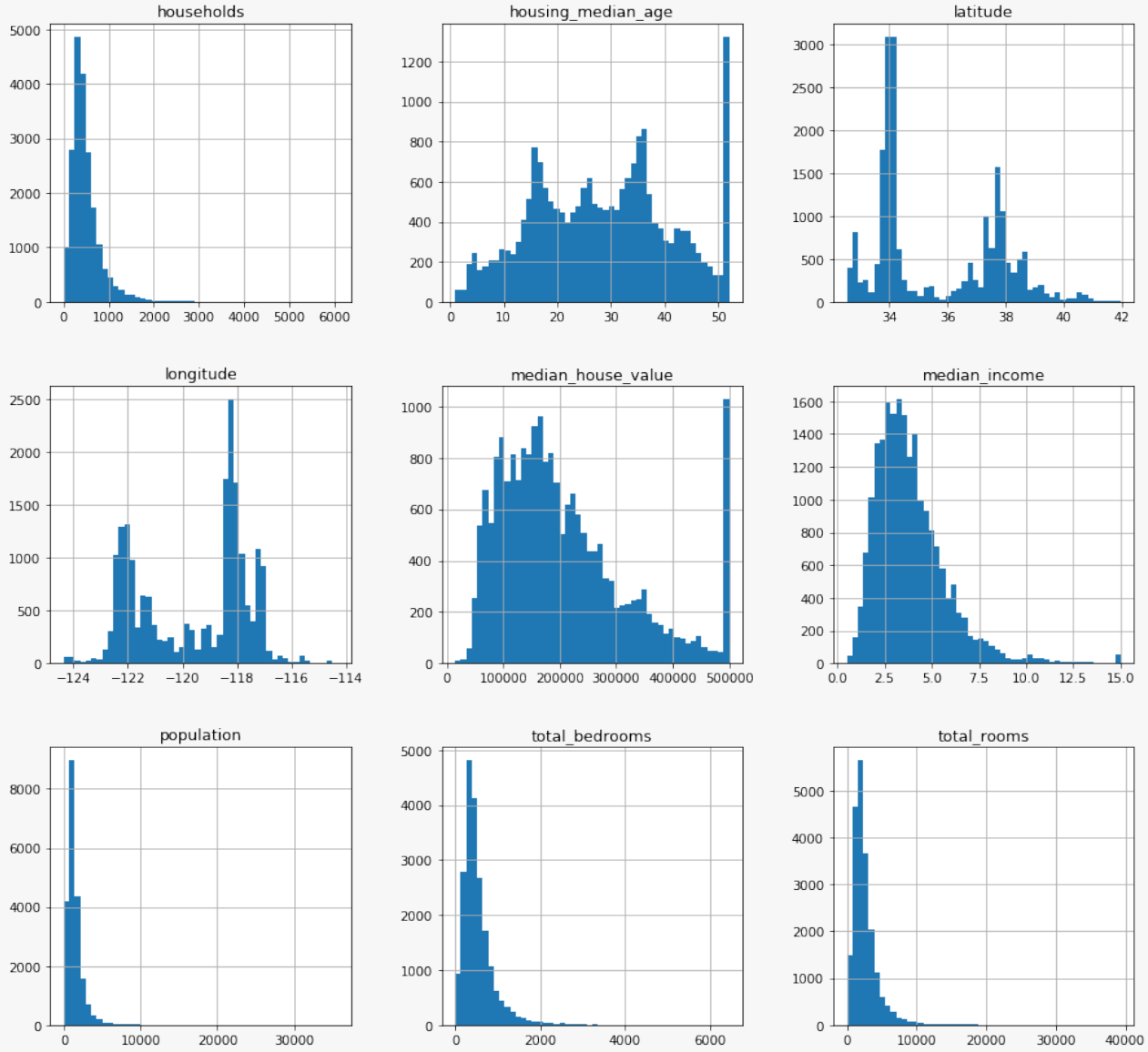
Input(3): housing.head(10)

Input(4): housing.describe()

Input(5): housing.boxplot(['median_house_value'], figsize=(10, 10))

Input(6): housing.hist(bins=50, figsize=(15, 15))

Output(6):




```
Input(7): housing['ocean_proximity'].value_counts()
```

```
Output(7):
```

<1H OCEAN 9136

INLAND 6551

NEAR OCEAN 2658

NEAR BAY 2290

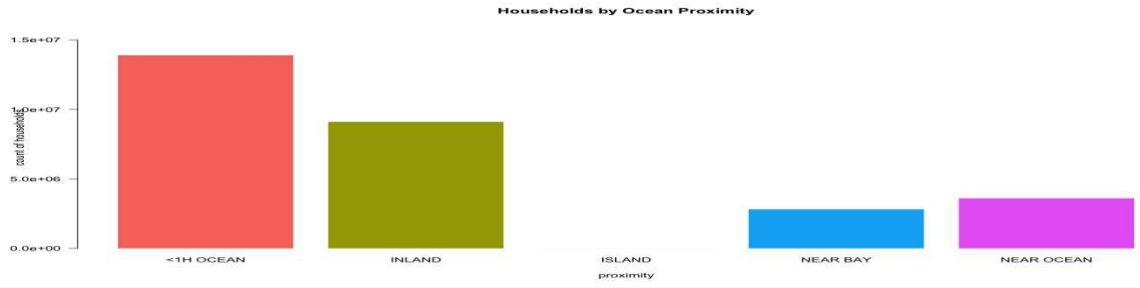
ISLAND 5

Name: ocean_proximity, dtype: int64

Input(8):

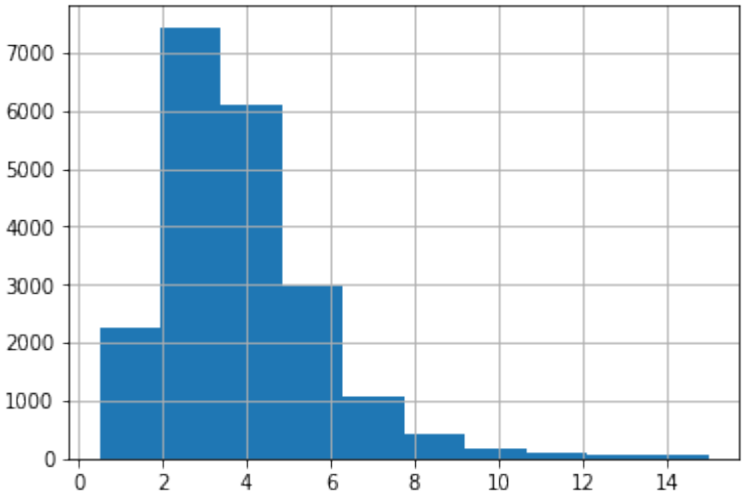
```
op_count = housing['ocean_proximity'].value_counts()
plt.figure(figsize=(10,5))
sns.barplot(op_count.index, op_count.values, alpha=0.7)
plt.title('Ocean Proximity Summary')

plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Ocean Proximity', fontsize=12)
plt.show()
# housing['ocean_proximity'].value_counts().hist()
```

Output(8):



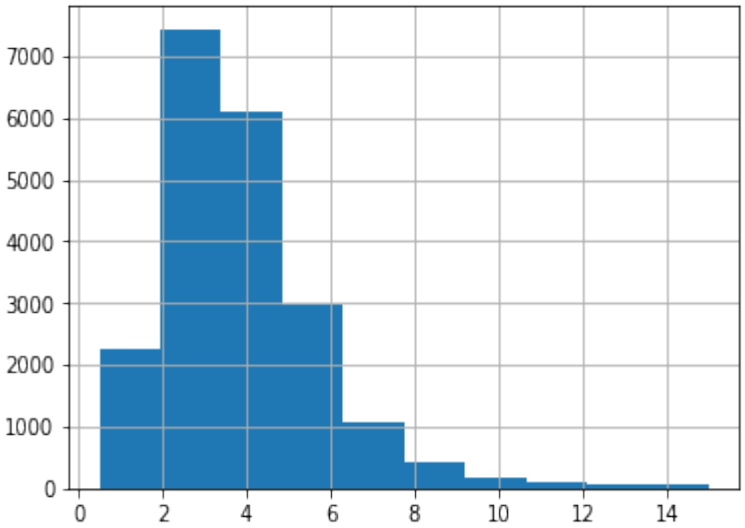
Input(9): housing['median_income'].hist()

Output(9): <matplotlib.axes._subplots.AxesSubplot at 0x7f264523cb00>



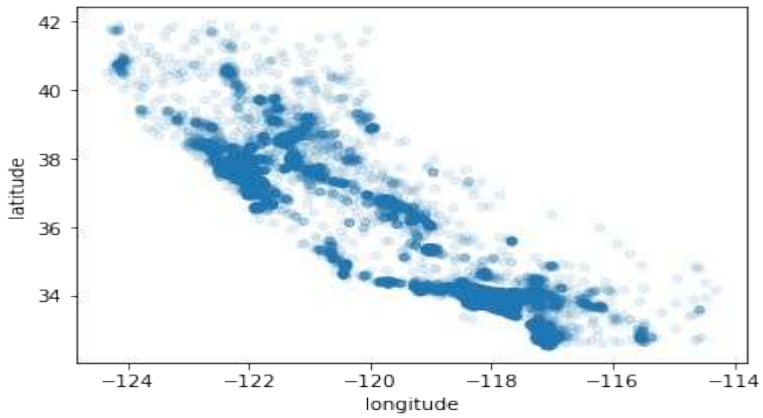
Input(10): housing['median_income'].hist()

Output(10): <matplotlib.axes._subplots.AxesSubplot at 0x7f264523cb00>



Input(11):housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)

Output(11): <matplotlib.axes._subplots.AxesSubplot at 0x7f2645224e10>



Input(12):

```
# Pearson's r, aka, standard correlation coefficient for every pair
corr_matrix = housing.corr()
# Check the how much each attribute correlates with the median house value
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Output(12):

```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64.
```

PROGRAM 4:

THE SCIKIT-LEARNS's KNN CLASSIFIER TO CLASSIFY REAL vs. FAKE NEWS HEADLINES.

AIM:

The aim of this question is for you to read the scikit-learn API and get comfortable with training/validation splits. Use California Housing Datasets.

DESCRIPTION:

Classification with Nearest Neighbors. In this question, you will use the scikit-learn's KNN classifier to classify real vs. fake news headlines. The aim of this question is for you to read the scikit-learn API and get comfortable with training/validation splits. Use California Housing Datasets.

SOURCE CODE:

```
import csv

import random

import math

import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[])
:
with open(filename, 'rb') as csvfile
:
lines = csv.reader(csvfile)

dataset = list(lines)

for x in range(len(dataset)-1):

for y in range(4):

dataset[x][y] = float(dataset[x][y])

if random.random() < split: trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):

distance = 0
```

```
for x in range(length):
distance += pow((instance1[x] - instance2[x]), 2)
return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
distances = []
length = len(testInstance)-1
for x in range(len(trainingSet)):
 dist = euclideanDistance(testInstance, trainingSet[x], length) distances.append((trainingSet[x], dist))
 distances.sort(key=operator.itemgetter(1))
 neighbors = []
 for x in range(k):
neighbors.append(distances[x][0])
return neighbors
def getResponse(neighbors):
 classVotes = {}
for x in range(len(neighbors)):
 response = neighbors[x][-1]
 if response in classVotes:
 classVotes[response] += 1
else:
 classVotes[response] = 1

sortedVotes =
 sorted(classVotes.iteritems(),
```

```
reverse=True)
  return sortedVotes[0][0]

  def getAccuracy(testSet,
  predictions): correct = 0
  for x in
  range(len(testSet)):
  key=operator.itemgetter(1);
  if testSet[x][-1] == predictions[x]:
  correct += 1
  return (correct/float(len(testSet))) * 100.0

  def main():
  # prepare
  Data
  trainingSet=
  [] testSet=[]
  split = 0.67
  loadDataset('knndat.data', split, trainingSet,
   testSet) print('Train set: ' + repr(len(trainingSet)))
  print('Test set: ' + repr(len(testSet)))
  # generate
  predictions
  predictions=[]
  k=3
   for x in range(len(testSet)):
```

```
neighbors = getNeighbors(trainingSet, testSet[x],

k) result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-

) accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) +

1])   '%') main()
```

OUTPUT:

Confusion matrix is as follows:

```
[[11 0 0]
 [0 9 1]
 [0 1 8]]
```

Accuracy metrics:

0 1.00 1.00 1.00 11

1 0.90 0.90 0.90 10

2 0.89 0.89 0.89 9

Avg/Total: 0.93 0.93 0.93 30.

PROGRAM 5:

ANALYZE DELTAS BETWEEN TRAINING SET AND VALIDATION SET RESULTS

AIM:

To implement the experiment with validation sets and test sets using the datasets.

DESCRIPTION:

To experiment with validation sets and test sets using the datasets. Split a training set into a smaller training set and a validation set. Analyze deltas between training set and validation set results. Test the trained model with a test set to determine whether your trained model is over fitting. Detect and fix a common training problem.

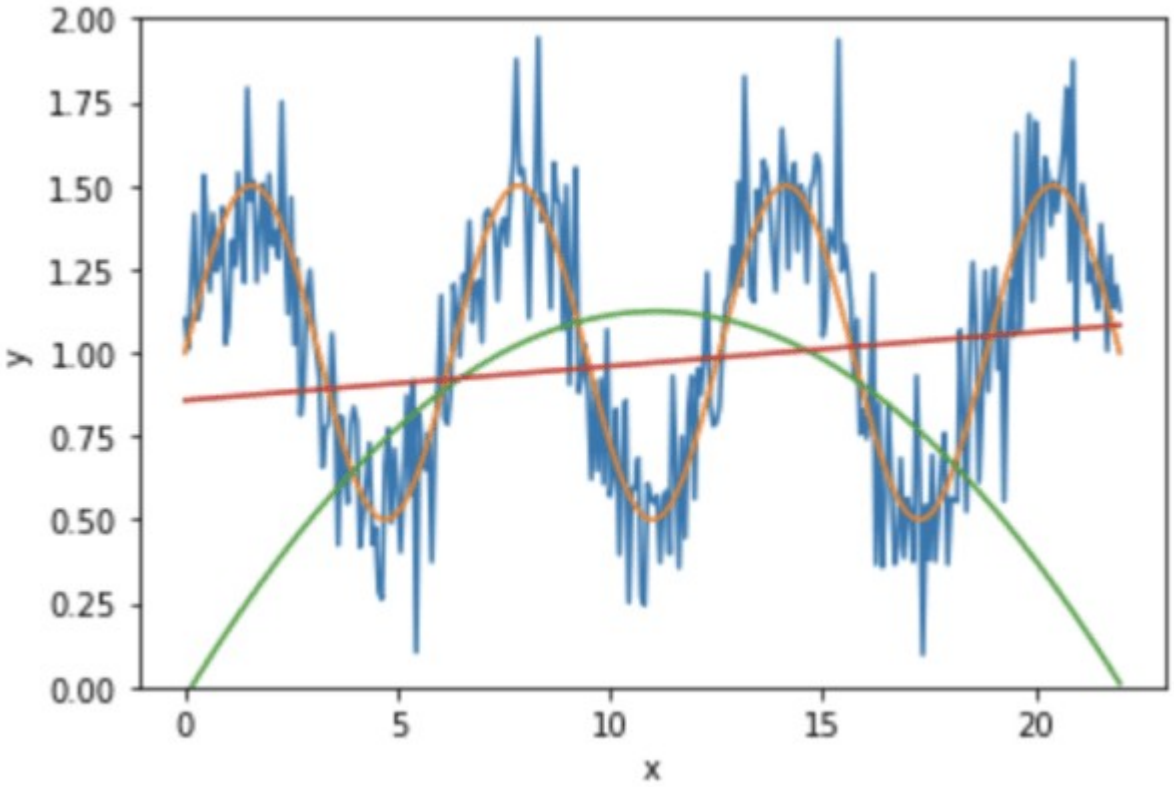
SOURCE CODE:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_validate, train_test_split
from sklearn.preprocessing import Polynomial Features, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
np.random.seed(42)
# Generate data and plot
N = 300
x = np.linspace(0, 7*np.pi, N)
smooth = 1 + 0.5*np.sin(x)
y = smooth + 0.2*np.random.randn(N)
plt.plot(x, y)
plt.plot(x, smooth)
plt.xlabel("x")
plt.ylabel("y")
plt.ylim(0,2)
plt.show()
```

```
# Train-test split, intentionally use shuffle=False
X = x.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, shuffle=False)

# Create two models: Polynomial and linear regression
degree = 2
polyreg = make_pipeline(Polynomial Features(degree), LinearRegression(fit_intercept=False))
linreg = LinearRegression()
# Cross-validation
scoring = "neg_root_mean_squared_error"
polyscores = cross_validate(polyreg, X_train, y_train, scoring=scoring, return_estimator=True)
linscores = cross_validate(linreg, X_train, y_train, scoring=scoring, return_estimator=True)
# Which one is better? Linear and polynomial
print("Linear regression score:", linscores["test_score"].mean())
print("Polynomial regression score:", polyscores["test_score"].mean())
print("Difference:", linscores["test_score"].mean() - polyscores["test_score"].mean())
print("Coefficients of polynomial regression and linear regression:")
# Let's show the coefficient of the last fitted polynomial regression
# This starts from the constant term and in ascending order of powers
print(polyscores["estimator"][0].steps[1][1].coef_)
# And show the coefficient of the last-fitted linear regression
print(linscores["estimator"][0].intercept_, linscores["estimator"][-1].coef_)
# Plot and compare
plt.plot(x, y)
plt.plot(x, smooth)
```

```
plt.plot(x, polyscores["estimator"][0].predict(X))
plt.plot(x, linscores["estimator"][0].predict(X))
plt.ylim(0,2)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
# Retrain the model and evaluate
import sklearn
linreg = sklearn.base.clone(linreg)
linreg.fit(X_train, y_train)
print("Test set RMSE:", mean_squared_error(y_test, linreg.predict(X_test), squared=False))
print("Mean validation RMSE:", -linscores["test_score"].mean()).
```

OUTPUT:



PROGRAM 6:

IMPLEMENT A BINARY CLASSIFICATION MODEL

AIM:

Implement a binary classification model. Binary question such as "Are houses in this neighborhood above a

certain price?

DESCRIPTION:

Implement a binary classification model. Binary question such as "Are houses in this neighborhood above a certain price?"(use data from exercise 1). Modify the classification threshold and determine how that modification influences the model. Experiment with different classification metrics to determine your model's effectiveness.

SOURCE CODE:

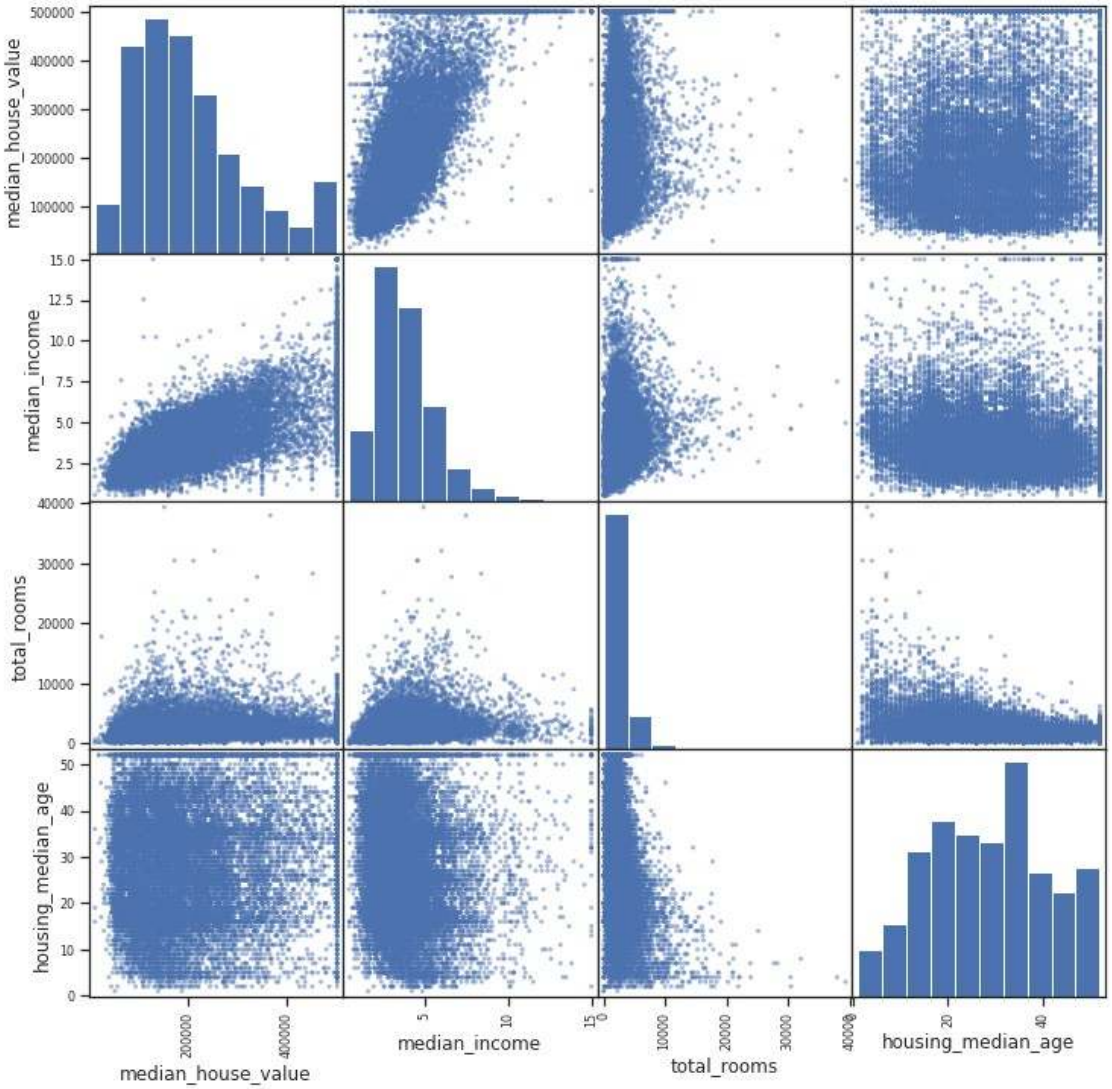
```
train_df = pd.read_csv("https://download.mlcc.google.com/mledu datasets/california_housing_train.csv")

test_df = pd.read_csv("https://download.mlcc.google.com/mledu-datasets/california_housing_test.csv")

train_df = train_df.reindex(np.random.permutation(train_df.index))

# shuffle the training set

threshold = 265000 # This is the 75th percentile for median house values.

train_df_norm["median_house_value_is_high"] = ? Your code here

test_df_norm["median_house_value_is_high"] = ? Your code here

# Print out a few example cells from the beginning and

# middle of the training set, just to make sure that

# your code created only 0s and 1s in the newly created

# median_house_value_is_high column

train_df_norm["median_house_value_is_high"].head(8000)

inputs = {

# Features used to train the model on.
  'median_income': tf.keras.Input(shape=(1,)),

  'total_rooms': tf.keras.Input(shape=(1,))

}


# The following variables are the hyperparameters.
```

```
learning_rate = 0.001

epochs = 20

batch_size = 100

classification_threshold = 0.35

label_name = "median_house_value_is_high"


# Modify the following definition of METRICS to generate

# not only accuracy and precision, but also recall:

METRICS = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy',

                                    threshold=classification_threshold),
    tf.keras.metrics.Precision(thresholds=classification_threshold,

                               name='precision'
                               ),
    ?  # write code here
]
```

```
# Establish the model's topography.

my_model = create_model(inputs, learning_rate, METRICS)
```

```
# Train the model on the training set.

epochs, hist = train_model(my_model, train_df_norm, epochs,

                           label_name, batch_size)
# Plot metrics vs. Epochs
list_of_metrics_to_plot = ['accuracy', 'precision', 'recall']

plot_curve(epochs, hist, list_of_metrics_to_plot)
```

OUTPUT:



PROGRAM 7:

IMPLEMENT THE FINITE WORDS CLASSIFICATION SYSTEM USING BACKPROPAGATIONALGORITHM

AIM:

To implement the finite words classification system using Back-propagation algorithm.

DESCRIPTION:

What is back propagation?

We can define the back propagation algorithm as an algorithm that trains some given feed forward Neural Network for a given input pattern where the classifications are known to us. At the point when every passage of the example set is exhibited to the network, the network looks at its yield reaction to the example input pattern. After that, the comparison done between output response and expected output with the error value is measured. Later, we adjust the connection weight based upon the error value measured.

It was first introduced in the 1960s and 30 years later it was popularized by David Rumelhart, Geoffrey Hinton, and Ronald Williams in the famous 1986 paper. In this paper, they spoke about the various neural networks. Today, back propagation is doing good. Neural network training happens through back propagation. By this approach, we fine-tune the weights of a neural net based on the error rate obtained in the previous run. The right manner of applying this technique reduces error rates and makes the model more reliable. Back propagation is used to train the neural network of the chain rule method. In simple terms, after each feed-forward passes through a network, this algorithm does the backward pass to adjust the model's parameters based on weights and biases. A typical supervised learning algorithm attempts to find a function that maps input data to the right output. Back propagation works with a multi-layered neural network and learns internal representations of input to output mapping.

The Back propagation algorithm is a supervised learning method for multi layer feed forward networks from the field of Artificial Neural Networks.

Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to other cell's dendrites.

The principle of the back propagation approach is to model a given function by modifying internal

weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.

Technically, the back propagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer.Back propagation can be used for both classification and regression problems.

In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as [1, 0] and [0, 1] for A and B respectively. This is called a one hot encoding.

How does back propagation work?

Let us take a look at how back propagation works. It has four layers: input layer, hidden layer, hidden layer II and final output layer. So, the main three layers are:

1. Input layer
2. Hidden layer
3. Output layer.

Each layer has its own way of working and its own way to take action such that we are able to get the desired results and correlate these scenarios to our conditions. Let us discuss other details needed to help summarizing this algorithm.

SOURCE CODE:

```
import pandas as pd
fromsklearn.model_selection import train_test_split
fromsklearn.feature_extraction.text import
```

```
CountVectorizer from sklearn.neural_network

import MLPClassifier

fromsklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score

msg = pd.read_csv('document.csv',

names=['message', 'label']) print("Total Instances of

Dataset: ", msg.shape[0]) msg['labelnum'] =

msg.label.map({'pos': 1, 'neg': 0})

X =

msg.me

ssage y

=

msg.lab

Elnum

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)

count_v = CountVectorizer()

Xtrain_dm =

count_v.fit_transform(Xtrain)

Xtest_dm =

count_v.transform(Xtest)

df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())

clf = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2),

random_state=1)

clf.fit(Xtrain_dm, ytrain) pred = clf.predict(Xtest_dm)

print('Accuracy Metrics:')
```

print('Accuracy: ', accuracy_score(ytest, pred)) print('Recall: ', recall_score(ytest, pred))

print('Precision: ', precision_score(ytest, pred))

print('Confusion Matrix: \n', confusion_matrix(ytest, pred)),

document.csv:

I love this sandwich

,pos This is

an amazingplace,

pos KG CO

I feel very good about these

beers,pos This is my best

work,pos

What an awesome view,pos

I do not like this

restaurant,neg I am

tired of this stuff,neg

I can't deal with this,neg

He is my sworn enemy,ne

My boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this

juice,neg I love to dance,pos

I am sick and tired

of this place,neg

What a great holiday,pos

That is a bad locality to stay

,neg We will have good fun

tomorrow,pos I went to my

enemy's house today,neg

OUTPUT:

Total Instances of Dataset: 18

Accuracy Metrics:

Accuracy: 0.8

Recall: 1.0

Precisio n: 0.75

Confusi on Matrix:

[[1 1]

[0 3]}

VIVA QUESTIONS:

What is machine learning?

Define supervised learning?

Define unsupervised learning?

Define semi supervised learning?

Define reinforcement learning?

What do you mean by hypotheses is classification?

What is clustering?

Define precision, accuracy and recall 10.Define entropy?

Define regression?

How Knn is different from k-means clustering?

What is concept learning?

Define specific boundary and general boundary?

Define target function 16.Define decision tree?

What is ANN?

Explain gradient descent approximation?

State Bayes theorem?

Define Bayesian belief networks?

Differentiate hard and soft clustering?

Define variance 23. What is inductive machine learning?

Why K nearest neighbour algorithm is lazy learning algorithm?

Why naïve Bayes is naïve?

Mention classification algorithms?

Define pruning 28.Differentiate Clustering and classification?

Mention clustering algorithms and Define Bias?