HITAM
find your path

HYDERABAD INSTITUTE OF TECHNOLOGY AND MANAGEMENT

» ACCREDITATED BY NAAC, NBA [CSE, ECE] » APPROVED BY AICTE » AFFILIATED TO JNTUH

CERTIFICATE

Laboratory _____ Academic Year _____

Certified that this is the bonafide record of work done by

Mr./Ms./Mrs. _____

of _____ year B.Tech / M.Tech _____ semester,

Branch _____ with Hall Ticket No. _____ and

performed _____ no. of experiments under my supervision.

_____ _____

Faculty In-charge
Date:

Head of the Department
(with seal)

_____ _____

Internal Examiner

External Examiner

Gowdavelly, Near Kompally, Medchal (Dist.), Telangana. 501401. email: info@hitam.org | www.hitam.org

INDEX

HITAM
find your path

S.No	Name of the Experiment	Page #	Date		Marks	Initial
			Conducted	Submitted		

Average Marks

INDEX

HITAM
find your path

S.No	Name of the Experiment	Page #	Date		Marks	Initial
			Conducted	Submitted		

Average Marks

1.The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

```
      #Probability that it is Friday and that a student is absent =
0.03

->FandA= 0.03

#Probability it is Friday = 0.2
->F=0.2

# We need to find the probability that a student is absent given that
today is Friday.

# So, We will use Conditional Probability

->print("student is absent|today is Friday=",FandA/F)
#print(FandA/F)
```

Output:

```
student is absent|today is Friday= 0.15
```

2. Extract the data from database using python

```
->import pandas as pd
->dataset=pd.read_csv("student.csv")
->dataset
```

Output:

	Maths	Physics	Chemistry	Result
0	17	27	22	0
1	72	82	77	1
2	97	18	13	0
3	8	42	37	0
4	32	25	20	0
...	...	...	...	...
995	4	48	64	0
996	63	22	88	0
997	90	64	43	1
998	67	41	6	0
999	92	74	9	0

1000 rows × 4 columns

```
->dataset.head()
```

Output:

	Maths	Physics	Chemistry	Result
0	17	27	22	0
1	72	82	77	1
2	97	18	13	0
3	8	42	37	0
4	32	25	20	0
5	15	73	68	0
6	63	67	62	1
7	97	70	65	1
8	57	93	88	1
9	60	58	53	1

```
->dataset.tail()
```

Output:

	Maths	Physics	Chemistry	Result
995	4	48	64	0
996	63	22	88	0
997	90	64	43	1
998	67	41	6	0
999	92	74	9	0

```
->dataset.tail(10)
```

Output:

	Maths	Physics	Chemistry	Result
990	76	50	77	1
991	2	26	53	0
992	96	43	37	0
993	27	58	21	0
994	87	24	15	0
995	4	48	64	0
996	63	22	88	0
997	90	64	43	1
998	67	41	6	0
999	92	74	9	0

3. Implement k-nearest neighbours classification using python

```
->import pandas as pd
->dataset = pd.read_csv('/content/drive/MyDrive/dataset/diabetes.csv')
->dataset.shape
```

Output:

```
(768, 9)
```

```
->dataset.head()
```

Output:

	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

```
->dataset.tail()
```

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
->dataset.head()
```

Output:

	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

```
->X = dataset.drop('Outcome',axis=1).values
  y = dataset['Outcome'].values

->from sklearn.model_selection import train_test_split
  X_train,X_test,y_train,y_test =
  train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)

->from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  scaler.fit(X_train)
  X_train = scaler.transform(X_train)
  X_test = scaler.transform(X_test)

->from sklearn.neighbors import KNeighborsClassifier
  classifier = KNeighborsClassifier(n_neighbors = 8)
  classifier.fit(X_train, y_train)
```

Output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                     weights='uniform')

->from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
  result = confusion_matrix(y_test, y_pred)
  print("Confusion Matrix:")
  print(result)
  result1 = classification_report(y_test, y_pred)
  print("Classification Report:",)
  print (result1)
  result2 = accuracy_score(y_test,y_pred)
  print("Accuracy:",result2)
```

Output:

```
Confusion Matrix:
[[176  25]
 [ 54  53]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       201
           1       0.68      0.50      0.57       107

    accuracy                           0.74       308
   macro avg       0.72      0.69      0.69       308
weighted avg       0.74      0.74      0.73       308

Accuracy: 0.7435064935064936
```

4. Implement linear regression using python.

```
->import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
print(x)
```

Output:

```
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
```

```
->model = LinearRegression()
  model.fit(x, y)
  model = LinearRegression().fit(x, y)
  r_sq = model.score(x, y)
  print('coefficient of determination:', r_sq)
```

Output:

```
coefficient of determination: 0.7158756137479542
```

```
->print('intercept:', model.intercept_)
```

Output:

```
intercept: 5.633333333333329
```

```
->print('slope:', model.coef_)
```

Output:

```
slope: [0.54]
```

```
->y_pred = model.predict(x)
  print('predicted response:', y_pred, sep='\n')
```

Output:

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

```
->y_pred = model.intercept_ + model.coef_ * x
  print('predicted response:', y_pred, sep='\n')
```

Output:

```
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

```
->x_new = np.arange(5).reshape((-1, 1))
print(x_new)
```

Output:

```
[[0]
 [1]
 [2]
 [3]
 [4]]
```

```
->y_new = model.predict(x_new)
print(y_new)
```

Output:

```
[5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]
```

5. Implement Naïve Bayes theorem to classify the English text

```
->#data importing
import pandas as pd

columns = ['sent', 'class']
rows = []

rows = [['This is my book', 'stmt'],
        ['They are novels', 'stmt'],
        ['have you read this book', 'question'],
        ['who is the author', 'question'],
        ['what are the characters', 'question'],
        ['This is how I bought the book', 'stmt'],
        ['I like fictions', 'stmt'],
        ['what is your favorite book', 'question']]

training_data = pd.DataFrame(rows, columns=columns)
training_data
```

Output:

	sent	class
0	This is my book	stmt
1	They are novels	stmt
2	have you read this book	question
3	who is the author	question
4	what are the characters	question
5	This is how I bought the book	stmt
6	I like fictions	stmt
7	what is your favorite book	question

```
->#Term-Document Matrix (TDM) for 'statement' class
from sklearn.feature_extraction.text import CountVectorizer

stmt_docs = [row['sent'] for index,row in training_data.iterrows() if
row['class'] == 'stmt']

vec_s = CountVectorizer()
X_s = vec_s.fit_transform(stmt_docs)
tdm_s = pd.DataFrame(X_s.toarray(), columns=vec_s.get_feature_names())
```

```
tdm_s
```

Output:

	are	book	bought	fictions	how	is	like	my	novels	the	they	this
0	0	1	0	0	0	1	0	1	0	0	0	1
1	1	0	0	0	0	0	0	0	1	0	1	0
2	0	1	1	0	1	1	0	0	0	1	0	1
3	0	0	0	1	0	0	1	0	0	0	0	0

```
->#Term-Document Matrix (TDM) for 'question' class.
q_docs = [row['sent'] for index,row in training_data.iterrows() if
row['class'] == 'question']

vec_q = CountVectorizer()
X_q = vec_q.fit_transform(q_docs)
tdm_q = pd.DataFrame(X_q.toarray(), columns=vec_q.get_feature_names())

tdm_q
```

Output:

	are	author	book	characters	favorite	have	is	read	the	this	what	who	you	your
0	0	0	1	0	0	1	0	1	0	1	0	0	1	0
1	0	1	0	0	0	0	1	0	1	0	0	1	0	0
2	1	0	0	1	0	0	0	0	1	0	1	0	0	0
3	0	0	1	0	1	0	1	0	0	0	1	0	0	1

```
->#Frequency of words for Statement Class:
word_list_s = vec_s.get_feature_names();
count_list_s = X_s.toarray().sum(axis=0)
freq_s = dict(zip(word_list_s,count_list_s))
Freq_s
```

Output:

```
{'are': 1,
 'book': 2,
 'bought': 1,
 'fictions': 1,
 'how': 1,
 'is': 2,
 'like': 1,
 'my': 1,
 'novels': 1,
```

```
 'the': 1,
 'they': 1,
 'this': 2}
```

```
->#Frequency of words for Question Class:
word_list_q = vec_q.get_feature_names();
count_list_q = X_q.toarray().sum(axis=0)
freq_q = dict(zip(word_list_q,count_list_q))
freq_q
```

Output:

```
{'are': 1,
 'author': 1,
 'book': 2,
 'characters': 1,
 'favorite': 1,
 'have': 1,
 'is': 2,
 'read': 1,
 'the': 2,
 'this': 1,
 'what': 2,
 'who': 1,
 'you': 1,
 'your': 1}
```

```
->#Probabilities of words in Statement Class:
prob_s=[]
for word,count in zip(word_list_s,count_list_s):
prob_s.append(count/len(word_list_s))
dict(zip(word_list_s,prob_s))
```

Output:

```
{'are': 0.08333333333333333,
 'book': 0.16666666666666666,
 'bought': 0.08333333333333333,
 'fictions': 0.08333333333333333,
 'how': 0.08333333333333333,
 'is': 0.16666666666666666,
 'like': 0.08333333333333333,
 'my': 0.08333333333333333,
 'novels': 0.08333333333333333,
 'the': 0.08333333333333333,
 'they': 0.08333333333333333,
 'this': 0.16666666666666666}
```

```
->#Probabilities of words in Question Class:
prob_q = []
for count in count_list_q:
prob_q.append(count/len(word_list_q))
dict(zip(word_list_q,prob_q))
```

Output:

```
{'are': 0.07142857142857142,
 'author': 0.07142857142857142,
 'book': 0.14285714285714285,
 'characters': 0.07142857142857142,
 'favorite': 0.07142857142857142,
 'have': 0.07142857142857142,
 'is': 0.14285714285714285,
 'read': 0.07142857142857142,
 'the': 0.14285714285714285,
 'this': 0.07142857142857142,
 'what': 0.14285714285714285,
 'who': 0.07142857142857142,
 'you': 0.07142857142857142,
 'your': 0.07142857142857142}
```

```
->#Total count of all features in the training set
from sklearn.feature_extraction.text import CountVectorizer

docs = [row['sent'] for index,row in training_data.iterrows()]

vec = CountVectorizer()
X = vec.fit_transform(docs)

total_features = len(vec.get_feature_names())
total_features
```

Output:

```
21
```

6.Implement an algorithm to demonstrate the significance of genetic algorithm

```
->import numpy as np
import pandas as pd
import random
import matplotlib.pyplot
%matplotlib inline

->from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

->#import the breast cancer dataset
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
label=cancer["target"]

->#splitting the model into training and testing set
  X_train, X_test, y_train, y_test = train_test_split(df,
                                                      label, test_size=0.30,
                                                      random_state=101)

->#training a logistics regression model
  logmodel = LogisticRegression()
  logmodel.fit(X_train,y_train)
  predictions = logmodel.predict(X_test)
  print("Accuracy = "+ str(accuracy_score(y_test,predictions)))
```

Output:

```
Accuracy = 0.9239766081871345

->#defining various steps required for the genetic algorithm
def initilization_of_population(size,n_feat):
   population = []
   for i in range(size):
       chromosome = np.ones(n_feat,dtype=np.bool)
       chromosome[:int(0.3*n_feat)]=False
       np.random.shuffle(chromosome)
       population.append(chromosome)
   return population
```

```
def fitness_score(population):
    scores = []
    for chromosome in population:
        logmodel.fit(X_train.iloc[:,chromosome],y_train)
        predictions = logmodel.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,predictions))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])

def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range(n_parents):
        population_nextgen.append(pop_after_fit[i])
    return population_nextgen

def crossover(pop_after_sel):
    population_nextgen=pop_after_sel
    for i in range(len(pop_after_sel)):
        child=pop_after_sel[i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen.append(chromosome)
    #print(population_nextgen)
    return population_nextgen

def generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,
                                    X_test, y_train, y_test):
    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population(size,n_feat)
```

```
        for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_nextgen)
        print(scores[:2])
        pop_after_sel = selection(pop_after_fit,n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo,best_score

->chromo,score=generations(size=200,n_feat=30,n_parents=100,mutation_rate=0.10,

n_gen=38,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_test)
logmodel.fit(X_train.iloc[:,chromo[-1]],y_train)
predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
print("Accuracy score after genetic algorithm is=
"+str(accuracy_score(y_test,predictions)))
```

Output:

```
[0.9590643274853801, 0.9590643274853801]
Accuracy score after genetic algorithm is= 0.9415204678362573
```