University of Delhi

Atma Ram Sanatan Dharma College

NAME :- ASHUTOSH KUMAR PANDEY

Roll No:- 18081

Course :- B.Sc.(Hons) Computer Science

Subject :- Machine Learning

Practicals

Teacher :

Ms. Uma Ojha

Q1. 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. (use enjoysport.csv)

```
import pandas as pd
import numpy as np
df = pd.read_excel('enjoysport.xlsx')
df
def find_s(data):
    hypothesis = ['0'] * (len(data.columns) - 1)  # Initialize the
hypothesis as the most general hypothesis
    for _, example in data.iterrows():
        if example.iloc[-1] == 'yes':
            for i in range(len(example) - 1):
                if hypothesis[i] == '0':
                    hypothesis[i] = example[i]
                elif hypothesis[i] != example[i]:
                    hypothesis[i] = '?'
    return hypothesis
hypothesis = find_s(df)
print('Final hypothesis:', hypothesis)
```

Q2. Implement email spam classification using naive Bayes algorithm.

```
import pandas as pd
df = pd.read_csv('spam.csv')
df
df.groupby('Category').describe()
df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)
df.head()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
 train_test_split(df.Message,df.spam,test_size=0.20)
 from sklearn.feature_extraction.text import CountVectorizer
```

```
  v= CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()

from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train_count,y_train)
emails = [
    'Hey mohan, can we get together to watch football game
tomorrow?',
    'Upto 20% discount on parking, exclusive offer just for you. Dont
miss this reward!'
]
emails_count = v.transform(emails)
emails_count.toarray()
model.predict(emails_count)
x_test_count = v.transform(X_test)
model.score(x_test_count,y_test)
from sklearn.pipeline import Pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
clf.predict(emails)
```

Q3. Implement Linear regression to predict house prices using (i)Least squared method (ii) Normal equations.
(use homeprice_uni.csv)

```
import pandas as pd
import numpy as np
df = pd.read_excel('homeprices_uni.xlsx')
df
from matplotlib import pyplot as plt
from sklearn import linear_model
 %matplotlib inline
 plt.xlabel('area')
```

```
  plt.ylabel('price')
plt.scatter(df.area, df.price, color="blue",marker='+')
new_df = df.drop('price', axis='columns')
model = linear_model.LinearRegression()
model.fit(new_df, df.price)
model.predict([[3300]])
model.coef_
model.intercept_
price = model.intercept_ + model.coef_*3300
price
```

Q3. Implement Linear regression to predict house prices using gradient descent algorithm. (use homeprice_uni.csv)

LINEAR REGRESSION MULTIPLE VARIABLE

```
import pandas as pd
import numpy as np
df = pd.read_excel('homeprices_multivariate.xlsx')
df
df.tail()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
from sklearn.linear_model import LinearRegression
mulreg = LinearRegression()
new_df = df.drop('price', axis='columns')
new_df
mulreg.fit(new_df, df.price)
print("Weight: ",mulreg.coef_," Intercept: ",mulreg.intercept_)
mulreg.predict([[3000, 3, 40]])
price = mulreg.intercept_ + mulreg.coef_[0]*3000 +
mulreg.coef_[1]*3 + mulreg.coef_[2]*40
print("the predicted price is: ",price)
```

Q4. Implement Linear regression to predict house prices based on multiple variables. (use homeprice_multivariate. csv)

```
from sklearn.datasets import fetch_openml
 boston = fetch_openml('boston',version=1)
 boston
```

```
boston.feature_names
boston.target_names
import pandas as pd
df = pd.DataFrame(boston.data, columns=boston.feature_names)
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(df['RM'], boston.target)
plt.xlabel('RM')
plt.ylabel('Price')
plt.title('Price vs RM')
df
df.drop(['CHAS'], axis=1, inplace=True)
df
df[boston.target_names[0]] = boston.target
df
df.isna().any()
X = df.drop(boston.target_names[0], axis=1)
y = df[boston.target_names[0]]
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
x_train = x_train.apply(pd.to_numeric, errors='coerce')
y_train = y_train.apply(pd.to_numeric, errors='coerce')
x_test = x_test.apply(pd.to_numeric, errors='coerce')
y_test = y_test.apply(pd.to_numeric, errors='coerce')
x_train = x_train.fillna(0)
y_train = y_train.fillna(0)
x_test = x_test.fillna(0)
y_test = y_test.fillna(0)
```

L1 Regularized: Lasso And L2 Regularized: Ridge

```
from sklearn.linear_model import Lasso, Ridge
model_lasso = Lasso(alpha=0.1,max_iter=100,tol=0.1)
model_ridge = Ridge(alpha=0.1,max_iter=100,tol=0.1)
model_lasso.fit(x_train, y_train)

model_ridge.fit(x_train, y_train)
model_lasso.score(x_train, y_train)
```

```
  model_lasso.score(x_test, y_test)
model_ridge.score(x_train, y_train)
model_ridge.score(x_test, y_test)
Normal Linear Regression
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression().fit(x_train, y_train)
model_lr.score(x_train, y_train)
model_lr.score(x_test, y_test)
```

Q5. Implement Linear regression to predict house prices based on multiple variables using regularization techniques and explain how regularisation overcome overfitting problem. (use inbuilt dataset boston- from sklearn.datasets import load_boston)

```
from sklearn.naive_bayes import MultinomialNB, GaussianNB
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
df
df.drop(['Name', 'SibSp', 'Parch', 'Ticket', 'Cabin',
'Embarked','PassengerId','Fare'], axis='columns', inplace=True)

df
df.Age = df.Age.fillna(df.Age.mean())
df
from sklearn.model_selection import train_test_split
df['female'] = df['Sex'].apply(lambda x: 1 if x == "female" else 0)
df
df.drop(['Sex'], axis='columns', inplace=True)
df
model = MultinomialNB()
model2 = GaussianNB()
X_train, X_test, Y_train, Y_test = train_test_split(df.drop(['Survived'],
axis='columns'),df['Survived'],test_size = 0.20)
model.fit(X_train, Y_train)
model2.fit(X_train, Y_train)
model.score(X_test, Y_test)

 model2.score(X_test, Y_test)
```

```
  model.predict(X_test)
model2.predict(X_test)
X_test
```

Q6. Implement titanic survival prediction using Naive Bayes algorithm.

```
import pandas as pd
import numpy as np
df = pd.read_excel('insurance_data.xlsx')
df
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(df['age'], df['bought_insurance'], color='blue',marker='+')
plt.xlabel('Age')
plt.ylabel('Insurance')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],
df.bought_insurance, test_size=0.2)
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)
Y_pred = model.predict(X_test)
X_test
y_test
Y_pred
```

Q7. Predict a person would buy life insurance based on his age using logistic regression. (insurance_data.csv)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
 from tensorflow.keras.layers import Dense
 from tensorflow.keras.datasets import mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train_flat)
# Logistic Regression model for classification
lr_model = LogisticRegression(multi_class='multinomial',
solver='lbfgs')
lr_model.fit(x_train_scaled, y_train)

# Define neural network architecture
nn_model = Sequential()
nn_model.add(Dense(128, activation='relu', input_shape=(784,)))  #
Input layer with 784 features
nn_model.add(Dense(64, activation='relu'))  # Hidden layer with 64
neurons
nn_model.add(Dense(10, activation='softmax'))  # Output layer with
10 neurons (one for each digit)

# Compile the model
nn_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
nn_model.fit(x_train_flat, y_train, epochs=10, batch_size=32,
validation_data=(x_test_flat, y_test))

# Evaluate Logistic Regression model
lr_score = lr_model.score(x_test_flat, y_test)
print("Logistic Regression Accuracy:", lr_score)

# Evaluate Neural Network model
nn_loss, nn_accuracy = nn_model.evaluate(x_test_flat, y_test)
print("Neural Network Accuracy:", nn_accuracy)
```

Q8. Implement neural network or Logistic regression to recognise hand writen digits.

```
import pandas as pd
import numpy as np
df = pd.read_csv('income.csv')
df
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(df['Age'], df['Income($)'])
xlabel = 'Age'
ylabel = 'Income($)'
plt.xlabel(xlabel)
plt.ylabel(ylabel)

from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])
y_predicted
df['Cluster'] = y_predicted
df.head()
km.cluster_centers_
df1 = df[df.Cluster==1]
df2 = df[df.Cluster==2]
df0 = df[df.Cluster==0]
plt.scatter(df0.Age, df0['Income($)'], label = 'Cluster 0',color='black')
plt.scatter(df1.Age, df1['Income($)'], color='green', label = 'Cluster
1')
plt.scatter(df2.Age, df2['Income($)'], color='red', label = 'Cluster 2')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='pu
rple',marker='*',label='centroid')
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.legend()
Scaler = MinMaxScaler()

 Scaler.fit(df[['Income($)']])
```

```
  df['Income($)'] = Scaler.transform(df[['Income($)']])
Scaler.fit(df[['Age']])
df['Age'] = Scaler.transform(df[['Age']])
df.head()
plt.scatter(df.Age,df['Income($)'])
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
df['Cluster']=y_predicted
df.head()
km.cluster_centers_
df1 = df[df.Cluster==0]
df2 = df[df.Cluster==1]
df3 = df[df.Cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='pu
rple',marker='*',label='centroid')
plt.legend()
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Q9. Implement K-means clustring.

```
import pandas as pd
import numpy as np
def entropy(target_col):
    elements,counts = np.unique(target_col,return_counts = True)
```

```
    entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
  return entropy
def InfoGain(data,split_attribute_name,target_name="class"):
  total_entropy = entropy(data[target_name])
  vals,counts= np.unique(data[split_attribute_name],return_counts=True)
  Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i in range(len(vals))])
  Information_Gain = total_entropy - Weighted_Entropy
  return Information_Gain
def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class = None):
  if len(np.unique(data[target_attribute_name])) <= 1:
    return np.unique(data[target_attribute_name])[0]
  elif len(data)==0:
    return np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name],return_counts=True)[1])]
  elif len(features) ==0:
    return parent_node_class
  else:
    parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],return_counts=True)[1])]
    item_values = [InfoGain(data,feature,target_attribute_name) for feature in features]
    best_feature_index = np.argmax(item_values)
    best_feature = features[best_feature_index]
    tree = {best_feature:{}}
    features = [i for i in features if i != best_feature]
    for value in np.unique(data[best_feature]):
      value = value
      sub_data = data.where(data[best_feature] == value).dropna()
```

```
            subtree =

ID3(sub_data,originaldata,features,target_attribute_name,parent_node_
class)
            tree[best_feature][value] = subtree
        return(tree)
def predict(query,tree,default = 1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result = tree[key][query[key]]
            except:
                return default
            result = tree[key][query[key]]
            if isinstance(result,dict):
                return predict(query,result)
            else:
                return result
from sklearn.model_selection import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
iris
df = pd.DataFrame(data = np.c_[iris['data'], iris['target']],columns=
iris['feature_names'] + ['target'])
df
train, test = train_test_split(df, test_size = 0.2)
features = train.columns[:-1]
features
tree = ID3(train,train,features,'target')
query = test.iloc[0,:].to_dict()
query.pop('target')
prediction = predict(query,tree,1)
print('The predicted class is:', prediction)
print('The actual class is:', test.iloc[0, -1])
query
```

Q10. Implement decision tree using ID3 algorithm.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
digits = load_digits()
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)
lrModel = LogisticRegression()
lrModel.fit(x_train, y_train)
lrModel.score(x_test, y_test)
svm = SVC()
svm.fit(x_train, y_train)
svm.score(x_test, y_test)
rfmodel = RandomForestClassifier(n_estimators=40)
rfmodel.fit(x_train, y_train)
rfmodel.score(x_test, y_test)
from sklearn.model_selection import KFold
KFold(n_splits=10,random_state=None,shuffle=False)
array = np.array([1,2,3,4,5,6,7,8,9])
for train_index, test_index in KFold(n_splits=3).split(array):
    print(array[train_index],array[test_index])
def get_score(model,x_train,x_test,y_train,y_test):
    model.fit(x_train,y_train)
    return model.score(x_test,y_test)
scores_logistic = []
scores_svm = []
scores_rf = []
for train_index, test_index in KFold(n_splits=3,shuffle=
False,random_state=None).split(digits.data,digits.target):
    X_train, X_test, Y_train, Y_test = digits.data[train_index],
digits.data[test_index], digits.target[train_index],
digits.target[test_index]
```

```
scores_logistic.append(get_score(LogisticRegression(),X_train,X_test,
Y_train,Y_test))

scores_svm.append(get_score(SVC(),X_train,X_test,Y_train,Y_test))

scores_rf.append(get_score(RandomForestClassifier(),X_train,X_test,Y
_train,Y_test))
scores_logistic
scores_rf
scores_svm
from sklearn.model_selection import cross_val_score
cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr'),
digits.data, digits.target,cv=3)

cross_val_score(SVC(gamma='auto'), digits.data, digits.target,cv=3)

cross_val_score(RandomForestClassifier(n_estimators=40),digits.data,
digits.target,cv=3)
```

Q11. Implement

- kfold cross valudation technique.
  12. Implement KNN classification technique.

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
iris.feature_names
iris.target_names
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
df['target'] = iris.target
df.head()
 df[df.target==1].head()
  df[df.target==2].head()
```

```
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])
df.head()
df1 = df[df.flower_name=='setosa']
df2 = df[df.flower_name=='versicolor']
df3 = df[df.flower_name=='virginica']
df1
df2
df3
import matplotlib.pyplot as plt
%matplotlib inline

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df1['sepal length (cm)'], df1['sepal width
(cm)'],color="green",marker='+')
plt.scatter(df2['sepal length (cm)'], df2['sepal width
(cm)'],color="blue",marker='.')
plt.scatter(df3['sepal length (cm)'], df3['sepal width
(cm)'],color="red",marker='*')
plt.legend(['setosa', 'versicolor','virginica'])
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df1['petal length (cm)'], df1['petal width
(cm)'],color="green",marker='+')
plt.scatter(df2['petal length (cm)'], df2['petal width
(cm)'],color="blue",marker='.')
plt.scatter(df3['petal length (cm)'], df3['petal width
(cm)'],color="red",marker='*')
plt.legend(['setosa', 'versicolor','virginica'])
from sklearn.model_selection import train_test_split
X = df.drop(['target','flower_name'], axis='columns')
Y = df.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
len(X_train)
len(X_test)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
```

```
  knn.fit(X_train, Y_train)
knn.score(X_test, Y_test)
ypred = knn.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, ypred)
cm
%matplotlib inline
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
from sklearn.metrics import classification_report
print   (classification_report(Y_test, ypred))
```