Assignment No: 1 - Linear and Logistic Regression

1. **Set A1 - Create 'sales' Data set having 5 columns namely: ID, TV, Radio, Newspaper and Sales.(random 500 entries) Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets. then divide the training and testing sets into a 7:3 ratio, respectively and print them. Build a simple linear regression model.**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
df = pd.read_csv('/home/ctbora/Downloads/Advertising.csv') # Importing the data set
new_df = df[['TV','Sales']]
X = np.array(new_df[['TV']]) # Storing into X the 'Engine HP' as np.array
y = np.array(new_df[['Sales']]) # Storing into y the 'MSRP' as np.array
print(X.shape) # Vewing the shape of X
print(y.shape) # Vewing the shape of y
plt.scatter(X,y,color="red") # Plot a graph X vs y
plt.title('TV vs. Sales')
plt.xlabel('TV')
plt.ylabel('Sales')
plt.show()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.333,random_state=15)
regressor = LinearRegression() # Creating a regressior
print("Training Set :")
print(X_train)
print("Test Set :")
print(X_test)

regressor.fit(X_train,y_train) # Fiting the dataset into the model
plt.scatter(X_test,y_test,color="green") # Plot a graph with X_test vs y_test
plt.plot(X_train,regressor.predict(X_train),color="red",linewidth=3) # Regressior line showing
plt.title('Regression(Test Set)')
plt.xlabel('TV')
plt.ylabel('Sales')
plt.show()
y_pred = regressor.predict(X_test)
print('R2 score: %.2f' % r2_score(y_test,y_pred)) # Priniting R 2 Score
print('Mean Error :',mean_squared_error(y_test,y_pred)) # Priniting the mean error
```

2. **Set A2 - Create 'realestate' Data set having 4 columns namely: ID,flat, houses and purchases (random 500 entries). Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
df = pd.read_csv('/home/ctbora/Downloads/Realestate.csv') # Importing the data set
new_df = df[['X4 number of convenience stores','Y house price of unit area']]
X = np.array(new_df[['X4 number of convenience stores']]) # Storing into X the 'Engine HP' as np.array
y = np.array(new_df[['Y house price of unit area']]) # Storing into y the 'MSRP' as np.array
print(X.shape) # Vewing the shape of X
print(y.shape) # Vewing the shape of y
plt.scatter(X,y,color="red") # Plot a graph X vs y
plt.title('X4 number of convenience stores vs. Y house price of unit area')
plt.xlabel('X1 transaction date')
plt.ylabel('Y house price of unit area')
plt.show()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.333,random_state=15)
regressor = LinearRegression() # Creating a regressior
print("Training Set :")
print(X_train)
print("Test Set :")
print(X_test)

regressor.fit(X_train,y_train) # Fiting the dataset into the model
plt.scatter(X_test,y_test,color="green") # Plot a graph with X_test vs y_test
plt.plot(X_train,regressor.predict(X_train),color="red",linewidth=3) # Regressior line showing
plt.title('Regression(Test Set)')
plt.xlabel('X4 number of convenience stores')
plt.ylabel('Y house price of unit area')
plt.show()
y_pred = regressor.predict(X_test)
print('R2 score: %.2f' % r2_score(y_test,y_pred)) # Priniting R 2 Score
print('Mean Error :',mean_squared_error(y_test,y_pred)) # Priniting the mean error
```

3. **Set A3 - Create 'User' Data set having 5 columns namely: User ID, Gender, Age, EstimatedSalary and Purchased. Build a logistic regression model that can predict whether on the given parameter a person will buy a car or not.**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
data = pd.read_csv("/home/ctbora/Downloads/suvdata.csv")
x = data.iloc[:, [2,3]].values
y = data.iloc[:, 4].values
print("x")
print(x)
print("y")
print(y)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=0)
logistic_regression= LogisticRegression()
logistic_regression.fit(x_train,y_train)
y_pred=logistic_regression.predict(x_test)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
print (x_test)
print (y_pred)
```

4. **Set B1 - Build a simple linear regression model for Fish Species Weight Prediction. (download dataset https://www.kaggle.com/aungpyaeap/fish-market?select=Fish.csv)**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
data = pd.read_csv("/home/ctbora/Downloads/Fish.csv")
x = data.iloc[:, [1,3]].values
y = data.iloc[:,0 ].values
print("x")
print(x)
print("y")
print(y)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.5,random_state=0)
logistic_regression= LogisticRegression()
logistic_regression.fit(x_train,y_train)
y_pred=logistic_regression.predict(x_test)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
print (x_test)
print (y_pred)
```

Assignment No: 2 - Frequent itemset and Association rule mining

1. **Set A1 - Create the following dataset in python. Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min_sup values**

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data= [['Bread','Milk'],
       ['Bread','Diaper','Beer','Eggs'],
       ['Milk','Diaper','Beer','Coke'],
       ['Bread','Milk','Diaper','Beer'],
       ['Bread','Milk','Diaper','Coke']
       ]

df = pd.DataFrame(data)
print(df)
from mlxtend.preprocessing import TransactionEncoder
te=TransactionEncoder()
te_array=te.fit(data).transform(data)
df=pd.DataFrame(te_array, columns=te.columns_)
print(df)
freq_items = apriori(df, min_support = 0.5, use_colnames = True)
print(freq_items)
rules = association_rules(freq_items, metric ='support', min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])
print(rules
```

2. **Set A2 - Create your own transactions dataset and apply the above process on your dataset.**

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data = [['Sheldon', 'Penny', 'Amy', 'Penny', 'Raj', 'Sheldon'],
      ['male', 'female', 'female', 'female', 'male', 'male']]
print(data)
df = pd.DataFrame(data)
print(df)
from mlxtend.preprocessing import TransactionEncoder
te=TransactionEncoder()
te_array=te.fit(data).transform(data)
df=pd.DataFrame(te_array, columns=te.columns_)
print(df)
freq_items = apriori(df, min_support = 0.5, use_colnames = True)
print(freq_items)
rules = association_rules(freq_items, metric ='support', min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])
print(rules)
```

3. **Set B1 - Download the Market basket dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.**

```
import pandas as pd
import io
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

d = pd.read_csv('/home/ctbora/DataAnalytics/ItemList.csv')
print(d)
d=d.fillna("Hi");
print(d)

te=TransactionEncoder()
te_array=te.fit(d).transform(d)
df=pd.DataFrame(te_array, columns=te.columns_)
print(df)

freq_items = apriori(df, min_support = 0.00000001, use_colnames = True)
print(freq_items)


rules = association_rules(freq_items, metric ='support', min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])
print(rules)
```

Assignment No: 3 - Text and Social Media Analytics

1. **Set A1 - Consider any text paragraph. Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process.**

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import nltk
#nltk.download('all')
#Preprocessing
import re
text="""
If98745345b43b54b bkb6jkb36k36b3j6h called with no arguments, ``download()`` will display an interactive
interface which can be used to download and install new packages.
If Tkinter is available, then a graphical interface will be shown,
otherwise a simple text interface will be provided.

Individual packages can be downloaded by calling the ``download()``
function with a single argument, giving the package identifier for the
package that should be downloaded:
"""
text = re.sub(r'[[0-9]*]', ' ', text)
text = re.sub(r's+', ' ', text)
formatted_text = re.sub('[^a-zA-Z]', ' ', text)
print(formatted_text)
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stopWords = set(stopwords.words("english"))
words = word_tokenize(formatted_text)
# Creating a frequency table of words
wordfreq = {}
for word in words:
 if word in stopWords:
  continue
 if word in wordfreq:
  wordfreq[word] += 1
else: wordfreq[word] = 1
#Compute the weighted frequencies
maximum_frequency = max(wordfreq.values())
for word in wordfreq.keys():
 wordfreq[word] = (wordfreq[word]/maximum_frequency)
# Creating a dictionary to keep the score # of each sentence
sentences = sent_tokenize(text)
sentenceValue = {}
for sentence in sentences:
 for word, freq in wordfreq.items():
  if word in sentence.lower():
   if sentence in sentenceValue:
    sentenceValue[sentence] += freq
   else:sentenceValue[sentence] = freq
import heapq
summary = ''
summary_sentences = heapq.nlargest(4, sentenceValue, key=sentenceValue.get)
summary = ' '.join(summary_sentences)
print(summary)
```

2. **Set A2 - Consider any text paragraph. Remove the stopwords. Tokenize the paragraph to extract words and sentences. Calculate the word frequency distribution and plot the frequencies. Plot the wordcloud of the text.**

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
# Textual data to remove stopwords
paragraph_text="""Hello all, Welcome to Python Programming Academy. Python
Programming Academy is a nice platform to learn new programming skills. It is
difficult to get enrolled in this Academy."""
# Word Tokenization
tokenized_words=word_tokenize(paragraph_text)
# It will find the stowords in English language.
stop_words_data=set(stopwords.words("english"))
# Create a stopwords list to filter it from original text
filtered_words_list=[]
for words in tokenized_words:
 if words not in stop_words_data:
  filtered_words_list.append(words)
print("Tokenized Words : \n",tokenized_words,"\n")
print("Filtered Words : \n",filtered_words_list,"\n")
```

3. **Set A3 - Consider the following review messages. Perform sentiment analysis on the messages.**
   i. I purchased headphones online. I am very happy with the product.
   ii. I saw the movie yesterday. The animation was really good but the script was ok.
   iii. I enjoy listening to music
   iv. I take a walk in the park everyday

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
vader_analyzer=SentimentIntensityAnalyzer()
text1="I purchased headphones online. I am very happy with the product."# The text is positive.
print(vader_analyzer.polarity_scores(text1))
text1="I saw the movie yesterday. The animation was really good but the script was ok."
print(vader_analyzer.polarity_scores(text1))
text1="I enjoy listening to music"
print(vader_analyzer.polarity_scores(text1))
text1="I take a walk in the park everyday"
print(vader_analyzer.polarity_scores(text1))
text1="Very Bad day"
print(vader_analyzer.polarity_scores(text1))
```

4. **Set A4 - Perform text analytics on WhatsApp data :**
   Write a Python script for the following :
   i. First Export the WhatsApp chat of any group. Read the exported ".txt" file using open() and read() functions.
   ii. Tokenize the read data into sentences and print it.
   iii. Remove the stopwords from data and perform lemmatization.
   iv. Plot the wordcloud for the given data.

```
import regex
import pandas as pd
import numpy as np
import emoji
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
def date_time(s):
    pattern = '^([0-9]+)(\/)([0-9]+)(\/)([0-9]+), ([0-9]+):([0-9]+)[ ]?(AM|PM|am|pm)? -'
    result = regex.match(pattern, s)
    if result:
        return True
    return False

def find_author(s):
    s = s.split(":")
    if len(s)==2:
        return True
    else:
        return False

def getDatapoint(line):
    splitline = line.split(' - ')
    dateTime = splitline[0]
    date, time = dateTime.split(", ")
    message = " ".join(splitline[1:])
    if find_author(message):
        splitmessage = message.split(": ")
        author = splitmessage[0]
        message = " ".join(splitmessage[1:])
    else:
        author= None
    return date, time, author, message
data = []
conversation = 'WhatsApp Chat with T.Y.B.Sc(Comp)Boys-21-22.txt'
with open(conversation, encoding="utf-8") as fp:
    fp.readline()
    messageBuffer = []
    date, time, author = None, None, None
    while True:
        line = fp.readline()
        if not line:
            break
        line = line.strip()
        if date_time(line):
            if len(messageBuffer) > 0:
                data.append([date, time, author, ' '.join(messageBuffer)])
```

```
            messageBuffer.clear()
            date, time, author, message = getDatapoint(line)
            messageBuffer.append(message)
        else:
            messageBuffer.append(line)
df = pd.DataFrame(data, columns=["Date", 'Time', 'Author', 'Message'])
df['Date'] = pd.to_datetime(df['Date'])
print(df)
print(df.info())
print(df.Author.unique())
import nltk
nltk.download('wordnet')
# Lemmatization
from nltk.stem.wordnet import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
word_text=message
print("Lemmatized Word : ",lemmatizer.lemmatize(word_text,"v"))
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from wordcloud import WordCloud, get_single_color_func
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
movies_reviews=df.values.astype(str)
movies_reviews1=np.array_str(movies_reviews)
# It will find the stowords in English language.
stop_words_data=set(stopwords.words("english"))
words=movies_reviews1.split()
final_data=[]
for w in words:
 if w not in final_data:
  final_data.append(w)
# Create dictionaries to store positive and negative words with polarity.
positive_words=dict()
negative_words=dict()
# Create lists to store positive and negative words without polarity.
positive=[]
negative=[]
# Sentiment Analysis
sentiment_analyzer=SentimentIntensityAnalyzer()
for i in words:
 if not i.lower() in stop_words_data: # It will remove stopwords.
  polarity=sentiment_analyzer.polarity_scores(i)
  if polarity['compound']>=0.05: # Positive Sentiment
   positive_words[i]=polarity['compound']
  if polarity['compound']<=-0.05: # Negative Sentiment
   negative_words[i]=polarity['compound']
# Append the positive and negative words from dictionaries to lists i.e.positive[] and negative[]
for key,value in positive_words.items():
 positive.append(key)
for key,value in negative_words.items():
 negative.append(key)
# Create a dictionary to mention the colors : green for positive and red for negative
coloured_words={"green":positive,"red":negative}
# Implement separate colour assignments
class ColourAssignment(object):
# Functions to give different colours on the basis of sentiments.
 def __init__(self,coloured_words,default):
  self.coloured_words=[
```

```
   (get_single_color_func(colour),set(words))
  for (colour,words) in coloured_words.items()]
  self.default=get_single_color_func(default)

def get_colour(self,word):
 try:
  colour=next(
  colour for (colour,words) in self.coloured_words
  if word in words)
 except StopIteration:
  colour=self.default
 return colour
def __call__(self,word, **kwargs):
 return self.get_colour(word) (word, **kwargs)
word_cloud=WordCloud(collocations=False,background_color='black').generate(movies_reviews1)
# Neutral words will be visible as black
group_color=ColourAssignment(coloured_words, 'white')
#word_cloud.recolor(color_func=group_color)
plt.figure()
plt.imshow(word_cloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```