Data Science and Engineering

Imports

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import seaborn as sn
5. from sklearn.model_selection import train_test_split
6. from sklearn.linear_model import LinearRegression
7. from sklearn.neighbors import KNeighborsClassifier
8. from sklearn.preprocessing import StandardScaler
9. from sklearn import metrics
10. from sklearn.tree import DecisionTreeClassifier
11. from IPython.display import Image
12. from sklearn import tree
13. import nltk
14. from sklearn.naive_bayes import MultinomialNB
15. from nltk.corpus import stopwords
16. from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
17. from sklearn.svm import SVC
18. from apyori import apriori
19. from sklearn.cluster import KMeans
20. from sklearn.feature_extraction.text import TfidfTransformer

General Stuff Same In All Programs

```
    test_url = './datasets/p9_test.csv'
    df = pd.read_csv(train_url)

    df.describe()
    df.head()
    df.columns
    df.shape

    # plotting graphs
```

IMPORTANT STUFF

program1(correlation matrix):

```
# generating correlation matrix:
corrMatrix = df.corr()

# plotting correlation matrix heat map
sn.heatmap(corrMatrix, annot=True)
plt.show()
```

program2(data preprocessing):

```
# printing rows containing missing values in "runtime" column
df[pd.isnull(df["runtime"])]

# filled missing value with mean
df["runtime"] = df["runtime"].fillna(df["runtime"].mean())

#data descritization
df['status'] = np.where(df['vote_average']>=6,'HIT','FLOP')

#normalizing budget (absolute maximum scaling)
df["revenue_scaled"] = df['budget']/df['budget'].abs().max()
```

program3(linear regression):

```
df = df.fillna(method='ffill')

# plotting from df
df.plot(x='MinTemp', y='MaxTemp', style='.')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()

X = df['MinTemp'].values.reshape(-1,1) #df[['MeanTemp', 'MinTemp']].values.reshape(-1,2)
y = df['MaxTemp'].values.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the model

#To retrieve the intercept:
print(regressor.intercept_)#For retrieving the slope:
print(regressor.coef_)

y_pred = regressor.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

#line learnt by the model
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```

program4(KNN):

```
x = df[x_colms].values
y = df['quality'].values
```

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

score = []
k = range(1,40)
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    score.append(metrics.accuracy_score(y_test,y_pred))
score

plt.figure(figsize =(15,10))
plt.plot(k,score,markersize = 10,color = 'red',linestyle = 'dashed',marker =
'o',markerfacecolor= 'blue')
plt.title('optimal k vlue')
plt.xlabel('knn value')

plt.ylabel('testing accuracy')
plt.show()
```

program5(decision tree):

```
feature_cols = ['Pregnancies','Glucose','BloodPressure' ,'SkinThickness','Insulin'
,'BMI','DiabetesPedigreeFunction','Age']

x = df[feature_cols]
y = df["Outcome"]

X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.2, random_state
= 1)

clf = DecisionTreeClassifier(criterion = 'entropy', max_depth = 4)
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test,y_pred))

fig = plt.figure(figsize=(35,30))
_ = tree.plot_tree(clf,
                   feature_names=feature_cols,
                   filled=True)
```

program6(Naive bayes):

```
# remove punctuations and stopwords
def text_process(mess):
    nopunc =[char for char in mess if char not in string.punctuation]
    nopunc=''.join(nopunc)
    clean = [word for word in nopunc.split() if word.lower() not in
stopwords.words('english')]
    if(len(clean) < 9):
        for i in range(9-len(clean)):
            clean.append(11304)
    return clean[:9]

df['v2'] = df['v2'].apply(text_process)
df['length'] = df['v2'].apply(len)
unique_words = []
for msg in df['v2']:
    for word in msg:
        unique_words.append(word)
unique_words = set(unique_words)
df.head()

vocab = {}
i=0
for word in unique_words:
    vocab[word]=i
    i+=1

def transform_data(data_set,vocab):
    ds=[]
    for row in data_set:
        temp = []
        for word in row:
            temp.append(vocab[word])
        ds.append(temp)
    return ds

msg_train,msg_test,label_train,label_test =
train_test_split(transform_data(df['v2'],vocab),df['v1'],test_size=0.2)

# improves accuracy significantly
tfidf_transformer = TfidfTransformer(use_idf = False)
msg_train=tfidf_transformer.transform(msg_train)
msg_test=tfidf_transformer.transform(msg_test)
msg_train.shape

spam_detect_model = MultinomialNB().fit(msg_train,label_train)
y_pred = spam_detect_model.predict(msg_test)

print("accuracy: ",accuracy_score(label_test,y_pred))
print(classification_report(label_test,y_pred))
print(confusion_matrix(label_test,y_pred))
```

program7(svm):

same as prev with only one change

```
spam_detect_model = SVC()
```

program8(apriori):

```
path = './datasets/P8_store_data.csv'
df = pd.read_csv(path, header=None)

records = []
for i in range(df.shape[0]):
    records.append([str(df.values[i,j]) for j in range(df.shape[1]) if (str(df.values[i,j]) != 'nan')])

association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)
association_results = list(association_rules)

num_associations = 0
for item in association_results:
    for i in range(len(item.ordered_statistics)):
        print("Rule: " + str(list(item.ordered_statistics[i].items_base)) + " -> " + str(list(item.ordered_statistics[i].items_add)))
        print("Confidence: " + str(item[2][i][2]))
        print("Lift: " + str(item[2][i][3]))
        print("=====================================")
        num_associations += 1
print(f"total number of association rules = {num_associations}")
```

program9(kmeans Clustering):

```
# fill missing values with mean
df_train.fillna(df_train.mean(),inplace=True)
df_test.fillna(df_test.mean(),inplace=True)

# to check number of missing values
df_train.isna().sum()

# find percentage of survivers based on another column
df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean()
df_train[["Sex", "Survived"]].groupby(['Sex']).mean()

selected_colms = ["Survived","Pclass","Sex","Age","SibSp","Parch","Fare"]
df_train = df_train[selected_colms]

# one hot encode categorical values
df_train = pd.concat([df_train,pd.get_dummies(df_train['Sex'], prefix='Sex',dummy_na=False)],axis=1).drop(['Sex'],axis=1)

x = np.array(df_train.drop(["Survived"],1))
y = np.array(df_train["Survived"])
```

```
kmeans = KMeans(n_clusters=2,max_iter=600, algorithm = 'auto') # You want cluster the passenger records into 2: Survived or Not survived
kmeans.fit(x)

# calculating accuracy
y_pred = kmeans.predict(x)
print("accuracy is: ",accuracy_score(y_pred,y))
```