UCS2612 Machine Learning Laboratory

k-Nearest Neighbor algorithm

Name : Mega V

Reg No : 3122 21 5001 051

Develop a python program to predict the Online Shoppers Purchasing Intention using K-Nearest Neighbour algorithm. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library

Code :

```
import pandas as pd
import os
import math
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
import statsmodels.api as sm
from sklearn.neighbors import KNeighborsClassifier
```

```
df = pd.read_csv("online_shoppers_intention.csv")
df
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	Op
0	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	0.000000	0.0	Feb	
1	0	0.0	0	0.0	2	64.000000	0.000000	0.100000	0.000000	0.0	Feb	
2	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	0.000000	0.0	Feb	
3	0	0.0	0	0.0	2	2.666667	0.050000	0.140000	0.000000	0.0	Feb	
4	0	0.0	0	0.0	10	627.500000	0.020000	0.050000	0.000000	0.0	Feb	
...	...	...	...	...	...	...	...	...	...	...	...	
12325	3	145.0	0	0.0	53	1783.791667	0.007143	0.029031	12.241717	0.0	Dec	
12326	0	0.0	0	0.0	5	465.750000	0.000000	0.021333	0.000000	0.0	Nov	
12327	0	0.0	0	0.0	6	184.250000	0.083333	0.086667	0.000000	0.0	Nov	
12328	4	75.0	0	0.0	15	346.000000	0.000000	0.021053	0.000000	0.0	Nov	
12329	0	0.0	0	0.0	3	21.250000	0.000000	0.066667	0.000000	0.0	Nov	

12330 rows × 18 columns

```
# Null Values :
```

```
print(df.isnull().sum())
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

```
df['Month'] = df['Month'].map({'Feb':0, 'Mar':1, 'May':2, 'Oct':3,
'June':4, 'Jul':5, 'Aug':6, 'Nov':7, 'Sep':8, 'Dec':9})
df['VisitorType'] = df['VisitorType'].map({'Returning_Visitor':0,
'New_Visitor':1, 'Other':2})
df['Weekend'] = df['Weekend'].map({False:0, True:1})
df['Revenue'] = df['Revenue'].map({False:0, True:1})
```

```
numeric_cols = ['Administrative',
'Administrative_Duration',  'Informational', 'Informational_Duration',
'ProductRelated', 'ProductRelated_Duration', 'BounceRates',
'ExitRates', 'PageValues', 'SpecialDay']
scaler_minmax = MinMaxScaler()
df[numeric_cols] = scaler_minmax.fit_transform(df[numeric_cols])
```

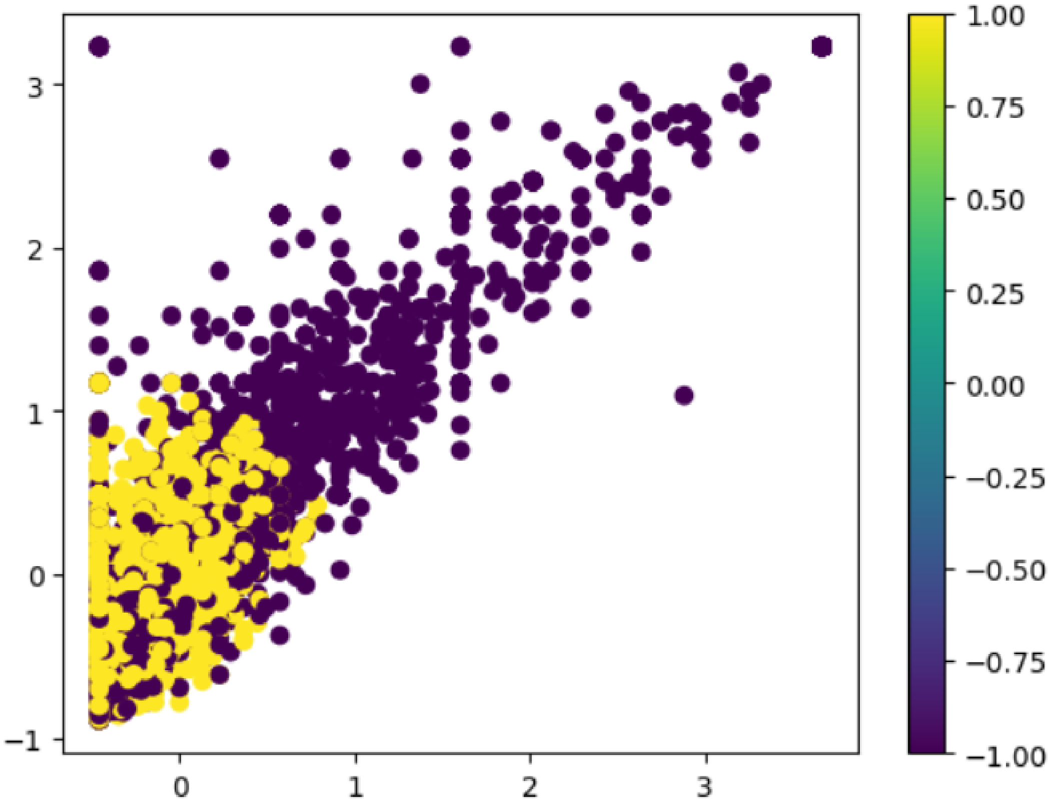
```
scaler_standard = StandardScaler()
df[numeric_cols] = scaler_standard.fit_transform(df[numeric_cols])
df
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	Ope
0	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316	-0.317178	-0.308821	0	
1	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.590903	-0.457683	1.171473	-0.317178	-0.308821	0	
2	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316	-0.317178	-0.308821	0	
3	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.622954	0.573535	1.994610	-0.317178	-0.308821	0	
4	-0.696993	-0.457191	-0.396478	-0.244931	-0.488636	-0.296430	-0.045196	0.142551	-0.317178	-0.308821	0	
...	...	...	...	...	...	...	...	...	...	...	...	
12325	0.206173	0.363075	-0.396478	-0.244931	0.478227	0.307822	-0.310366	-0.288966	0.342125	-0.308821	9	
12326	-0.696993	-0.457191	-0.396478	-0.244931	-0.601062	-0.380957	-0.457683	-0.447364	-0.317178	-0.308821	7	
12327	-0.696993	-0.457191	-0.396478	-0.244931	-0.578577	-0.528063	1.261014	0.897093	-0.317178	-0.308821	7	
12328	0.507228	-0.032916	-0.396478	-0.244931	-0.376210	-0.443536	-0.457683	-0.453140	-0.317178	-0.308821	7	
12329	-0.696993	-0.457191	-0.396478	-0.244931	-0.646033	-0.613243	-0.457683	0.485525	-0.317178	-0.308821	7	

```
outlier_detector = IsolationForest(contamination=0.5)
```

```
df['Outlier'] = outlier_detector.fit_predict(df[numeric_cols])
plt.scatter(df['BounceRates'], df['ExitRates'], c=df['Outlier'],
cmap='viridis')
plt.colorbar()
plt.show()

df = df[df['Outlier'] == 1].drop('Outlier', axis=1)
```




```
X = df.drop("Revenue", axis = 1)
Y = df["Revenue"]

X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
```

```
def kNNModelEuclidean(k, x_train, x_test, y_train, y_test):
    neigh = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    neigh.fit(x_train, y_train)
    y_pred = neigh.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    return accuracy

def kNNModelManhattan(k, x_train, x_test, y_train, y_test):
    neigh = KNeighborsClassifier(n_neighbors=k, metric='manhattan')
    neigh.fit(x_train, y_train)
```

```
    y_pred = neigh.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    return accuracy

def kNNModelMinkowski(k, x_train, x_test, y_train, y_test):
    neigh = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p =
3)
    neigh.fit(x_train, y_train)
    y_pred = neigh.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    return accuracy
```

```
print("Without Feature Engineering : \n")
euc_acc = []
man_acc = []
min_acc = []
for k in range(2,20):
    euc_acc.append((k,round(kNNModelEuclidean(k, X_train, X_test,
y_train, y_test),3)))
    man_acc.append((k,round(kNNModelManhattan(k, X_train, X_test,
y_train, y_test),3)))
    min_acc.append((k,round(kNNModelMinkowski(k, X_train, X_test,
y_train, y_test),3)))

print("Euclidean - k = ",max(euc_acc)[0], "Accuracy : ", max(euc_acc))
print("Manhattan - k = ",max(man_acc)[0], "Accuracy : ", max(man_acc))
print("Minkowski - k = ",max(min_acc)[0], "Accuracy : ", max(min_acc))
```

```
Without Feature Engineering :

Euclidean - k =  19 Accuracy :  (19, 0.899)
Manhattan - k =  19 Accuracy :  (19, 0.897)
Minkowski - k =  19 Accuracy :  (19, 0.902)
```

```
alpha = 0.01
lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X_train, y_train)
feature_importance = lasso_model.coef_
selected_features = np.where(feature_importance != 0)[0]
X_train_selected_lasso = X_train.iloc[:, selected_features]
X_test_selected_lasso = X_test.iloc[:, selected_features]
```

```
print("After Lasso Feature Reduction Technique :\n")

euc_acc = []
```

```
man_acc = []
min_acc = []
for k in range(2,20):
    euc_acc.append((k,round(kNNModelEuclidean(k,
X_train_selected_lasso, X_test_selected_lasso, y_train, y_test),3)))
    man_acc.append((k,round(kNNModelManhattan(k,
X_train_selected_lasso, X_test_selected_lasso, y_train, y_test),3)))
    min_acc.append((k,round(kNNModelMinkowski(k,
X_train_selected_lasso, X_test_selected_lasso, y_train, y_test),3)))

print("Euclidean - k = ",max(euc_acc)[0], "Accuracy : ",
max(euc_acc)[1])
print("Manhattan - k = ",max(man_acc)[0], "Accuracy : ",
max(man_acc)[1])
print("Minkowski - k = ",max(min_acc)[0], "Accuracy : ",
max(min_acc)[1])
```

```
After Lasso Feature Reduction Technique :

Euclidean - k =  19 Accuracy :  0.924
Manhattan - k =  19 Accuracy :  0.924
Minkowski - k =  19 Accuracy :  0.924
```

```
# Ridge Feature Reduction Technique :

alpha = 1.0
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(X_train, y_train)
feature_importance = ridge_model.coef_
selected_features = feature_importance != 0
X_train_selected_ridge = X_train.loc[:, selected_features]
X_test_selected_ridge = X_test.loc[:, selected_features]
```

```
print("After Ridge Feature Reduction Technique :\n")

euc_acc = []
man_acc = []
min_acc = []
for k in range(2,20):
    euc_acc.append((k,round(kNNModelEuclidean(k,
X_train_selected_ridge, X_test_selected_ridge, y_train, y_test),3)))
    man_acc.append((k,round(kNNModelManhattan(k,
X_train_selected_ridge, X_test_selected_ridge, y_train, y_test),3)))
    min_acc.append((k,round(kNNModelMinkowski(k,
X_train_selected_ridge, X_test_selected_ridge, y_train, y_test),3)))
```

```
print("Euclidean - k = ",max(euc_acc)[0], "Accuracy : ",
max(euc_acc)[1])
print("Manhattan - k = ",max(man_acc)[0], "Accuracy : ",
max(man_acc)[1])
print("Minkowski - k = ",max(min_acc)[0], "Accuracy : ",
max(min_acc)[1])
```

```
After Ridge Feature Reduction Technique :

Euclidean - k =  19 Accuracy :  0.899
Manhattan - k =  19 Accuracy :  0.897
Minkowski - k =  19 Accuracy :  0.902
```

```
# Backward Feature Elimination :

def backward_elimination(X, y, significance_level=0.05):
    features = X.columns.tolist()
    num_features = len(features)

    for i in range(num_features, 0, -1):
        X_with_constant = sm.add_constant(X)
        model = sm.OLS(y, X_with_constant).fit()
        max_p_value = max(model.pvalues[1:])

        if max_p_value > significance_level:
            removed_feature = model.pvalues.idxmax()[:]
            print(f"Removing feature: {removed_feature} (p-value:
{max_p_value:.4f})")
            X = X.drop(removed_feature, axis=1)
        else:
            break

    return X

X_train_backward = backward_elimination(X_train, y_train)
X_test_backward = X_test[X_train_backward.columns]
```

```
Removing feature: TrafficType (p-value: 0.8939)
Removing feature: SpecialDay (p-value: 0.7614)
Removing feature: Browser (p-value: 0.5067)
Removing feature: Region (p-value: 0.5265)
Removing feature: ExitRates (p-value: 0.5175)
Removing feature: ProductRelated_Duration (p-value: 0.4116)
Removing feature: Administrative (p-value: 0.1013)
Removing feature: Administrative_Duration (p-value: 0.3635)
Removing feature: Informational_Duration (p-value: 0.0652)
```

```
print("After Backward Feature Elimination :\n")

euc_acc = []
man_acc = []
min_acc = []
for k in range(2,20):
    euc_acc.append((k,round(kNNModelEuclidean(k, X_train_backward,
X_test_backward, y_train, y_test),3)))
    man_acc.append((k,round(kNNModelManhattan(k, X_train_backward,
X_test_backward, y_train, y_test),3)))
    min_acc.append((k,round(kNNModelMinkowski(k, X_train_backward,
X_test_backward, y_train, y_test),3)))

print("Euclidean - k = ",max(euc_acc)[0], "Accuracy : ",
max(euc_acc)[1])
print("Manhattan - k = ",max(man_acc)[0], "Accuracy : ",
max(man_acc)[1])
print("Minkowski - k = ",max(min_acc)[0], "Accuracy : ",
max(min_acc)[1])
```

```
After Backward Feature Elimination :

Euclidean - k =  19 Accuracy :  0.908
Manhattan - k =  19 Accuracy :  0.905
Minkowski - k =  19 Accuracy :  0.908
```

Learning Outcome :

- The Categorical values are converted to numerical values
- The numerical values are normalized and Standarized
- For a different feature selection algorithms the accuracy may differ
- The accuracy mainly depends on the K value