MACHINE INTELLIGENCE SYSTEMS

CODING

Decision Tree

```
# importing necessary libraries
import numpy as py
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# libraries used for the actual model
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
from sklearn.metrics import classification_report

model_comparison = {}

# read dataset using pandas
pd.set_option("display.max_rows", 100, "display.max_columns", 100)
df = pd.read_csv("bank-additional-full.csv", delimiter=';')

# prints the first 5 rows
print("First 5 rows of dataset")
print(df.head(5))
print("\n")

# quick summary of the Dataframe
print("Information about the Dataset")
df.info()

# maps 'yes' values to 1 and 'no' values to 0
df['y'] = df['y'].map({'yes': 1, 'no': 0})
df.pdays[df.pdays == -1] = 0
```

```
print("Unique values in dataset")
print(df.nunique().sort_values(ascending=True))
print("\n")

print("Null values in dataset")
print(df.isnull().sum())
print("\n")

# Skewed Distribution
plt.figure(figsize=(18, 18))
for i, col in enumerate(df.drop(['y'], axis=1).select_dtypes(include=['int',
'float']).columns):
    plt.rcParams['axes.facecolor'] = 'black'
    ax = plt.subplot(4,3, i+1)
    sns.histplot(data=df, x=col, ax=ax,color='red',kde=True)
plt.suptitle('Data distribution of continuous variables')
plt.tight_layout()

# Box Plot
plt.figure(figsize=(18, 18))
for i, col in enumerate(df.drop(['y'], axis=1).select_dtypes(include=['int',
'float']).columns):
    plt.rcParams['axes.facecolor'] = 'black'
    ax = plt.subplot(4, 3, i+1)
    sns.boxplot(data=df, x=col, ax=ax,color='red')
plt.suptitle('Data distribution of continuous variables')
plt.tight_layout()

# Heatmap
plt.figure(figsize=(15,10))
sns.heatmap(df.select_dtypes(include=['int', 'float']).corr(), annot=True,
center=0)
plt.show()

# removing pdays to avoid multicollinearity
del df['pdays']

# Bar plot of target variable
plt.figure(figsize=(15, 10))
for i, col in enumerate(df.drop(['y'], axis=1).select_dtypes(include=['int',
'float']).columns):
    plt.rcParams['axes.facecolor'] = 'black'
    ax = plt.subplot(3, 3, i+1)
    sns.barplot(data=df, x='y', y=col, ax=ax,
edgecolor="black",palette='viridis_r')
```

```python
font = {'fontsize': 25, 'color': 'grey'}
plt.suptitle('Data distribution of Target variable', fontsize=40)
plt.tight_layout()

# Pie Chart of target class
target_var = pd.crosstab(index=df['y'],columns='% observations')
plt.pie(target_var['% observations'], labels=target_var['% observations'].index,
autopct='%.0f%%')
plt.title('has the client subscribed a term deposit?')
plt.show()

# Bar plot of target class
sns.barplot(x=target_var.index, y=target_var['% observations'])
plt.title('has the client subscribed a term deposit?')
plt.show()

# applying One-Hot Encoding
df1 = pd.get_dummies(df,drop_first=True)

X = df1.drop(['y'], axis=1)
y = df1['y']

print("\n")
# printing after encoding and dropping, the first 5 rows
X.head(5)

# selection of features
clf = ExtraTreesClassifier(n_estimators=100)
clf = clf.fit(X.values, y)
clf.feature_importances_
model = SelectFromModel(clf, prefit=True)
X_new = model.transform(X.values)

# splitting of dataset for training and testing
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.25,
stratify=y, random_state=0)

# standardization
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

print("\n")
# Performance Metrics
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(f"Model Accuracy : {accuracy_score(y_pred,y_test)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_pred,y_test,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=5,
scoring="recall")
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_pred, y_test,zero_division=1))
model_comparison['Decision Tree'] = [accuracy_score(y_pred, y_test),
f1_score(y_pred, y_test, average='weighted'),
                                     (accuracies.mean()), (accuracies.std())]
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

K-Nearest Neighbor

```
# importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
from sklearn.metrics import classification_report

# read dataset using pandas
df = pd.read_csv('bank-additional-full.csv', delimiter=';')

df = df[['age', 'job', 'marital', 'education', 'default', 'housing',
'loan','campaign', 'pdays', 'previous', 'poutcome', 'y']]
df.info()

# pre-process categorical data
objfeatures = df.select_dtypes(include="object").columns
le = preprocessing.LabelEncoder()

# tranforms features
for feat in objfeatures:
    df[feat] = le.fit_transform(df[feat].astype(str))
```

```python
X = df.drop('y', axis=1)
y = df['y']

# normalization
X = preprocessing.StandardScaler().fit_transform(X.astype(int))

# splitting into training and testing tests
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=4)

# knn is run twice
# the first time is using 'Euclidean Distance' to find optimum K
# the second time is using 'Manhattan Distance' to find optimum K

# euclidean distance
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(weights='distance',
n_neighbors=i).fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

# plot the minimum error
plt.figure(figsize=(10,6))
plt.plot(range(1, 40), error_rate,color='black', linestyle='dashed',
         marker='o',markerfacecolor='yellow', markersize=6)
plt.title('Error Rate vs. K Value with Distance Metric: Euclidean')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:", min(error_rate), "at K =",
error_rate.index(min(error_rate)))

# euclidean distance
acc = []
for i in range(1, 40):
    neigh = KNeighborsClassifier(weights='distance', n_neighbors=i).fit(X_train,
y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

# plot the maximum accuracy
plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), acc, color='black', linestyle='dashed',
         marker='o', markerfacecolor='yellow', markersize=6)
```

```
plt.title('Accuracy vs. K Value with Distance Metric: Euclidean')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-", max(acc), "at K =", acc.index(max(acc)))

# performance metrics with euclidean distance
classifier = KNeighborsClassifier(weights='distance', n_neighbors=i).fit(X_train,
y_train)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(f"Model Accuracy : {accuracy_score(y_pred,y_test)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_pred,y_test,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=5,
scoring="recall")
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_pred, y_test,zero_division=1))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

# manhattan distance
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(weights='distance', n_neighbors=i,
p=1).fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

# plot the minimum error
plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), error_rate,color='black', linestyle='dashed',
         marker='o',markerfacecolor='yellow', markersize=6)
plt.title('Error Rate vs. K Value with Distance Metric: Manhattan')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:", min(error_rate), "at K =",
error_rate.index(min(error_rate)))

# manhattan distance
acc = []
for i in range(1, 40):
    neigh = KNeighborsClassifier(weights='distance', n_neighbors=i,
p=1).fit(X_train, y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))
```

```
# plot the maximum accuracy
plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), acc, color='black', linestyle='dashed',
         marker='o', markerfacecolor='yellow', markersize=6)
plt.title('Accuracy vs. K Value with Distance Metric: Manhattan')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-", max(acc), "at K =", acc.index(max(acc)))

# performance metrics with manhattan distance
classifier = KNeighborsClassifier(weights='distance', n_neighbors=i,
p=1).fit(X_train, y_train)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(f"Model Accuracy : {accuracy_score(y_pred,y_test)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_pred,y_test,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=5,
scoring="recall")
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_pred, y_test,zero_division=1))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```