k-Nearest Neighbors: Fit

In this exercise, you will build your first classification model using the churn_df dataset, can be loaded from churn_df.csv.

The features to use will be "account_length" and "customer_service_calls". The target, "churn", needs to be a single column with the same number of observations as the feature data.

You will convert the features and the target variable into NumPy arrays, create an instance of a KNN classifier, and then fit it to the data.

Instructions

- Import KNeighborsClassifier from sklearn.neighbors.
- Create an array called X containing values from the "account_length" and "customer_service_calls" columns, and an array called y for the values of the "churn" column.
- Instantiate a KNeighborsClassifier called knn with 6 neighbors.
- Fit the classifier to the data using the .fit() method.

```
X = churn_df[['total_day_charge', 'total_eve_charge']].values
y = churn_df['churn'].values
print(X.shape, y.shape)
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X,y)
X_new = np.array([[56.8, 17.5],
                  [24.4, 24.1],
                  [50.1, 10.9]])
print(X_new.shape)
predictions = knn.predict(X_new)
print(f'Predictions: {predictions}')
```

k-Nearest Neighbors: Predict

Now you have fit a KNN classifier, you can use it to predict the label of new data points. All available data was used for training, however, fortunately, there are new observations available, X_new.

The model knn, which you created and fit the data in the last exercise, will be used. You will use your classifier to predict the labels of a set of new data points:

X_new = np.array([[30.0, 17.5],
[107.0, 24.1],
[213.0, 10.9]])


```
X = churn_df[['total_day_charge', 'total_eve_charge']].values
y = churn_df['churn'].values
print(X.shape, y.shape)
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X,y)
X_new = np.array([[56.8, 17.5],
                  [24.4, 24.1],
                  [50.1, 10.9]])
print(X_new.shape)
predictions = knn.predict(X_new)
print(f'Predictions: {predictions}')
```

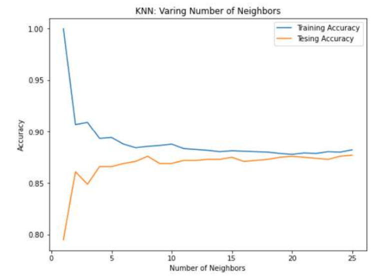

Instructions

- Create y_pred by predicting the target values of the unseen features X_new.
- Print the predicted labels for the set of predictions.

```
Predictions: [0 1 0]
```

```
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1,26)
print(neighbors)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

#print(train_accuracies.values())
print(test_accuracies.values())
my_train = list(train_accuracies.values())
my_test = list(test_accuracies.values())

plt.figure(figsize=(8,6))
plt.title("KNN: Varing Number of Neighbors")
plt.plot(neighbors, my_train, label="Training Accuracy")
plt.plot(neighbors, my_test, label="Tesing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

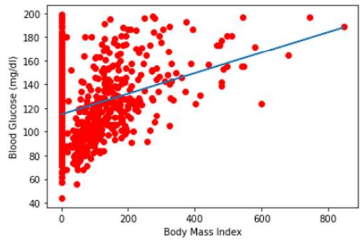
```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
dict_values([0.795, 0.861, 0.849, 0.866, 0.866, 0.869, 0.871, 0.876, 0.869, 0.869, 0.872, 0.872, 0.873, 0.873, 0.875, 0.871, 0.
872, 0.873, 0.875, 0.876, 0.875, 0.874, 0.873, 0.876, 0.877])
```




```
import pandas as pd
import numpy as np
sales_df = pd.read_csv('sales_df.csv')

X = sales_df['radio'].values
y = sales_df['sales'].values

X = X.reshape(-1, 1)
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (4546, 1)
Shape of y: (4546,)
```

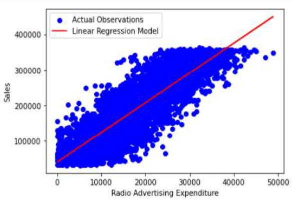
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_bmi, y)
predictions = reg.predict(X_bmi)
plt.scatter(X_bmi, y, color="Red")
plt.plot(X_bmi, predictions)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```




```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

predictions = model.predict(X)

print("Five prediction values:", predictions[:5])
```

```
Five prediction values: [ 95491.17119147 117829.51
 111137.28167129]
```

```
import matplotlib.pyplot as plt
plt.scatter(X, y, color='blue', label='Actual Observations')

plt.plot(X, predictions, color='red', label='Linear Regression Model')

plt.xlabel('Radio Advertising Expenditure')
plt.ylabel('Sales')
plt.legend(loc='best')
plt.show()
```




```
from sklearn.model_selection import train_test_split

print(cc_apps.corr())

#Drop the features 11 and 13
cc_apps = cc_apps.drop([11, 13], axis=1)

# Split into train and test sets
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, random_state=42)
```

	2	7	10	14
2	1.000000	0.298902	0.271207	0.123121
7	0.298902	1.000000	0.322330	0.051345
10	0.271207	0.322330	1.000000	0.063692
14	0.123121	0.051345	0.063692	1.000000

```
# Import numpy
import numpy as np

# Replace the '?'s with NaN in the train and test sets
cc_apps_train = cc_apps_train.replace('?', np.NaN)
cc_apps_test = cc_apps_test.replace('?', np.NaN)
```

```
# Impute the missing values with mean imputation
cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)

# Count the number of NaNs in the datasets and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
```

```
for col in cc_apps_train.columns: # Iterate over each column of cc_apps_train

    if cc_apps_train[col].dtypes == 'object': # Check if the column is of object type
        # Impute with the most frequent value
        # The value_counts() function returns a Series that contain counts of unique val
        # descending order so that its first element will be the most frequently-occurre
        cc_apps_train = cc_apps_train.fillna(cc_apps_train[col].value_counts().index[0])
        cc_apps_test = cc_apps_test.fillna(cc_apps_train[col].value_counts().index[0])

# Count the number of NaNs in the dataset and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
# At this point, there is no missing values.
```

```
# Convert the categorical features in the train and test sets independently
print(cc_apps_train)
cc_apps_train = pd.get_dummies(cc_apps_train)
cc_apps_test = pd.get_dummies(cc_apps_test)
print(cc_apps_train)
# Reindex the columns of the test set aligning with the train set
cc_apps_test = cc_apps_test.reindex(columns=cc_apps_train.columns, fill_value=0)
```

```
# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# Segregate features and labels into separate variables
X_train, y_train = cc_apps_train.iloc[:, :-1].values, cc_apps_train.iloc[:, [-1]].values
X_test, y_test = cc_apps_test.iloc[:, :-1].values, cc_apps_test.iloc[:, [-1]].values

# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
```

```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train,y_train)
```

```
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set and store it
y_pred = logreg.predict(rescaledX_test)

# Get the accuracy score of logreg model and print it
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_test,y_test))

# Print the confusion matrix of the logreg model
confusion_matrix(y_test,y_pred)
```

```
Accuracy of logistic regression classifier:  1.0

array([[103,   0],
       [  0, 125]], dtype=int64)
```