KNN REFERENCE

Data Pre-Processing Step:

1.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

2.

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

```
dataset = pd.read_csv('diabetes.csv')
print( len(dataset) )
print( dataset.head() )
```

```
# Replace zeroes
zero_not_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Insulin']

for column in zero_not_accepted:
    dataset[column] = dataset[column].replace(0, np.NaN)
    mean = int(dataset[column].mean(skipna=True))
    dataset[column] = dataset[column].replace(np.NaN, mean)
```

```
# split dataset
X = dataset.iloc[:, 0:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
```

```
#Feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```python
# split dataset
X = dataset.iloc[:, 0:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
```

```python
#Feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```python
# Define the model: Init K-NN
classifier = KNeighborsClassifier(n_neighbors=11, p=2,metric='euclidean')
```

3.

```python
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from seaborn import load_dataset
```

```python
# Loading the penguins dataset
df = load_dataset('penguins')
print(df.head())
```

```python
# Splitting our DataFrame into features and target
df = df.dropna()

X = df[['bill_length_mm']]
y = df['species']
```

```python
# Splitting data into training and testing data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
100)
```

4.

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Get the Data

Set index_col=0 to use the first column as the index.

```python
df = pd.read_csv("Classified Data",index_col=0)
```

```python
df.head()
```

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

```python
from sklearn.preprocessing import StandardScaler
```

```python
scaler = StandardScaler()
```

```python
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```python
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
```

```python
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Train Test Split ¶

```python
from sklearn.model_selection import train_test_split
```

```python
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
                                                    test_size=0.30)
```

5.

```python
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
print(knn.predict(X_test))

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

Fitting K-NN classifier to the Training data:

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

1.

```
KNeighborsClassifier(
    n_neighbors=5,          # The number of neighbours to consider
    weights='uniform',      # How to weight distances
    algorithm='auto',       # Algorithm to compute the neighbours
    leaf_size=30,           # The leaf size to speed up searches
    p=2,                    # The power parameter for the Minkowski
metric
    metric='minkowski',     # The type of distance to use
    metric_params=None,     # Keyword arguments for the metric function
    n_jobs=None             # How many parallel jobs to run
)
```

2.

```
# Creating a classifier object in sklearn
clf = KNeighborsClassifier(p=1)
```

```
# Fitting our model
clf.fit(X_train, y_train)
```

Using KNN

Remember that we are trying to come up with a model to predict whether someone will TARGET CLASS or not. We'll start with k=1.

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

```
pred = knn.predict(X_test)
```

3.

Predicting the Test Result:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```
1.

```
# Predict the test set results
y_pred = classifier.predict(X_test)
y_pred
```
2.

```
print(f1_score(y_test, y_pred))
```

```
0.6956521739130436
```

```
print(accuracy_score(y_test, y_pred))
```

```
0.8181818181818182
```

```
# Making predictions
predictions = clf.predict(X_test)
print(predictions)
```
3.

```
# Making your own predictions
predictions = clf.predict([[44.2]])
print(predictions)
```

```
# Measuring the accuracy of our model
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, predictions))
```

Predictions and Evaluations

Let's evaluate our KNN model!

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,pred))
```

```
[[125  18]
 [ 13 144]]
```

```
print(classification_report(y_test,pred))
```

```
             precision    recall  f1-score   support

          0       0.91      0.87      0.89       143
          1       0.89      0.92      0.90       157

avg / total       0.90      0.90      0.90       300
```
4.

Creating the Confusion Matrix:

```
#Creating the Confusion matrix
  from sklearn.metrics import confusion_matrix
  cm= confusion_matrix(y_test, y_pred)
```

1.

```
# Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

2.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         9
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.88      0.78      0.82         9

       accuracy                           0.90        30
      macro avg       0.91      0.90      0.90        30
   weighted avg       0.90      0.90      0.90        30

[[ 9  0  0]
 [ 0 11  1]
 [ 0  2  7]]
```

3.

Visualizing the Training set result:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

1.

```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

2.

Here we can see that that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
# NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

3.

Visualizing the Test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

REFERENCE

https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
https://datagy.io/python-knn/
https://www.youtube.com/watch?v=4HKqjENq9OU
https://www.youtube.com/watch?v=otolSnbanQk
https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/