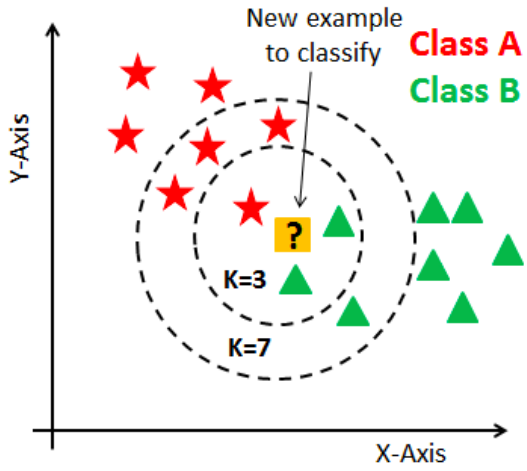
National University of Singapore
School of Computing
Tutorial 3:
K-NEAREST NEIGHBORS

K-Nearest Neighbours is a **supervised learning technique** that is used mostly for classification, but sometimes for regression as well. The 'K' in KNN is the number of nearest neighbors used to classify/predict a test sample.



Produce a Jupyter Notebook code to answer questions 1 to 7.

1) Import the Wisconsin Breast Cancer dataset from Sklearn datasets. What format is it in? Inspect the keys. What are the names of the keys?

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import neighbors
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

breast_cancer_dataset = load_breast_cancer()
breast_cancer_dataset.keys()
```

2) Create your X variable (the features) and the y variable (the labels).

```
X = breast_cancer_dataset.data
y = breast_cancer_dataset.target
```

3) Create a train-test split in your data using the SKLearn Train-Test split library.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

4) Fit the SKLearn KNeighborsClassifier with a n_neighbors value of 3. What is the accuracy score?

```
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn_model_1 = knn.fit(X_train, y_train)

print(knn_model_1.score(X_test, y_test))
```

5) Create predictions on the test set and use the SKLearn Classification_report library to generate a classification report. Discuss your results.

```
y_true, y_pred = y_test, knn_model_1.predict(X_test)
print(classification_report(y_true, y_pred))
```

6) Visualize the dataset you have as a histogram. Normalize your data using SKLearn's standard scaler and re-run the classifier on the data. Why do we need to normalize our data, and why does our result change? Discuss the results that you have obtained.

```
breast_cancer_df = pd.DataFrame(data = X,
                                columns = breast_cancer_dataset.feature_names)

pd.DataFrame.hist(breast_cancer_df, figsize=[15,15]);

scaler = StandardScaler()
X_train_transform = scaler.fit_transform(X_train)
X_test_transform = scaler.fit_transform(X_test)

breast_cancer_train_transformed_df = pd.DataFrame(data = X_train_transform,
                                                  columns = breast_cancer_dataset.feature_names)

pd.DataFrame.hist(breast_cancer_train_transformed_df, figsize=[15,15]);

knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn_model_2 = knn.fit(X_train_transform, y_train)

print(knn_model_2.score(X_test_transform, y_test))

y_true, y_pred = y_test, knn_model_2.predict(X_test_transform)

print(classification_report(y_true, y_pred))
```

7) Use an SVM to conduct the same classification. What are the differences in result?

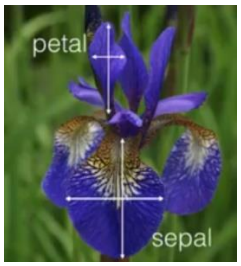
```
svm_classifier = SVC(gamma='auto')
svm_classifier.fit(X_train_transform, y_train)

print(svm_classifier.score(X_test_transform, y_test))

y_true, y_pred = y_test, svm_classifier.predict(X_test_transform)

print(classification_report(y_true, y_pred))
```

8) Produce a Jupyter Notebook code and use KNN classification on the IRIS dataset contained in the SKLearn datasets library (i.e. sklearn.datasets.load_iris).




```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import neighbors
```

a) What are the features and species of flowers that are measured in this dataset?

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
iris_dataset.feature_names
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'pet
al width (cm)']
```

```
iris_dataset.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

b) Print first 10 measurements taken in this dataset

```
X2 = iris_dataset.data
iris_df = pd.DataFrame(data = X2,
                       columns = iris_dataset.feature_names)
iris_df[:10]
```

c) Using only Sepal length and Sepal width to classify flowers, create a color-coded scatterplot

```
import matplotlib.pyplot as plt
```

```
from matplotlib.colors import ListedColormap

X2a = X2[:, (0,1)]
y2 = iris_dataset.target

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA','#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00AA00','#00AAFF'])

# Plot also the training points
plt.scatter(X2a[:, 0], X2a[:, 1], c=y2, cmap=cmap_bold)
plt.xlabel(iris_dataset.feature_names[0])
plt.ylabel(iris_dataset.feature_names[1])
plt.show()
```

d) Using only Petal length and Petal width to classify flowers, create a color-coded scatterplot

```
X2a = X2[:, (2,3)]
y2 = iris_dataset.target

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA','#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00AA00','#00AAFF'])

# Plot also the training points
plt.scatter(X2a[:, 0], X2a[:, 1], c=y2, cmap=cmap_bold)
plt.xlabel(iris_dataset.feature_names[2])
plt.ylabel(iris_dataset.feature_names[3])
plt.show()
```

e) Choose two features and classify using K nearest neighbor and plot the decision boundaries using np.meshgrid, np.ravel and plt.colormesh

```
#Choose two features from ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
a = 1
b = 3
X2a = X2[:, (a,b)]
y2 = iris_dataset.target

h = .02  # step size in the mesh

# Calculate min, max and limits
x_min, x_max = X2a[:, 0].min() - 1, X2a[:, 0].max() + 1
y_min, y_max = X2a[:, 1].min() - 1, X2a[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# we create an instance of Neighbours Classifier and fit the data.
n_neighbors=3
```

```
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X2a, y2)

# predict class for all points on mesh grid using kNN classifier
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
#Color mesh

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X2a[:, 0], X2a[:, 1], c=y2, cmap=cmap_bold)
plt.xlabel(iris_dataset.feature_names[a])
plt.ylabel(iris_dataset.feature_names[b-1])
plt.title("3-Class classification (k = %i)" % (n_neighbors))
plt.show()
```

9) Produce a Jupyter Notebook code and use KNN classification on the wine dataset contained in the SKLearn datasets library (i.e. sklearn.datasets.load_wine). What is the optimal value for n_neighbors? What is the accuracy score? Produce a classification report and discuss your results.

```
from sklearn.datasets import load_wine

wine_dataset = load_wine()
wine_dataset.keys()

X = wine_dataset.data
y = wine_dataset.target

#Create a train-test split in your data using the SKLearn Train-Test split library
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

#Normalize your data using SKLearn's standard scaler
scaler = StandardScaler()
X_train_transform = scaler.fit_transform(X_train)
X_test_transform = scaler.fit_transform(X_test)

#Run classifier on data
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn_model_2 = knn.fit(X_train_transform, y_train)

print(knn_model_2.score(X_test_transform, y_test))

y_true, y_pred = y_test, knn_model_2.predict(X_test_transform)
```

```
print(classification_report(y_true, y_pred))

# Conduct grid search
np.random.seed(0)
best_knn = None
knn_best_test = -1

results = {}
n_neighbors = [2, 3, 4, 5, 6, 7, 8, 9, 10]
weights = ['uniform'] #, 'distance']
grid_search = [ (nn,weight) for nn in n_neighbors for weight in weights ]

for nn, weight in grid_search:
    # Create knn model
    knn = neighbors.KNeighborsClassifier(n_neighbors = nn, weights = weight)

    # Train phase
    knn_model_2 = knn.fit(X_train_transform, y_train)

    # Accuracy
    train_accuracy = knn_model_2.score(X_train_transform, y_train)
    test_accuracy = knn_model_2.score(X_test_transform, y_test)

    # Classification report
    y_true, y_pred = y_test, knn_model_2.predict(X_test_transform)

    results[nn,weight] = (train_accuracy,test_accuracy)

    # Save best model
    if test_accuracy > knn_best_test:
        knn_best_test = test_accuracy
        best_knn = knn_model_2

# Print out results.
for g, k in sorted(results):
    train_accuracy, test_accuracy = results[(g, k)]
    print('n_neighbors %d weights %s train accuracy: %f test accuracy: %f' % (
            g, k, train_accuracy, test_accuracy))


#print accuracy score of highest test accuracy
print(knn_best_test)

#Print classification report
y_true, y_pred = y_test, best_knn.predict(X_test_transform)
print(classification_report(y_true, y_pred))
```