COMP9517 - Pattern Recognition

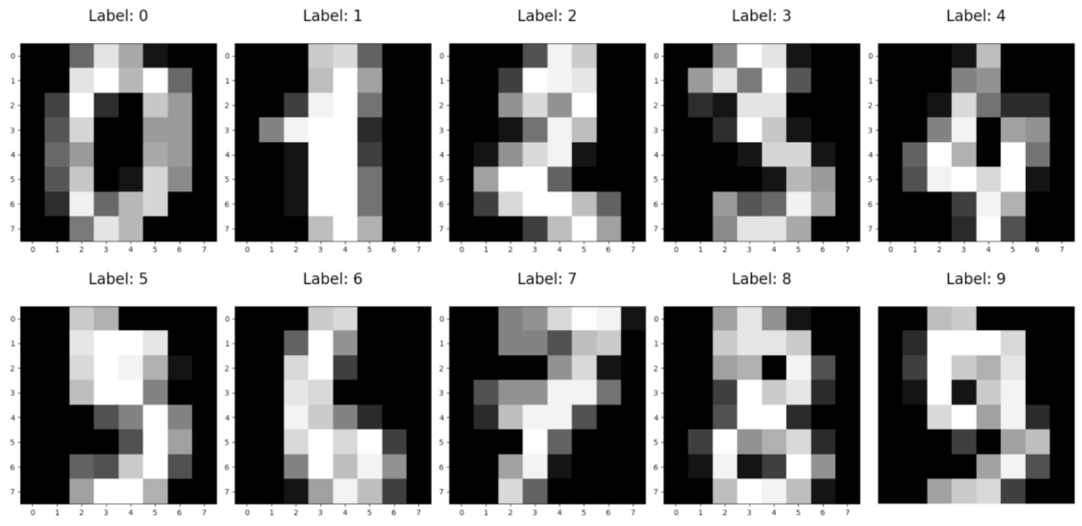
Lab 4, T3 2019

The goal of this lab is to become familiar with the training/testing process for pattern recognition / machine learning algorithms. Background information is provided AFTER all the questions.
In this lab, we will implement a K-Nearest Neighbour (kNN) classifier, a Stochastic Gradient Descent (SGD) classifier and a Decision Tree (DT) classifier.

Code and results should be submitted via WebCMS3 for marking (1 mark).
Submission is due at 23:59:59 on October 18th, 2019. Submission instructions will be posted prior to the lab.

Preliminaries

The following experiments will be based on sci-kit learn's **digits** dataset, which has 1797 images and 1797 corresponding labels. This dataset contains low resolution (8x8) images of digits ranging from 0 to 9, and was designed to test classification algorithms.



Sample of the first 10 training images and their corresponding labels.

We will predominantly be using sci-kit learn for this lab, so make sure it is downloaded. The following sci-kit learn libraries will need to be imported:

```
from sklearn import metrics
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

Lab Task - Image Classification (1 mark)

Develop a program to perform digit recognition. Classify the digits using the three classifiers mentioned above, and compare their results. The program should contain the following steps:

Set Up

1. Import relevant packages (most listed above)
2. Load the images using sklearn's `load_digits()`
   a. [optional] familiarize yourself with the dataset. For example, find out how many images and labels there are, the size of each image, and perhaps display some of the images and their labels. The code below will plot the first entry (digit 0):

```
plt.imshow(np.reshape(digits.data[0], (8, 8)), cmap='gray')
plt.title('Label: %i\n' % digits.target[0], fontsize=25)
```

3. Split the images using sklearn's `train_test_split()` with a test size anywhere between 20% and 30% (inclusive)

Classification (Repeat for each model)

4. Initialize the model (KNeighborsClassifier, SGDClassifier, DecisionTreeClassifier)
5. Fit the model to the training data
6. Use the trained/fitted model to evaluate the test data.

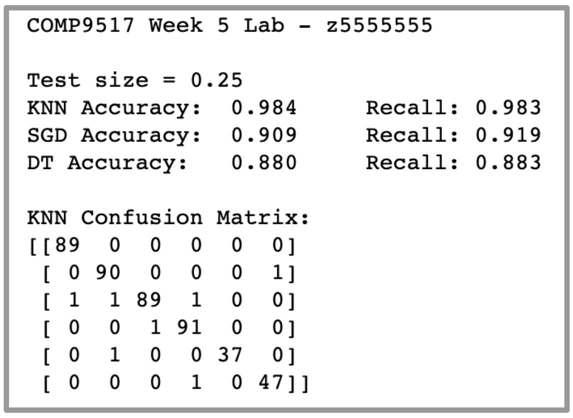
Evaluation

7. For each of the classifiers, evaluate the digit classification performance by calculating the accuracy, recall and generating the confusion matrix.

You are encouraged to experiment with the number of neighbours used in the KNN classifier in an attempt to find the best number for this dataset. You can adjust the number of neighbours with the `n_neighbours` parameter. The default value is `n_neighbours=5`

SUBMISSION

Print the **accuracy and recall** of all three classifiers, and the **confusion matrix** of the best performing classifier. **Submit** a screenshot for marking.
An example (for a 6 class model) can be seen here:

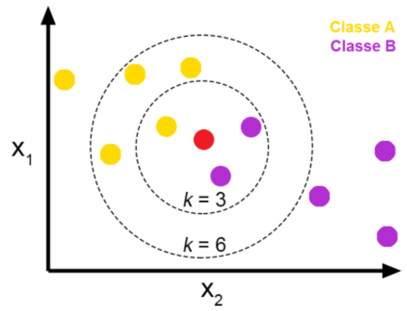


```
COMP9517 Week 5 Lab - z5555555

Test size = 0.25
KNN Accuracy:  0.984     Recall: 0.983
SGD Accuracy:  0.909     Recall: 0.919
DT Accuracy:   0.880     Recall: 0.883

KNN Confusion Matrix:
[[89  0  0  0  0  0]
 [ 0 90  0  0  0  1]
 [ 1  1 89  1  0  0]
 [ 0  0  1 91  0  0]
 [ 0  1  0  0 37  0]
 [ 0  0  0  1  0 47]]
```

Background Information

K - Nearest Neighbours

The KNN algorithm is very simple and very effective. The model representation for KNN is the entire training dataset. Predictions are made for a new data point by searching through the entire training set for the K most similar instances (the neighbours) and summarizing the output variable for those K instances. For regression problems, this might be the mean output variable, for classification problems this might be the mode (or most common) class value. The trick is in how to determine the similarity between the data instances.



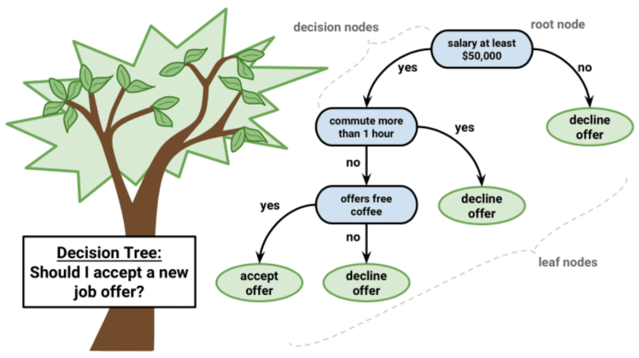
2 Class KNN example with 3 and 6neighbours (Towards Data Science)

Similarity

In order to make predictions we need to calculate the similarity between any two given data instances. This is needed so that we can locate the k most similar data instances in the training dataset for a given member of the test dataset and in turn make a prediction.

For a numeric dataset, we can directly use the Euclidean distance measure. This is defined as the square root of the sum of the squared differences between the two arrays of numbers.

Decision Tree Algorithm (see also https://en.wikipedia.org/wiki/Decision_tree_learning)



Algorithm for Construction of Decision Tree

1. Select a feature to place at the node (the first one is the root)
2. Make one branch for each possible value
3. For each branch node, repeat step 1 and 2
4. When all instances at a node have the same classification, stop developing that part of the tree

How to determine which feature to split on?

5. One way is to use measures from information theory
6. ***Entropy*** and ***Information Gain***

Entropy
To construct optimal decision trees from training data, we need a definition of optimality One simple criterion is ***entropy***, based on information theory. Entropy may be viewed as the average uncertainty of the information source.

Information Gain
Information gain is an entropy-based measure to evaluate features and produce optimal decision trees. When splitting, we use the feature with highest information gain to split on.

Note: the decision tree for this classification has been uploaded in webCMS.

SGD Classifier (See https://scikit-learn.org/stable/modules/sgd.html)

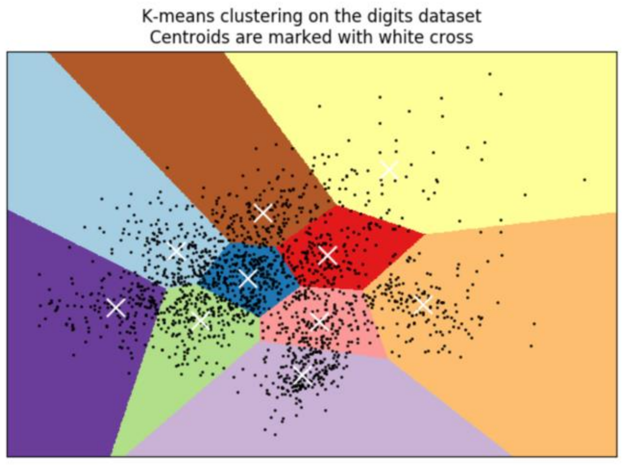
Extensions from this lab
Experiment with different k values and different parameters in KNeighborsClassifier from scikit-learn, and see their effects on the recognition accuracy and efficiency.

You can adjust the number of neighbours with the `n_neighbours` parameter. The default value is `n_neighbours=5`

Refer to the scikit-learn documentation for available parameters:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Experiment with different classifiers: https://scikit-learn.org/stable/modules/multiclass.html

There are many more models to experiment with. Here is an example of a clustering model on this dataset:



References:

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_index.html
https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html