NNFS Lab -1

Handwritten Digits Recognition

AIM: Training, evaluating and testing a machine learning algorithm to identify handwritten digits.

PLATFORM & TOOLS USED:

1. Google colab
2. Python
3. Sklearn

THEORY:

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

- Supervised learning: The computer is presented with example inputs and their desired outputs, the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent).

The problem discussed in this report belongs to the Supervised Learning Category.

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.
Types of supervised learning algorithms include active learning, classification and regression.
Our problem belongs to the classification category.

Classification Models:

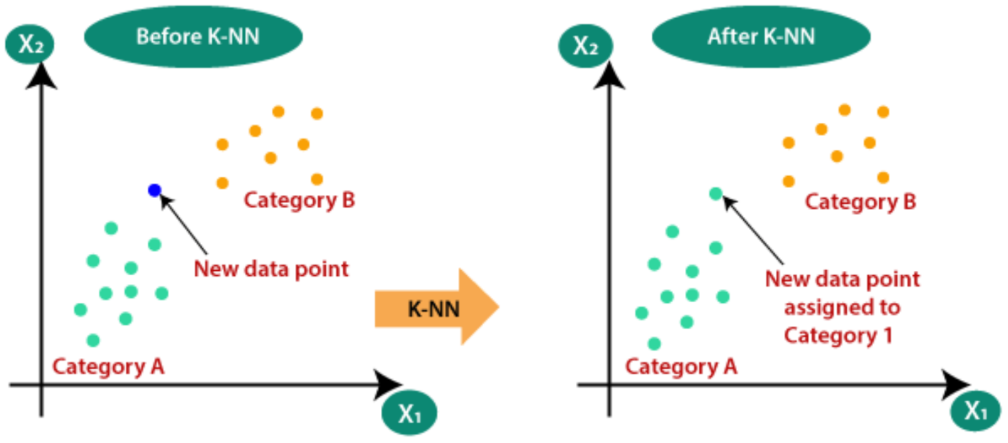
Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. As an example, for a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

1. **K- Nearest Neighbours (KNN) Model:**

   KNN is supervised machine learning algorithm which can be used for both classification and regression problems. In the case of classification K - Nearest Neighbour can be used for both binary and multi-class classifications. K-nearest neighbour or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line. Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to over fit the data and resulting in complex models. Note: It's very important to have the right k-value when analysing the dataset to avoid over-fitting and under-fitting of the dataset. Using the k-nearest neighbour algorithm we fit the historical data (or train the model) and predict the future.

Why do we need KNN algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.



To implement KNN algorithm you need to follow following steps.

Step-1: Select the number K of the neighbours

Step-2: Calculate the Euclidean distance of K number of neighbours

Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.

Step-4: Among these k neighbours, count the number of the data points in each category.

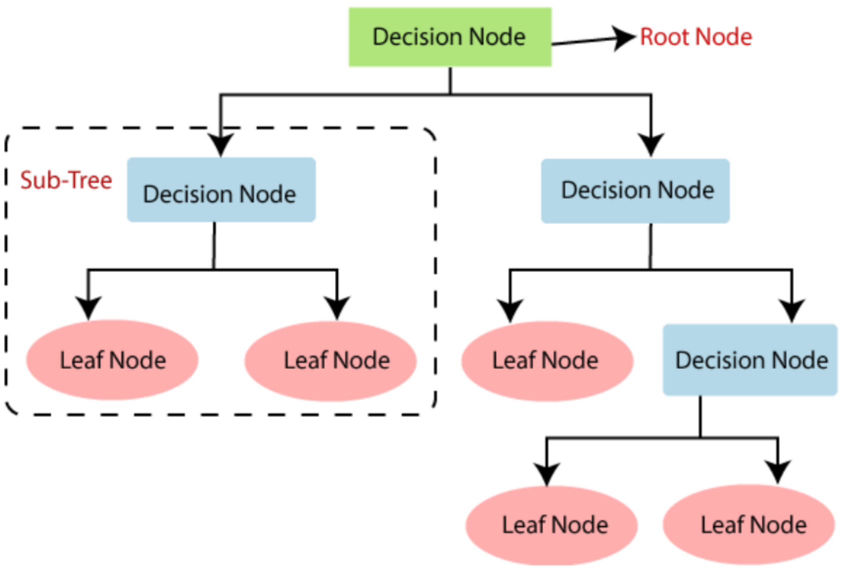
Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.

Step-6: Our model is ready.

2. Decision Tree Model:

Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub-trees.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

1. Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
2. The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

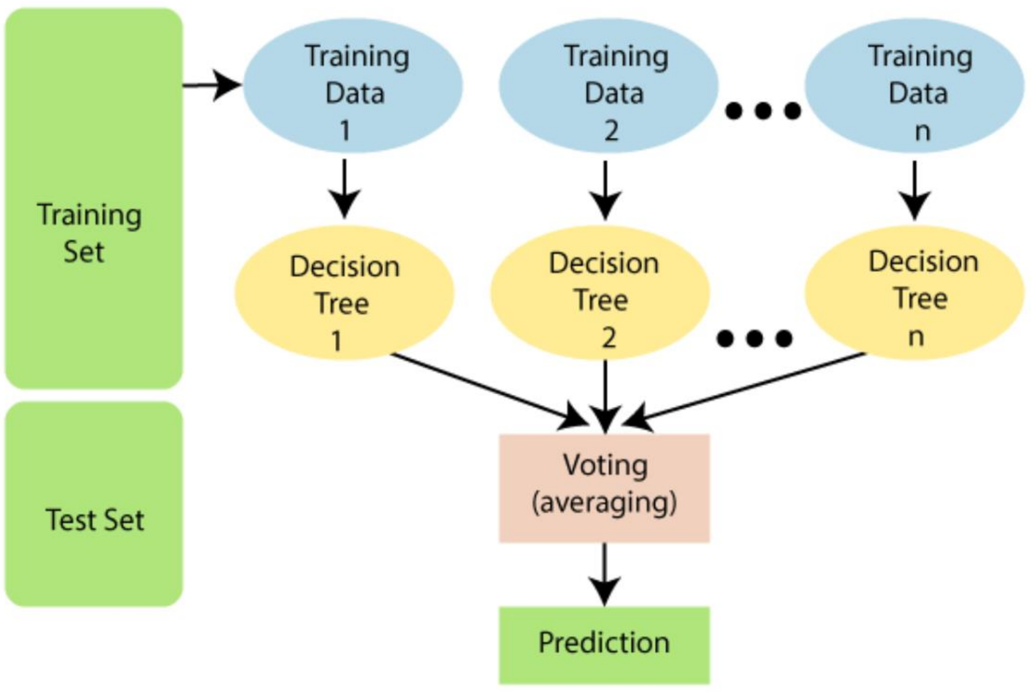
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

3. **Random Forest model:**

The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then. It collects the votes from different decision trees to decide the final prediction. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

1. It takes less training time as compared to other algorithms.
2. It predicts output with high accuracy, even for the large dataset it runs efficiently.
3. It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

CODE:

```
from sklearn.datasets import load_digits

dataset = load_digits()
data=dataset.data
```

```
images=dataset.images
target=dataset.target

print('data shape:',data.shape)
print('images shape:',images.shape)
print('data shape:',target.shape)
from matplotlib import pyplot as plt
plt.imshow(images[17],cmap='gray')

from sklearn.model_selection import train_test_split
train_data,test_data,train_target,test_target=train_test_split(data,target,test_siz
e=0.2)

from sklearn.neighbors import KNeighborsClassifier
#import KNN Algorithm
model=KNeighborsClassifier(n_neighbors=3) #loading our algorithm into an object cal
led model. Setting K=3.
#New KNN Algorithm (Object)
model.fit(train_data,train_target) #we have trained our model now.
#training the KNN Algorithm

#Applying testing images into the trained model
predicted_targets = model.predict(test_data)

from sklearn.metrics import accuracy_score
acc=accuracy_score(test_target,predicted_targets)
print('Accuracy:',acc)
```

KNN from Scratch

```
# distance metrics
import numpy as np
from scipy.stats import mode
def eucledian(p1, p2):
    dist = np.sqrt(np.sum((p1-p2)**2))
    return dist

def manhattan(p1, p2):
    dist = sum(np.abs(p1-p2))
    return dist
# function to calculate KNN
def knn_classify(x_train, y, x_input, k):
    op_labels = []

    #Loop through the Datapoints to be classified
    for item in x_input:

        #Array to store distances
        point_dist = []

        #Loop through each training Data
        for j in range(len(x_train)):
            distances = eucledian(np.array(x_train[j,:]) , item)
            #Calculating the distance
            point_dist.append(distances)
        point_dist = np.array(point_dist)
        #Sorting the array while preserving the index
        #Keeping the first K datapoints
        dist = np.argsort(point_dist)[:k]

        #Labels of the K datapoints from above
        labels = y[dist]

        #Majority voting
        lab = mode(labels)
        lab = lab.mode[0]
        op_labels.append(lab)

    return op_labels

predicted_data = knn_classify(train_data, train_target, test_data, 3)
from sklearn.metrics import accuracy_score
acc_scr=accuracy_score(test_target,predicted_data)
print('Accuracy:',acc_scr)
```

Using Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
dt_classifier.fit(train_data, train_target)
y_pred_dt = dt_classifier.predict(test_data)
print("The accuracy is : ", accuracy_score(test_target, y_pred_dt))
```

Using Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', r
andom_state = 0)
rf_classifier.fit(train_data, train_target)
y_pred_rf = rf_classifier.predict(test_data)
from sklearn.metrics import confusion_matrix, accuracy_score
print("The accuracy is : ", accuracy_score(test_target, y_pred_rf))
```

EXPLANATION:

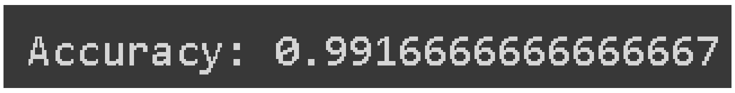
1. The problem belongs to classification category. We have to classify the handwritten digits into classes numbered from 0-9.
2. There are many Machine Learning approaches for classification problems. We chose KNN, Decision Tree and Random Forest algorithms out of them.
3. We first classified the test data using the inbuilt KNN algorithm provided by sklearn. Then we also implemented a KNN algorithm from scratch and compared the accuracies given by both the algorithms.
4. We also observed the effects on the results by changing the parameters such as “k”, “distance metrics”.
5. To explore further we also classified the data using random forest and decision tree algorithms to compare the results with KNN.

RESULTS:

Success of the Prediction is determined by the accuracy between the predicted value and the original value. Hence we use accuracy as a parameter to compare various models. Accuracy of various approaches are shown below and are deemed as results of this experiments.

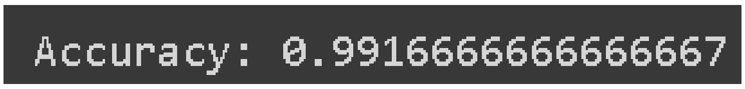
1. Using KNN method with from sklearn:

   With k = 3



```
Accuracy: 0.9916666666666667
```

   With k = 5

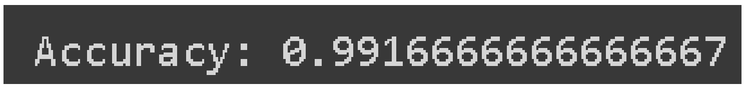


```
Accuracy: 0.9916666666666667
```

   With k = 7
   Same as above

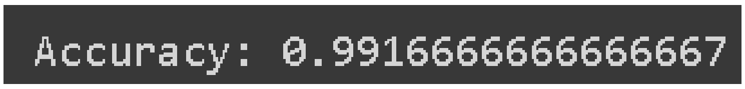
2. Using self-made KNN (KNN from scratch):

   Distance Metric = “Euclidian Distance”
   With k = 3



```
Accuracy: 0.9916666666666667
```

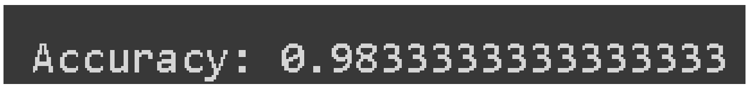
   With k = 5



```
Accuracy: 0.9916666666666667
```

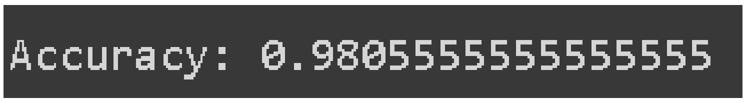
   With k = 7
   Same as above

   Distance Metric = “Manhattan Distance”
   With k = 3



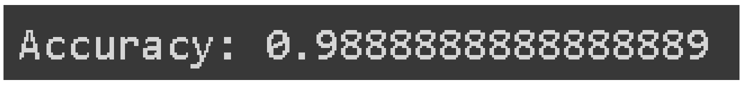
```
Accuracy: 0.9833333333333333
```

   With k = 5



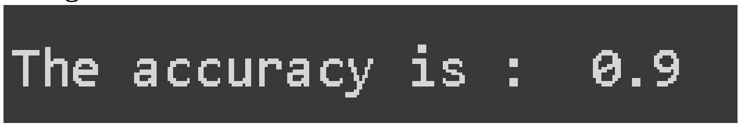
```
Accuracy: 0.9805555555555555
```

   With k = 7



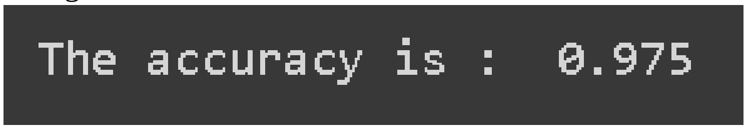
```
Accuracy: 0.9888888888888889
```

3. Using Decision Tree:



```
The accuracy is :  0.9
```

4. Using Random Forest:



```
The accuracy is :  0.975
```

CONCLUSION:

1. KNN algorithm provided by sklearn is by far the best algorithm this problem.
2. Self-made KNN algorithm from scratch gives same accuracy as that given by KNN from sklearn, however takes more time to run.
3. The accuracy when Manhattan distance is used, is less than when Euclidean distance is used, because Euclidean distance is symmetric in all directions while Manhattan distance is more limiting than Euclidean distance.
4. Because of high accuracy of KNN algorithm, the value of “k” does not seem to be affecting the result.
5. Decision Tree gives less accuracy than KNN, because DT not only checks for numeric features (distance in this case) but also extracts qualitative features. Also DT works better for small number of classes and tends to be overfitting when number of classes increase.
6. Decision Tree completed instantly with 90 % accuracy , Random Forest with 97.5 % accuracy with very less running time and KNN with 99.167 % accuracy with considerable running time and occupying the resources all along.