# Assignment No : 01 ( Final )
# Assignment name : K-Nearest Neighbor Algorithm

CSE-0408 Summer 2021

Farhana Rahman
*Department of Computer Science and Engineering*
*State University of Bangladesh (SUB)*
Dhaka, Bangladesh
swapnarahman40@gmail.com

***Abstract*—Many algorithms have been implemented to the problem of text categorization. Most of the work in this area was carried out for the English text; on the other hand very few researches have been carried out for the Arabic text. In this project we have implemented the key Nearest Neighbor (kNN) algorithm, which is known to be one of top performing classifiers applied for the English text along with the Support Vector Machines (SVMs) algorithm. However the nature of Arabic text is different than that of the English text and the preprocessing of the Arabic text is different and a little bit more challenging. For the problem of keyword extraction and reduction we have implemented a method to extract keywords based on the Document Frequency threshold (DF) method. The results show that kNN is applicable to Arabic text; we have reached a 0.95 micro-average precision and recall scores, using a data set of 621 Arabic text documents that belong to 6 different categories for training and testing.**

***Index Terms*—Text Categorization, kNN, Similarity Measuring, Vector Model, Terms Weighting, Keywords Extracting,**

## I. Introduction

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. KNN also known as K-nearest neighbour is a supervised and pattern classification learning algorithm which helps us find which class the new input(test value) belongs to when k nearest neighbours are chosen and distance is calculated between them.

## II. LITERATURE REVIEW

Step – 1: Establish the type of training examples. The user needs to courage the type(s) of data that will be used as a training set. Step – 2: Converge a training set. The training set ambition to be delegate of the real-world use of the function. As a effort, a set of input objects is collected that remains and analogous outputs are also collected. Step – 3: Resolve the input feature illustration of the learned function / learned attribute. The accurateness of the learned function is securely based on the input object is representation. Step – 4: Resolve the formation of the learned function and comparable machine learning algorithm. Step – 5: Assimilate the design and execute the learning algorithm on the collected training set. Step – 6: Evaluate the accurateness / correctness of the learned function. Then, parameter adapt and learning may be performed on the resulting function and needs to be measured on a test data set that is break up from the training set.

## III. PROPOSED METHODOLOGY

When we classified a data set including large number of unlabeled data, if only utilize the few training examples available, then we can't obtain a high accuracy classifier with inadequate training examples; if we want to obtain a classifier of high performance, then labeling the unlabeled data is necessary, but labeling vast unlabeled data wastes time and consumes strength. In this paper, we propose a novel method which uses SVM cooperated with KNN for classification based on semi supervised learning theory. The general model is depicted as above (See Figure 2). To begin with, we construct a weaker classifier SVM according to the few training examples available, then using the weaker SVM classifies the remaining large number of unlabeled data in the data set, picking out n examples belonging to each class around the decision boundary by calculating Euclidean distance in the feature space, because the examples located around the boundary are easy to be misclassified, but they are likely to be the support vectors, we call them boundary vectors, so picking out these boundary vectors whose labels are fuzzy labeled by the weaker classifier SVM. Secondly we recognize these boundary vectors as testing set while recognize initial training examples as training set, use KNN method to classify them and recognize the results as the labels for boundary vectors. In the end, we put these boundary vectors and their labels into initial training set to enlarge the number of the training examples then retrain a SVM, iteratively until the number of the training examples is m times of the whole data set. The experimental results on three UCI data sets indicate that the final classifier SVM has significant improvement on accuracy.

## IV. Conclusion

Every learning algorithm will tend to suit some problem types better than others, and will typically have many different parameters and configurations to be adjusted before achieving optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best outof-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples. The supervised machine learning algorithms such as decision trees and support vector machine are capable enough to deal with big data mining tasks. Even though the algorithms efficiency considerably improving there is a need for adaptive boosting process required in order to increase the predictive accuracy much more. The following are the findings from this survey research manuscript. (i) Fuzzy logic which is a soft computing technique is incorporated with the decision tree machine learning algorithm in order to rule out the ambiguity in the datasets. (ii) Example – dependent along with cost sensitive factors helps the decision trees to proclaim more independency in machine learning process. (iii) Error margins based methods reduce the false negative values while making use of decision trees. (iv) Interactions between behaviour variables tend to improve the performance of the decision trees. (v) Weight based structural information helps the support vector machine to quickly train the machine learning algorithm. (vi) Relationships between inter-class and intra-class surely will increase the effectiveness of the support vector machine. (vii) Decomposition of the attributes also significantly improves the effectiveness of the classifier. (viii)Noise detection process will be helpful to increase the accuracy of the machine learning algorithm. (ix) Cluster based boosting still has further scope of research by making use of optimization techniques. From the above findings it is interesting to note that the clustering or classification accuracy directly depends on the employment of boosting process. Not only that the overall computational complexity would be reduced then. This survey research article chooses two machines learning algorithm and one boosting technique and portrayed on the recent research works carried out during 2014 to 2017.

```
In [14]:
# 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))
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
In [13]:
# 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
```

*a) fig :1.0*

*:*

```
# 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)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
0.9666666666666667
In [12]:
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

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)
```

*b) fig :1.1*

:

```
# 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)

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 A
plt.plot(neighbors, train_accuracy, label = 'Training dataset

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

In [ ]:
```

c) *fig :1.2*

:

```
#!/usr/bin/env python
# coding: utf-8

# In[14]:


# 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))


# In[13]:


# Import necessary modules
```

d) *fig :2.0*

:

```
# 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)

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


# In[12]:


# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
```

e) *fig :2.1*


## ACKNOWLEDGMENT

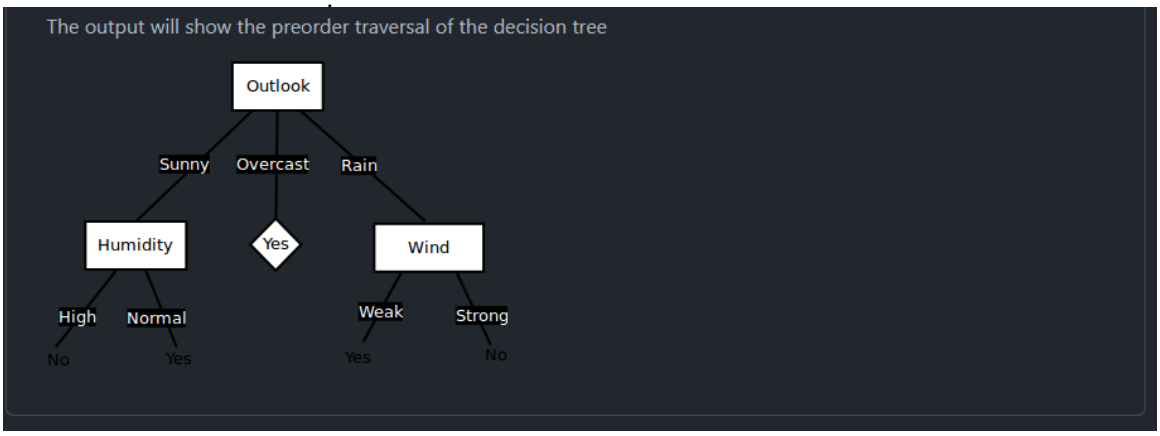
I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

:

```
import matplotlib.pyplot as plt

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)

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()
# In[ ]:
```

*f) fig :2.2*

:




*g) fig :2.3*

:

## REFERENCES


[1] X.J. Zhu. Semi-supervised learning literature survey[R]. Technical Report 1530, Department of Computer Sciences, University of Wisconsin at Madison, Madison, WI, December, 2007.

[2] Miller, D. J., and Uyar, H. S. A mixture of experts classifier with learning based on both labeled and unlabeled data. Advance in NIPS 9.571–577, 1997.

[3] Fung, G., and Mangasarian, O. Semi-supervised support vector machines for unlabeled data classification (Technical Report 99-05). Data Mining Institute, University of Wisconsin Madison, 1999

[4] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien. Semi- Supervised Learning [M]. The MIT Press, 2006

[5] Zhou, Z.-H., Zhan, D.-C., and Yang, Q. Semi-supervised learning with very few labeled training examples. TwentySecond AAAI Conference on Artificial Intelligence (AAAI-07), 2007.