**Project: Letter Recognition**

**Huseyin Cem Ates / S13869**

Table of Contents

[1. Data source 1](#_Toc535780741)

[2. Task Descriptions 1](#_Toc535780742)

[3. Task 1](#_Toc535780743)

[4. Implement the classification algorithm 1](#_Toc535780744)

[5. Testing model 1](#_Toc535780745)

[Number of Instances: 2](#_Toc535780746)

[Number of Attributes: 2](#_Toc535780747)

[Attribute Information: 2](#_Toc535780748)

[Missing Attribute Values: 2](#_Toc535780749)

[Class Distribution: 3](#_Toc535780750)

[Algorithms Description 3](#_Toc535780751)

[Compiling & Running: 3](#_Toc535780752)

[Data Inputs & Results: 4](#_Toc535780753)

[Conclusion 5](#_Toc535780754)

[Dependencies 6](#_Toc535780755)

[\*[GCC, the GNU Compiler Collection Version 4.2.1] (https://gcc.gnu.org/) 6](#_Toc535780756)

[References 6](#_Toc535780757)

# Data source

The data used to recognize the letters come from the Machine Learning Repository UCI and are available under the link:

<https://archive.ics.uci.edu/ml/datasets/letter+recognition>

# Task Descriptions

The objective is to identify each of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 examples. Each example was converted into 16 primitive numerical attributes which were scaled to fit into a range of integer values from 0 through 15. We typically train on the first 16000 items and then use the resulting model to predict the letter category for the remaining 4000.

# Task

Apply multilayer neural network (NN) to recognize the labels of the letters.

# Implement the classification algorithm

• Tune the parameters to find the best neural network for the given problem.

• Test the classifier for the unseen (testing) examples. Give the classification accuracy, describe in details the experimental results, show observations, remarks and the final conclusion.

# Testing model

• Training data: 16000 first examples in the file letter-recognition.data.

• Testing data: 4000 last examples in the same file.

• Evaluation measure: Classification accuracy - percent of the correctly classified examples in the testing set.

Data Set Information:

**Number of Instances:**

20000

## **Number of Attributes:**

17 (Letter category and 16 numeric features)

## **Attribute Information:**

1. lettr capital letter (26 values from A to Z)

2. x-box horizontal position of box (integer)

3. y-box vertical position of box (integer)

4. width width of box (integer)

5. high height of box (integer)

6. onpix total # on pixels (integer)

7. x-bar mean x of on pixels in box (integer)

8. y-bar mean y of on pixels in box (integer)

9. x2bar mean x variance (integer)

10. y2bar mean y variance (integer)

11. xybar mean x y correlation (integer)

12. x2ybr mean of x \* x \* y (integer)

13. xy2br mean of x \* y \* y (integer)

14. x-ege mean edge count left to right (integer)

15. xegvy correlation of x-ege with y (integer)

16. y-ege mean edge count bottom to top (integer)

17. yegvx correlation of y-ege with x (integer)

## **Missing Attribute Values:**

None

## **Class Distribution:**

789 A 766 B 736 C 805 D 768 E 775 F 773 G

734 H 755 I 747 J 739 K 761 L 792 M 783 N

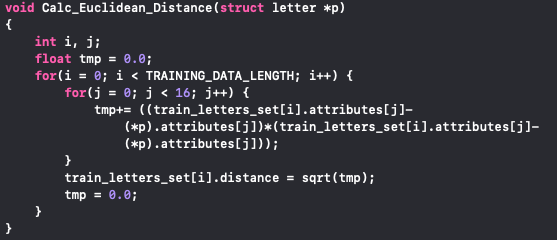
753 O 803 P 783 Q 758 R 748 S 796 T 813 U

764 V 752 W 787 X 786 Y 734 Z

# Algorithms Description

The case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. In my instances, I use K = 1, then the case is simply being assigned to the class of its nearest neighbor. And then increasing the K value to test if when changing the k, will improve classification accuracy But the results were decreasing.

And, I am taking a sample from dataset as training data. For calculating distance between test point and all the training points I am calculating the Euclidean Distance between all training points and test point using square root operation. The nearest k neighbors are stored in the list.



# Compiling & Running:

**$** gcc letters.c -o knn

**$** ./knn

# 

# Data Inputs & Results:

─ test 4000.txt (test 4000 contains 4000 remaining item samples, acting as testing set.)

─ train16000.txt (train16000 contains 16000 first item samples, acting as training set.)

# Pros and Cons of knn

Pros

* Can work with large sample size classification problem.
* Good efficiency. (Relatively speaking)
* Algorithm's concept is easy to comprehend and then implement.
* It's a 'Incremental learning' model like Naive Bayesian. Does not retrain the model while input data grow.

Cons

* Every sample in training set requires global search, will affect efficiency when data size is huge (due to the amount of calculation).
* Misclassification happen when sample distribution is uneven. (Value majority more than minority)
* Needs more physical memory when data size increase.

**1st Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 1

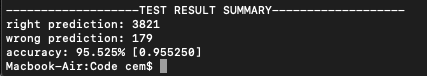
---------**TEST RESULT SUMMARY**---------

right prediction: 3821

wrong prediction: 179

accuracy: 95.525% [0.955250]

---------------------------------------

****

**2nd Experimental Result:**

---------------------------------------

rows of training data set: 15000

rows of test data set: 5000

input K in K-nearest-neighbor algorithm: 1

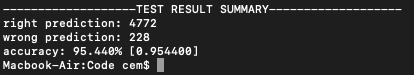
---------**TEST RESULT SUMMARY**---------

right prediction: 4772

wrong prediction: 228

accuracy: 95.440% [0.954400]

---------------------------------------



**3rd Experimental Result:**

---------------------------------------

rows of training data set: 19900

rows of test data set: 100

input K in K-nearest-neighbor algorithm: 1

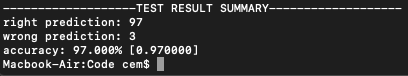
---------**TEST RESULT SUMMARY**---------

right prediction: 97

wrong prediction: 3

accuracy: 97.000% [0.970000]

---------------------------------------

****

**4rd Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 2

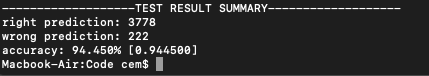
---------**TEST RESULT SUMMARY**---------

right prediction: 3778

wrong prediction: 222

accuracy: 94.450% [0.944500]

---------------------------------------



**5th Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 5

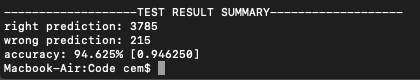
---------**TEST RESULT SUMMARY**---------

right prediction: 3785

wrong prediction: 215

accuracy: 94.625% [0.946250]

---------------------------------------



**6th Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 10

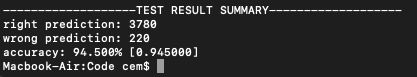
---------**TEST RESULT SUMMARY**---------

right prediction: 3780

wrong prediction: 220

accuracy: 94.500% [0.945000]

---------------------------------------



**7th Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 50

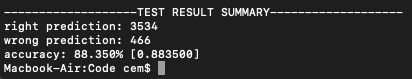
---------**TEST RESULT SUMMARY**---------

right prediction: 3534

wrong prediction: 466

accuracy: 88.350% [0.883500]

---------------------------------------



**8th Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 50

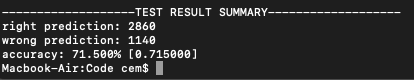
---------**TEST RESULT SUMMARY**---------

right prediction: 2860

wrong prediction: 1140

accuracy: 71.500% [0.715000]

---------------------------------------



**Weka Experimental Result:**

---------------------------------------

rows of training data set: 16000

rows of test data set: 4000

input K in K-nearest-neighbor algorithm: 5

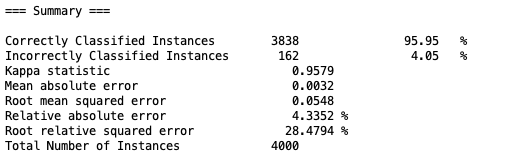
---------**TEST RESULT SUMMARY**---------

right prediction: 3838

wrong prediction: 162

accuracy: 95.95%

---------------------------------------



# Conclusion

*After Learning for 80% of Data Set (Training Sets)*

*Neural Network properly able to classify for 20% remaining Training Set.*

The developed package can be used in the future in rapid prototyping of simple ensemble models, using different kernel models. What is more, for certain results, our implementation performed nearly as good as the baseline. We also confirmed our early assumption, that generally increasing the value of the defined parameters.

# Dependencies

# \***[**GCC, the GNU Compiler Collection Version 4.2.1**]** (<https://gcc.gnu.org/)>

# \***[**Weka: Data Mining Software**]** (<https://www.cs.waikato.ac.nz/ml/weka/)>

# References

\* <https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>

\* <https://medium.com/@sunnerli/visual-attention-in-deep-learning-77653f611855>

\* <https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/>

\* Ensemble learners

<https://www.youtube.com/watch?v=Un9zObFjBH0>

\* Bootstrap aggregating bagging

<https://www.youtube.com/watch?v=2Mg8QD0F1dQ>

\* Bagging

<https://www.youtube.com/watch?v=sVriC_Ys2cw>

\* Random Forest

<https://en.wikipedia.org/wiki/Random_forest>

\* Recursive Partitioning

\*\* <https://en.wikipedia.org/wiki/Recursive_partitioning>

\* Feature Selection

\*\* <https://en.wikipedia.org/wiki/Feature_selection>