LAPORAN TUGAS 1
KECERDASAN KOMPUTASIONAL

Implementasi Algoritma k-Nearest Neighbors (k-NN) Menggunakan Python dan Weka dengan Variasi *k* dan Variasi Fungsi Jarak

KELAS: C

5115100043 HAFARA FIRDAUSI

Dosen:

Dini Adni Navastara, S.Kom.,M.Sc

Asisten:

M. ANWAR HIDAYAT

Jurusan Teknik Infomatika - Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember (ITS)
Surabaya

2017

BAB I
PENDAHULUAN

1.1 Latar Belakang

Machine Learning adalah pengambilan pengetahuan dari suatu data. Penerapan metode *Machine Learning* dalam beberapa tahun terakhir telah berkembang di mana-mana dalam kehidupan sehari-hari. Dari rekomendasi film, makanan apa yang harus dipesan atau produk mana yang akan dibeli, ke radio *online* yang telah dipersonalisasi dan mengenali teman di foto Anda, banyak situs web dan perangkat modern memiliki algoritma *Machine Learning* pada intinya. Ketika Anda melihat situs web yang kompleks seperti *Facebook, Amazon,* atau *Netflix,* kemungkinan besar setiap bagian situs berisi beberapa model *Machine Learning*.

Salah satu jenis Algoritma *Machine Learning* adalah Algoritma *Supervised Learning*, yaitu algoritma yang terdiri dari variabel dependen (data *train*) yang akan menjadi bahan prediksi dari himpunan prediktor (data *test*). Salah satu contoh Algoritma *Supervised Learning* yang paling populer adalah *k-Nearest Neighbor (k*-NN). Oleh karena itu, kita sebagai anak informatika seharusnya mengerti bagaimana mengimplementasikan Algoritma *k*-NN dengan baik menggunakan Python dan Weka.

1.2 Tujuan

- Dapat mengimplementasikan Algoritma *k*-NN dengan baik menggunakan Python dan Weka
- Dapat menganalisis dan menyimpulkan hasil uji coba dengan variasi *k* dan variasi fungsi jarak

1.3 Detail Tugas

Implementasi *k*-NN menggunakan Python dan Weka

1. Menggunakan Data UCI Machine Learning: Pima Indian Diabetes Data Set dan Housing Data Set
2. Menggunakan fungsi jarak *(distance)*: Euclidean Distance, Manhattan Distance, Cosine Similarity, dll
3. Nilai *k* dinamis/ bervariasi (bisa di*input*kan)
4. Menggunakan fungsi normalisasi
5. Menggunakan 10-*Fold Cross Validation*
6. Menghitung Akurasi untuk klasifikasi Pima Indians Diabetes Data Set
7. Menggunakan fungsi *k*-NN untuk regresi pada Boston Housing Data Set dan klasifikasi pada Pima Indian Diabetes Data Set

BAB II
METODE

2.1 Waktu dan Tempat

Waktu pengerjaan tugas adalah 1 minggu dan disusul dengan demo tugas kepada asisten dosen. Tempat pengerjaan tugas adalah bebas.

2.2 Kebutuhan (*Requirements)*

- *Personal Computer*/ Laptop
- Sistem Operasi Windows/ Linux (bebas)
- Bahasa Pemrograman Python beserta *environment*-nya
  Ada 5 *library* utama yang harus terinstall :
  1. Scipy
  2. Numpy
  3. Matplotlib
  4. Pandas
  5. Sklearn
- Software Weka 3: Data Mining Software in Java
- Data Set :
  1. Pima Indian Diabetes Data Set
  2. Boston Housing Data Set

2.3 Prosedur Kerja

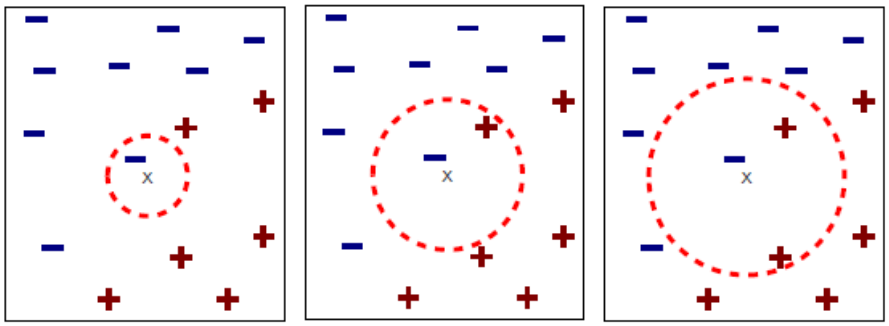
1. Meng-*install* Python *environment* ke dalam sistem operasi yang digunakan (saya menggunakan Linux Ubuntu 16.04), beserta 5 *library* yang harus terinstall
2. Meng-*install* Weka 3: Data Mining Software in Java (bisa di download dari web resminya, *http://www.cs.waikato.ac.nz/ml/weka/*)
3. Men-*download* Pima Indian Diabetes Data Set *(https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes)* dan Boston Housing Data Set *(https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)* di web resmi UCI Machine Learning Repository *(https://archive.ics.uci.edu/ml/datasets.html)*
4. Melakukan uji coba implementasi k-NN *algorithm* terhadap dataset menggunakan Python dan Weka dengan variasi *k* dan variasi fungsi jarak
5. Melakukan analisa terhadap hasil yang didapat

BAB III
DASAR TEORI

3.1 *k-Nearest Neighbors* (*k*-NN)

3.1.1 Definisi *k*-NN

***k-Nearest Neighbor* (*k*-NN)** adalah sebuah metode untuk melakukan klasifikasi terhadap objek berdasarkan data *training* yang jaraknya paling dekat dengan objek tersebut. Data *training* diproyeksikan *k*-ruang berdimensi banyak, dimana masing-masing dimensi merepresentasikan fitur dari data. Tujuan dari algoritma ini adalah mengklasifikasikan objek baru berdasarkan atribut dan data *training.*



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Nearest Neighbor

Dalam pengenalan pola, *k-Nearest Neighbor* (*k*-NN) adalah metode non-parametrik (model yang tidak mengasumsikan apa-apa mengenai distribusi instance di dalam dataset) yang digunakan untuk **klasifikasi** dan **regresi**. Dalam kedua kasus tersebut, *input* terdiri dari *k*-data *training* terdekat di ruang fitur. *Output*-nya bergantung pada apakah *k*-NN digunakan untuk klasifikasi atau regresi:

- Jika **Klasifikasi *k*-NN**, *output*-nya adalah *class*. Data *test* diklasifikasikan berdasarkan *majority vote* dari *class* tetangga terdekatnya *(nearest neighbor)* sejumlah *k*
- Jika **Regresi *k*-NN**, output-nya adalah nilai properti, yaitu nilai rata-rata dari nilai tetangga terdekatnya *(nearest neighbor)* sejumlah *k*

Kelebihan dari *k*-NN yaitu:

1. Tangguh terhadap data *training* yang memiliki banyak *noise*
2. Efektif apabila data *training*-nya besar

Sedangkan, kelemahan dari *k*-NN adalah:

1. *K*-NN perlu menentukan nilai dari parameter *k* (jumlah dari tetangga terdekat)
2. *Training* berdasarkan jarak tidak jelas mengenai jenis jarak apa yang harus digunakan
3. Atribut mana yang harus digunakan untuk mendapatkan hasil yang terbaik
4. Biaya komputasi cukup tinggi karena diperlukan perhitungan jarak dari tiap data *test* pada keseluruhan data *training*

3.1.2 Algoritma *k*-NN

Untuk setiap objek data *test* (X) dalam dataset:

1. Menghitung jarak antara X dengan setiap data *training*
2. Mengurutkan jarak dari yang paling kecil (kedekatan terbesar)
3. Mengambil *k*-data dengan jarak paling kecil dengan X
4. Mencari *majority vote* dari *class* diantara *k* data tersebut
5. Mengembalikan *majority vote* dari *class* tersebut sebagai prediksi class dari X

Cara memilih nilai *k* yang baik:

1. Jangan terlalu kecil, karena akan menghasilkan data *noise*
2. Jangan terlalu besar, karena tetangga (*neighborhood)* mungkin saja memasukkan data dari kelas lain

3.1.3 Perhitungan Jarak dalam *k*-NN

Dalam percobaan implementasi ini, saya hanya menggunakan 3 macam Algoritma perhitungan jarak, yaitu:

1. ***Euclidean Distance***

   Euclidean Distance membandingkan jarak minimum data *test* dengan data *training. Euclidean Distance* dihitung dengan persamaan:

$$dist = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2}$$

   Semakin kecil nilai d(x,y), maka semakin mirip kedua data yang dibandingkan. Sebaliknya, semakin besar nilai d(x,y), maka semakin berbeda kedua data yang dibandingkan.

2. ***Cosine Similarity***

   Cosine Similarity dihitung dengan persamaan:

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / \|d_1\| \|d_2\|$$

3. ***Manhattan Distance***
   Prosedur ini disebut blok absolut atau lebih dikenal dengan *city block distance. Manhattan Distance* dihitung dengan persamaan:

$$d(x,y) = L_p = i(x,y) = \sum_i^n \|x_i - y_i\|$$

3.1.4 Cara Implementasi Algoritma *k*-NN ke dalam Python

1. ***Handle Data*****:** Membuka dataset dari CSV dan membaginya menjadi data *training* dan data *testing*
2. ***Similarity*****:** Menghitung jarak antara 2 data
3. ***Neighbors:*** Menemukan *k* data yang paling mirip
4. ***Response:*** Mendapatkan hasil *majority vote* dari *class* dari tetangga
5. ***Accuracy:*** Meringkas ke akuratan prediksi
6. ***Main:*** Menggabungkan itu semua

Kita juga bisa menambahkan fungsi berikut dalam *code*:

- **Regresi:** Mengatasi masalah regresi (memprediksi atribut bernilai riil). Untuk mencari *instances* terdekat, dilakukan pengambilan rata-rata dari atribut yang di prediksi.
- **Normalisasi:** Mengatasi masalah perbedaan satuan ukuran antar atribut, yang menyebabkan suatu atribut sangat mendominasi atribut yang lain dalam hal kontribusi penghitungan jarak. Misal: Atribut A = 5 kg, Atribut B = Rp10.000, terlihat sangat tampak kesenjangan nilai antar 2 atribut (satuan dan puluh ribuan). Hal ini tentunya harus dinormalisasi karena kisaran range variabel yang besar akan menjadi bias. Caranya adalah dengan me-*rescale* semu data atribut dalam *range* 0-1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- **Ukuran Jarak Alternatif:** Ada banyak persamaan lain untuk menghitung jarak, seperti Manhattan Distance dan Cosine Similarity.

BAB IV
HASIL IMPLEMENTASI

4.1 Pima Indians Diabetes Data Set

Abstract: From National Institute of Diabetes and Digestive and Kidney Diseases; Includes cost data (donated by Peter Turney)

Data Set Characteristics:	Multivariate	**Number of Instances:**	768	**Area:**	Life
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	8	**Date Donated**	1990-05-09
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	314438

4.1.1 Python

- **Source Code**

1. ***Import Library***

```
### IMPORT LIBRARY ###
import csv
import random
import math
import operator
```

2. ***Handle Data:*** Membuka dataset dari CSV dan membaginya menjadi data *training* dan data *testing*. Tambahkan pula fungsi yang meng*handle missing value* dan fungsi normalisasi

```
### HANDLE DATA ###
def loadDataset (filename, split, trainingSet=[], testSet=[]):
        with open(filename, 'rb') as csvfile:
                lines = csv.reader(csvfile)
                dataset = list(lines)
                flag = 0

        # Handle missing value
        for x in range(len(dataset)):
                for y in range (len (dataset[x]) - 1):
                        if float (dataset[x][y]) == 0.0:
                                kolom = [float (i[y]) for i in dataset]
                                tidakNol = len(kolom) - kolom.count (0)
                                dataset[x][y] = float(sum(kolom)/tidakNol)
                        else:
                                dataset[x][y] = float(dataset[x][y])

        # Fungsi Normalisasi
        for x in range(len(dataset)):
                minx = min([i for i in dataset[x][:-1]])
                maxx = max([i for i in dataset[x][:-1]])
                for y in range (len (dataset[x]) - 1):
                        dataset[x][y] = (dataset[x][y] - minx) / (maxx - minx)

        # Membagi data training dan data testing
        for x in range(len(dataset)-1):
                for y in range(4):
                        dataset[x][y] = float (dataset[x][y])
                if flag < split * len(dataset):
                        trainingSet.append (dataset[x])
                        flag += 1
                else:
                        testSet.append (dataset[x])
```

3. ***Similarity:*** Menghitung jarak antara 2 data. Saya menggunakan 3 variasi fungsi jarak

```
### SIMILARITY ###
#EUCLIDEAN DIST
def euclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += float(pow((float(instance1[x]) - float(instance2
[x])), 2))
        return float(math.sqrt(distance))

#MANHATTAN DIST
def manhattanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                temp = instance1[x] - instance2[x]
        if temp < 0:
                temp*=-1
        distance += temp
        return math.sqrt(distance)

#COSINE SIMILARITY
def cosineSimilarity(instance1, instance2, length):
        distance = 0
        sumxx, sumxy, sumyy = 0, 0, 0
        for i in range(length):
                x = float(instance1[i])
                y = float(instance2[i])
                sumxx += x * x
                sumyy += y * y
                sumxy += x * y
        return 1 - (sumxy / math.sqrt(sumxx * sumyy))
```

4. ***Neighbors:*** Menemukan *k* data yang paling mirip

```
### K NEIGHBORS ###
def getNeighbors(trainingSet, testInstance, k, pilih):
        distances = []
        length = len(testInstance) - 1
        for x in range(len(trainingSet)):
                if pilih==1:
                        dist = euclideanDistance(testInstance,trainingSet[x],length)
                elif pilih==2:
                        dist = manhattanDistance(testInstance,trainingSet[x],length)
                elif pilih==3:
                        dist = cosineSimilarity(testInstance,trainingSet[x],length)
                distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
                neighbors.append(distances[x][0])
        return neighbors
```

5. ***Response:*** Mendapatkan hasil *majority vote* dari *class* dari tetangga

```
### RESPONSE -- MAJORITY VOTE OF CLASS ###
def getResponse(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
        if response in classVotes:
                classVotes[response] += 1
        else:
                classVotes[response] = 1
        sortedVotes = sorted(classVotes.items(),key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]
```

6. ***Accuracy***: Meringkas keakuratan prediksi

```
### ACCURACY ###
def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][-1] is predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0
```

7. ***Main:*** Menggabungkan semua fungsi

```
### MAIN ###
def main():
        # prepare data
        trainingSet = []
        testSet = []
        split = 0.8
        loadDataset('pima-indians-diabetes.data', split, trainingSet, testSet)
        print('\nData Train: ' + repr(len(trainingSet)))
        print('Data Test: ' + repr(len(testSet)))

        # generate predictions
        predictions = []
        k = input("\nMasukkan nilai k : ")
        print "\n1. Euclidean Distance\n2. Manhattan Distance\n3. Cosine Similarity\n"
        pilih = input("Masukkan pilihan fungsi jarak yang ingin digunakan : ")
        for x in range(len(testSet)):
                neighbors = getNeighbors(trainingSet, testSet[x], k, pilih)
                result = getResponse(neighbors)
                predictions.append(result)
                print('> Predicted=' + repr(result) + ', Actual=' + repr(testSet[x][-1]))
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: ' + repr(accuracy) + '%')

main()
```

- ***Testing***

  K = 3, Menggunakan fungsi jarak Euclidean Distance

```
root@hf-VirtualBox:/home/hf/Downloads/kk/tugas# python knn-klasifikasi.py

Data Train: 615
Data Test: 152

Masukkan nilai k : 3

1. Euclidean Distance
2. Manhattan Distance
3. Cosine Similarity

Masukkan pilihan fungsi jarak yang ingin digunakan : 1
> Predicted='1', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
> Predicted='1', Actual='1'
> Predicted='0', Actual='0'
> Predicted='0', Actual='1'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
> Predicted='1', Actual='1'
> Predicted='1', Actual='0'
> Predicted='1', Actual='0'
> Predicted='1', Actual='1'
> Predicted='1', Actual='0'
> Predicted='0', Actual='0'
> Predicted='0', Actual='1'
> Predicted='0', Actual='0'
> Predicted='0', Actual='1'
> Predicted='1', Actual='1'
> Predicted='1', Actual='1'
> Predicted='1', Actual='0'
> Predicted='1', Actual='0'
> Predicted='0', Actual='1'
> Predicted='1', Actual='1'
> Predicted='0', Actual='1'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
> Predicted='1', Actual='0'
> Predicted='0', Actual='0'
> Predicted='1', Actual='0'
> Predicted='0', Actual='0'
> Predicted='1', Actual='1'
Accuracy: 62.5%
```

- **Hasil**

  Total Data : 768

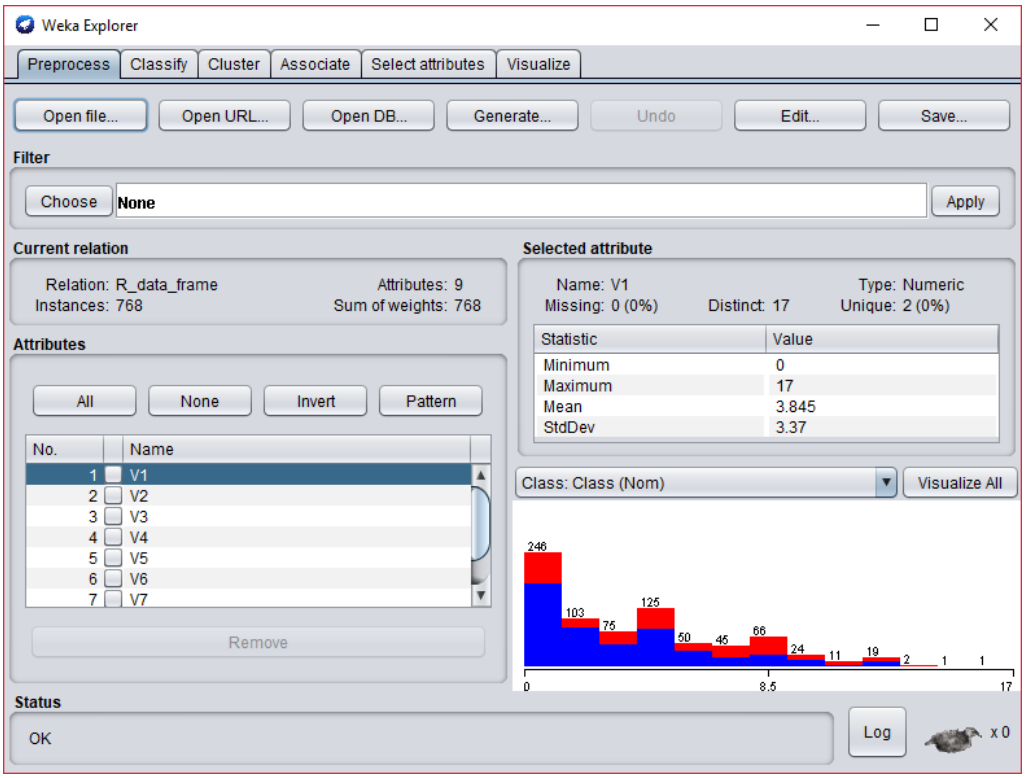
  Data *Train* : 615 (80% total data)

  Data *Test* : 152 (20% total data)

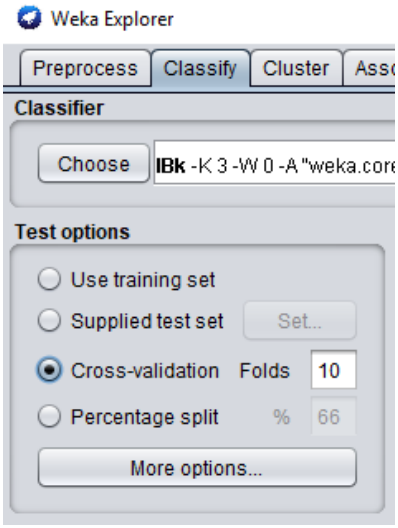
	Akurasi				
	k = 3	*k* = 5	*k* = 7	*k* = 9	*K* = 11
Euclidean Dist.	62.5%	68.4210%	61.1842%	63.8157%	65.7895%
Manhattan Dist.	52.6316%	56.5789%	56.5789%	58.5526%	50.6579%
Cosine Similarity	66.4473%	59.2105%	59.2105%	64.4737%	62.5%

4.1.2 Weka

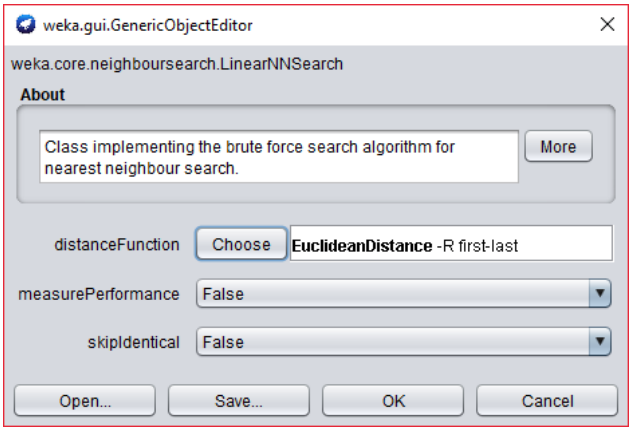
1. Buka Weka Explorer dan masukkan data pima-indians-diabetes.arff (*https://raw.githubusercontent.com/lpfgarcia/ucipp/master/uci/pima-indians-diabetes.arff*) ke dalam Weka melalui *Open File*



2. Atur sedemikian rupa seperti ini:
    a) 10-Cross Validation



3. Klasifikasi menggunakan *k*-NN/IBK dengan 2 variasi fungsi jarak
    - **Euclidean Distance**



a) *k* = 3

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         558               72.6563 %
Incorrectly Classified Instances       210               27.3438 %
Kappa statistic                          0.3822
Mean absolute error                      0.3092
Root mean squared error                  0.4525
Relative absolute error                 68.0324 %
Root relative squared error             94.9365 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.820    0.448    0.774      0.820    0.796      0.384    0.742     0.804     1
                 0.552    0.180    0.622      0.552    0.585      0.384    0.742     0.569     2
Weighted Avg.    0.727    0.354    0.721      0.727    0.722      0.384    0.742     0.722

=== Confusion Matrix ===

   a   b   <-- classified as
 410  90 |   a = 1
 120 148 |   b = 2
```

b) *k* = 5

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         562               73.1771 %
Incorrectly Classified Instances       206               26.8229 %
Kappa statistic                          0.3874
Mean absolute error                      0.3165
Root mean squared error                  0.4318
Relative absolute error                 69.6387 %
Root relative squared error             90.5982 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.836    0.463    0.771      0.836    0.802      0.390    0.766     0.828     1
                 0.537    0.164    0.637      0.537    0.583      0.390    0.766     0.619     2
Weighted Avg.    0.732    0.358    0.724      0.732    0.726      0.390    0.766     0.755

=== Confusion Matrix ===

   a   b   <-- classified as
 418  82 |   a = 1
 124 144 |   b = 2
```

c) *k* = 7

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         574               74.7396 %
Incorrectly Classified Instances       194               25.2604 %
Kappa statistic                          0.4189
Mean absolute error                      0.3178
Root mean squared error                  0.4209
Relative absolute error                 69.9184 %
Root relative squared error             88.3093 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.856    0.455    0.778      0.856    0.815      0.424    0.785     0.853     1
                 0.545    0.144    0.670      0.545    0.601      0.424    0.785     0.643     2
Weighted Avg.    0.747    0.347    0.740      0.747    0.740      0.424    0.785     0.780

=== Confusion Matrix ===

   a   b   <-- classified as
 428  72 |   a = 1
 122 146 |   b = 2
```

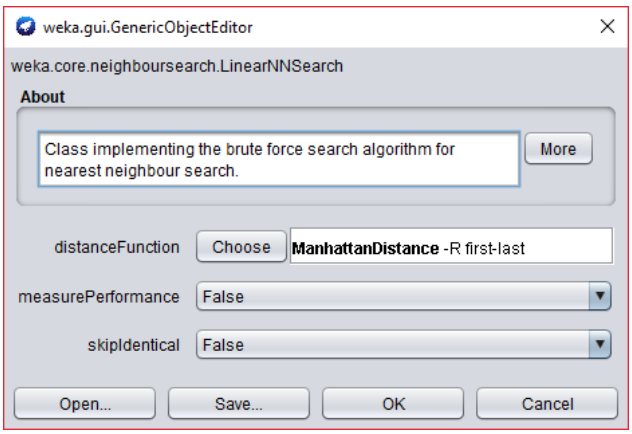
d) *k* = 9

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         554               72.1354 %
Incorrectly Classified Instances       214               27.8646 %
Kappa statistic                          0.3602
Mean absolute error                      0.3221
Root mean squared error                  0.4187
Relative absolute error                 70.8694 %
Root relative squared error             87.8344 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.834    0.489    0.761      0.834    0.796      0.364    0.790     0.859     1
                 0.511    0.166    0.623      0.511    0.561      0.364    0.790     0.643     2
Weighted Avg.    0.721    0.376    0.713      0.721    0.714      0.364    0.790     0.783

=== Confusion Matrix ===

   a   b   <-- classified as
 417  83 |   a = 1
 131 137 |   b = 2
```

e) *k* = 11

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         550               71.6146 %
Incorrectly Classified Instances       218               28.3854 %
Kappa statistic                          0.3446
Mean absolute error                      0.3254
Root mean squared error                  0.4167
Relative absolute error                 71.6048 %
Root relative squared error             87.4307 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.836     0.507     0.755     0.836     0.793      0.349     0.794     0.866     1
                 0.493     0.164     0.617     0.493     0.548      0.349     0.794     0.650     2
Weighted Avg.    0.716     0.388     0.706     0.716     0.708      0.349     0.794     0.791

=== Confusion Matrix ===

   a   b   <-- classified as
 418  82 |   a = 1
 136 132 |   b = 2
```

- **Manhattan Distance**



a) *k* = 3

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         545               70.9635 %
Incorrectly Classified Instances       223               29.0365 %
Kappa statistic                          0.3351
Mean absolute error                      0.3209
Root mean squared error                  0.4646
Relative absolute error                 70.6083 %
Root relative squared error             97.4825 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.822     0.500     0.754     0.822     0.787      0.338     0.723     0.790     1
                 0.500     0.178     0.601     0.500     0.546      0.338     0.723     0.554     2
Weighted Avg.    0.710     0.388     0.701     0.710     0.703      0.338     0.723     0.707

=== Confusion Matrix ===

   a   b   <-- classified as
 411  89 |   a = 1
 134 134 |   b = 2
```

b) $k = 5$

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         556               72.3958 %
Incorrectly Classified Instances       212               27.6042 %
Kappa statistic                          0.365
Mean absolute error                      0.3194
Root mean squared error                  0.4374
Relative absolute error                 70.2686 %
Root relative squared error             91.7663 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.838    0.489    0.762      0.838    0.798      0.369    0.755     0.821     1
                 0.511    0.162    0.628      0.511    0.564      0.369    0.755     0.604     2
Weighted Avg.    0.724    0.375    0.715      0.724    0.716      0.369    0.755     0.745

=== Confusion Matrix ===

   a   b   <-- classified as
 419  81 |   a = 1
 131 137 |   b = 2
```

c) $k = 7$

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         558               72.6563 %
Incorrectly Classified Instances       210               27.3438 %
Kappa statistic                          0.3675
Mean absolute error                      0.3278
Root mean squared error                  0.4305
Relative absolute error                 72.1275 %
Root relative squared error             90.3125 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.846    0.496    0.761      0.846    0.801      0.373    0.769     0.842     1
                 0.504    0.154    0.637      0.504    0.563      0.373    0.769     0.603     2
Weighted Avg.    0.727    0.377    0.718      0.727    0.718      0.373    0.769     0.758

=== Confusion Matrix ===

   a   b   <-- classified as
 423  77 |   a = 1
 133 135 |   b = 2
```

d) $k = 9$

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         554               72.1354 %
Incorrectly Classified Instances       214               27.8646 %
Kappa statistic                          0.3496
Mean absolute error                      0.3308
Root mean squared error                  0.4275
Relative absolute error                 72.7786 %
Root relative squared error             89.6991 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.852    0.522    0.753      0.852    0.799      0.357    0.776     0.852     1
                 0.478    0.148    0.634      0.478    0.545      0.357    0.776     0.607     2
Weighted Avg.    0.721    0.392    0.711      0.721    0.710      0.357    0.776     0.767

=== Confusion Matrix ===

   a   b   <-- classified as
 426  74 |   a = 1
 140 128 |   b = 2
```

e) *k* = 11

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         557               72.526  %
Incorrectly Classified Instances       211               27.474  %
Kappa statistic                          0.3628
Mean absolute error                      0.3275
Root mean squared error                  0.4199
Relative absolute error                 72.0474 %
Root relative squared error             88.0862 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.848    0.504    0.758      0.848    0.801      0.369    0.787     0.858     1
                 0.496    0.152    0.636      0.496    0.558      0.369    0.787     0.637     2
Weighted Avg.    0.725    0.381    0.716      0.725    0.716      0.369    0.787     0.781

=== Confusion Matrix ===

   a   b   <-- classified as
 424  76 |   a = 1
 135 133 |   b = 2
```

4.2 Boston Housing Data Set

Concerns housing values in suburbs of Boston

Number of Instances: 506

4.2.1 Python

- **Source Code**

1. Import Library

```
### IMPORT LIBRARY ###
import csv
import random
import math
import operator
import numpy
```

2. ***Handle Data:*** Membuka dataset dari CSV dan membaginya menjadi data *training* dan data *testing*. Tambahkan pula fungsi yang meng*handle missing value* dan fungsi normalisasi

```
### HANDLE DATA ###
def loadDataset (filename, split, trainingSet=[], testSet=[]):
        with open(filename, 'rb') as csvfile:
                lines = csv.reader(csvfile, delimiter = ' ', skipinitialspace = True)
                dataset = list(lines)
                flag = 0

        # Handle missing value
        for x in range(len(dataset)):
                for y in range (len (dataset[x]) - 1):
                        if float (dataset[x][y]) == 0.0:
                                kolom = [float (i[y]) for i in dataset]
                                tidakNol = len(kolom) - kolom.count (0)
                                dataset[x][y] = float(sum(kolom)/tidakNol)
                        else:
                                dataset[x][y] = float(dataset[x][y])

        # Fungsi Normalisasi
        for x in range(len(dataset)):
                minx = min([i for i in dataset[x][:-1]])
                maxx = max([i for i in dataset[x][:-1]])
                for y in range (len (dataset[x]) - 1):
                        dataset[x][y] = (dataset[x][y] - minx) / (maxx - minx)

        # Membagi data training dan data testing
        for x in range(len(dataset)-1):
                for y in range(4):
                        dataset[x][y] = float (dataset[x][y])
                if flag < split * len(dataset):
                        trainingSet.append (dataset[x])
                        flag += 1
                else:
                        testSet.append (dataset[x])
```

3. ***Similarity:*** Menghitung jarak antara 2 data. Saya menggunakan 3 variasi fungsi jarak

```
### SIMILARITY ###
#EUCLIDEAN DIST
def euclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += float(pow((float(instance1[x]) - float(instance2
[x])), 2))
        return float(math.sqrt(distance))

#MANHATTAN DIST
def manhattanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                temp = instance1[x] - instance2[x]
        if temp < 0:
                temp*=-1
        distance += temp
        return math.sqrt(distance)

#COSINE SIMILARITY
def cosineSimilarity(instance1, instance2, length):
        distance = 0
        sumxx, sumxy, sumyy = 0, 0, 0
        for i in range(length):
                x = float(instance1[i])
                y = float(instance2[i])
                sumxx += x * x
                sumyy += y * y
                sumxy += x * y
        return 1 - (sumxy / math.sqrt(sumxx * sumyy))
```

4. ***Neighbors:*** Menemukan *k* data yang paling mirip

```
### K NEIGHBORS ###
def getNeighbors(trainingSet, testInstance, k, pilih):
        distances = []
        length = len(testInstance) - 1
        for x in range(len(trainingSet)):
                if pilih==1:
                        dist = euclideanDistance(testInstance,trainingSet[x],length)
                elif pilih==2:
                        dist = manhattanDistance(testInstance,trainingSet[x],length)
                elif pilih==3:
                        dist = cosineSimilarity(testInstance,trainingSet[x],length)
                distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
                neighbors.append(distances[x][0])
        return neighbors
```

5. ***Response:*** Mendapatkan hasil *majority vote* dari *class* dari tetangga

```
### RESPONSE -- MAJORITY VOTE OF CLASS ###
def getResponse(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
        if response in classVotes:
                classVotes[response] += 1
        else:
                classVotes[response] = 1
        sortedVotes = sorted(classVotes.items(),key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]
```

6. ***Accuracy****:* Meringkas keakuratan prediksi

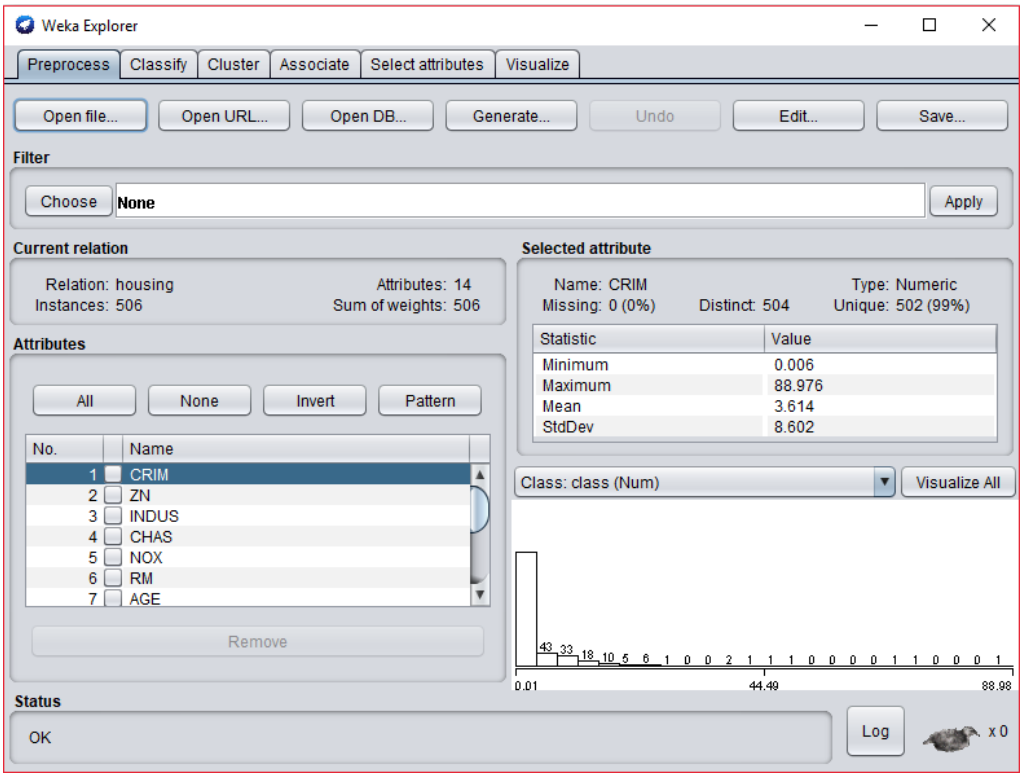
```
### ACCURACY ###
def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][-1] is predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0
```

7. ***Main:*** Menggabungkan semua fungsi

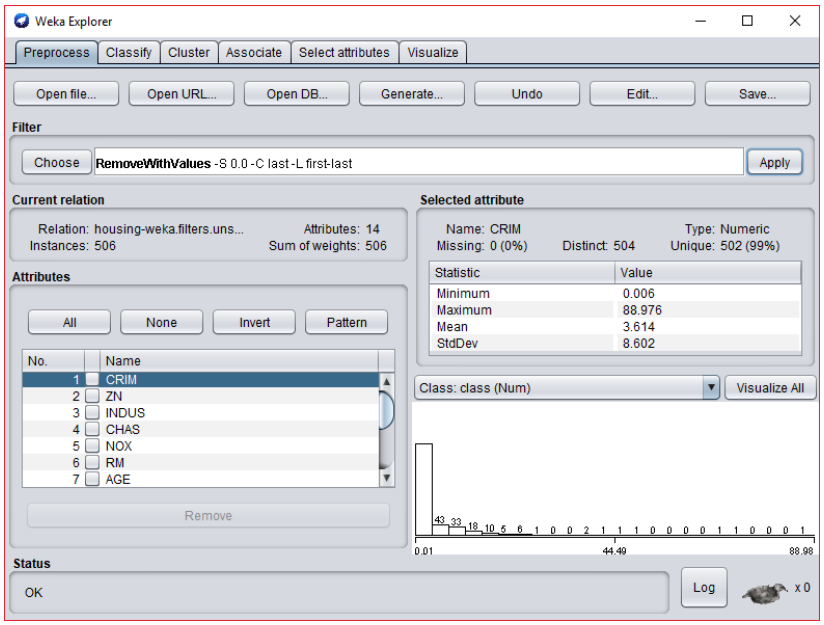
```
### MAIN ###
def main():
        # prepare data
        trainingSet = []
        testSet = []
        split = 0.8
        total_regresi = 0
        loadDataset('housing.data', split, trainingSet, testSet)
        print('\nData Train: ' + repr(len(trainingSet)))
        print('Data Test: ' + repr(len(testSet)))

        # generate predictions
        predictions = []
        print "\nMasukkan nilai k : "
        k = int(raw_input())
        print "\n1. Euclidean Distance\n2. Manhattan Distance\n3. Cosine Similarity\n"
        print "\nMasukkan pilihan algoritma yang ingin digunakan : "
        typeDist = [euclideanDistance, manhattanDistance, cosineSimilarity][int(raw_input())]
        for x in range(len(testSet)):
                neighbors = getNeighbors(trainingSet, testSet[x], k, typeDist)
                regresiBukan = sum ([n[-1] for n in neighbors]) / k
                regresi = sum ([typeDist(testSet[x], n, len(testSet[x])) for n in neighbors]) / k
                total_regresi += regresi
                result = getResponse(neighbors)
                predictions.append(result)
                print('> Predicted=' + repr(result) + ', Actual=' + repr(testSet[x][-1]))
                for n in neighbors:
                        print (n[-1])
                print ('Regresi bukan rumus : ' + str(regresiBukan))
        accuracy = getAccuracy(testSet, predictions)
        #print('Accuracy: ' + repr(accuracy) + '%')
        print('Regresi : ' + str(total_regresi/len(testSet)))

main()
```

4.2.2 Weka

1. Buka Weka Explorer dan masukkan data pima-indians-diabetes.arff (*http://tunedit.org/repo/UCI/numeric/housing.arff*) ke dalam Weka melalui *Open File*

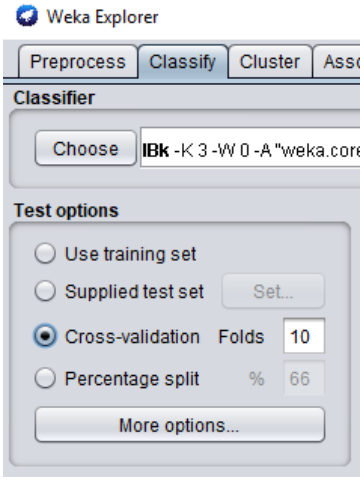


2. *Handle MissingValue*



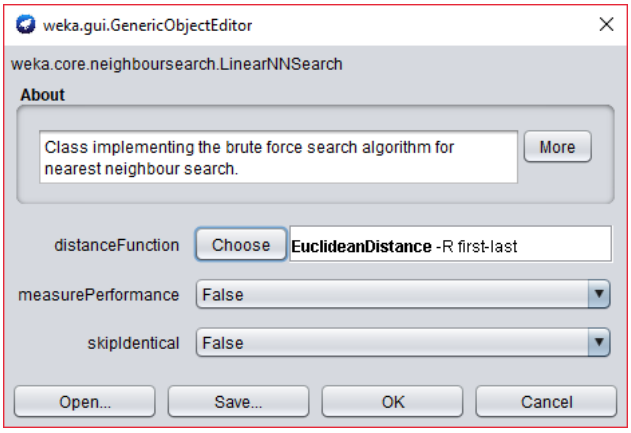
3. Atur sedemikian rupa seperti ini:

   b) 10-Cross Validation



4. Klasifikasi menggunakan *k*-NN/IBK dengan 2 variasi fungsi jarak

   - **Euclidean Distance**



a) *k* = 3

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 3 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.871
Mean absolute error                      2.9544
Root mean squared error                  4.5644
Relative absolute error                 44.3116 %
Root relative squared error             49.5106 %
Total Number of Instances              506
```

b) *k* = 5

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8591
Mean absolute error                      2.9811
Root mean squared error                  4.8173
Relative absolute error                 44.7119 %
Root relative squared error             52.2536 %
Total Number of Instances              506
```

c) *k* = 7

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 7 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8448
Mean absolute error                      3.1655
Root mean squared error                  5.039
Relative absolute error                 47.4771 %
Root relative squared error             54.6591 %
Total Number of Instances              506
```

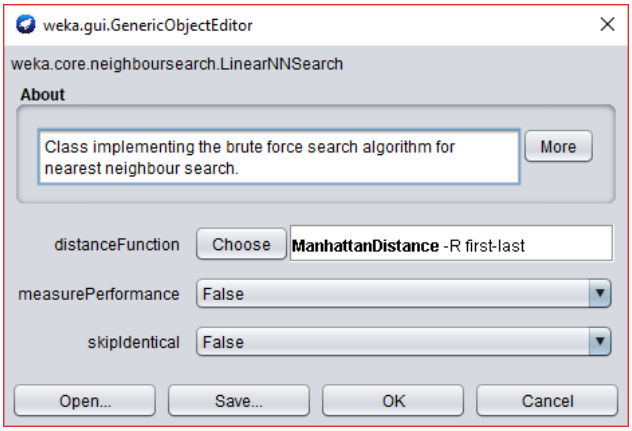
d) *k* = 9

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 9 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8264
Mean absolute error                      3.3603
Root mean squared error                  5.3075
Relative absolute error                 50.3989 %
Root relative squared error             57.5713 %
Total Number of Instances              506
```

e) *k* = 11

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 11 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8181
Mean absolute error                      3.4018
Root mean squared error                  5.4424
Relative absolute error                 51.0221 %
Root relative squared error             59.0338 %
Total Number of Instances              506
```

- **Manhattan Distance**



a) ***k*** **= 3**

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 3 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8931
Mean absolute error                      2.7304
Root mean squared error                  4.1612
Relative absolute error                 40.9522 %
Root relative squared error             45.1373 %
Total Number of Instances              506
```

b) ***k*** **= 5**

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8807
Mean absolute error                      2.8708
Root mean squared error                  4.4186
Relative absolute error                 43.0579 %
Root relative squared error             47.9289 %
Total Number of Instances              506
```

c) ***k*** **= 7**

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 7 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8714
Mean absolute error                      2.9727
Root mean squared error                  4.6084
Relative absolute error                 44.5858 %
Root relative squared error             49.9877 %
Total Number of Instances              506
```

d) *k* = 9

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 9 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8533
Mean absolute error                      3.1184
Root mean squared error                  4.909
Relative absolute error                 46.7708 %
Root relative squared error             53.2487 %
Total Number of Instances              506
```

e) *k* = 11

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 11 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8482
Mean absolute error                      3.1725
Root mean squared error                  5.0096
Relative absolute error                 47.5824 %
Root relative squared error             54.3397 %
Total Number of Instances              506
```

BAB V
ANALISA DATA

5.1 Pima Indians Diabetes Data Set

- **Hasil Python**

  Total Data : 768
  Data *Train* : 615 (80% total data)
  Data *Test* : 152 (20% total data)

	Akurasi				
	k = 3	*k* = 5	*k* = 7	*k* = 9	*K* = 11
Euclidean Dist.	62.5%	68.4210%	61.1842%	63.8157%	65.7895%
Manhattan Dist.	52.6316%	56.5789%	56.5789%	58.5526%	50.6579%
Cosine Similarity	66.4473%	59.2105%	59.2105%	64.4737%	62.5%

Dari hasil pengumpulan data diatas, dapat kita analisa bahwa:

a) Rata-rata tingkat akurasi klasifikasi tertinggi adalah Euclidean Distance
b) Rata-rata tingkat akurasi klasifikasi terendah adalah Manhattan Distance
c) Tingkat akurasi tertinggi pada Euclidean Distance adalah 68.4210% dengan k=5
d) Tingkat akurasi tertinggi pada Manhattan Distance adalah 58.5526% dengan k=9
e) Tingkat akurasi tertinggi pada Cosine Similarity adalah 66.4473% dengan k=3

BAB VI
PENUTUP

5.1 Kesimpulan

Setelah kita melakukan uji coba dan analisa, dapat kita tarik kesimpulan bahwa:

- Nilai *k* tidak boleh terlalu besar maupun terlalu kecil. Semakin besar *k,* klasifikasi data cenderung semakin tidak akurat
- Euclidean Distance adalah metode perhitungan jarak yang lebih akurat (dibuktikan dengan rata-rata tingkat akurasi tertinggi) dibandingkan metode perhitungan jarak yang lain

DAFTAR PUSTAKA

- https://archive.ics.uci.edu/ml/datasets.html
- https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
- http://omahti.web.id/blog/langkah-pertama-belajar-machine-learning-menggunakan-python-part-1/
- https://mragungsetiaji.wordpress.com/2016/06/23/machine-learning-knn-k-nearest-neighbors-menggunakan-python/
- https://tentangdata.wordpress.com/2015/09/16/knn-perhitungan-jarak-serta-keunggulan-dan-batasan/
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm