FACULTY OF INGENIEURSWETENSCHAPPEN
Applied Computer Science

Techniques of Artificial Intelligence

Techniques of Artificial Intelligence Report

Yanfang GUO, Jin LI

Enroll number: 0535945,0535128

Prof: Bart Jansen, Dipankar Sengupta, Roxana Radulescu

May 2017

Abstract

This project mainly focuses on the classification problem. It includes the lazy learning distance model K-Nearest Neighbors, probability model Naive Bayes Classifier, and tree model ID3(Iterative Dichotomiser 3). The project also implements a lazy learning distance model K-means which focused on clustering.

This project uses *Iris, Car Evaluation, Blood Transfusion Donating, Balance Scale* and *Lenses* data sets. The attributes of Iris and Blood Transfusion Donating are numeric while the other three are categorical. The Iris and Balance Scale dataset are balanced, while the others are imbalanced. More information about the dataset include the correlation , visualization and other details can be found in Section 1.

Section 2 gives the details of implementations of algorithms. The implementations are based on python, and follow the OOP concepts. We use the accuracy, TP, FP, TN, FN, MCC score, kappa statistics and confusion matrix to evaluate the algorithms. Section also includes the performance comparisons between the implemented algorithms and algorithms in Weka.

Section 3 gives comparisons between 3 classification algorithms KNN, ID3 and Naive Bayes. The analysis includes the accuracy, kappa score both on balanced, clear boundary datasets and imbalanced and vague boundary datasets. It also analyze the training time and predict time cost of different algorithms.

Section 4 introduces the format of data set and shows the data preprocessing process. In addition, some normalization functions are also implemented to adjust different scales to a notionally common scale.

1 Data Set

1.1 Iris Data set

1.1.1 Iris Dataset Description

This is a well-known data set about iris(plant)[1] classification. The data set contains 3 classes of 150 samples totally, and each class refers to one type.[2] There are 4 attributes: *sepal length, sepal width, petal length* and *petal width*, and 3 class: *Setosa, Versicolour* and *Virginica.* Below the table 1 shows the detailed information.

Table 1: Iris data set table.

Data Set Characteristics	Multivariate	**Number of Sample**	150
Attribute Characteristics	Real	**Number of Attribute**	4
Associated Task	Classification	**Missing Value**	No
Creator	R.A. Fisher	**Class**	3

The iris dataset is a balanced dataset. For each Class, it has the the equal percentage Samples.(50 for each class).

Table 2: Iris class distribution.

Class	**N**	**N(%)**
Setosa	50	33.3
Versicolour	50	33.3
Virginica	50	33.3

1.1.2 Iris Dataset Visualization

The python package **seaborn** is used to draw the pairplot of iris, Figure 1 gives a clear view of the distribution of the iris dataset. The boundaries between different class are clear.

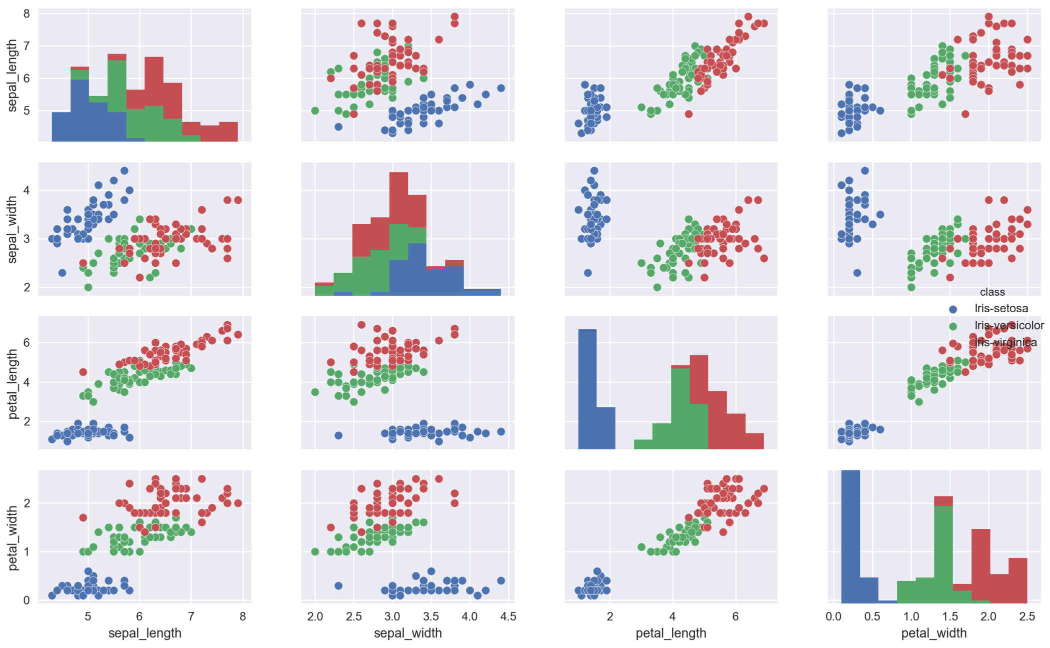


Figure 1: Iris Dataset Visualization

From the correlation map, the *Sepal.length, petal.length* and *petal.width* have strong correlation with the *Variety* and the correlations between *petal.width* and *petal.length*, *sepal.length* are strong. Figure 2 demonstrates the correlation.

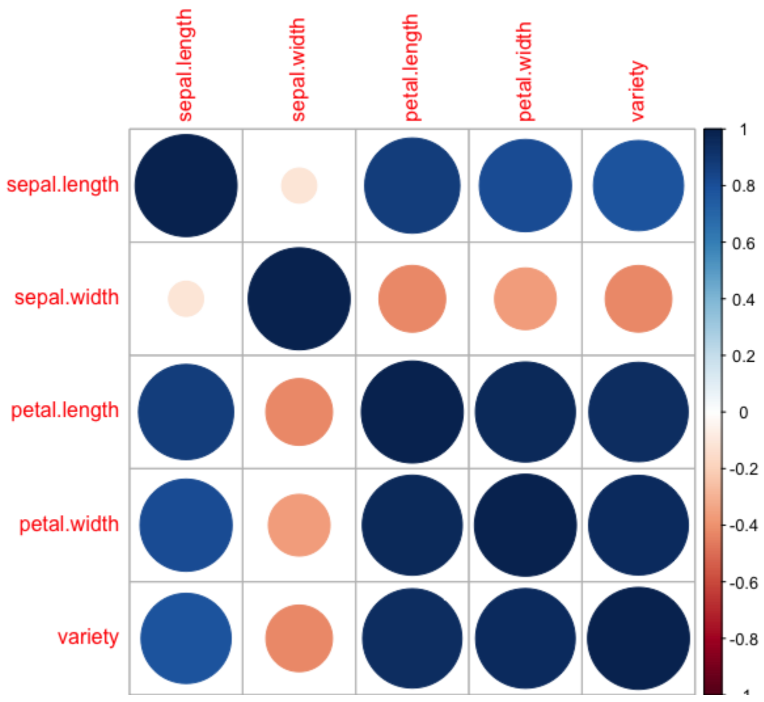


Figure 2: Iris correlation

1.2 Car Evaluation Dataset

1.2.1 Car Evaluation Dataset Description

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX.[1] The car evaluation dataset contains 1728 instances and 6 attributes. Below the table 3 show the detailed information. All the attributes in this dataset are categorical and this dataset is usually used for multi-attribute decision making.

Table 3: Car Evaluation Dataset Description table.

Data Set Characteristics	Multivariate	**Number of Sample**	1728
Attribute Characteristics	Categorical	**Number of Attribute**	6
Associated Task	Classification	**Missing Value**	No
Creator	R.A. Fisher	**Class**	4

From the table class distribution, it is an imbalanced dataset, the *unacc* accounts for 70.023%, while *acc,good,v-good* account for 22.2%,3.933% ,3.762% respectively. The table 4 shows the distribution information.

Table 4: Car Evaluation Classification Distribution.

Class	**N**	**N(%)**
unacc	1210	70.023
acc	384	22.22
good	69	3.933
v-good	65	3.762

All 6 attributes of this dataset are categorical, the meanings and the levels are described in the table 5

Table 5: The Attributes of Car Evaluation Dataset

Attribute	**Meaning**	**Levels**
buying	buying price	v-high, high, med, low
maint	price of the maintenance	v-high, high, med, low
doors	number of doors	2, 3, 4, 5-more
persons	capacity in terms of persons to carry	2, 4, more
lug_boot	the size of luggage boot	small, med, big
safety	estimated safety of the car	low, med, high

1.2.2 Car Evaluation Dataset Visualization

Car evaluation dataset has 6 attributes and 4 classes, here we use RadViz function in python **seaborn** package to visualizing multi-variate data. Figure 3 shows the result.

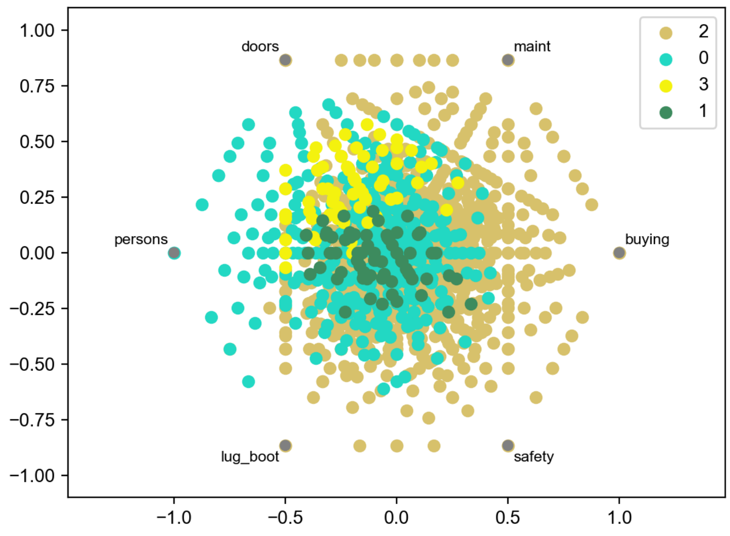


Figure 3: Car dataset visualization

From figure 4 below, the correlations between attributes and ratings are low, which means the connections between attributes and attributes are weak.

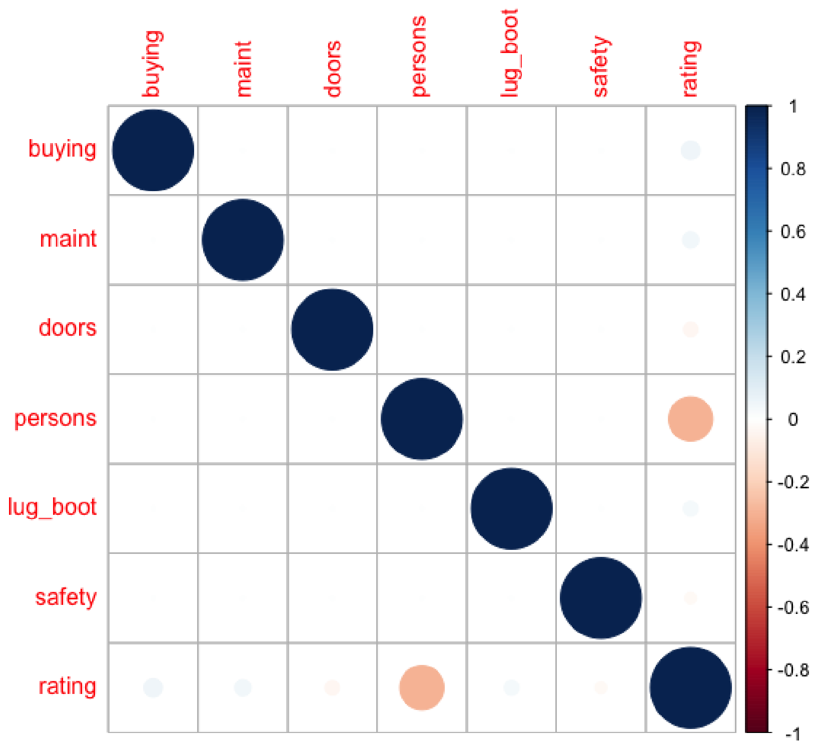


Figure 4: Car dataset correlation

1.3 Blood Transfusion Donating Dataset

1.3.1 Blood Transfusion Donating Dataset Description

The Blood Transfusion Donating Data Set[3] is taken from the Blood Transfusion Service Center in Hsin-Chu City in Taiwan. The center passes their blood transfusion service bus to one university in Hsin-Chu City to gather blood donated about every three months. This dataset contains 748 donors at random from the donor database. This dataset contains 748 Samples and the table 6 below shows the corresponding information

Table 6: Transfusion Service Center Data Set

Data Set Characteristics	Multivariate	**Number of Sample**	748
Attribute Characteristics	Real	**Number of Attribute**	5
Associated Task	Classification	**Missing Value**	No
Creator	Yeh, I-Cheng.	**Class**	2

These 748 donor data, each one includes 4 attributes including *Recency,Frequency,Time,Monetary*. This 4 attributes are all numeric type and the statistical details are described in table 7

Table 7: The Attributes information of Blood Transfusion Dataset

Attribute	**Meaning**	**Min**	**Max**	**Mean**	**Std**
Recency	months since last donation	0.03	74.4	9.74	8.07
Frequency	total number of donation	1	50	5.51	5.84
Time	months since first donation	250	12500	1378.68	1459.83
Monetary	total blood donated in c.c.	2.27	98.3	34.42	24.32

This dataset is divided into two categories by whether he/she donated blood in March 2007, where 0 stands for not donating blood, 1 stands for donating blood. This is an unbalanced dataset, details are given in the table 8 below.

Table 8: Blood Transfusion Dataset Classification Distribution

Class	**N**	**N(%)**
1(donated)	568	76
0(not donated)	180	24

1.3.2 Blood Transfusion Visualization

We use the python **seaborn** package to draw the pairplot for Blood Transfusion donating. We can notice that the dataset have vague boundaries between classes and the attributes do not satisfy normal distribution.

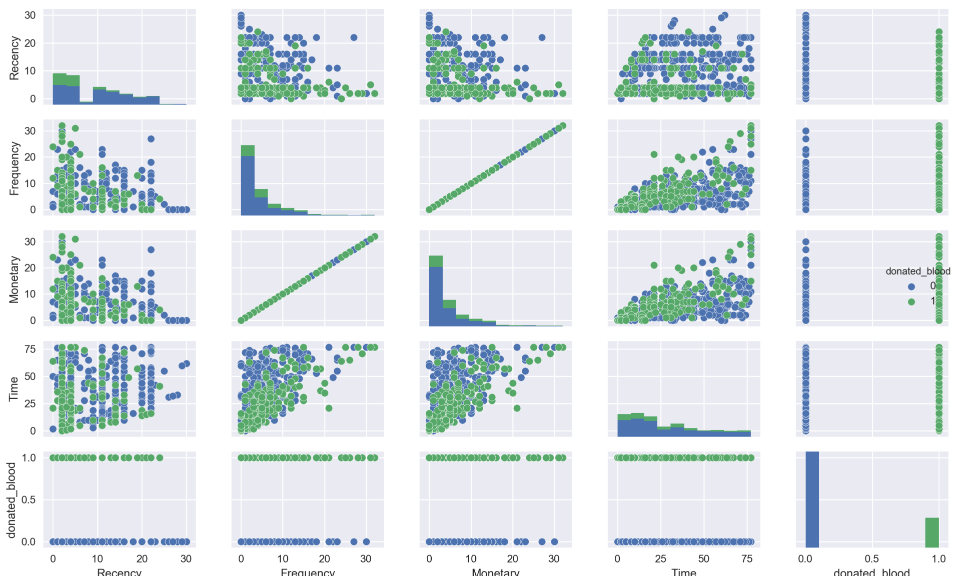


Figure 5: Blood Transfusion Pairplot

Blood transfusion dataset has 4 attributes and 2 classes, here we use RadViz function in python **seaborn** package to visualizing multi-variate data. Figure 6 shows the result. The blood transfusion dataset do not have clear boundaries between classes.

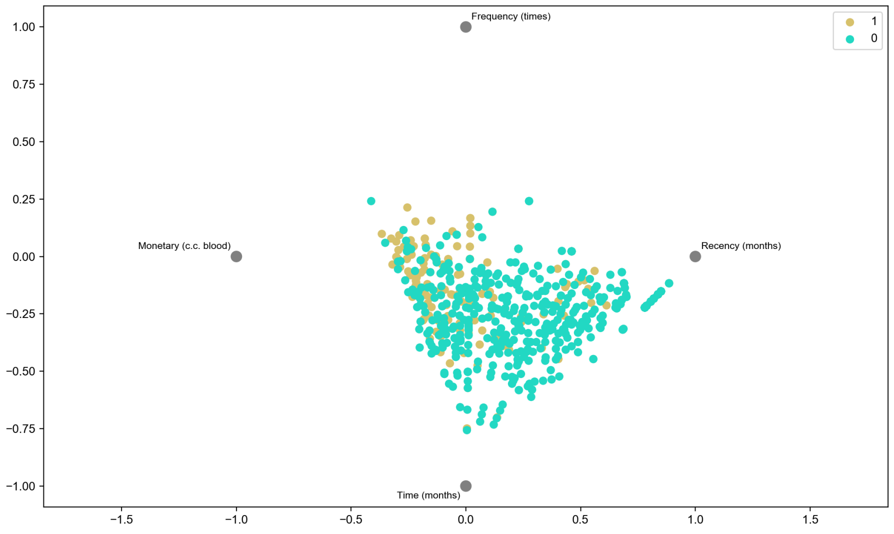


Figure 6: Blood transfusion visualization

Figure 7 displays the dataset correlation, the *Frequency* and *Monetary* have strong correlations.

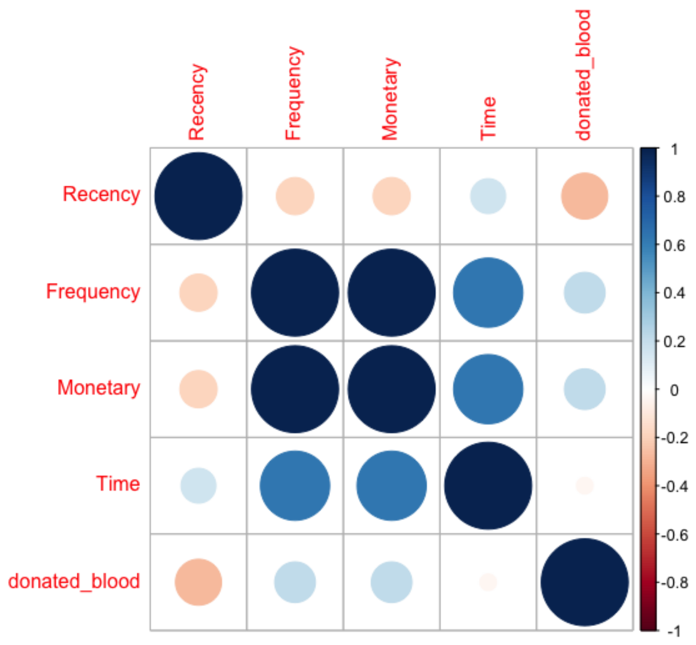


Figure 7: Blood transfusion Correlation

1.4 Balance Scale Data Set

1.4.1 Data Description

This data set was used to model psychological experimental results. The attributes are the left weight, the left distance, the right weight, and the right distance. There are three balanced classifications. If the result is greater than $(left\_distance * left\_weight)$, the scale tips to the right. If the result is greater than $(right\_distance * right\_weight)$, its tips to the left, otherwise, it's balanced. [4]

Table 9: Balance Scale data set.

Data Set Characteristics	Multivariate	**Number of Sample**	625
Attribute Characteristics	Categorical	**Number of Attribute**	4
Associated Task	Classification	**Missing Value**	No
Creator	Siegler, R. S.	**Class**	3

1.4.2 Data Visualization

Figure 8 shows the 4 attributes and their relationship, the dataset is well balanced.

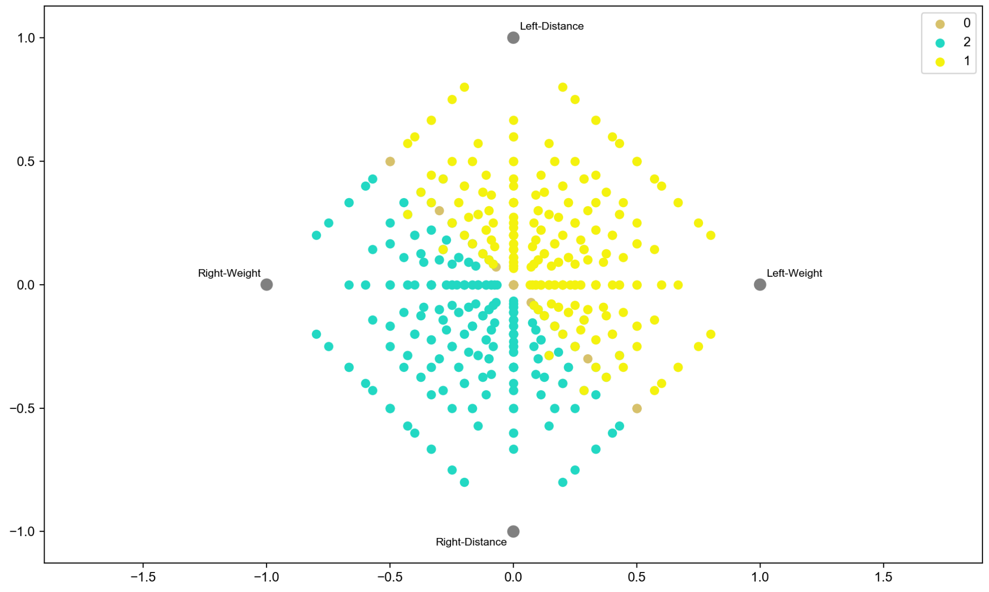


Figure 8: Balanced Scale Dataset visualization

From figure 9 below, those four attributes have the same strong correlation.

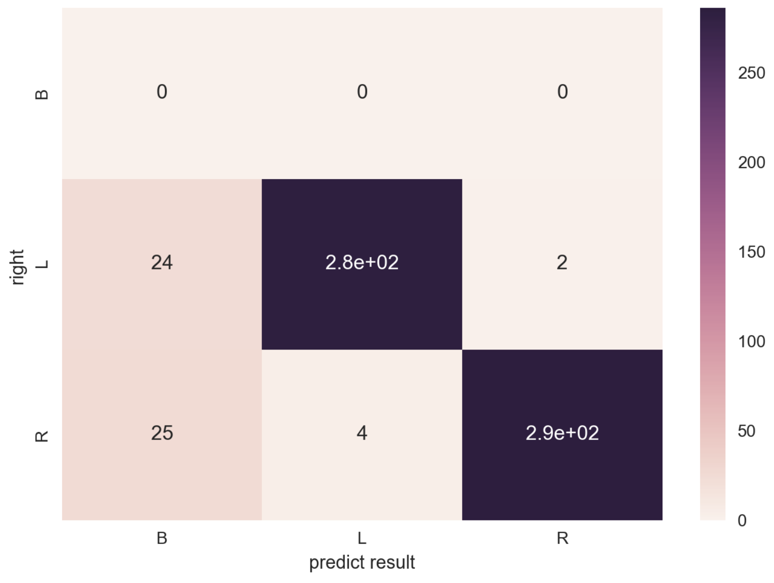


Figure 9: Balanced Scale Dataset Correlation

1.5 Lenses Data Set

1.5.1 Lenses Data Set Description

This dataset describes whether the patients fit with contact lenses and what kind of contact lenses are they fitted. This is a small dataset, and we only use in ID3 algorithms for visualizing purpose. Table 10 shows the corresponding information.[5]

Table 10: Lenses data set

Data Set Characteristics	Multivariate	**Number of Sample**	24
Attribute Characteristics	Categorical	**Number of Attribute**	4
Associated Task	Classification	**Missing Value**	No
Creator	Cendrowska, J.	**Class**	3

The data set simplified the practical problems. It converted abstract medical data into integer. Four main attributes are listed below:

- age of the patient: 1) young, 2) pre-presbyopic, 3) presbyopic
- spectacle prescription: 1) myope, 2) hypermetrope
- astigmatic: 1) no, 2) yes
- tear production rate: 1) reduced, 2) normal

And there are 3 classes:

1. the patient should be fitted with hard contact lenses
2. the patient should be fitted with soft contact lenses
3. the patient should not be fitted with contact lenses

1.5.2 Lenses Data Set Visualization

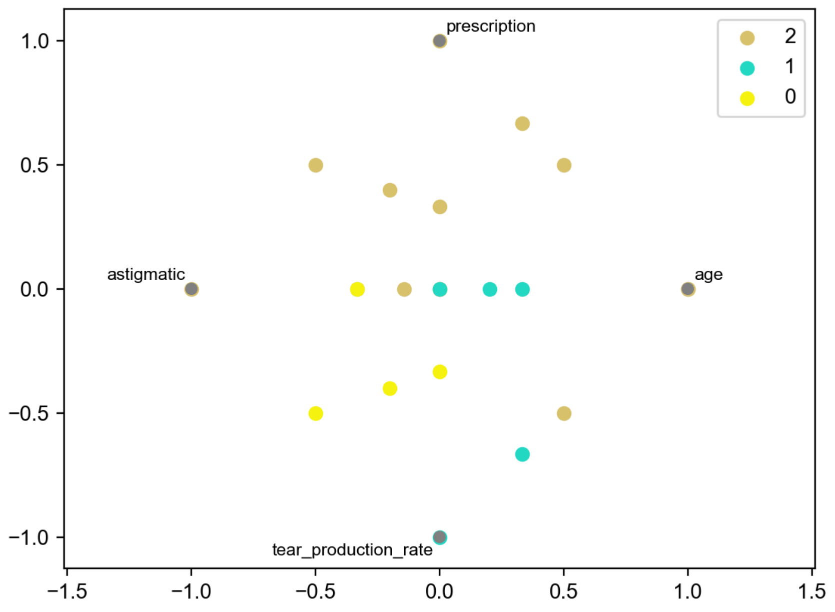


Figure 10: Dataset visualization

2 Algorithm Implement

In this section, we implement ID3, Naive Bayes, KNN and K-means algorithm.The whole process of coding will be displayed in this section. This part only include the core part of the algorithm.

We use accuracy, TP, TN, FP, FN , MCC , Kappa and confusion matrix to evaluate the algorithm.We also compare the result in **Weka**. The performances of our implementations are almost the same as the models in **Weka**

2.1 ID3 (Iterative Dichotomiser 3)

ID3 is used to generate a decision tree. It's a classification algorithm. Figure 11 shows a simple example. The dataset is from the text book[6]. It shows how to judge the weather is suitable for playing tennis by provided parameters.

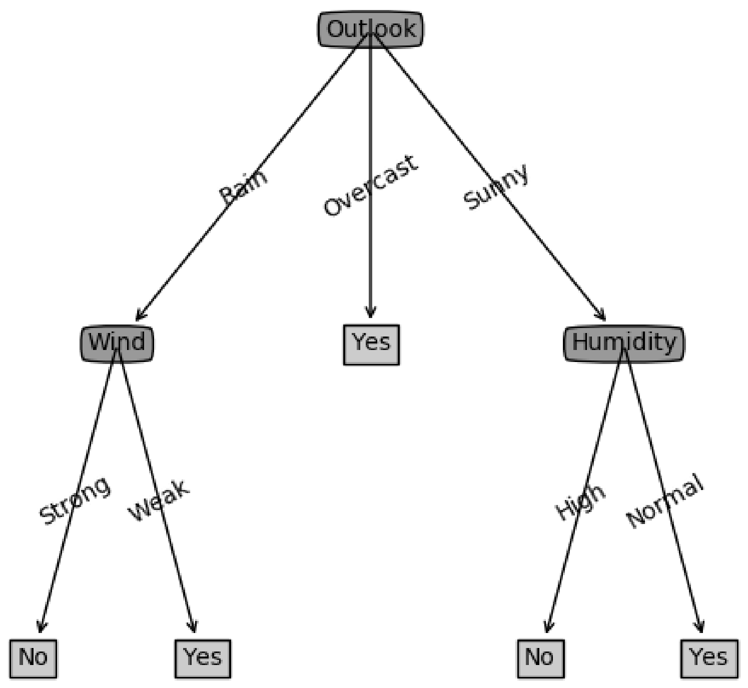


Figure 11: An example of ID3

2.1.1 Algorithm Details

In decision tree, each internal node represents an attribute, each edge or branch is corresponding to an attribute value, and each leaf node assigns a classification.

ID3 is suitable for datasets:

1. instances are described by attribute-value pairs
2. target function is discrete valued
3. maybe require disjunctive hypothesis
4. training data may be noisy

ID3 algorithm firstly calculates entropy of all attributes in the set,

$$Entropy(S) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

and chooses the root node which has the smallest entropy.

```
def __Entropy(self,datas):
  data_entropy = 0.0
  freq_list = {}
  for data in datas:
    freq_list[data[-1]] = 1.0 + freq_list.get(data[-1],0.0)

  for freq in freq_list.values():
    data_entropy += -freq/len(datas) * np.log2(freq/len(datas))
  return data_entropy
```

Then, the data set is split into different subset by the selected attribute value, for each subset, the algorithm continues to calculate Entropy and information gain.

$$Gain(S, A) = Entropy(S) - \sum_{v}^{Values(A)} \frac{| S_v |}{| S |} Entropy(S_v)$$

and selects the decision tree node which has the smallest information gain.

Continue iterating until all attributes are exhausted.

In this program, the decision tree is stored in nested dictionary. The key is the node, the value is edge or leaf.

What's more, we can use the inverse process to predict results and use **matplotlib.pyplot** package to draw the tree.

2.1.2 Algorithm Validation and Performance

This program uses *lenses* dataset. Figure 12 demonstrates the id3 tree created by this program. And Below using 10-fold cross validation to show the performance of ID3.

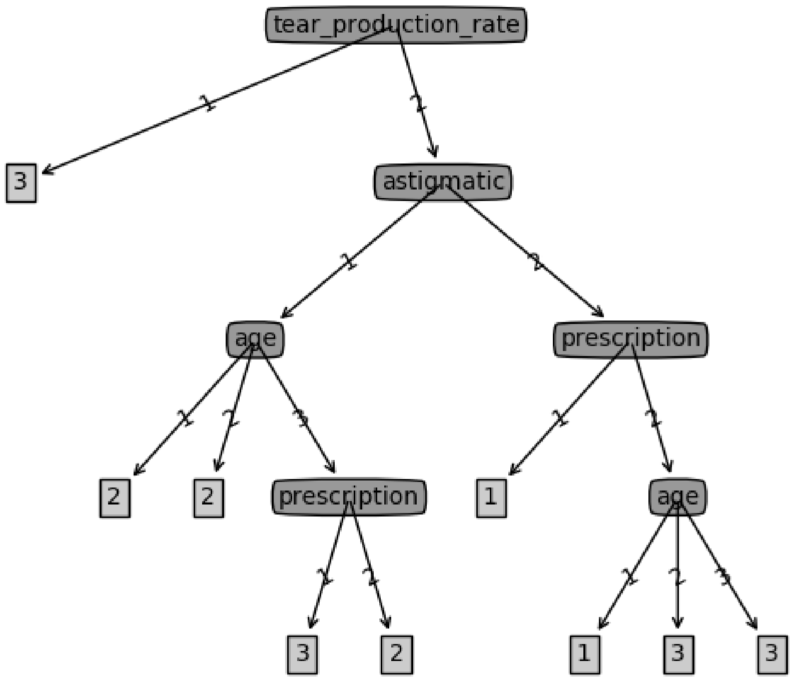
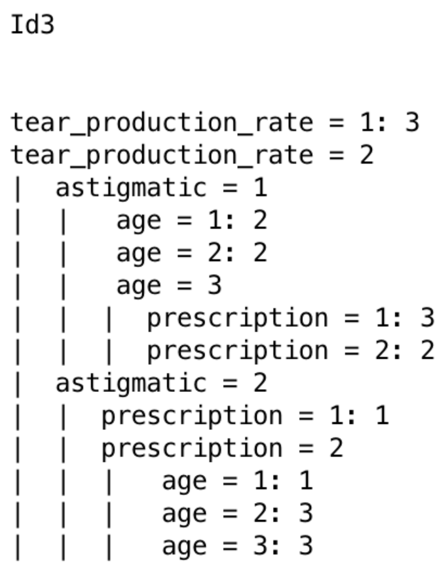


Figure 12: Lenses ID3 tree

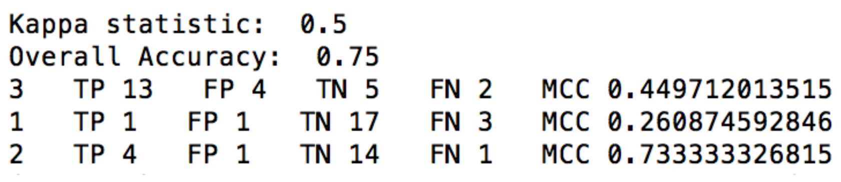
And figure 13 shows the tree built by weka.



```
Id3


tear_production_rate = 1: 3
tear_production_rate = 2
|  astigmatic = 1
|  |   age = 1: 2
|  |   age = 2: 2
|  |   age = 3
|  |  |  prescription = 1: 3
|  |  |  prescription = 2: 2
|  astigmatic = 2
|  |  prescription = 1: 1
|  |  prescription = 2
|  |  |   age = 1: 1
|  |  |   age = 2: 3
|  |  |   age = 3: 3
```

Figure 13: Lenses ID3 tree in Weka

Figure 14 shows the output running by this program, TP, FP, FN, TN and *kappa* value are displayed below. The accuracy is 0.75.



```
Kappa statistic:  0.5
Overall Accuracy:  0.75
3   TP 13   FP 4   TN 5   FN 2   MCC 0.449712013515
1   TP 1    FP 1   TN 17  FN 3   MCC 0.260874592846
2   TP 4    FP 1   TN 14  FN 1   MCC 0.733333326815
```

Figure 14: ID3 result

Figure 15 shows the confusion matrix created by Python **seaborn** package.

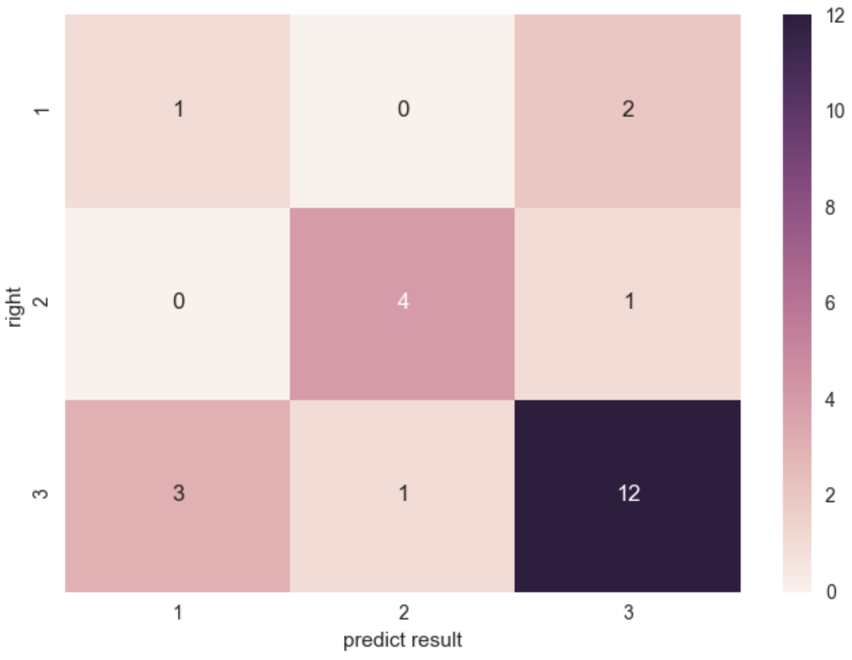


Figure 15: ID3 confusion matrix

Figure 16 lists the output. From the result, we can see the program works well.

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

Correctly Classified Instances          17               70.8333 %
Incorrectly Classified Instances         7               29.1667 %
Kappa statistic                          0.4381
Mean absolute error                      0.1944
Root mean squared error                  0.441
Relative absolute error                 51.4706 %
Root relative squared error            100.965  %
Total Number of Instances               24

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.250    0.100    0.333      0.250    0.286      0.169    0.575     0.208     1
                 0.800    0.053    0.800      0.800    0.800      0.747    0.874     0.682     2
                 0.800    0.444    0.750      0.800    0.774      0.365    0.678     0.725     3
Weighted Avg.    0.708    0.305    0.691      0.708    0.698      0.412    0.701     0.630

=== Confusion Matrix ===

  a  b  c   <-- classified as
  1  0  3 |  a = 1
  0  4  1 |  b = 2
  2  1 12 |  c = 3
```

Figure 16: Weka output

2.2 Naive Bayes Classifier

Naive Bayes Classifier is a classification algorithm[7] based on the Bayes Theorem with an assumption of independence among predictors.Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of features in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

2.2.1 Bayes theory and Maximum likelihood

Bayes theorem describes the relation between $P(c|x)$ and $P(c)$, $P(x)$ and $P(x|c)$. $P(c|x)$ is the posterior probability, $P(x|c)$ is the likelihood function. It expresses how probable the observed data set is for different settings of the parameter vector x. The $P(c)$ and $P(x)$ is the observed probability for classification and attributes.

$$P(c|x) = \frac{P(X|c)P(c)}{P(x)}$$

where $P(X|c) = P(x_1|c)P(x_2|c) \cdots P(x_n|c)$

2.2.2 Algorithm Description

The process of Naive Bayes Classifier algorithm is described in the figure 17.We assume that all the attributes are independent and numeric attributes are all satisfy the normal distribution. Our train process finish after calculating all the probability. On predict procedure, we apply the maximum likelihood theorem, the result is the class with the max posterior probability.

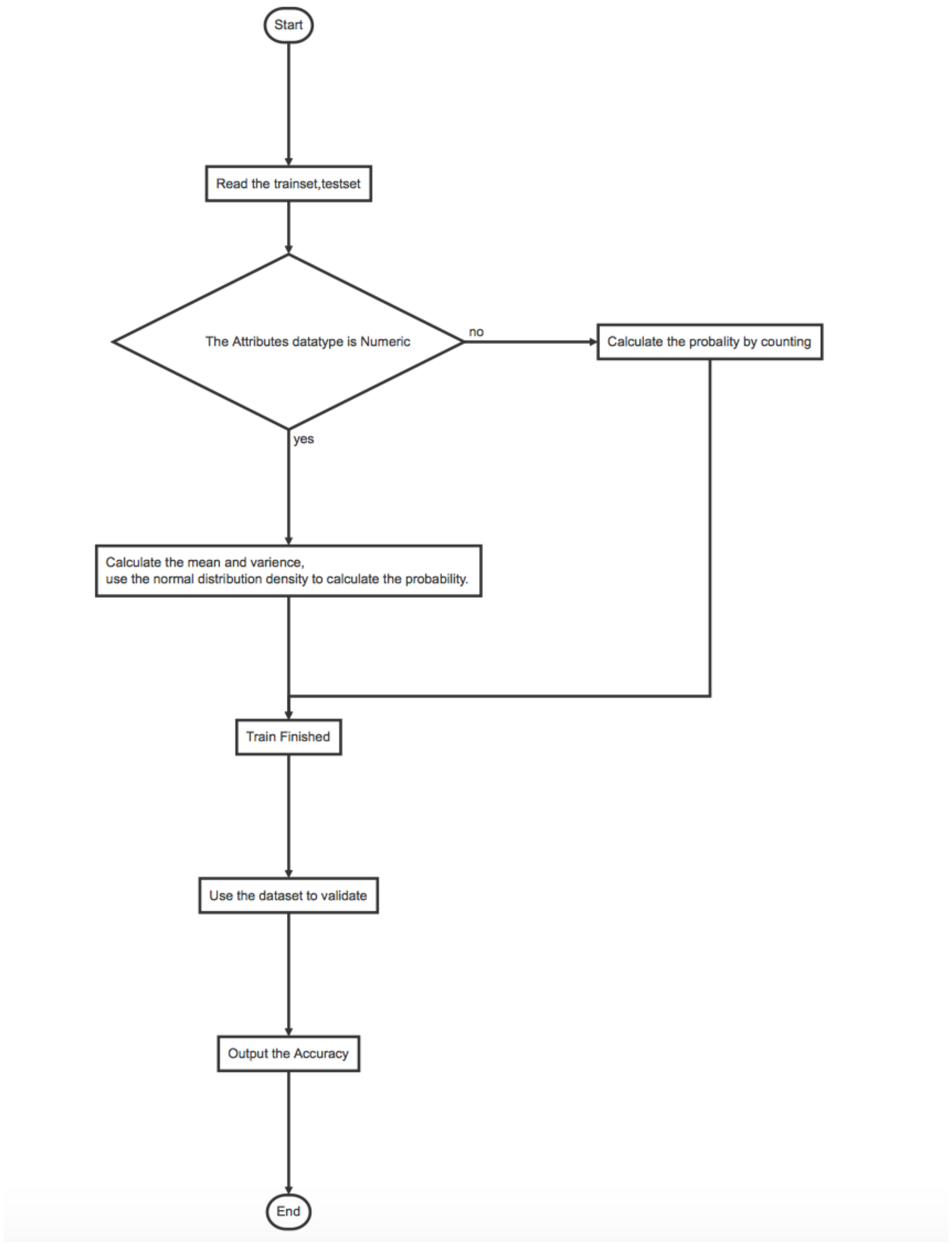

Start
Read the trainset,testset
The Attributes datatype is Numeric
no
Calculate the probality by counting
yes
Calculate the mean and varience,
use the normal distribution density to calculate the probability.
Train Finished
Use the dataset to validate
Output the Accuracy
End

Figure 17: Naive Bayes Classifier Work flow

The input datatype is *pd.data.frame* In the calculating probability procedure for categorical attributes, When a feature value does not occur with every class value, a value 0.001 is added to the count of every feature value-class combination.

```
    def count_px(self):
        for i in range(len(self.__attributes)):
            temp = self.__trainSet[:,i]
            tdict = self.count(temp)
            self.__px.append(tdict)

    def count_pc(self):
        tag = self.__trainSet[:,-1]
        self.__pc = self.count(tag)
    def count_pxc(self):
        for i in range(len(self.__trainSet)):
            for j in range(len(self.__attributes)):
                self.__pxc[self.__trainSet[i][-1]][j][self.__trainSet[i][j]] +=
                                                        1

    def train(self):
        self.count_px()
        self.count_pc()
        for i in self.__pc.keys():
            t = []
            for j in range(len(self.__attributes)):
                tdict = {}
                for k in self.__px[j].keys():
                    tdict[k] = 0.001
                t.append(tdict)
            self.__pxc[i] = t

        self.count_pxc()
```

The predict procedure for Categorical attributes.

```
    def predict(self, blist):
        result = {}
        for i in self.__pc.keys():
            pxc = 1
            px = 1
            for j in range(len(self.__attributes)):
                pxc *= self.__pxc[i][j][blist[j]] / self.__pc[i]
                px *= self.__px[j][blist[j]] / len(self.__trainSet)

            result[i] = pxc * self.__pc[i] / len(self.__trainSet) / px
        largest = max(result.values())
        for i in self.__pc.keys():
            if result[i] == largest:
                t = i
        return t
```

In the calculating probability procedure for numeric attributes

$$P(X = x|C = c) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean value and the σ is the variance.

```
   def count(self, list):
        dict = {}

        dict['mu'] = mean(list)
        dict['sigma'] = var(list)

        return dict
   def count_px(self):
        for i in self.__attributes[:-1]:
            temp = self.__trainSet[i]
            tdict = self.count(temp)
            self.__px.append(tdict)
```

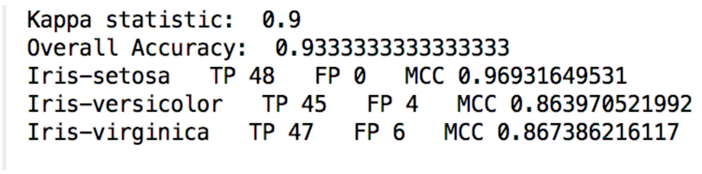
The Predict Procedure for numeric attributes

```
    def count(self, list):
        dict = {}
        dict['mu'] = mean(list)
        dict['sigma'] = var(list)

        return dict

    def count_px(self):
        for i in range(len(self.__attributes)):
            temp = self.__trainSet[:, i]
            tdict = self.count(temp)
            self.__px.append(tdict)

    def count2(self, list):
        dict = {}
        for i in list:
            if i not in dict.keys():
                dict[i] = 1
            else:
                dict[i] += 1

        return dict

    def count_pc(self):
        tag = self.__trainSet[:, -1]
        self.__pc = self.count2(tag)


    def count_pxc(self):
        for i in self.__pc.keys():
            t = []
            for j in range(len(self.__trainSet)):
                if self.__trainSet[j][-1] == i:
                    t.append(self.__trainSet[j])
            t2=[]
            for k in range(len(t[0])-1):
                t2.append(self.count([t[j][k] for j in range(len(t))]))

            self.__pxc[i] = t2
```

2.2.3 Algorithm Validation and Performance

Here I use the famous dataset Iris whose attributes are numeric and the dataset balance scale whose attributes are categorical.

Below show the performance of Naive Bayes on Iris Dataset, using the 10-fold cross validation.



```
Kappa statistic:  0.9
Overall Accuracy:  0.9333333333333333
Iris-setosa    TP 48    FP 0    MCC 0.96931649531
Iris-versicolor    TP 45    FP 4    MCC 0.863970521992
Iris-virginica    TP 47    FP 6    MCC 0.867386216117
```

Figure 18:   Naive Bayes Score on Iris Dataset

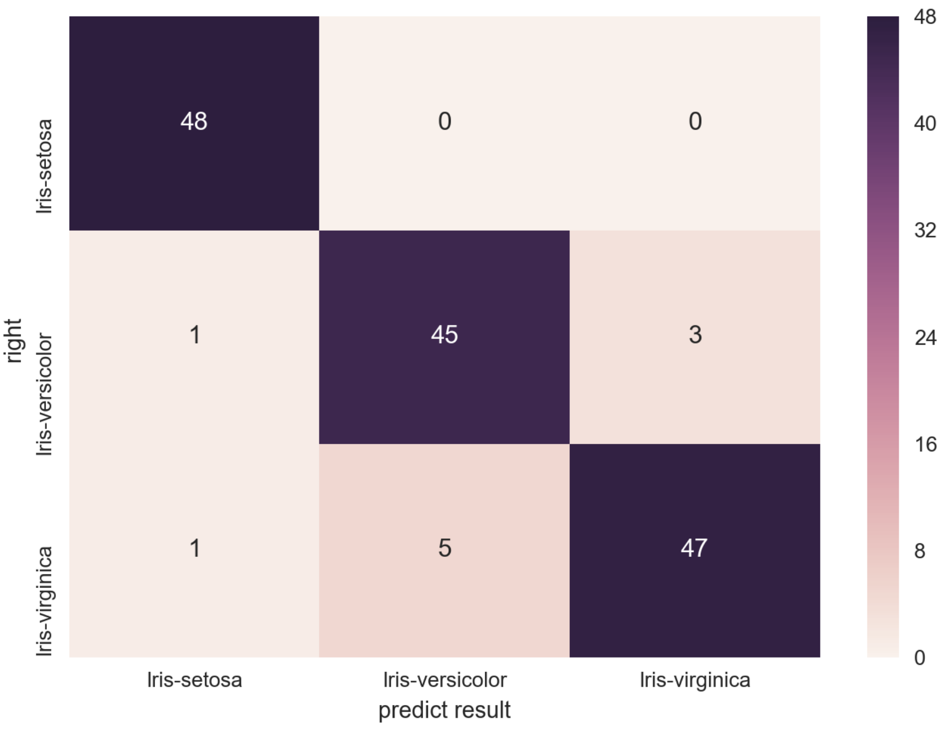
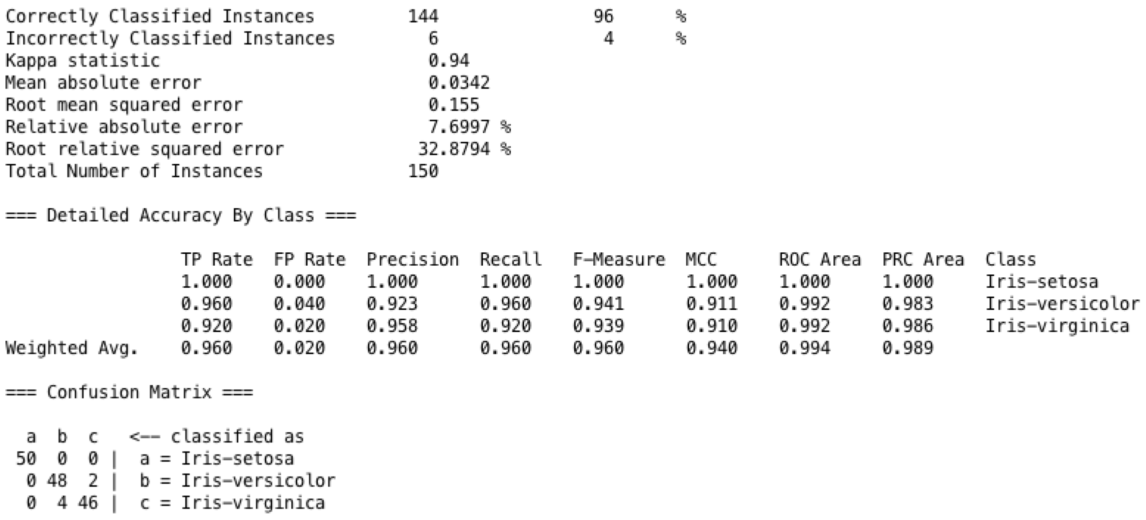


Figure 19: Naive Bayes Confusion Matrix on Iris Dataset

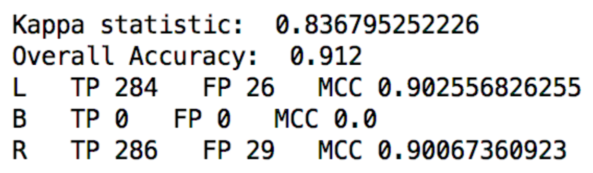
Below show the Weka result.



```
Correctly Classified Instances         144               96      %
Incorrectly Classified Instances         6                4      %
Kappa statistic                          0.94
Mean absolute error                      0.0342
Root mean squared error                  0.155
Relative absolute error                  7.6997 %
Root relative squared error             32.8794 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 0.960    0.040    0.923      0.960    0.941      0.911    0.992     0.983     Iris-versicolor
                 0.920    0.020    0.958      0.920    0.939      0.910    0.992     0.986     Iris-virginica
Weighted Avg.    0.960    0.020    0.960      0.960    0.960      0.940    0.994     0.989

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 48  2 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

Figure 20: weka result using Naive Bayes Classifier on Iris Dataset

Below show the performance of Naive Bayes on balance scale Dataset, using the 10-fold cross validation.



```
Kappa statistic:  0.836795252226
Overall Accuracy:  0.912
L   TP 284   FP 26   MCC 0.902556826255
B   TP 0   FP 0   MCC 0.0
R   TP 286   FP 29   MCC 0.90067360923
```

Figure 21: Naive Bayes Score on balance scale Dataset

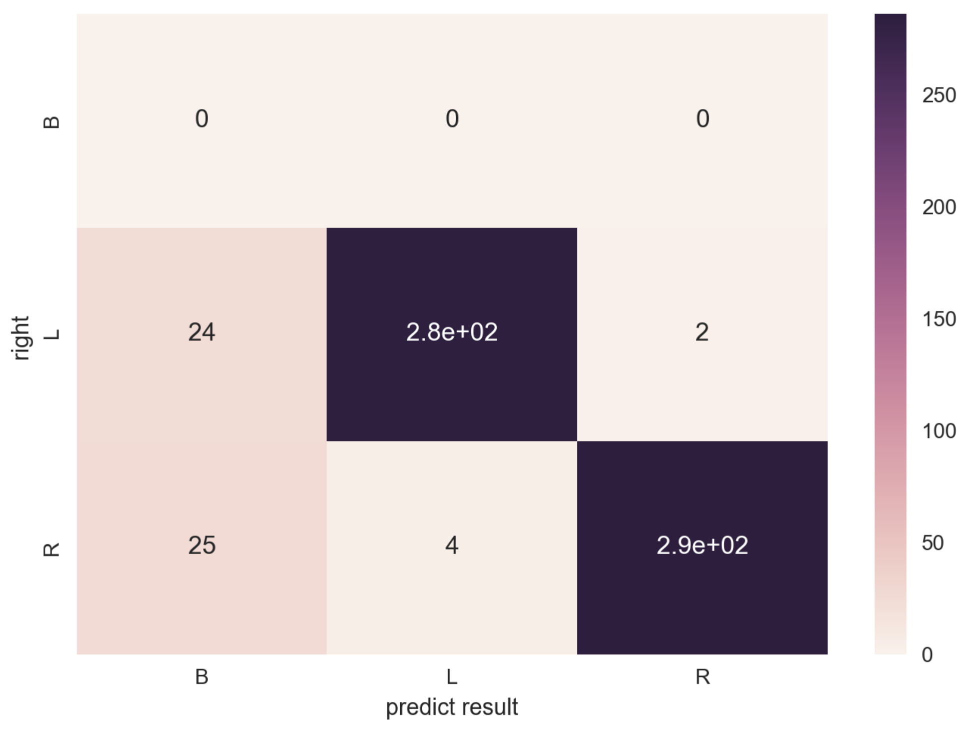
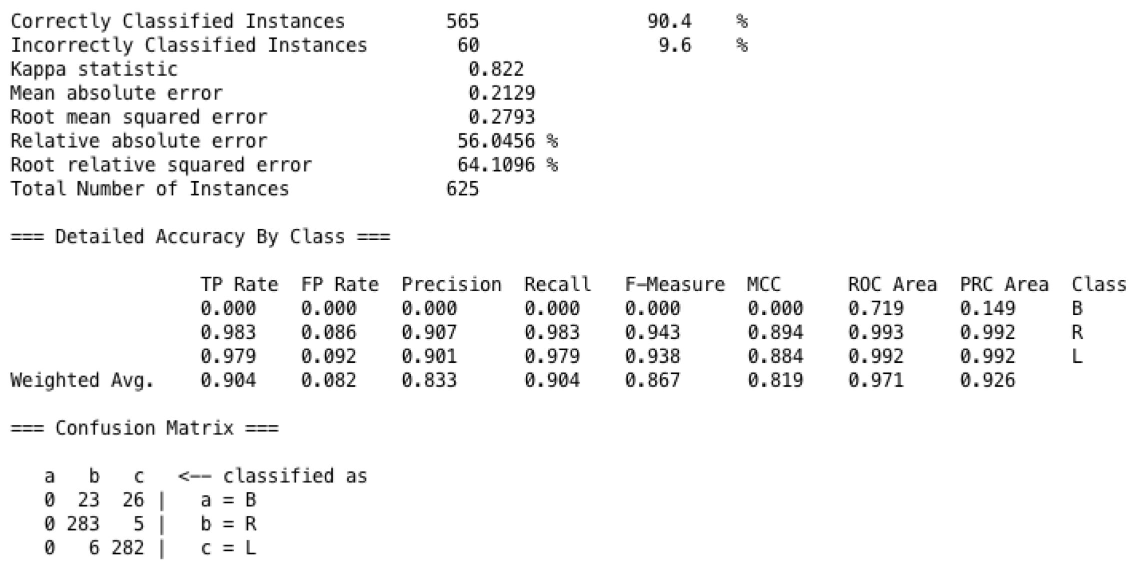


Figure 22: Naive Bayes Confusion Matrix on balance scale dataset

Below show the Weka result.



```
Correctly Classified Instances         565               90.4    %
Incorrectly Classified Instances        60                9.6    %
Kappa statistic                          0.822
Mean absolute error                      0.2129
Root mean squared error                  0.2793
Relative absolute error                 56.0456 %
Root relative squared error             64.1096 %
Total Number of Instances              625

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    0.000      0.000    0.000      0.000    0.719     0.149     B
                 0.983    0.086    0.907      0.983    0.943      0.894    0.993     0.992     R
                 0.979    0.092    0.901      0.979    0.938      0.884    0.992     0.992     L
Weighted Avg.    0.904    0.082    0.833      0.904    0.867      0.819    0.971     0.926

=== Confusion Matrix ===

   a   b   c   <-- classified as
   0  23  26 |   a = B
   0 283   5 |   b = R
   0   6 282 |   c = L
```

Figure 23: weka result using Naive Bayes Classifier on balance scale dataset

2.3 K Nearest Neighborhood

KNN algorithm is a lazy algorithm, figure 24 shows a simple way to understand the algorithm. There are 2 classes *green star* and *red dot*, and the *blue square (?)* is unknown one. If we want to predict a unknown object, we could count its nearest object, and vote the most one, then we can predict the result. In figure 24, if we count the object in the inner circle, the *?* is *red dot*; if we count all object including the outer circle, the *?* is *green star*.

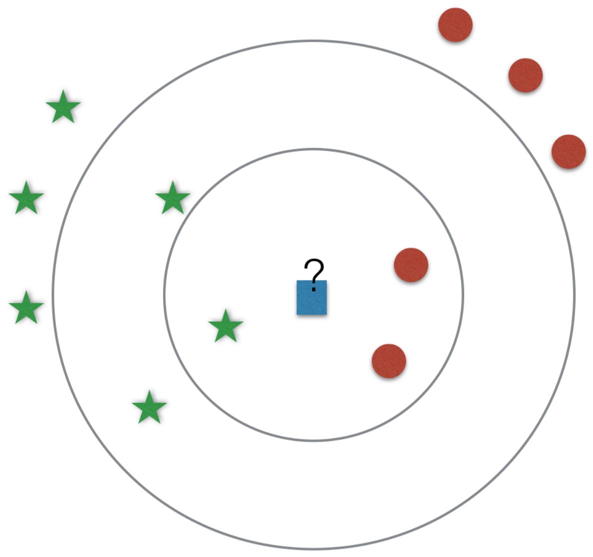


Figure 24: KNN example

2.3.1 Algorithm Description

1. Calculate the Euclidean distance between the test set and every train set instance, and sort the results.
2. Choose the first K smallest results, and vote the results, usually K is an odd number, therefore, choose the majority one.

2.3.2 Algorithm Implement

The following code get the first n nearest neighborhood. It calculates the Euclidean distance, and add the distance and corresponding neighbor into a list together, finally sort the list and choose the first n smallest tuple.

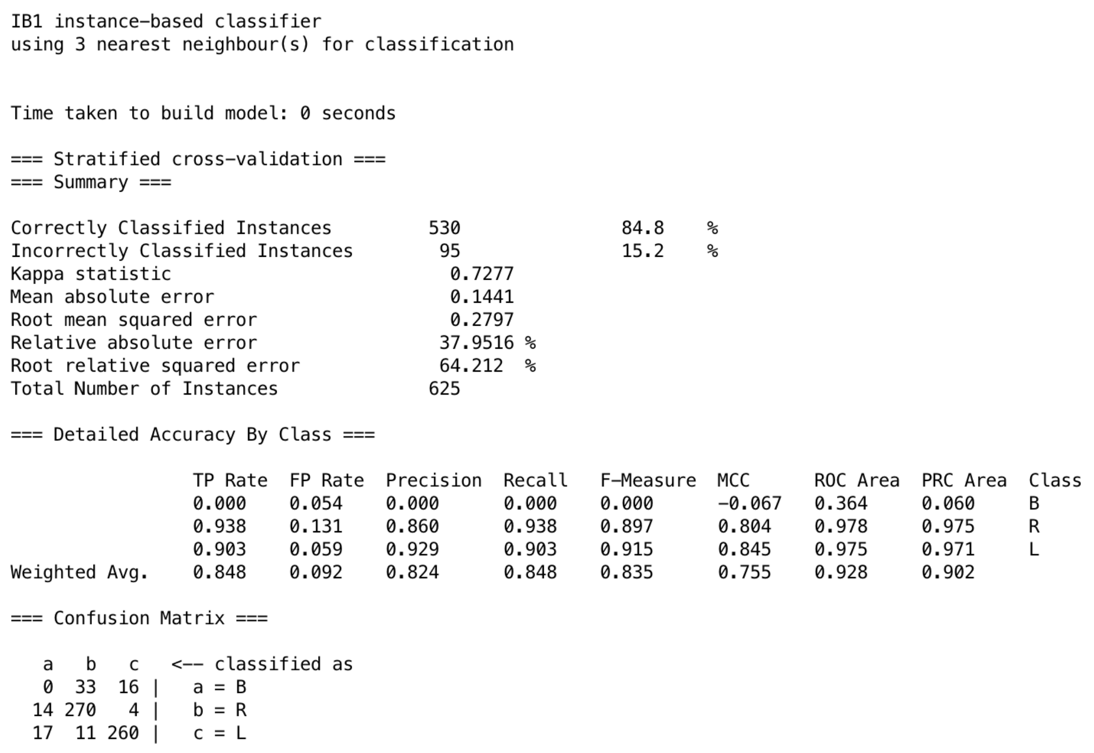
```
def getKNearNeighbors(trainSet, testInstance, n=3):
  distances = []
  neighbors = []
  for i in range(len(trainSet)):
    dist = EuclideanDistance(testInstance,trainSet[i])
  distances.append((trainSet[i],dist))
  distances.sort(key=itemgetter(1))
  for i in range(n):
    neighbors.append(distances[i][0])
  return neighbors
```

The following code shows how to predict the unknown object.

```
def getClassification(neighbors):
  dic = {} # use dictionary to count
  for i in range(len(neighbors)):
    response = neighbors[i][-1]   # result
    dic.setdefault(response,dic.get(response,0)+1)
  temp = sorted(dic.items(),key=itemgetter(1))
  temp.reverse()
  #choose the best one
  return temp[0][0]
```

2.3.3 Algorithm Validation and Performance

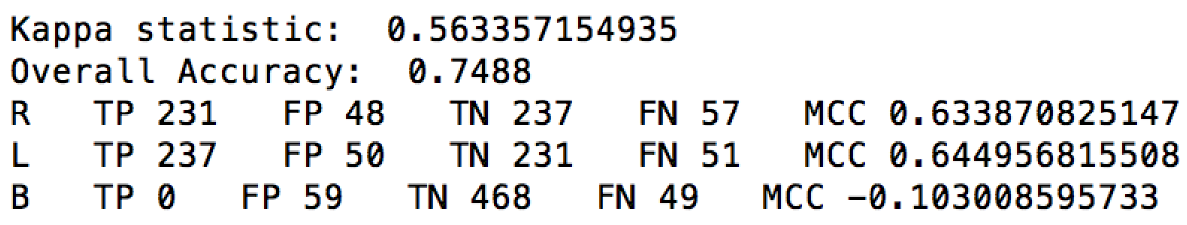
Figure 25 shows the dataset *balance-scale* classifier output using Weka.



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


Time taken to build model: 0 seconds

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

Correctly Classified Instances         530               84.8    %
Incorrectly Classified Instances        95               15.2    %
Kappa statistic                          0.7277
Mean absolute error                      0.1441
Root mean squared error                  0.2797
Relative absolute error                 37.9516 %
Root relative squared error             64.212  %
Total Number of Instances              625

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.054    0.000      0.000    0.000      -0.067   0.364     0.060     B
                 0.938    0.131    0.860      0.938    0.897      0.804    0.978     0.975     R
                 0.903    0.059    0.929      0.903    0.915      0.845    0.975     0.971     L
Weighted Avg.    0.848    0.092    0.824      0.848    0.835      0.755    0.928     0.902

=== Confusion Matrix ===

   a   b   c   <-- classified as
   0  33  16 |   a = B
  14 270   4 |   b = R
  17  11 260 |   c = L
```

Figure 25: KNN in Weka

Figure 26 shows the accuracy, TP, FP, FN and TN value and Kappa statistic. And the accuracy is 74.88%.



```
Kappa statistic:  0.563357154935
Overall Accuracy:  0.7488
R   TP 231   FP 48   TN 237   FN 57   MCC 0.633870825147
L   TP 237   FP 50   TN 231   FN 51   MCC 0.644956815508
B   TP 0   FP 59   TN 468   FN 49   MCC -0.103008595733
```

Figure 26: KNN result

Figure 27 shows the Confusion Matrix of balance-scale dataset. And the Accuracy is 74.88%

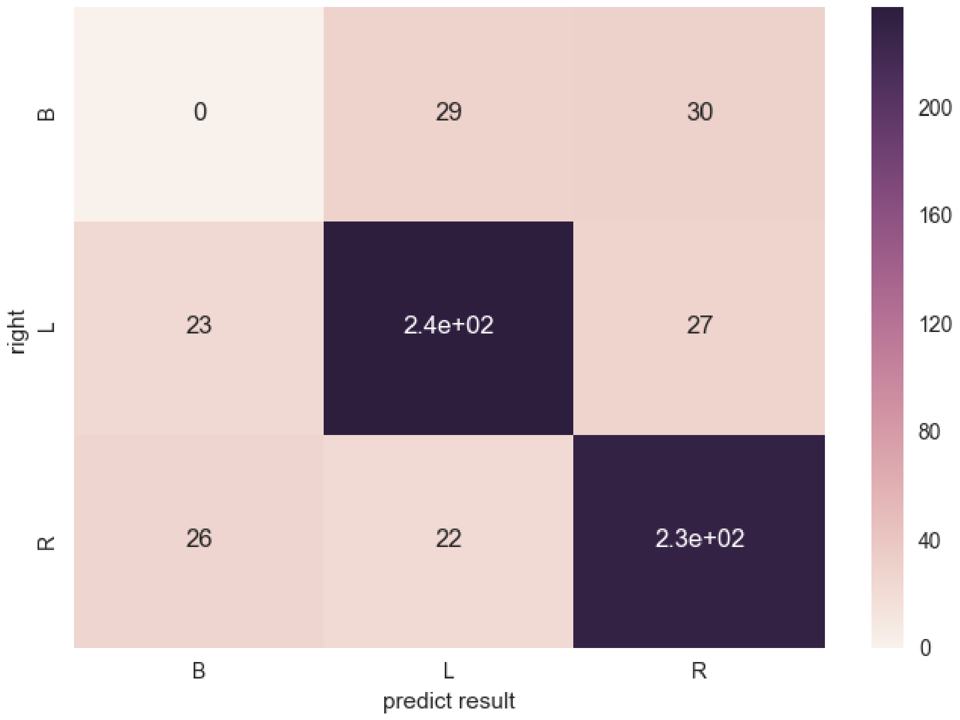


Figure 27: KNN Confusion Matrix

2.4 K-Means Clustering

K-means belongs to unsupervised algorithm. It's a widely used machine learning algorithm.

2.4.1 Algorithm Description

The following items show the main idea:

1. Choose K points randomly before clustering, those points represent initial centroids.
2. Calculate the Euclidean distance between centroids and other points, then classify the data into K clusters.
3. Re-calculate the centroids in each clusters.
4. Repeat Steps 2 and 3 until the centroids no longer change.

Time complexity is

$$O(n * k * t)$$

Among them, n is the length of instances, k is the number of cluster and t is the number of iteration.

2.4.2 Algorithm Implements

The following pseudo code shows the core part, firstly it chooses k centroids, and continue to iterative until no changing.

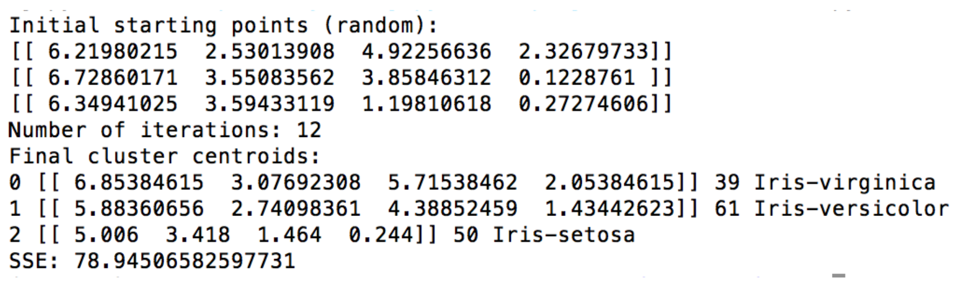
```
row, col = dim(train Set)
cluster_points = zeros((k,col)) # k * col matrix
for j in range(col):
  min_value = min(trainSet[:,j])
  max_value = max(trainSet[:,j])
# find random cluster points,
# add row by row
  cluster_points[:,j] = min_value + (max_value-min_value)*random.rand(k,1)
clusterTable = zeros((row,3)) #record index, dist, cluster
for i in range(row):
  clusterTable[i,0] = i  #initial

flag = True
while flag:
  flag = False
  for i in range(row):
    min_dist = INFINITY
    min_index = -1
    for j in range(k):
      dist = EuclideanDistance(cluster_points[j,:],trainSet[i,:])
      if dist < min_dist:
        min_dist = dist
        min_index = j
      if clusterTable[i,1] != min_index:
                # centroids moves/changes
        flag = True
        clusterTable[i,:] = i,min_index,min_dist**2

  for kk in range(k):
    new_cluster = zeros((1,col))
    count = 0
      for j in range(row):
        if (clusterTable[j,1]) == kk:
          count = count + 1
          new_cluster = new_cluster + trainSet[j,:]
      new_cluster = new_cluster / count
      #re-calculate the centroids
      for r in range(col):
        cluster_points[kk,r] = new_cluster[0,r]
```

2.4.3 Algorithm Validation and Performance

This program used Iris data set.

At first, the program chose 3 random points as centroids, and after 12 times iterations, the dataset was divided into 3 clusters, each cluster and its centroid are displayed in figure 28.

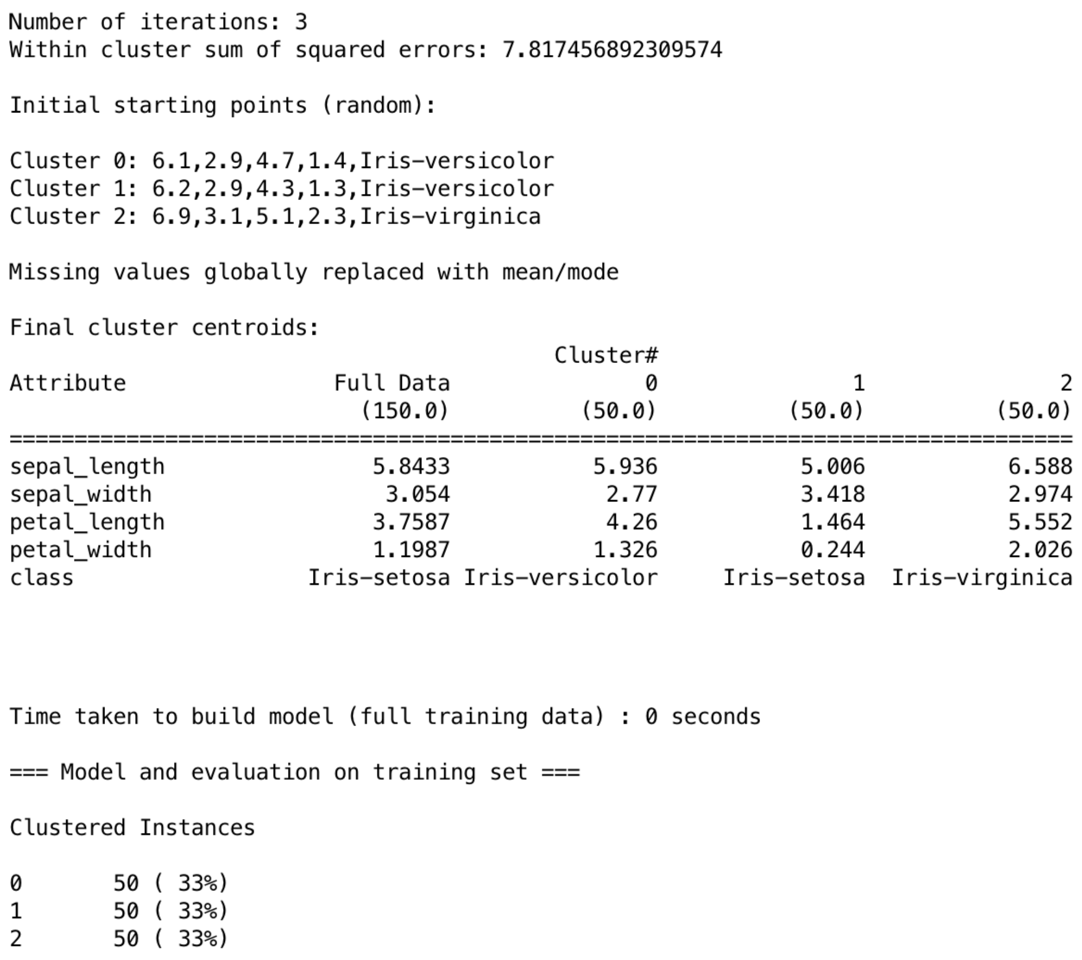
And sum of squared error is 78.945.



```
Initial starting points (random):
[[ 6.21980215  2.53013908  4.92256636  2.32679733]]
[[ 6.72860171  3.55083562  3.85846312  0.1228761 ]]
[[ 6.34941025  3.59433119  1.19810618  0.27274606]]
Number of iterations: 12
Final cluster centroids:
0 [[ 6.85384615  3.07692308  5.71538462  2.05384615]] 39 Iris-virginica
1 [[ 5.88360656  2.74098361  4.38852459  1.43442623]] 61 Iris-versicolor
2 [[ 5.006   3.418   1.464   0.244]] 50 Iris-setosa
SSE: 78.94506582597731
```

Figure 28: K-Means

Figure 29 shows the number of iterations, sum of squared errors, initial random cluster points, final cluster points and results in weka.



```
Number of iterations: 3
Within cluster sum of squared errors: 7.817456892309574

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica

Missing values globally replaced with mean/mode

Final cluster centroids:
                                          Cluster#
Attribute                   Full Data            0               1               2
                              (150.0)       (50.0)          (50.0)          (50.0)
==================================================================================
sepal_length                   5.8433        5.936           5.006           6.588
sepal_width                     3.054         2.77           3.418           2.974
petal_length                   3.7587         4.26           1.464           5.552
petal_width                    1.1987        1.326           0.244           2.026
class                     Iris-setosa Iris-versicolor    Iris-setosa  Iris-virginica




Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       50 ( 33%)
1       50 ( 33%)
2       50 ( 33%)
```

Figure 29: K-Means in Weka

2.4.4 Shortcomings and Improvements

According to the pseudo code, it's easy to implement the algorithm, but there are some shortcomings in the code, for example, each attribute (variable) is spherical, because they are classified by the distance. What's more, in the step 1, the program chooses the centroid randomly, if some centroids are very closely, it maybe cause to *local minimum*.

Figure 30 shows an inappropriate situation: when $k = 3$, but the result only demonstrates two classification. (It uses Iris data set, in order to print a 2-D figure, we only use the first two columns.)

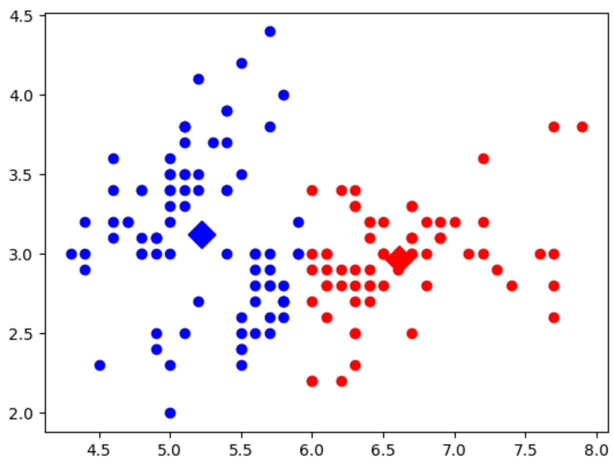


Figure 30: Wrong classifications

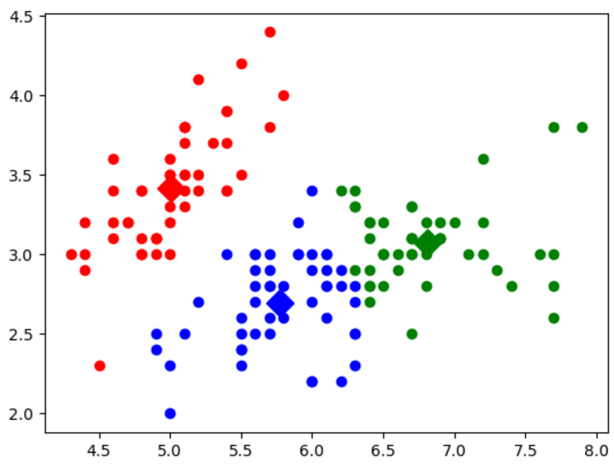


Figure 31: Correct classifications

Run multiple times can get better results. Figure 31 demonstrates the right classification. Or we can use more efficient way to find best clustering centroids.[8]

3 Comparison of algorithms

This Section gives a detail comparison of our 3 algorithms *Naive Bayes Classifier, KNN, ID3* which focus on classification on 4 datasets *iris,blood transfusion donating, car evaluation, balance scale*. The table 11 shows the main feature of 4 datasets.

Table 11: Dataset information

Data Set	**Attribute Characteristic**	**Number**	**Boundary**	**balanced ?**
Iris	numeric	150	clear	Yes
Blood transfusion donate	numeric	748	vague	No
Balance Scale	Categorical	625	clear	Yes
Car	Categorical	1728	Vague	No

3.1 Accuracy and Kappa Statistics

3.1.1 Performance on iris dataset

Iris dataset is a balance dataset with clear boundary between different class.The figure 1 *Iris Dataset Visualization* gives a straight view of boundary information. Below 32 show the overall accuracy and kappa statistics of KNN and Naive Bayes classifier run 100 times on Iris Dataset(use 10-fold cross validation, and the parameter of KNN is k=3).

From the graph, the performance of KNN and Naive Bayes Classifier are stable, both the accuracy and kappa statistic of KNN are slightly higher than Naive Bayes Classifier.

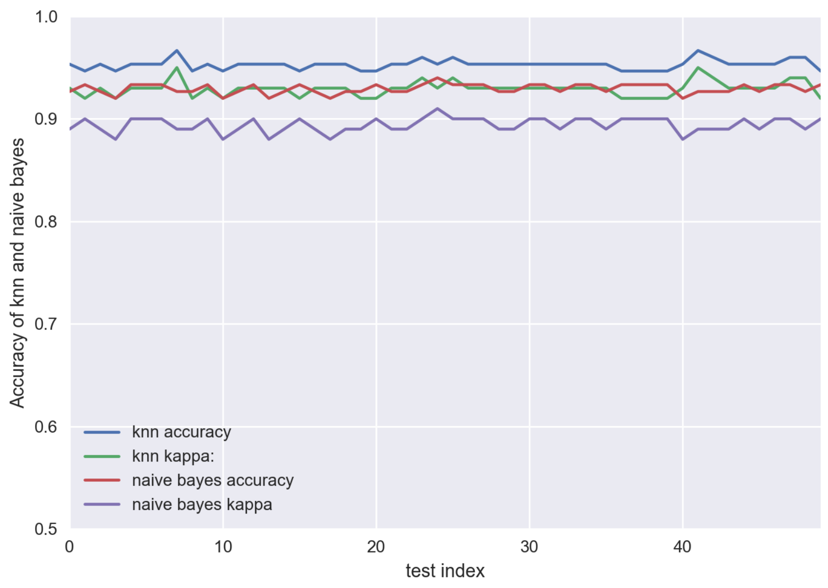


Figure 32: Performance on iris dataset

3.1.2 Performance on blood dataset

Blood transfusion donating dataset is an unbalanced dataset with vague boundary. The figure 5 gives a straight view of distribution information. From the figure 33, both the KNN and Naive Bayes Classifier show good performance on accuracy score, but show really bad performance on kappa statistics.

The figure 35 shows the confusion matrix, most donating samples are classified into not donating. For Naive Bayes Classifier figure 35, all donating samples are classified into not donating.

The blood transfusion dataset is unbalanced, and for Naive Bayes Classifier which needs the independence between attributes, while the blood transfusions have strong correlations between attributes(figure7),what's more, the attributes of Blood transfusion do not satisfy normal distribution, those are the reasons that why Naive bayes classifier gives such bad kappa statistics.

The Naive Bayes classifier can give predictions quickly, but it also have some limitations. It requires the independences among predictors and for numeric values, we also need to apply the right density functions.(In this project, We assume the numeric predictors satisfy the normal distribution and use normal distribution density function, but the blood transfusion donating dataset do not satisfy the assumption.)

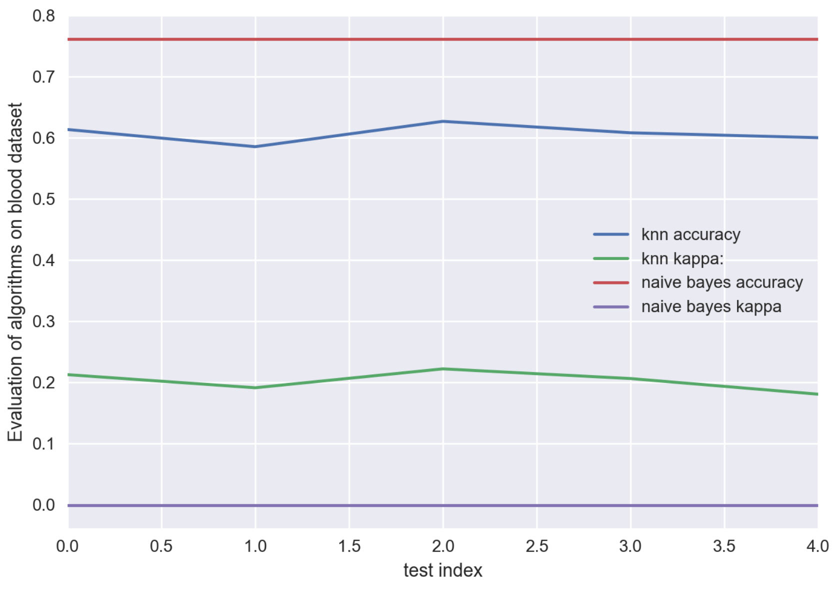


Figure 33: Performance on blood dataset

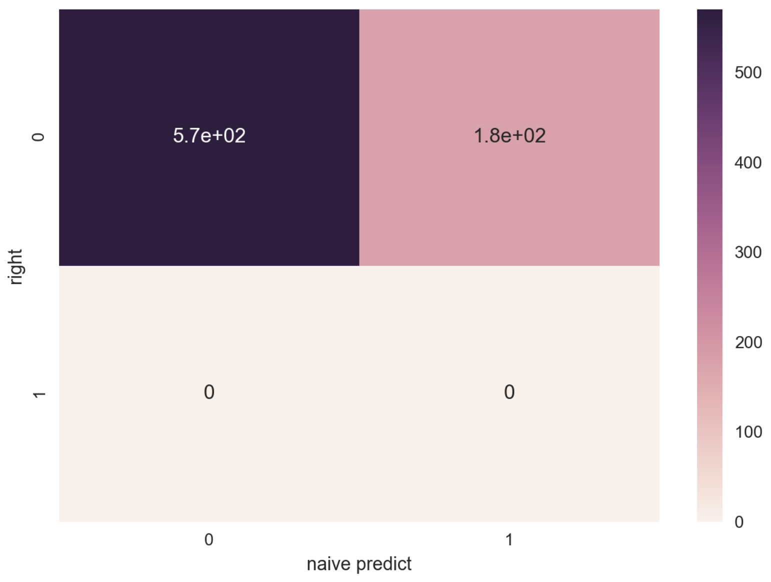


Figure 34: Naive Bayes Confusion Matrix on Blood Transfusion dataset

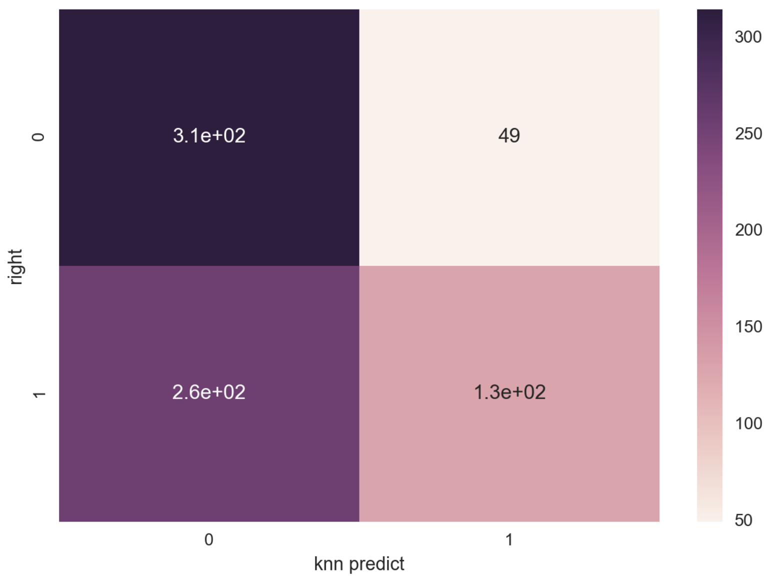


Figure 35: KNN Confusion Matrix on Blood Transfusion dataset

3.1.3 Performance on car dataset

ID3 and Naive Bayes Classifier are used to evaluate the car. The accuracy of ID3 and Naive Bayes Classifier are similar, but ID3 gives better kappa statistics than Naive Bayes.

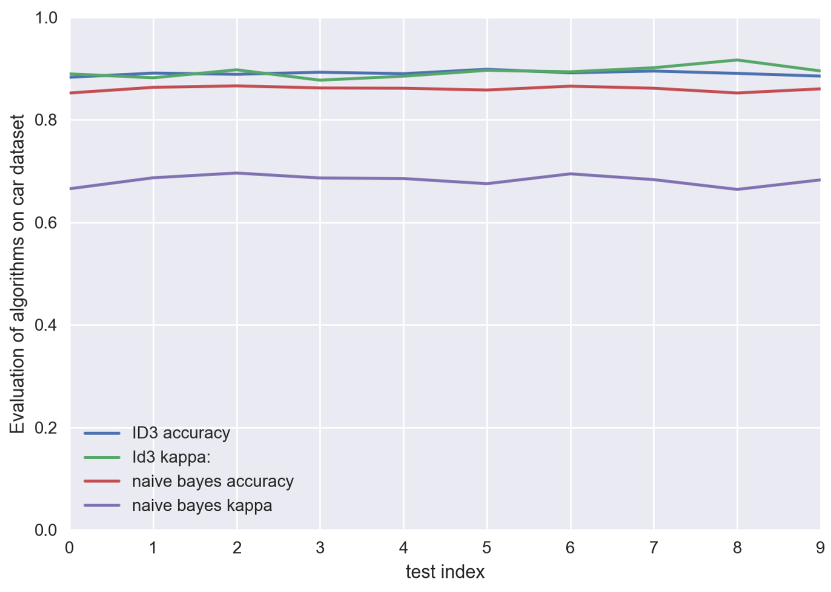


Figure 36: Performance on car dataset

3.1.4 Performance on balance scale dataset

ID3 and Naive Bayes Classifier are used to evaluate the car. The accuracy of ID3 and Naive Bayes Classifier are similar, but ID3 gives better kappa statistics than Naive Bayes.

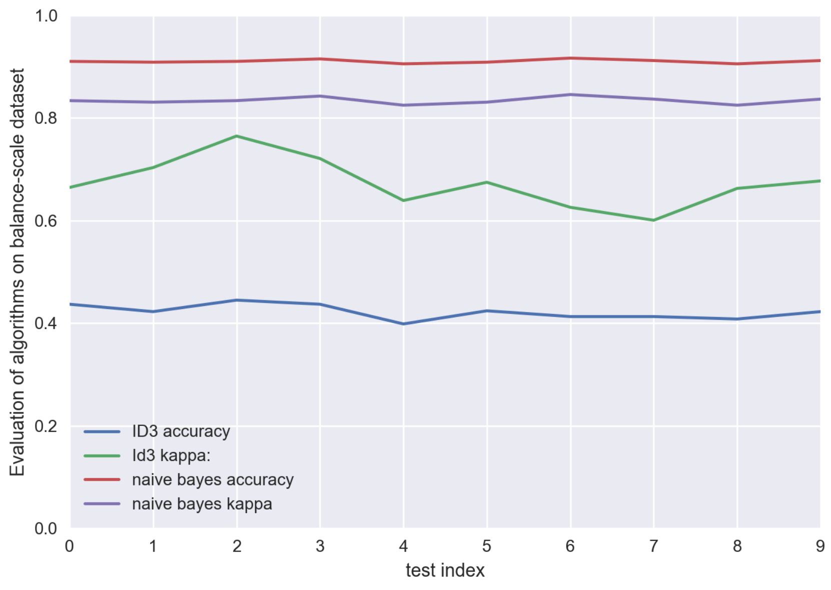


Figure 37: Performance on balance scale dataset

Balance scale dataset is an balanced dataset with clear boundary(figure 8), but the figure 37 shows that ID3 gives a bad performance.

3.2 Time Comparisons

Figure 38 shows the training and predict time using the *iris dataset*. KNN is a lazy algorithm, it need less time to train and more time to predict when it compare to Naive Bayes Algorithm.

Figure 39 shows the time cost when algorithms perform on a larger dataset *Car evaluation*, the predict cost of KNN is larger than ID3 and Naive Bayes. We should avoid using lazy learning algorithms such as KNN on large datasets.

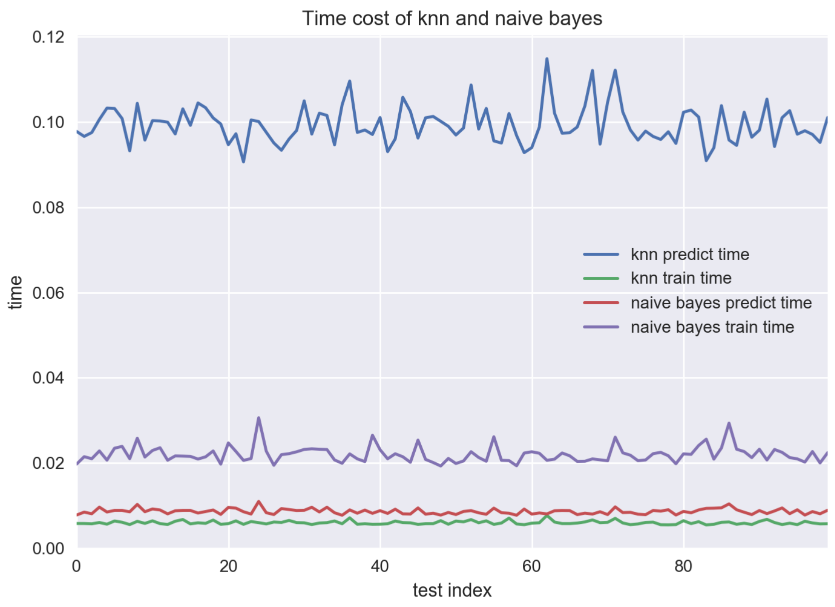


Figure 38: Training and predict time between Naive Bayes and KNN

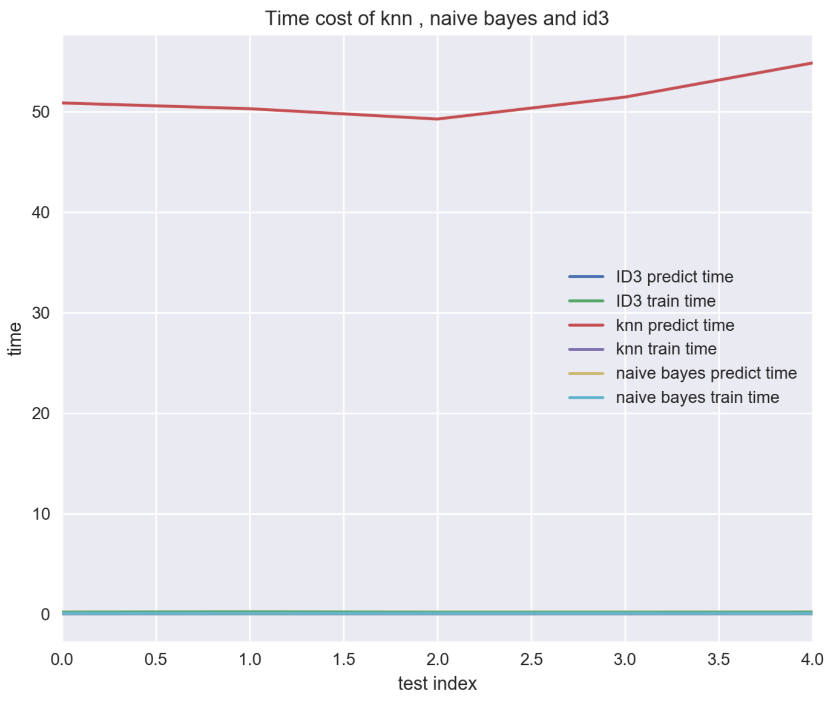


Figure 39: Training and predict time of KNN, Naive Bayes and ID3

Figure 40 shows the time cost of Naive Bayes and ID3. The train time of ID3 is longer than that of Naive Bayes, but the predict time of ID3 is shorter than that of Naive Bayes.

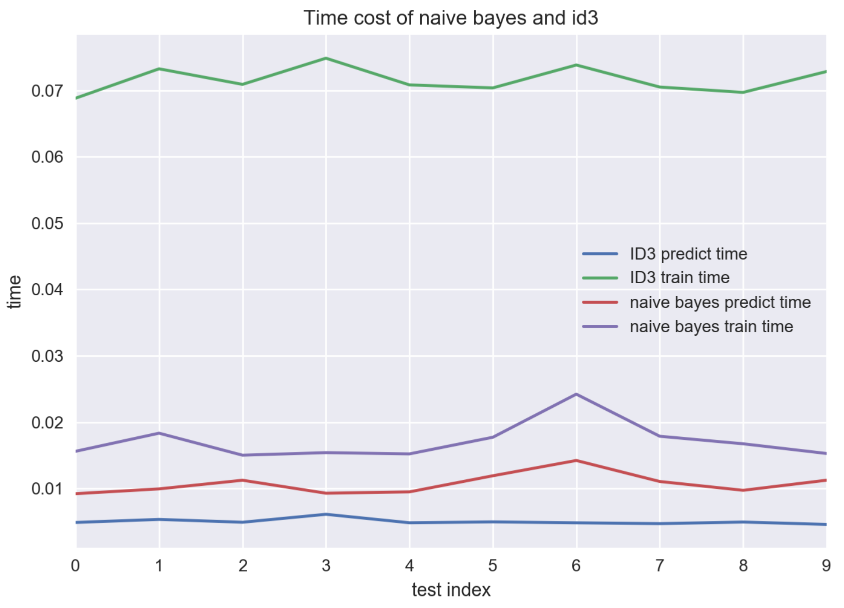


Figure 40: Training and predict time

4 Data Analysis and Preprocessing

Before running the algorithm and building a model, the dataset must be imported in the specified way. In this project, the format of dataset is CSV format, and this program provides two way to read dataset file.

The first one is $readCsv()$, it returns attributes and data in data.frame format. The other one is $readDataSet()$, this function returns two matrix, all attributes are stored in an one-dimensional matrix, other instances are stored in a multi-dimensional matrix.

This project use **Pandas** and **numpy** package to import dataset from local files.

4.1 Normalization

After importing the dataset. This program provides different kinds of normalization ways to adjust values which measured on different scales to a notionally common scale.

- Min Max Normalization

$$x_{normalization} = \frac{x - Min}{Max - Min}$$

Min-max normalization scales the wide range of data between the 0 and 1. The minimum value would be 0, the maximum value converts to 1.

- Z-score Normalization

$$x_{normalization} = \frac{x - \mu}{\sigma}$$

It's used to reduce dimension when using distance to measure similarity, or using PCA technology to reduce dimension.

$$x_{normalization} = log_{10}(x)$$

- Log Normalization

$$x_{normalization} = log_{10}(x)$$

Log Normalization can make the data smooth.

- Arctan Normalization

$$x_{normalization} = atan(x)$$

This normalization is to speed up the convergence of the training network.

In this project, we mainly use the first two normalization function.

4.2 Split Dataset

This program provide a function to divide dataset into training set and test set.

```
def splitDataSet(matrix,split=0.66):
  #define training set and test set
  for i in range(row):
      if np.random.rand() <= split:
            #add to training Set
        else:
            #add to test Set
    return training Set,test Set
```

From the pseudo code, the default split coefficient is 0.66, users can modify the parameter by themselves.

4.3 Cross Validation

The default cross fold is 10.

```
def cross_vali_split_data(dataset, cv=10):
    size = int(np.floor(len(dataset) / cv))
    result = []
    begin = 0
    end = size
    for i in range(cv - 1):
        testset = dataset[begin:end, :]
        trainset = np.row_stack((dataset[0:begin, :], dataset[end:, :]))
        begin = end
        end = end + size
        result.append([trainset, testset])
    result.append([dataset[:begin, :], dataset[begin:, :]])
    return result
```

4.4 Packages used in the implementations and Visualization

1. numpy
2. math
3. time
4. pandas
5. matplotlib
6. seaborn

References

[1] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[2] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[3] I.-C. Yeh, K.-J. Yang, and T.-M. Ting, "Knowledge discovery on rfm model using bernoulli sequence," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5866–5871, 2009.

[4] R. S. Siegler, “Three aspects of cognitive development,” *Cognitive Psychology*, vol. 8, pp. 481–520, 1976.

[5] J. CENDROWSKA, “Prism: An algorithm for inducing modular rules,” *International Journal of Man-Machine Studies*, vol. 27, pp. 349–370, 1987.

[6] J. R. Quinlan, “Induction of Decison Trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.

[7] I. Rish, “An empirical study of the naive bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.

[8] Z. Zhang, J. Zhang, and H. Xue, “Improved k-means clustering algorithm,” in *2008 Congress on Image and Signal Processing*, vol. 5, May 2008, pp. 169–172.