Fall 2019

Microarray Data Analysis and Classification of Cancers

Grant Gates
gjg25@zips.uakron.edu

Honors Project

Microarray Data Analysis and Classification of Cancers

Fall 2019

Grant Gates

Computer Science

11-20-2019

Table of Contents

Introduction

When it comes to cancer classification, there is no standardized approach for identifying new cancer classes nor is there a standardized approach for assigning cancer tumors to existing classes. These two ideas are known as class discovery and class prediction. For a cancer patient to receive proper treatment, it is important that the type of cancer be accurately identified. Different classification methods can yield different accuracies. For my honors project, I wanted to explore different classification algorithms to find the one that returned the best accuracy. I chose this topic because it gave me the opportunity to complete some research surrounding a topic which I found particularly interesting during my undergraduate studies. That is the topic of Bioinformatics. Bioinformatics incorporates a few of my favorite subjects into one including biology, computer science and statistics. An intricate method for class discovery and class prediction is to use microarray data analysis techniques with given gene expression data. The end goal of this project is to find the most accurate way to classify types of cancers with these gene expression levels.

The Data

In order to understand the project better, some background information surrounding the data is necessary. The dataset for this project was obtained from the Broad Institute. The Broad Institute is a Biomedical and Genomic center known for their research conducted in the area of Biology. The data obtained from the Broad Institute's website is split into 4 different files that each contain a respective purpose in the classification process.

These 4 files include:

- Train Dataset - which contains all the gene expression levels for the training data within a 2-dimensional matrix
- Train Classification Values – which contains the expected cancer classification values for the training data in an array
- Test Dataset – which contains all the gene expression levels for the test data within a 2-dimesional matrix
- Test Classification Values – which contains the expected cancer classification values for the test data in an array

In order to do a proper classification, the data needs to be broken into a Train and Test dataset. This can either be done before the classification process or during the classification process. In the case of my honors project, the files were separated prior to the classification process.

In this dataset, there are gene expression levels for two different types of cancer. These two cancers are:

- Acute Lymphoblastic Leukemia (ALL)
- Acute Myeloid Leukemia (AML)

In the classification data files, a 0 represents ALL while a 1 represents AML.

The train dataset will be used to feed the model. The classifier will then fit the model using the corresponding expected train classification values. These values represent the cancer types that go along with the expression levels. Once the model has been fit with the training data, the classifiers will make predictions on the test dataset. The classifier will output the predicted test classification values and those can be compared against the expected test classification values that were provided. From there, it will calculate the degree of certainty or accuracy score for that given classifier.

Preprocessing the Data

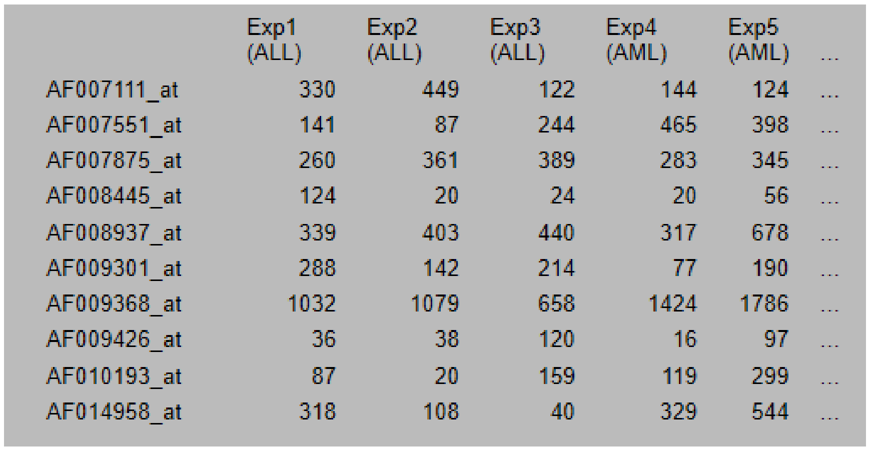
The initial stages of the project mainly focused around the pre-processing clean up and filtering out of all unwanted data from the four files. In order to use the classifiers correctly, the data must be formatted specifically in two respective files. One for train data and one for test data.

First thing to tackle as part of the pre-processing was removing the endogenous control genes from the datasets. These genes are useless for the scope of this project. These genes were marked with "(endogenous control)" in the gene name column, so it was simple to parse the dataset files and remove the rows corresponding to those control genes.

After those genes were removed, then the next course of action was to remove the genes containing absent expression levels. Genes with absent expression levels also did not help at all for the scope of this project. In order to classify genes based on an expression level, a value for that expression level needs to exist.

The last portion of the pre-processing is to set a minimum expression value. For this step, a minimum expression value of 20 was chosen. The reason behind doing this is the background signal level is about 20. If a gene has an expression level that is less than 20, that means the gene is not expressed in the cell. The numerical values that are less than 20 provide no additional information about the genes. Keeping values lower than 20, especially those that are negative would add unnecessary complexity to the classification process.

After the different steps of pre-processing, we are left with the meaningful genes to be used in the classification. This will leave two properly formatted files. The gene names would fall in the first column and then the gene expression levels would fall in the proceeding columns. On the top, the expected cancer classification values from the other file would be placed in the top row of the dataset file. This would then combine the files, so we have one file for train data and one file for test data. This pre-processing step was the same for each so that the files would mimic each other, to help with the accuracy of the classification. Once complete, each of the files should be formatted like so:



	Exp1 (ALL)	Exp2 (ALL)	Exp3 (ALL)	Exp4 (AML)	Exp5 (AML)	...
AF007111_at	330	449	122	144	124	...
AF007551_at	141	87	244	465	398	...
AF007875_at	260	361	389	283	345	...
AF008445_at	124	20	24	20	56	...
AF008937_at	339	403	440	317	678	...
AF009301_at	288	142	214	77	190	...
AF009368_at	1032	1079	658	1424	1786	...
AF009426_at	36	38	120	16	97	...
AF010193_at	87	20	159	119	299	...
AF014958_at	318	108	40	329	544	...

Feature Selection

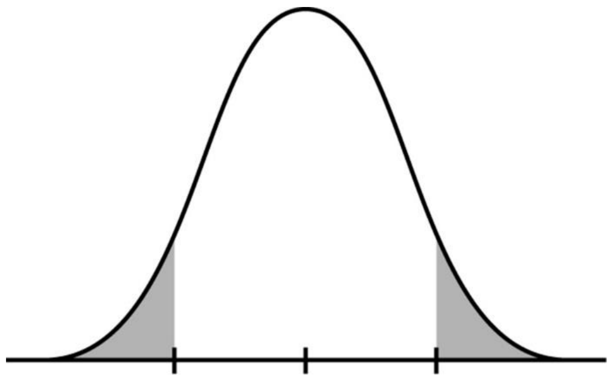
Once we have the files in an easy to read format, the next step in the project is doing feature selection on the training data. The goal of this step is to select the top N number of the most significantly different genes through a two-tailed t-test. These genes will be important in feeding the classifier. The lower the p-value from the two-tailed t-test, the more significantly different the expression levels of the gene in ALL vs AML cancers. For this, we look at the following null and alternative hypotheses:

Ho : The mean gene expression level for ALL equals AML

Ha : The mean gene expression level for ALL does not equal AML

The p-values were obtained for each of the genes by running a two-tailed t-test in excel with the ALL and AML samples. From there, the top N number of genes were selected where N could equal 5, 10, 20, 50 and 100. My experiments have shown that choosing more than 100 genes for an experiment in most cases actually decreased the classification accuracy. These genes were collected in respective excel files for the classification process.

The p-values were extremely small for all the genes collected meaning that these genes differed quite a bit in their expression levels between ALL and AML. As a matter of fact, the p-values obtained were significantly less than the typically used statistical significance of $p = 0.05$ or even $p = 0.01$. If I would have used all the genes with a significance less than $p = 0.05$ for example, it would have left me with around 700 genes.



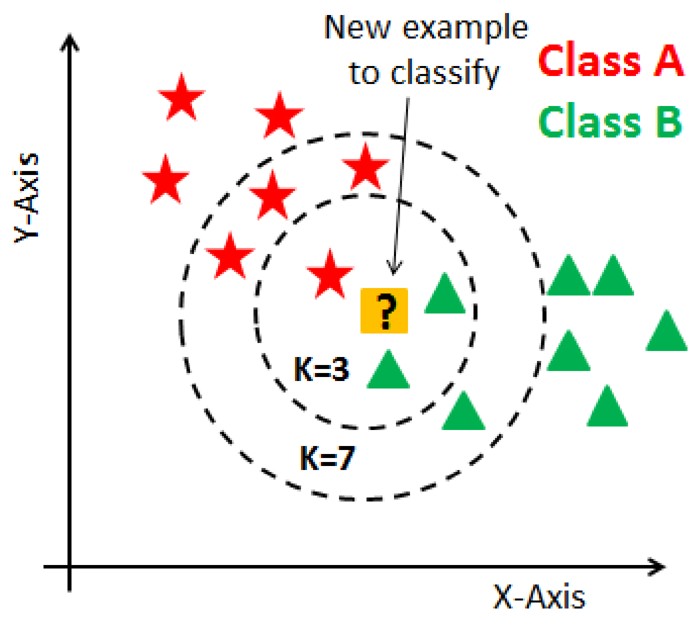
Even with 100 genes selected, the highest or least significant gene p-value obtained from that two tailed t-test was p = 0.0003.

Classification

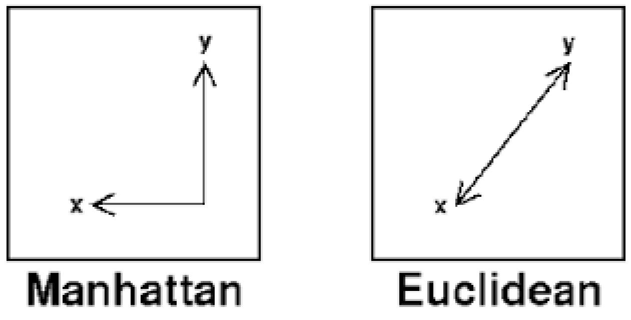
Once the data was in the format that I needed, I began doing the classification on the data. All the classifiers were implemented in a python script. The standard process for all the classifiers followed a similar approach. The data train and test data were read into respective 2-dimensional arrays and the expected train and test values were read into respective arrays. From there, each of the classifiers would fit the train data and expected values. Then, the classifier would predict the values for the train data. An accuracy score could easily be calculated using the predicted and expected cancer classification values (either a 0 or 1). The classifier would then predict the test data and compare predicted versus expected classification values as well. For each of the classifiers, a report would be generated and outputted to the screen with the predicted values array, expected values array and accuracy score in percentage. This process was replicated using N = 5, 10, 20, 50 and 100 genes to compare scores. For the scope of this project, I investigated 6 different classifiers:

k Nearest Neighbor Classifier

The way k Nearest neighbor works is that it makes a prediction on the value using a voting system based on the values of the nearby neighbors. The k value represents the number of neighbors that it will consider on the vote to assign it to a class. This is typically a small positive integer (such as 1, 3 or 5).

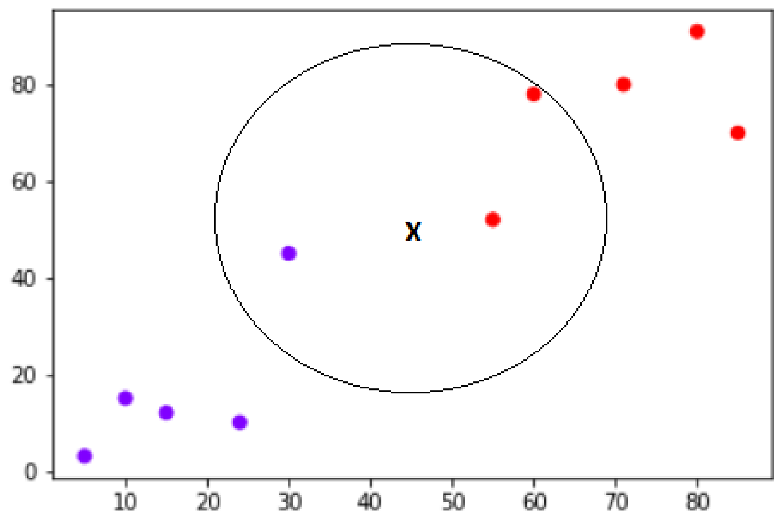


Aside from the k value, another metric that could be changed with the kNN classifier was the distance metric used to calculate distance for the nearest neighbors. There were two options in which I investigated. The first distance metric was Euclidean distance which calculates the distance between two points using a straight line. The second distance metric was Manhattan distance which calculates the distance using a block approach. It makes vertical and horizontal movements only.



Radius Neighbor Classifier

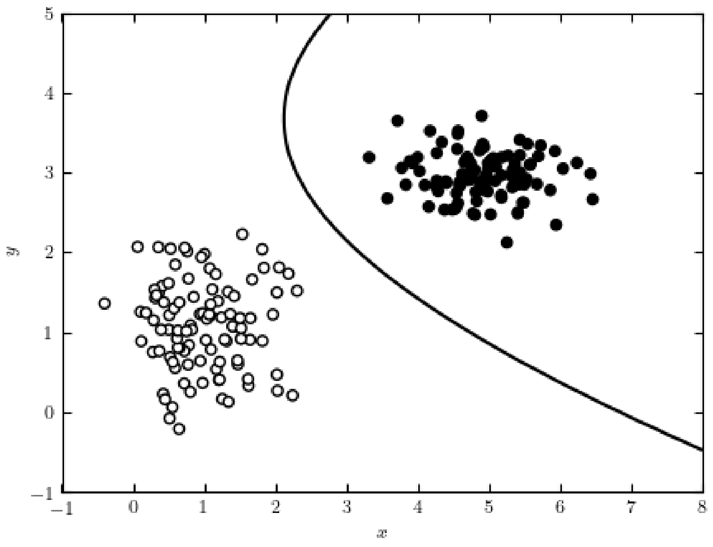
The radius neighbor classifier works very similar to the k Nearest Neighbor classifier. Instead of specifying a set number of neighbors, it makes a prediction using a voting system based on the neighbors that fall within a specified radius around that value. The larger that radius, the more values will be encompassed within the classification vote.



Gaussian Naïve Bayes Classifier

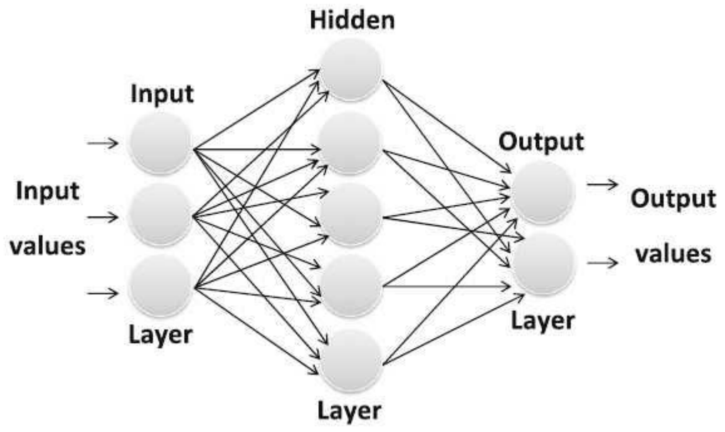
The Gaussian Naïve Bayes Classifier relies more on probability and statistics for making classifications than the previous two that were mentioned. These points can be distributed according to a Gaussian (aka Normal) Distribution where each point of data (gene) can be segmented into groups.

I had the ability to change the variance parameter which controlled the smoothing of the curve separating the two groups.



Multi-layer Perceptron Classifier

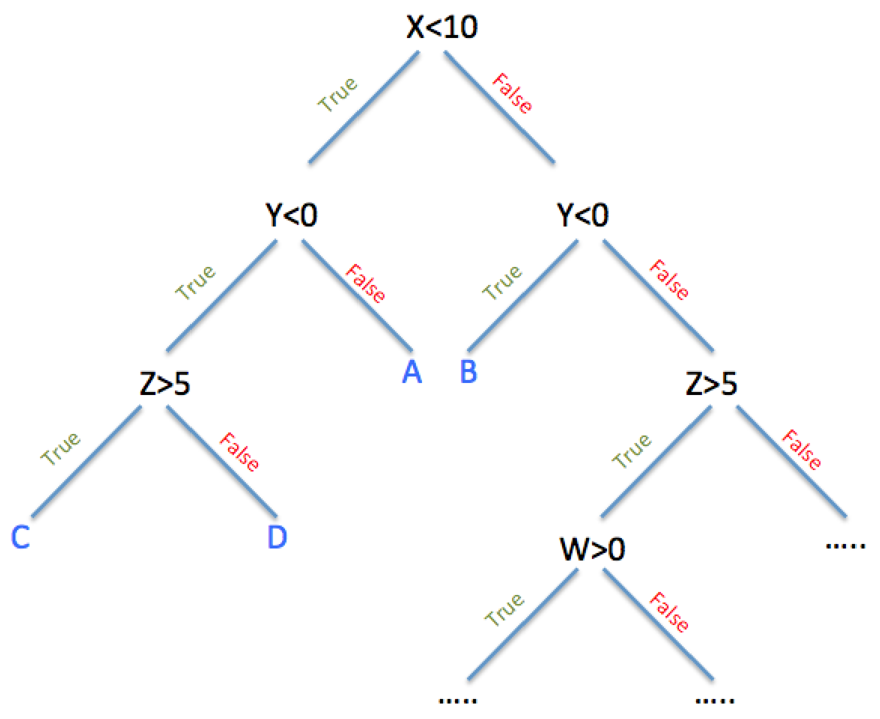
This classifier creates a forward feeding artificial neural network of multiple layers of perceptrons. Basically, these perceptrons are binary classifiers that determine whether it belongs to a specific class or not. There are typically several layers to this process. Input layers, hidden layers and output layers.



This classification method uses backpropagation (which is a machine learning algorithm) against the training data. From there it uses other high-level machine learning techniques to distinguish the class of nodes in the non-linear test dataset. In the end, it will output the classification values.

Decision Tree Classifier

The Decision Tree classifier is another machine learning and statistical method for classification. The name pretty much explains the way in which it works. It has sets of classification trees that have class labels and numerous branches. It flows down the branches in a binary fashion based on the label.



This is one of the simplest classification algorithms from an implementation standpoint. Trees are very simple data structures that can help make decisions from a top down approach. Eventually you will reach the leaf nodes which will have the classification values that we seek to find.

Extra Tree Classifier

An Extra Tree is also known as an extremely randomized tree. This algorithm functions the same as any other tree classifier, however there are very distinct differences. The cut-points for each of the features in the tree are randomized. This allows for more diversified trees which is expected to make more accurate predictions. Works in similar fashion to a random forest algorithm.

Implementation

Implementation of the various stages of the project is probably where I spent most of my time. All the implementation for this project was written in just a few different Python scripts:

Preprocess.py – This python script was written to accomplish all the preprocessing steps that were defined in the Preprocessing section. This included the data cleanup, manipulation and ultimately separation into two different files (a train data file and test data file).

Classification.py – This python script was written to accomplish the classification of the cancers in the test dataset using the train dataset. All 6 classifiers were incorporated in this python script and it spit out a classification report for all the classifiers and their accuracies on predictions.

Along side python I also used Excel extensively for completing portions of this project as well. I used Excel in the feature selection for obtaining the p-values of each gene through two tailed t-tests.

Certain aspects of implementation were difficult to accomplish. The one main example that comes to thought is the implementation of the Radius Neighbor Classifier. It was unclear what to specify as the radius. For this I had to use lots of trial and error. There were also other variables to experiment with, however exploring these was outside the scope of this project.

Results

Each of the classification methods made predictions on 35 patient samples representing either an ALL or AML cancer type. For each prediction test, either 5, 10, 20, 50 or 100 genes were provided to the classifier for all the patient samples to make those prediction.

kNN Classification (Manhattan Distance)

	5 genes	10 genes	20 genes	50 genes	100 genes
k=1	88.57%	91.43%	88.57%	88.57%	100.00%
k=3	94.29%	94.29%	94.29%	88.57%	100.00%
k=5	94.29%	94.29%	94.29%	85.71%	97.14%

The table above encapsulates all the results yielded from the kNN Classifier using Manhattan Distance measurement. In a few of the cases, it predicted all 35 cancer types correctly.

kNN Classification (Euclidean Distance)

	5 genes	10 genes	20 genes	50 genes	100 genes
k=1	82.86%	94.29%	88.57%	85.71%	88.57%
k=3	82.86%	97.14%	94.29%	88.57%	97.14%
k=5	97.14%	94.29%	91.43%	85.71%	94.29%

The table above encapsulates all the results yielded from the kNN Classifier using Euclidean Distance measurement. In a few of the cases, it predicted 34 out of the 35 cancer types correctly.

Radius Classification (Manhattan Distance)					
	5 genes	10 genes	20 genes	50 genes	100 genes
radius = 100000	60.00%	60.00%	60.00%	60.00%	82.86%

The table above encapsulates all the results yielded from the Radius Classifier using Manhattan Distance measurement. In the best case, this classifier correctly predicted 29 out of the 35 cancer types in the test patient samples.

Radius Classification (Euclidean Distance)					
	5 genes	10 genes	20 genes	50 genes	100 genes
radius = 100000	60.00%	60.00%	60.00%	60.00%	82.86%

The table above encapsulates all the results yielded from the Radius Classifier using Euclidean Distance measurement. The results mimicked the Radius Classifier using Manhattan Distance measurement having a best case of 29 out of the 35 cancer types in the test patient samples predicted correctly.

Gaussian Naïve Bayes Classifier					
	5 genes	10 genes	20 genes	50 genes	100 genes
var = 1e-9	82.86%	88.57%	85.71%	85.71%	85.71%

The table above encapsulates all the results yielded from the Gaussian Naïve Bayes Classifier. In the best case, this classifier accurately predicted 31 out of the 35 cancer types in the patient samples.

Multi-layer Perceptron Classifier					
	5 genes	10 genes	20 genes	50 genes	100 genes
	60.00%	42.86%	60.00%	40.00%	42.86%

The table above encapsulates all the results yielded from the Multi-layer Perceptron Classifier. This was the worst classifier, with a best case of 21 out of 35 accurately predicted cancer types.

Decision Tree Classifier					
	5 genes	10 genes	20 genes	50 genes	100 genes
	74.29%	74.29%	74.29%	74.29%	74.29%

The table above encapsulates all the results yielded from the Decision Tree Classifier. This classifier predicted 26 out of the 35 cancer types correct consistently across the board.

Extra Tree Classifier					
	5 genes	10 genes	20 genes	50 genes	100 genes
	74.29%	97.14%	85.71%	77.14%	68.57%

The table above encapsulates all the results yielded from the Extra Tree Classifier. This classifier had inconsistent results, but in one case predicted 34 out of the 35 cancer types correctly.

Analysis

Now it is time for the deep analytical dive into the data obtained from each of the 6 classifications on the data. There were a few classifiers that had really promising results and others that did not. From the results, the k Nearest Neighbor classifier obviously did the best at predicting the appropriate cancer type. Even in some cases, the kNN classifier was 100% correct in choosing the type of cancer.

Looking deeper into this specific classifier, more frequent than not the misclassified genes were the same throughout changing the variables (k = number of neighbors and n = number of genes). These misclassifications could be a certain gene that is more near the other type or just a misclassification of the data from the Broad Institute all together. However, I am not assuming that at all.

The second-best classifier ended up being the Gaussian Naïve Bayes Classifier with all the classifications being in the 80th percentile. You would expect a more statistical driven classifier to bring back some higher accuracies. The other classifiers did not deliver promising accuracies, but it was worth including them for comparison. Both tree driven classifiers came out with similar results except for the one-off occasion of the Extra Tree getting 97% accuracy.

These results only re-enforce the fact that the k Nearest Neighbor classifier is such a powerful classification method when it comes to bringing back very accurate results.

Conclusion

I had no expectation of curing cancer with this project, as that would be extremely far fetched and difficult. My objective with this project was to investigate the different classifiers and how well they performed the classification task with the data that I threw at them. I came across some promising results on some of the classifiers that I chose to investigate.

I can assume that doctors are looking for a consistently accurate way to diagnose cancers within patients. The k Nearest Neighbor classifier is the classifier I would personally trust the most to deliver both consistent and highly accurate results. The other classifiers did not do terrible, but considering the circumstances, we would like to cut down that margin of error as much as possible.

References

The Data

http://portals.broadinstitute.org/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=43

Classification – k Nearest Neighbor

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

http://www.ieee.ma/uaesb/pdf/distances-in-classification.pdf

Classification – Radius Neighbor Classifier

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html

https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

Classification - Gaussian Naïve Bayes Classifier

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

https://www.astroml.org/book_figures/chapter9/fig_simple_naivebayes.html

https://www.statlect.com/images/normal_distribution.png

Classification – Multilayer Perceptron

https://en.wikipedia.org/wiki/Multilayer_perceptron

https://www.researchgate.net/profile/Mouhammd_Al-Kasassbeh/publication/309592737/figure/fig2/AS:423712664100865@1478032379613/MultiLayer-Perceptron-MLP-sturcture-334-MultiLayer-Perceptron-Classifier-MultiLayer.png

Classification – Decision Tree

https://en.wikipedia.org/wiki/Decision_tree_learning

https://miro.medium.com/max/663/1*xGsYc6aXehD7lyoLEn-mMA.png

Classification – Extra Tree

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

Appendix (Source Code)

Preprocess.py

```
# Preprocess.py
# Grant Gates

import os

# file names
clsfile = "Leuk_ALL_AML.test.cls.txt"
resfile = "Leuk_ALL_AML.test.res"
processedfile = "test_training.txt"

# Read the .cls file to get subtype information
subtypes = []
numtypes = 0

# Place Files in Directory of py File
os.chdir(os.getcwd())

# Read in the .cls file
fileinput = []

with open(clsfile) as file:
    for line in file:
        for word in line.split():
            fileinput.append(word)

subtypes = fileinput[3:len(fileinput)].copy()
numtypes = int(fileinput[0])

# Read in the .res file
skipcount = ((numtypes * 4) + 7) - 1
count = 0
wantcount = (numtypes * 2) + 4 - 2
readincount = skipcount + wantcount

# Get total number of lines
totallinescount = 0

with open(resfile) as file:
    for line in file:
        for word in line.split('\t'):
            totallinescount = totallinescount + 1

totalgenescount = 7129

genedataarray = [[0 for x in range(wantcount)] for y in range(totalgenescount)]
row = 0
column = 0

with open(resfile) as file:
    for line in file:
        for word in line.split('\t'):
            if count < skipcount:
                count = count + 1
            elif count < readincount:
                genedataarray[column][row] = word
                row+=1
                count+=1
            else:
                readincount = readincount + wantcount
                row = 0
                column+=1

# Output to File
with open(processedfile, 'w') as f:
    count = 1
    while count <= numtypes:
        print('\tExp', count, file=f, sep='', end ='')
        count+=1

    print('\n', file=f, end='')

    count = 0
    while count < numtypes:
        if subtypes[count] == '0':
            print('\t', '(ALL)', file=f, sep='', end='')
        else:
            print('\t', '(AML)', file=f, sep='', end='')
        count+=1

    print('\n', file=f, end='')

    row = 0
```

```
Acount = 0
while row < totalgenescount:
    # if all A's, do not print
    column = 3
    Acount = 0
    while column < wantcount:
        if genedataarray[row][column] == 'A':
            Acount+=1
        column+=2

    # if gene has less than 2 fold change, do not print
    column = 2
    min = 10000
    max = -10000
    while column < wantcount:
        if int(genedataarray[row][column]) < min:
            min = int(genedataarray[row][column])
        if int(genedataarray[row][column]) > max:
            max = int(genedataarray[row][column])
        column += 2

    if min != 0:
        if (max/min) < 2:
            Acount = numtypes

    if Acount != numtypes:
        column = 0
        # do not print those with endogenous control
        if "endogenous control" not in genedataarray[row][column]:
            column+=1
            print(genedataarray[row][column], '\t', file=f, sep='', end='')
            column+=1
            while column < wantcount:
                # if below 20, set to 20
                if int(genedataarray[row][column]) < 20:
                    print('20', '\t', file=f, sep='', end='')
                else:
                    print(genedataarray[row][column], '\t', file=f, sep='', end='')
                column+=2
            print('\n', file=f, end='')
    row+=1
```

Classification.py

```
# Classification.py
# Grant Gates

import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import RadiusNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier

# For the labels...
# ALL = 0, AML = 1

# Read in the Training data
# Split into Train_Data and Train_Labels
training = pd.read_csv("Train5.csv")
Train_Data = np.array(training.drop(['type'], 1))
Train_Labels = np.array(training['type'])

# Read in the Test data
# Split into Test_Data and Test_Labels
test = pd.read_csv("Test5.csv")
Test_Data = np.array(test.drop(['type'], 1))
Test_Labels = np.array(test['type'])

# kNN Classifier - 1 Nearest Neighbor (Manhattan)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 1 Nearest Neighbor (Manhattan)")
Manhattan_1 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                      metric='minkowski', metric_params=None, n_jobs=1,
                      n_neighbors=1, p=1, weights='uniform')

Manhattan_1.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Manhattan_1.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(accuracy_score(Train_Labels, Train_Predict) * 100, end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Manhattan_1.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(accuracy_score(Test_Labels, Test_Predict) * 100, end = '')
print(" %")
print("-----------------------------------------")
print("")

# kNN Classifier - 3 Nearest Neighbor (Manhattan)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 3 Nearest Neighbor (Manhattan)")
Manhattan_3 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                    metric='minkowski', metric_params=None, n_jobs=1,
                    n_neighbors=3, p=1, weights='uniform')

Manhattan_3.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Manhattan_3.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Manhattan_3.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("-----------------------------------------")
print("")
```

```
# kNN Classifier - 5 Nearest Neighbor (Manhattan)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 5 Nearest Neighbor (Manhattan)")
Manhattan_5 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                    metric='minkowski', metric_params=None, n_jobs=1,
                    n_neighbors=5, p=1, weights='uniform')

Manhattan_5.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Manhattan_5.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Manhattan_5.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# kNN Classifier - 1 Nearest Neighbor (Euclidean)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 1 Nearest Neighbor (Euclidean)")
Euclidean_1 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                    metric='minkowski', metric_params=None, n_jobs=1,
                    n_neighbors=1, p=2, weights='uniform')

Euclidean_1.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Euclidean_1.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Euclidean_1.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# kNN Classifier - 3 Nearest Neighbor (Euclidean)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 3 Nearest Neighbor (Euclidean)")
Euclidean_3 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                     metric='minkowski', metric_params=None, n_jobs=1,
                     n_neighbors=3, p=2, weights='uniform')

Euclidean_3.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Euclidean_3.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Euclidean_3.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")
```

```
# kNN Classifier - 5 Nearest Neighbor (Euclidean)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
print("kNN Classifier - 5 Nearest Neighbor (Euclidean)")
Euclidean_5 = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                    metric='minkowski', metric_params=None, n_jobs=1,
                    n_neighbors=5, p=2, weights='uniform')

Euclidean_5.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Euclidean_5.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Euclidean_5.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Radius Neighbors Classifier - Radius of 100000.0 (Manhattan)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html
print("Radius Neighbors Classifier - Radius of 100000.0 (Manhattan)")
Radius_1 = RadiusNeighborsClassifier(radius=100000.0, weights='uniform', algorithm='auto',
leaf_size=30, p=1, metric='minkowski', outlier_label=None, metric_params=None, n_jobs=None)

Radius_1.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Radius_1.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Radius_1.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Radius Neighbors Classifier - Radius of 100000.0 (Euclidean)
# https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html
print("Radius Neighbors Classifier - Radius of 100000.0 (Euclidean)")
Radius_2 = RadiusNeighborsClassifier(radius=100000.0, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', outlier_label=None, metric_params=None, n_jobs=None)

Radius_2.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = Radius_2.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = Radius_2.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy:")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Gaussian Naive Bayes Classifier (Variance = 1e-9)
# https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
print("Gaussian Naive Bayes Classifier (Variance = 1e-9)")
```

```
GNB = GaussianNB()
GNB.fit(Train_Data, Train_Labels)
GaussianNB(priors=None, var_smoothing=1e-09)
print("Train Data Predictions:")
Train_Predict = GNB.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = GNB.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Multi-layer Perceptron Classifier
# https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
print("Multi-layer Perceptron Classifier")
MLP = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001,
                solver='sgd', verbose=10,  random_state=21,tol=0.000000001)
MLP.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = MLP.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = MLP.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Decision Tree Classifier
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
print("Decision Tree Classifier")
DT = DecisionTreeClassifier(random_state=0)
DT.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = DT.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")

print("Test Data Predictions:")
Test_Predict = DT.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")

# Extra Tree Classifier
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.ExtraTreeClassifier.html
print("Extra Tree Classifier")
ET = ExtraTreeClassifier()
ET.fit(Train_Data, Train_Labels)
print("Train Data Predictions:")
Train_Predict = ET.predict(Train_Data)
print(Train_Predict)
print("Train Data Actual Values")
print(Train_Labels)
print("Train Data Accuracy")
print(round(accuracy_score(Train_Labels, Train_Predict) * 100, 2), end = '')
print(" %")
print("Test Data Predictions:")
Test_Predict = ET.predict(Test_Data)
print(Test_Predict)
print("Test Data Actual Values")
print(Test_Labels)
print("Test Data Accuracy")
print(round(accuracy_score(Test_Labels, Test_Predict) * 100, 2), end = '')
print(" %")
print("----------------------------------------")
print("")
```