CSC 590: Final Project

Jennefer Maldonado

December 27, 2021

1 Introduction

Health is one of the most important things about ourselves that we need to take care of. If a doctor suspects something is wrong with a patient, they can send them to get blood work done. Different levels of bio-markers, whether it be hormones, insulin, proteins, or cholesterol, are indications of disease or infection. Hepatitis C is an infection of the liver that is spread by being in contact with the blood of an infected person. If hepatitis C is caught early, it can be cured in 8 to 12 weeks. If it is not detected it can become chronic leading to cirrhosis and liver cancer [CDC, 2020]. Using data from previous hepatitis C blood panels for patients ranging in age from 19 to 77 and data classifiers, it is possible to determine the progression of infection. This tool is important for doctors to properly diagnosis and take action to prevent serious illness and death in patients.

The data set has multiple lab records from blood donors and from patients diagnosed with hepatitis C, Fibrosis, and Cirrhosis. The table below discusses the attributes in more detail.

Attribute	Description
Patient ID	Numeric value associated with a patients lab work.
Category/Diagnosis	Contains the following values: '0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis'.
Age	The patient's age in years.
Sex	The patient's sex. The data set assumes only two choices: female or male.
ALB	Albumin, a protein created by the liver. Low levels indicate liver or kidney problems [MedlinePlus, 2020].
ALP	Alkaline phosphatase, an enzyme mainly found in the liver and kidneys. High levels may indicate liver disease [MedlinePlus, 2020].
ALT	Alanine transaminase, an enzyme released into the bloodstream when liver cells are damaged[MedlinePlus, 2020].
AST	Aspartate aminotransferase, an enzyme released into the bloodstream when the liver is damaged[MedlinePlus, 2020].
BIL	Bilirubin, the substance released when the body breaks down red blood cells. A healthy liver filters out the bilirubin from the. body [MedlinePlus, 2020].
CHE	Serum cholinesterase, enzyme that indicated liver cirrhosis [Ramachandran, 2014].
CHOL	Cholesterol, a substance similar to fat produced in the liver needed to keep cells and organs healthy[MedlinePlus, 2020].
CREA	Creatinine, waste produced by muscles filtered out by the kidneys. High levels can be a sign of kidney disease[MedlinePlus, 2020].
GGT	Gamma-glutamyl Transferase, enzyme found mainly in the liver. High levels in blood can indicate liver disease[MedlinePlus, 2020].
PROT	Total Protein and Albumin/Globulin (A/G) Ratio, measures the total amount of Albumin and Globulins in the blood[MedlinePlus, 2020].

Table 1: Attribute name and description from hepatitis C data

Using classifiers we will determine if these bio-markers are sufficient means of prediction for hepatitis C.

1.1 Missing Value Treatment

All patients are suggested to have the levels of bio-markers in Table 1 checked but there were a few instances where there were missing attribute values. This causes problems later on when trying to compute statistics and normalize data. I considered a few options like interpolation or averaging to fill in the missing attributes but was concerned a wrong interpretation for this value could lead to an incorrect diagnosis. In the real world, we want to prevent false negatives and false positives as much as possible to avoid patients getting the wrong treatment or no treatments at all. With this in mind and given the size of the data set I decided to drop rows missing any attribute data and only focus on patients who had all tests run.

2 Summary Statistics

Computing summary statistics on a data set is a good way of getting insight into what classifier is appropriate for the data.

2.1 Mean and Standard Deviation

The table below displays the mean and standard deviation of all attribute columns excluding the Sex and Category. The values range from 5.39 to 81.68 for the means and 1.13 to 54.26 for the standard deviations. The large range may become an issue and can be dealt with using data transformations.

Attribute	Mean	Standard Deviation
Age	47.41765704584041	9.922899410500207
ALB	41.624278438030565	5.756900755872436
ALP	68.12308998302207	25.899057870922846
ALT	26.575382003395585	20.845402038444735
AST	33.77283531409168	32.838958430080254
BIL	11.018166383701189	17.39178892279242
CHE	8.203633276740238	2.1892118670792398
CHOL	5.391341256366723	1.1279953825414155
CREA	81.66910016977928	50.653936532979145
GGT	38.19847198641765	54.256290327573964
PROT	71.89015280135824	5.344340402704473

Table 2: Means and Standard Deviations

3 Data Transformation

In the previous section the mean and standard deviation was discussed. It is important to take note that since some columns had much larger mean and standard deviation values than others, for example CHOL and PROT, there is a possibility for issues with analysis and classifying. I decided it would be best to normalize the data so that all values for the attributes are in $[0, 1]$. Also the sex attribute was specified by 'm' or 'f', using `data_frame['Sex'].astype('category').cat.codes` the values were converted to 0 and 1 respectively.

Attribute	Mean	Standard Deviation
Age	0.45217883418222976	0.18375739649074457
ALB	0.39709180442838876	0.08554087304416695
ALP	0.14020007397735523	0.0639009569709561
ALT	0.07914729347532548	0.06425832934169154
AST	0.07394012544381519	0.10478289224658666
BIL	0.04907860895149467	0.08353404862052075
CHE	0.45254391439227737	0.1460448210192955
CHOL	0.48074529810275773	0.1368926435123077
CREA	0.06877891902696227	0.04729151016056312
GGT	0.05213253710770059	0.08393609270973695
PROT	0.6496439520709408	0.12816164035262526

Table 3: Normalized Means and Standard Deviations

Now all attributes can be compared to one another without having issues with maximums and minimums.

3.1 Correlation Coefficient

For a thorough analysis of correlation in the data set, a for loop was used to compute the correlation coefficient in between each attribute. For a more detailed explanation of a correlation coefficient, see the appendix.

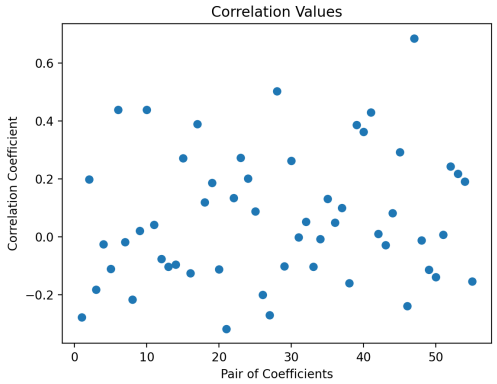

Figure 1: Correlation Values Scatter Plot

After going through each attribute, none of the correlation coefficients were close to 1 or -1. Therefore, there is no correlation between attribute pairings in the data set giving a good justification for using the K-Nearest Neighbors classifier.

4 Classifiers

4.1 K-Nearest Neighbors Classifier

There are a few different approaches that can be taken when using the K-Nearest Neighbors Classifier. The first analysis is simply, can the classifier predict the level of hepatitis C infection the patient has. Running the classifier given all attributes it has a 92.85% accuracy in predicting the infection level. Manually inspecting the labels it predicted incorrectly, the model seems to be more pessimistic. Most of the uninfected blood donors are labeled as having Cirrhosis or Hepatitis. If this tool was used to aid in a doctors diagnosis, the patient would often be misdiagnosed with a worse outcome than they were expecting.

4.2 Naive Bayes Classifier

The Naive Bayes approach requires more statistics to be computed in order to classify data. From previous sections we saw that no two columns were correlated with each other. Once the classifier ran it had very poor results only correctly labeling around 4% of the instances it tried to classify. This is possible despite not seeing correlation because there is an underlying relationship between the bio markers in this data set. Most are influenced by the kidney or the liver, this means that there is a chance that each attribute is not independent. This would lead to poor performance in the Naive Bayes classifier.

5 Conclusion

Classifiers are useful tools for predicting classes of data. For predicting hepatitis C, the K-Nearest Neighbors classifier has the most promise. There are many other iterations of work that could be done to improve the classifier. This could come from either a larger data set or even new and improved blood work done. For now it can be concluded that the current methods to diagnose patients are close to accurate in the data mining world.

6 Appendix

6.1 Technical Explanations

6.1.1 Correlation Coefficient

I used Pearson's correlation which is defined as,

$$corr(x, y) = \frac{cov(x, y)}{stddev(x) \times stddev(y)} \tag{1}$$

where covariance, standard deviation, and the mean are defined as follows.

$$cov(x, y) = \frac{1}{n-1} \sum_{k=1}^{n} (x_k - \bar{x})(y_k - \bar{y}) \tag{2}$$

$$stddev(x) = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} (x_k - \bar{x})^2} \tag{3}$$

$$\bar{x} = \frac{1}{n} \sum_{k=1}^{n} x_k \tag{4}$$

The correlation coefficient determines if there is a relationship between two attributes. A correlation coefficient of 1 means there is a strong positive correlation, -1 means a strong negative correlation, and 0 means no conclusion about correlation. We can use this to determine if there are relationships between attributes in data sets. This will then influence the choice of classifier used.

6.1.2 Min-Max Normalization

Normalization is important for data in a few cases. When training neural network large input data can lead to loss values approaching infinity. For classifiers, when attribute columns have different means and standard deviations it may be difficult to compare statistics or distances. Normalization allows for data to have a normal distribution throughout the data set. Min-Max normalization finds the minimum and maximum of a set of data points. Then takes each point, subtracts the minimum and divides by the maximum minus the minimum. This converts all data points to a value in $[0, 1]$.

6.1.3 KNN

KNN classifier's predict a class label based on the 'closest' neighbors to a data point. To find how close to data points are to each other we can compute the distance between the two. For my project I chose to use the Euclidean distance which is defined as,

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{5}$$

The code finds the euclidean distance of each test instance against every training instance in the data. The neighbors with the smallest distance between the test instance are then chosen. The class label that appears the most among the neighbors is then the predicted label for the test instance.

6.1.4 Naive Bayes

The Naive Bayes Classifier needs to maximize the probability that a given data point belongs to a certain class label. This is done by maximizing the following equation.

$$P(c) \prod_{i=1}^{n} P(x_i|c) \tag{6}$$

Where c is a class label and x_i is the attribute of a given test row. The code written represents the Gaussian Naive Bayes classifier. The Gaussian distribution function computes the relative likelihood that a data point would be close to the given attribute set.

$$\frac{1}{\sigma\sqrt{2 * \pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{7}$$

In the code, the class probability is computed first and then is multiplied by the distribution function for each attribute in the data set. This gives us the probability that the test instance has each class label.

6.2 Code

```
#!/usr/bin/env python3

'''
Jennefer Maldonado
Final Project for CSC 590

Detect Hepatitis C disease using blood biomarkers
'''

import time
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


'''
Missing Value Treatment
'''
def clean_data(data):
    num_rows = len(data)
    rows_to_drop = []
    for i in range(0, num_rows):
        current_row = data.iloc[i]
        for value in current_row:
            if value == 'NA':
                rows_to_drop.append(i)
    # make list unique
    rows_to_drop = list(set(rows_to_drop))
    clean_dataset = data.drop(rows_to_drop)
    return clean_dataset

def mean(data):
    '''
    Modified mean method from Naive Bayes Classifer
    '''
    mean_dict = {}
    for (col_name, col_data) in data.iteritems():
        if all ([col_name != 'Category', col_name != 'Unnamed: 0', col_name != 'Sex']):
            col_data = col_data.to_numpy()
            col_data = col_data.astype(np.float64)
            col_mean = np.mean(col_data)
            mean_dict[col_name] = col_mean
    return mean_dict

def standard_dev(data):
```

```
    std_dict = {}
    for (col_name, col_data) in data.iteritems():
        if all ([col_name != 'Category', col_name != 'Unnamed: 0', col_name != 'Sex']):
            col_data = col_data.to_numpy()
            col_data = col_data.astype(np.float64)
            std_dev = np.std(col_data)
            std_dict[col_name] = std_dev
    return std_dict

def normalize(data):
    '''
    Using Minimum and Maximum of column data to normalize a given data set
    '''
    norm_dataframe = data.copy()
    for (col_name, col_data) in data.iteritems():
        if all ([col_name != 'Category', col_name != 'Unnamed: 0', col_name != 'Sex']):
            current_col = pd.to_numeric(data[col_name])
            maximum = current_col.max()
            minimum = current_col.min()
            norm_dataframe[col_name] = (current_col- minimum)/(maximum-minimum)
    return norm_dataframe

def covariance(col_x,col_y):
    '''
    Compute the covariace suing numpy mean
    '''
    col_x = col_x.to_numpy()
    col_y = col_y.to_numpy()
    n = len(col_x)
    multiplier = 1/(n-1)
    x_bar = np.mean(col_x)
    y_bar = np.mean(col_y)
    total_sum = 0
    for i in range(0,n):
        total_sum += (col_x[i]- x_bar)*(col_y[i]-y_bar)
    return (multiplier*total_sum)

def correlation(col_x,col_y):
    '''
    Compute the correlation coefficient using the covariance and numpy standard deviation
    '''
    cov = covariance(col_x,col_y)
    std_dev_x = np.std(col_x)
    std_dev_y = np.std(col_y)
    return (cov/(std_dev_x*std_dev_y))

'''
KNN Implementation
'''
 # want to ignore col 0, 1, and 3

def euclid_dist(v1, v2):
    '''
    Computes the Euclidean Distance of two vectors v1 and v2
    Ignores the name and type column in distance calculation
    Params:
    v1 - list of attribute values
    v2 - list of attribute values
    Returns the euclidean distance of the two vectors
```

```
    '''
    dist = 0
    for vi in range(2, len(v1)):
        dist += (v1[vi] - v2[vi])**2
    return math.sqrt(dist)

def find_neighbors(current_vector, possible_neighbors, max_neighbors):
    '''
    Finds all possible neighbors by storing the distances in a dictionary,
    sorting the dictionary and finding the first k neighbors with the closest distances
    Params:
    current_vector - the row we are checking
    possible_neighbors - the set of possible rows to check against
    max_neighbors - the maximum number of neighbors per cluster
    '''
    all_distances = {}
    for row in possible_neighbors:
        all_distances[euclid_dist(current_vector, row)] = row
    sorted_dists = dict(sorted(all_distances.items(), key=lambda item: item[0]))
    keys = list(sorted_dists.values())
    return keys[:max_neighbors]

def predict(current_vector, possible_neighbors, max_neighbors):
    '''
    Calls find_neighbors and then slices the class labels out
    Finds the maximum of the class counts and returns the label for that class
    Params:
    current_vector - the row we are checking
    possible_neighbors - the set of possible rows to check against
    max_neighbors - the maximum number of neighbors per cluster
    '''
    current_neighbors = find_neighbors(current_vector, possible_neighbors, max_neighbors)
    classes = [row[1:2] for row in current_neighbors]
    return max(classes, key=classes.count)

def accuracy(actual, predicted):
    '''
    Computes the accuracy of the given model
    Params:
    actual - list of actual values
    predicted - list of predicted values
    Returns the percent of matching values
    '''
    total = 0
    for i in range(len(predicted)):
        if predicted[i] == actual[i]:
            total+=1
        #else:
            #print('Predicted' + str(predicted[i]))
            #print('Actual' + str(actual[i]))

    return (float(total)/float(len(predicted))) * 100.0


'''
Naive Bayes Implementation

'''
```

```
def statistics(training_data):
    '''
    Computes the length, mean, and standard deviation of each class in the training data
    Params:
    training_data - the data we wish to analyze
    Returns a dictionary with the class name and it's information in a list
    '''
    class_stats = {}
    df_class_type = training_data.groupby('Category')
    # iris dataset only has 3 classes
    classes = ['0=Blood Donor','0s=suspect Blood Donor','1=Hepatitis','2=Fibrosis','3=Cirrhosis']
    for c in classes:
        # sort dataframe by class type
        current_class_data = df_class_type.get_group(c)
        # drop type column for statistics
        current_class_data = current_class_data.drop(['Category','Unnamed: 0','Sex'],1)
        # store the length of the class data for later use
        col_stats = [len(current_class_data)]
        # loop through each attribute and find the mean and standard deviation
        for (col_name, col_data) in current_class_data.iteritems():
            col_data = col_data.to_numpy()
            col_data = col_data.astype(float)
            mean = np.mean(col_data)
            std_dev = np.std(col_data)
            col_stats.append([mean,std_dev])
        class_stats[str(c)] = col_stats
    return class_stats

def pdf(x, mean, std_dev):
    '''
    Computes the relative likelihood that another value
    would be close to the given value
    This helps in finding what class a given sample would fall into
    We are using normal distribution for this classifier
    Params:
    x -  the value we want to check the relative likelihood of
    mean - mean of the class we want to check against
    std_dev - standard deviation of the class we want to check against
    Returns the likelihood value
    '''
    y = math.exp(-((x-mean)**2 / (2*std_dev**2)))
    return (1/ (math.sqrt(2*math.pi)*std_dev )) * y

def compute_probability(current_row, class_statistics):
    '''
    Computes the class probabilities for the current row
    Returns a dictionary of the probabilities for each class labeled with the class label
    Params:
    current_row - the row to check 'closeness' of
    class_statistics - the dictionary holding the number of instances per class and the
        mean and standard deviation of each column of the class training dataset
    '''
    class_probabilties = {}
    # gets the class label and then values holds the list of [length, [mean, std], [mean,std]...]
    for class_label, values in class_statistics.items():
        # this computes the P(class_label) and stores it as the initial value in the dictionary
        class_probabilties[class_label] = float(values[0])/total_rows
        # now find the pdf of each value in the current row using the mean and standard deviation
            of each column
```

```
        for i in range(2, len(values)):
            # 1 skips the length stored in the first position
            if i != 3:
                mean, std_dev = values[i]
                class_probabilties[class_label] = class_probabilties[class_label] *
                    pdf(float(current_row[i]), mean, std_dev)
    return class_probabilties

def naive_prediction(current_row, class_statistics):
    '''
    Finds the probabilites per class and find the maximum one, this becomes the class label
    Params:
    current_row - the row to check 'closeness' of
    class_statistics - the dictionary holding the number of instances per class and the
        mean and standard deviation of each column of the class training dataset
    Returns the predicted class label
    '''
    probabilities = compute_probability(current_row, class_statistics)
    class_label = max(probabilities, key=probabilities.get)
    return class_label


'''
Get the data and clean up any missing values
'''
data_frame = pd.read_csv('hcvdat0.csv', keep_default_na=False)

# remove rows with missing values
data_frame = clean_data(data_frame)

# converts all males to category 1 and females to category 0
data_frame['Sex'] = data_frame['Sex'].astype('category').cat.codes

'''
Summary Statistics:
    used to understand the data well and get an overall
    picture of what we expect the data to look like

Mean and Standard Deviation are very far for each column
    Approach: Normalize all data in the dataset columns
    that are not category, name, and sex
'''
means = mean(data_frame)
stddevs = standard_dev(data_frame)
print('Unnormed means and standard deviation')
print(means)
print(stddevs)
norm_dataframe = normalize(data_frame)
norm_means = mean(norm_dataframe)
norm_std = standard_dev(norm_dataframe)
print('Normalized means and standard deviation')
print(norm_means)
print(norm_std)

df_train = norm_dataframe.sample(frac=1/3)
df_test = norm_dataframe[~(norm_dataframe.index.isin(df_train.index))]

'''
Now all data is inbetween [0,1]
```

```
Compute correlation between all numeric columns
'''
column_names = ['Age', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT']
cores = []
for i in range(0, len(column_names)):
    col_i = df_train[column_names[i]]
    for j in range(0, i):
        col_j = df_train[column_names[j]]
        current_cor = correlation(col_i,col_j)
        cores.append(current_cor)
        if current_cor< -0.5 or current_cor >0.5:
            print(''+str(column_names[i])+', '+ str(column_names[i])+ ': ' +str(current_cor))

'''
All correlation coefficients are around zero which means we are safe to use
KNN and Naive Bayes. No set of columns is correlated with each other.

'''


'''
K-Nearest Neighbors
'''

knn_df_train = df_train[:].values
knn_df_test = df_test[:].values

# loop through the number of neighbors and the rows
k_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for k in k_list:
    actual_labels = []
    predicted_labels = []
    start_time = time.time()
    for row in knn_df_train:
        actual_labels.append(row[1])
        output = predict(row, knn_df_test, k)
        predicted_labels.append(output)
    print('The accuracy for k='+ str(k)+ ' is: ' + str(accuracy(actual_labels,predicted_labels)))
    print("Ran for %s seconds" % (time.time() - start_time))


'''
Naive Bayes
'''
total_rows = len(df_train)
class_stats = statistics(df_train)
df_test = df_test[:].values
# empty lists for accuracy checking
naive_predictions = []
naive_actual = []
# loop through each data instance
for row in df_test:
    c = naive_prediction(row, class_stats)
    naive_predictions.append(c)
    naive_actual.append(row[1])
# output results
print('Naive Bayes classifier has accuracy: ' + str(accuracy(naive_actual, naive_predictions)))
```

References

[CDC, 2020] CDC (2020). Hepatitis c.

[MedlinePlus, 2020] MedlinePlus (2020). Medical tests.

[Ramachandran, 2014] Ramachandran, J. (2014). Serum cholinesterase is an excellent biomarker of liver cirrhosis.