“REPORT ON INDIAN LIVER PATIENT DATASET ANALYSIS”

STUDENT NAME: **KRISHNAN MONI**

REGISTRATION NUMBER: **201788865**

COURSE: **MSc DATA ANALYTICS**

SUBJECT CODE AND NAME:

CS986 FUNDAMENTALS OF MACHINE LEARNING FOR DATA ANALYTICS

FACULTY NAME: **DR. MARC ROPER**

DATE SUBMITTED: **20/02/2018**

PAGE COUNT: **13 (INCLUDING COVER PAGE AND APPENDIX)**

WORD COUNT: **3250**

SCHOOL OF COMPUTING AND INFORMATION SCIENCES

Selecting a dataset and identifying the problem

The dataset I chose to analyze was the “Indian Liver Patient Dataset” (Ramana B.V., et. al., May 2012). I downloaded this dataset from the UCI ML Repository. The reason why I chose this dataset was more for a personal reason, because “Liver Disease” is something that I can relate very much to some of my friends and relatives back home in India. When I realized that this dataset came from a town near mine, it shook me. When I went through the dataset, I felt quite interested in finding out more about how a person gets classified as a “liver patient”.

There are two questions (problems) I wish to investigate and answer by analyzing this dataset.

1. Which algorithm provides the best prediction (performance) of whether a person is a patient or not. I will be making a comparison between algorithms such as Support Vector Machines (SVM), Random Forest, Decision Tree, k-Nearest Neighbors Classifier and Gaussian Naïve Bayes.
2. Are there any features in the dataset that strongly determine the classification of a person having liver disease?

I also just want to add a note here that this dataset has been analyzed by me in the past (once), but the cleaning and visualizing approach I used earlier were different. For e.g., I filled in null values with “0” last time, but this time I filled it with the mean value of the column. I also did not do feature selection last time, but I am doing it this time. In terms of visualization, I did not use scatter plots or factor plots last time to visualize the independent variables determining the disease presence, but this time I am doing it. Even some of the algorithms I intend to use this time was not used last time, such as Random Forest Classifier and Decision Tree. I am using these this time. Also, I used normalization, one hot encoding, transformation and scaling techniques in my previous attempt and ended up with very low prediction accuracy (this could be due to the prediction algorithms I chose too). However, for demonstration, I have added screenshots (see Appendix) of my previous implementation using the above techniques to show that I am familiar with coding these techniques on the dataset. So, basically, I am trying to better my own work from last time, to see if I can improve the highest prediction accuracy I obtained last time of ~ 57%.

Summarizing and visualizing the dataset

This dataset is a pure classification problem, i.e., determining whether a person has liver disease or not. The data comes from a region in a South Indian state called Andhra Pradesh. It has 583 rows of data, and 11 attributes, such as the Liver Function Test counts, like Total Proteins, Albumin, Total Bilirubin, Direct Bilirubin, Alkaline Phosphatase, Alanine Aminotransferase, Aspertate Aminotransferase, ratio between Albumin and Globulins. Apart from these, there are attributes like age, gender and a classification label “Dataset” as (1) or (2), which differentiates a person from being a patient and not, respectively. Our main interest is in the classification label attribute. Rest all are features. There is data about 441 male and 142 female patients. As per the classification, there are 416 persons who are patients and 167 who are not. If a patient is more than 89 years old, it has been listed as ‘90’.

After loading the libraries such as Pandas, Numpy, Matplotlib, etc, I read in the dataset and tried to understand the structure of the data. For this, I first output the head (first few rows) as shown in Figure 1. Python is a zero-base index language (counts row number from 0, instead of 1).

```
In [6]: #Retrieving first few rows of the dataset
        disease_df.head()
```

Out[6]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumi
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	

Figure 1: First few rows of the dataset.

Next, I wanted to check if the dataset is complete. Figure 2 shows there are four missing (null) values in “Albumin and Globulin Ratio”. Also, gender is the only object data type. Rest all are numeric (integer, float) data types.

```
In [8]: #Finding out if there are any null values in the dataset
        disease_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
Age                           583 non-null int64
Gender                        583 non-null object
Total_Bilirubin               583 non-null float64
Direct_Bilirubin              583 non-null float64
Alkaline_Phosphotase          583 non-null int64
Alamine_Aminotransferase      583 non-null int64
Aspartate_Aminotransferase    583 non-null int64
Total_Protiens                583 non-null float64
Albumin                       583 non-null float64
Albumin_and_Globulin_Ratio    579 non-null float64
Dataset                       583 non-null int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

Figure 2: Finding out if the dataset is complete

The summary statistics is shown in Figure 3. From this, we can see the number of rows of data, the mean, standard deviation, the minimum and maximum values, the first, second and third quartile values for each feature.

```
In [10]: #Outputting summary statistics
         disease_df.describe()
```

Out[10]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Al
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852	
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519	
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000	
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000	
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	

Figure 3: Summary statistics

Figure 4 shows the column headers and number of null values in each column. We have seen evidence of the presence of null values in “Albumin and Globulin Ratio” earlier through a different code. Below is just another way of showing the same.

I will fix these null values in a later section.

```
In [12]: disease_df.columns

Out[12]: Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
                'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
                'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
                'Albumin_and_Globulin_Ratio', 'Dataset'],
               dtype='object')

In [14]: disease_df.isnull().sum()

Out[14]: Age                           0
         Gender                        0
         Total_Bilirubin               0
         Direct_Bilirubin              0
         Alkaline_Phosphotase          0
         Alamine_Aminotransferase      0
         Aspartate_Aminotransferase    0
         Total_Protiens                0
         Albumin                       0
         Albumin_and_Globulin_Ratio    4
         Dataset                       0
         dtype: int64
```

Figure 4: Column headers and number of null values in each column

Now, let us plot a couple of graphs to visualize the data. Figure 5 shows a bar plot of the number of persons with liver disease and no liver disease and prints the count of the same. Figure 6 similarly shows the representation of the number of males and females in the dataset.

```
In [15]: sns.countplot(data=disease_df, x="Dataset", label="Count")
         Patient, NotPatient = disease_df["Dataset"].value_counts()
         print("Number of patients with liver disease: ", Patient)
         print("Number of patients with no liver disease: ", NotPatient)

Number of patients with liver disease:  416
Number of patients with no liver disease:  167
```

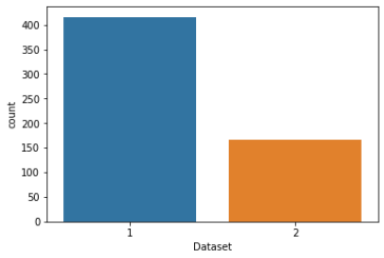

Figure 5: Liver disease patient count

```
In [16]: sns.countplot(data=disease_df, x="Gender", label="Count")
         Male, Female = disease_df["Gender"].value_counts()
         print("Number of male patients: ", Male)
         print("Number of female patients: ", Female)

Number of male patients:  441
Number of female patients:  142
```

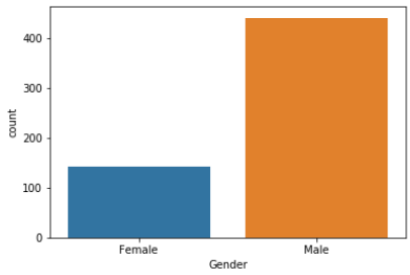

Figure 6: Gender count representation

Figure 7 shows a tabular representation of people with and without liver disease by age and gender.

```
In [19]: disease_df[["Gender", "Dataset", "Age"]].groupby(["Dataset", "Gender"], as_index=False).count().sort_values(by="Dataset", ascendi
```

Out[19]:

	Dataset	Gender	Age
0	1	Female	92
1	1	Male	324
2	2	Female	50
3	2	Male	117

Figure 7: Table showing count of people with and without liver disease by age and gender

Figure 8 shows Figure 7 in terms of mean age. This shows that mean age of people who have liver disease is between 43 and 47, whereas mean age of people who do not have liver disease is between 40 and 43.

```
In [20]: disease_df[["Gender", "Dataset", "Age"]].groupby(["Dataset", "Gender"], as_index=False).mean().sort_values(by="Dataset", ascendin
```

Out[20]:

	Dataset	Gender	Age
0	1	Female	43.347826
1	1	Male	46.950617
2	2	Female	42.740000
3	2	Male	40.598291

Figure 8: Figure 7 represented in terms of mean age

Figure 9 shows a histogram of the age distribution of patient count by gender for each classification (liver disease (1); no liver disease (2)).

```
In [23]: gen = sns.FacetGrid(disease_df, col="Dataset", row="Gender", margin_titles=True)
         gen.map(plt.hist,"Age", color="green", alpha=1, bins=10, ec="black")
         plt.subplots_adjust(top=0.9)
         gen.fig.suptitle("Disease by Gender and Age")

Out[23]: <matplotlib.text.Text at 0x1b5dc7b1898>
```

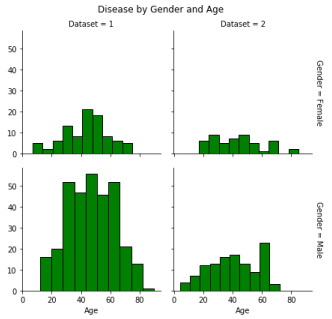

Figure 9: Histogram of age distribution of patient count by gender

Preparing dataset for analysis

To prepare the dataset for analysis, the first thing I had to do was to convert the categorical variable (Gender) to an indicator variable (binary). Figure 10 shows this.

```
In [28]: #Converting "Gender" categorical variable into indicator variable
         pd.get_dummies(disease_df['Gender'], prefix = 'Gender').head()
```

Out[28]:

	Gender_Female	Gender_Male
0	1	0
1	0	1
2	0	1
3	0	1
4	0	1

Figure 10: Converting Gender into indicator variable

Then I had to concatenate (link) the converted Gender variable as binary values into the dataset for further use. Figure 11 shows this. Once this is done, I dropped the original “Gender” column and “Dataset” column as we want to use only the features that determine liver disease.

```
In [36]: disease_df = pd.concat([disease_df, pd.get_dummies(disease_df['Gender'], prefix = "Gender")], axis=1)
         disease_df.head()
```

Out[36]:

ne_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	...	Gender_Female	Gender_Male
187	16	18	6.8	3.3	0.90	...	1	0
699	64	100	7.5	3.2	0.74	...	0	1
490	60	68	7.0	3.3	0.89	...	0	1
182	14	20	6.8	3.4	1.00	...	0	1
195	27	59	7.3	2.4	0.40	...	0	1

Figure 11: Concatenation of Gender as indicator variable into the dataset

Next is fixing the null values in “Albumin and Globulin Ratio”. First, we need to find out which rows these null values are located in. Figure 12 shows the row numbers where there are null values for this feature.

```
In [37]: disease_df[disease_df['Albumin_and_Globulin_Ratio'].isnull()]
```

Out[37]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamir
209	45	Female	0.9	0.3	189	
241	51	Male	0.8	0.2	230	
253	35	Female	0.6	0.2	180	
312	27	Male	1.3	0.6	106	

Figure 12: Row number of the location of null values

There are many ways of fixing null values. One way is to take the mean of the column for this feature and fill the null values with this mean. Figure 13 shows this. Figure 14 checks if the null values have been filled.

```
In [39]: #Filling the null values with the mean value of the column
         disease_df["Albumin_and_Globulin_Ratio"] = disease_df.Albumin_and_Globulin_Ratio.fillna(disease_df['Albumin_and_Globulin_Ratio'].
```

Figure 13: Filling in null values with mean of the column

```
In [46]: #Checking if the null values have been filled
         disease_df.loc[disease_df['Albumin_and_Globulin_Ratio'].isnull()]

Out[46]:
```

sferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	.

Figure 14: Checking if the null values are filled

Figure 15 shows the correlation between each variable. Higher the correlation value, the stronger its relationship. Similarly, lower the correlation value, the weaker its relation.

```
In [50]: disease_corr = x.corr()
         disease_corr
```

Out[50]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Prot
Age	1.000000	0.011763	0.007529	0.080425	-0.086883	-0.019910	-0.18
Total_Bilirubin	0.011763	1.000000	0.874618	0.206669	0.214065	0.237831	-0.00
Direct_Bilirubin	0.007529	0.874618	1.000000	0.234939	0.233894	0.257544	-0.00
Alkaline_Phosphotase	0.080425	0.206669	0.234939	1.000000	0.125680	0.167196	-0.02
Alamine_Aminotransferase	-0.086883	0.214065	0.233894	0.125680	1.000000	0.791966	-0.04
Aspartate_Aminotransferase	-0.019910	0.237831	0.257544	0.167196	0.791966	1.000000	-0.02
Total_Protiens	-0.187461	-0.008099	-0.000139	-0.028514	-0.042518	-0.025645	1.00
Albumin	-0.265924	-0.222250	-0.228531	-0.165453	-0.029742	-0.085290	0.78
Albumin_and_Globulin_Ratio	-0.216089	-0.206159	-0.200004	-0.233960	-0.002374	-0.070024	0.23
Gender_Female	-0.056560	-0.089291	-0.100436	0.027496	-0.082332	-0.080336	0.08
Gender_Male	0.056560	0.089291	0.100436	-0.027496	0.082332	0.080336	-0.08

Figure 15: Correlation between each variable

Let us see a diagrammatic representation of this. Figure 16 shows the correlation between each of the features of the dataset. As per the color bar, where there are dark bluish squares, those features are highly correlated, i.e., especially pairs of Total Proteins and Albumin; Alamine Aminotransferase and Aspartate Aminotransferase; Direct Bilirubin and Total Bilirubin and some significant correlation between Albumin and Globulin Ratio and Albumin. Figure 17 is a pair plot, which is another way of observing correlations between features.

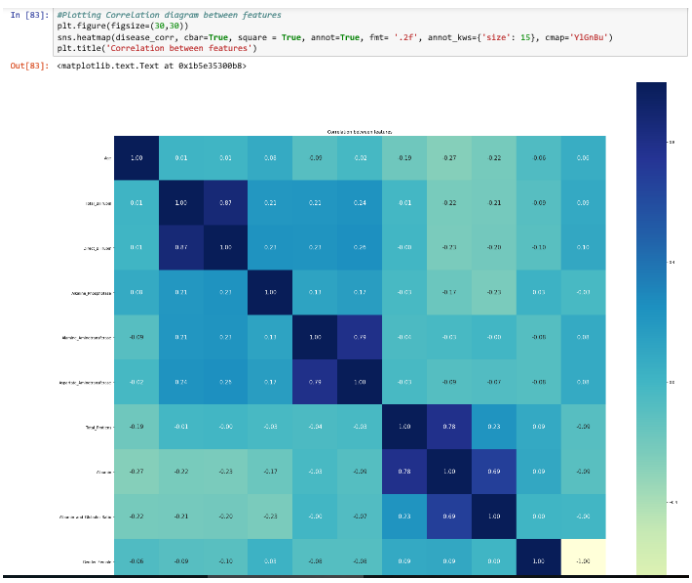

Figure 16: Correlation diagram between features

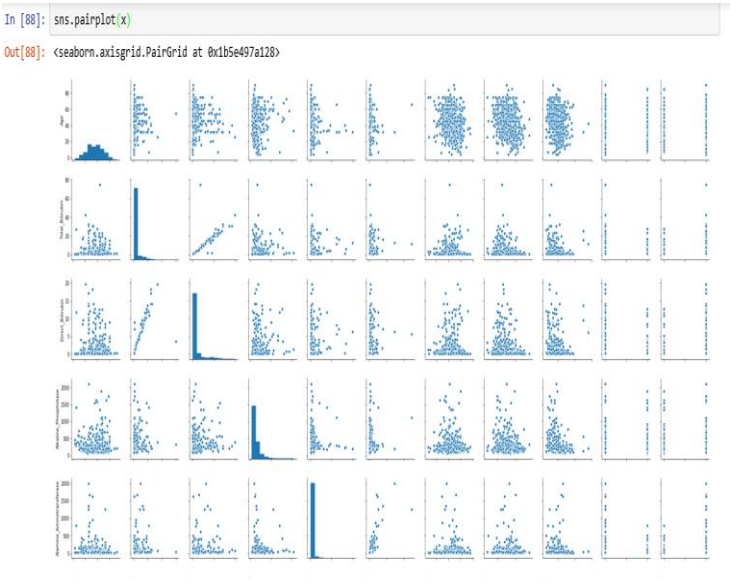

Figure 17: Pair plot between features

Figure 18 shows the list of columns we are going to be analyzing after the above cleansing.

```
In [85]: x.columns

Out[85]: Index(['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
       'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
       'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio',
       'Gender_Female', 'Gender_Male'],
      dtype='object')
```

Figure 18: List of columns for analysis

Figure 19 shows a factor plot of Age for the categorical variable Gender. It looks like age is a factor for liver disease for both male and female and at around 43 years of age, people of both genders are prone to get liver disease. So, we must consider “Age” as a feature for analysis and classifying a patient having liver disease.

Figure 19: Factor plot of Age VS Gender

From Figure 20, there is a positive relationship between Total Bilirubin and Direct Bilirubin. So, we can remove one of these features. Figure 21 is a regression join plot that shows evidence of linear regression between these two features.

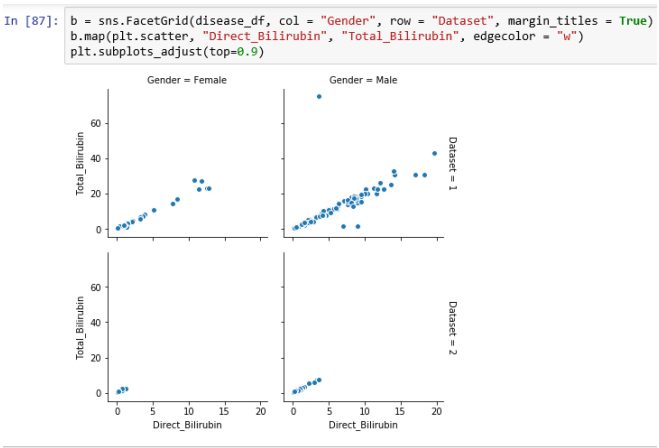

Figure 20: Scatterplot of Total Bilirubin and Direct Bilirubin for gender

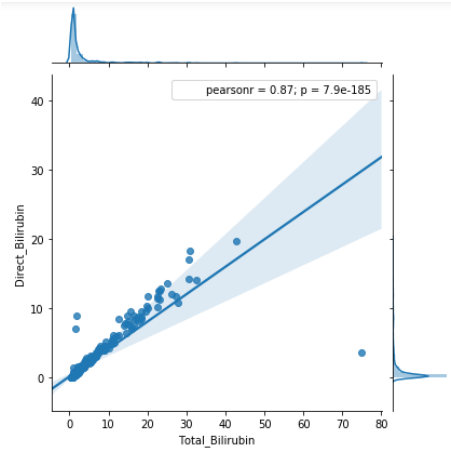

Figure 21: Regression joint plot of Figure 20

On the other hand, Figure 22 and 23 shows that there is no linear relationship or correlation between Alkaline Phosphotase and Alamine Aminotransferase. The Pearson correlation factor = 0.13 here, whereas it is = 0.87 in the above case.

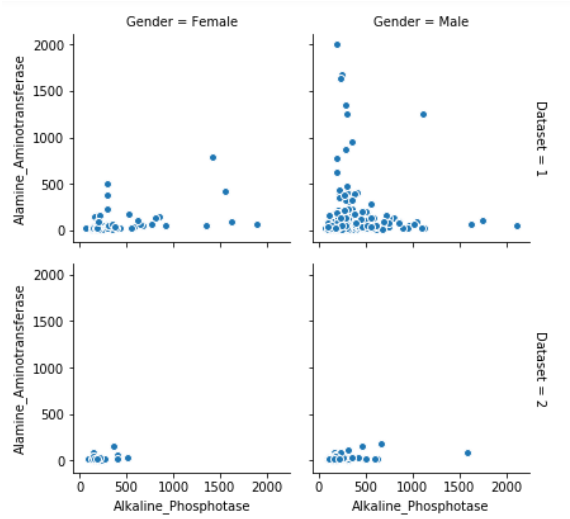

Figure 22: Scatterplot of another two features

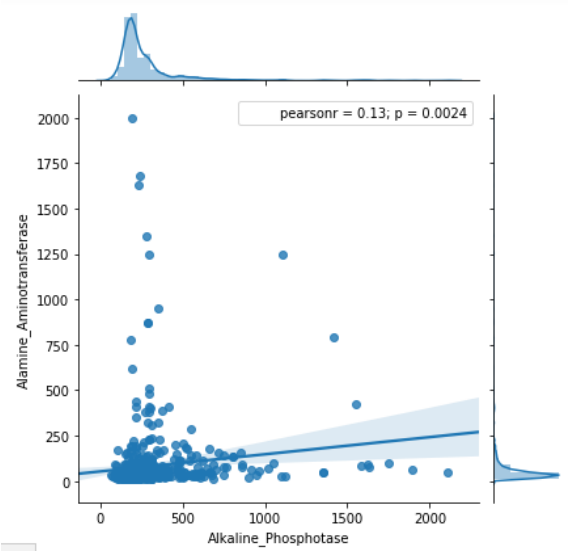

Figure 23: Regression joint plot of Figure 22

Figures 24 and 25 shows that Total Proteins and Albumin are highly correlated. So, we can remove one of these features.

```
In [94]: b = sns.FacetGrid(disease_df, col = "Gender", row = "Dataset", margin_titles = True)
         b.map(plt.scatter, "Total_Protiens", "Albumin", edgecolor = "w")
         plt.subplots_adjust(top=0.9)
```

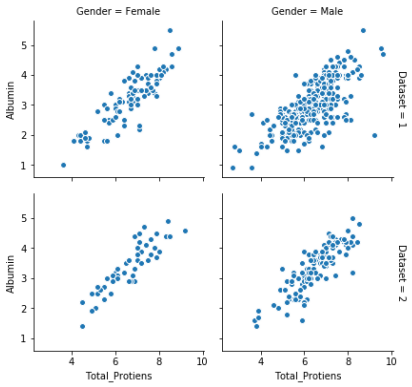

Figure 24: Scatterplot of another two features

```
In [95]: sns.jointplot("Total_Protiens", "Albumin", data=disease_df, kind="reg")
Out[95]: <seaborn.axisgrid.JointGrid at 0x1b5ee4a6d68>
```

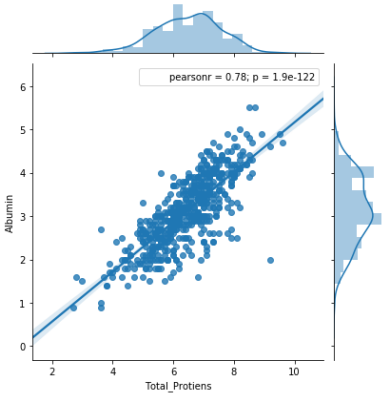

Figure 25: Regression plot of Figure 24

This way, I performed feature selection by plotting scatter plots and joint plots of various sets of features. From this, I found the set of highly correlated features to be, “Direct Bilirubin” and “Total Bilirubin”; “Aspartate Aminotransferase” & “Alamine Aminotransferase”; “Total Proteins” and “Albumin”; and “Albumin and Globulin Ration” and “Albumin”. This matches the same set of highly correlated features from the earlier correlation plot and pair plot. After this, I omitted one of the features and kept “Direct Bilirubin”, “Aspartate Aminotransferase”, “Total Proteins”, “Albumin” and “Albumin and Globulin Ration”.

Choice of models, application, evaluation and validation

I implemented five types of Machine Learning algorithms on the dataset to compare which provides the best prediction to classify if a patient has liver disease or not. The algorithms I used are namely, Support Vector Machine, Random Forest Classifier, KNeighbors Classifier, Gaussian Naïve Bayes (GNB) and Decision Tree Classifier. Except for GNB, all other algorithms are relevant to the dataset I chose as these are mostly associated with Classification type of problems where GNB is more for regression type of problems. I still wanted to test if there is any improvement in prediction accuracy.

First, I imported all the required ML packages for each of the algorithms and techniques as shown in Figure 26. Then I defined the split of my training data and testing data in 80-20 ratio (80% training data and 20% testing data) as Figure 27 shows.

```
In [120]: #Machine Learning part starts here
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

Figure 26: Importing all required ML packages

```
In [102]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.20, random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(466, 11)
(466,)
(117, 11)
(117,)
```

Figure 27: Splitting training and testing data

First, I implemented GNB. Figure 28 shows the coding and Figure 29 shows the results.

```
In [105]: #Gaussian Naive Bayes
gauss = GaussianNB()
gauss.fit(X_train, y_train)
gauss_predict = gauss.predict(X_test)
gaussian_score = round(gauss.score(X_train, y_train) * 100,2)
gauss_test = round(gauss.score(X_test, y_test) * 100,2)
```

```
In [110]: print("Gaussian Train Score: ", gaussian_score)
print("Gaussian Test Score: ", gauss_test)
print("Accuracy: \n", accuracy_score(y_test, gauss_predict))
print(confusion_matrix(y_test, gauss_predict))
print(classification_report(y_test, gauss_predict))
sns.heatmap(confusion_matrix(y_test, gauss_predict), annot=True, fmt= "d")
```

Figure 28: GNB algorithm coding

```
Gaussian Train Score:  55.36
Gaussian Test Score:  52.99
Accuracy:
 0.5299145299145
[[32 55]
 [ 0 30]]
              precision    recall  f1-score   support

           1       1.00      0.37      0.54        87
           2       0.35      1.00      0.52        30

   avg / total       0.83      0.53      0.53       117

Out[110]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f04cd400>
```

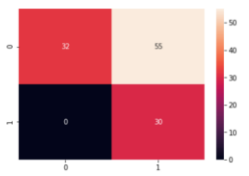

Figure 29: Results of GNB implementation

The accuracy of prediction during training and testing stages are quite less, at only 55.36% and 52.99%, respectively. I used a confusion matrix to look at the results more in depth. It shows that, out of a total of 117 instances, it made a correct prediction only 62 times, i.e., (True Positive + True Negative)/Total = Accuracy = 52.99%. The rest of the 55 times, it made an incorrect prediction, i.e., misclassification rate (error rate) = (False Positive + False Negative)/Total = 1 – Accuracy = 47.01% it predicts incorrectly. We can also see a “Classification Report”, where the testing dataset gets split randomly into two sets (Support) and calculations for Precision = True Positive / (True Positive + False Positive); Recall = True Positive / (True Positive + False Negative); F1-Score = Weighted mean of Precision and Recall have been done. Interestingly, the Precision and Recall scores are 100% alternatively in each split, but the average is only 83% and 53%, respectively. The F1-score is also only around 53% owing to 35% and 37% alternate Precision and Recall scores in each split. This generally means, if a patient does not have liver disease, it predicts that the person has the disease, which can be dangerous in the real world. So, this is not a good choice of an algorithm for prediction.

Next, I implemented SVM (Support Vector Machine) algorithm as shown below in Figure 30 and the results are shown in Figure 30. From this, it is evident that training score is 99.47%, which is very good, but the testing score is only 74.36%. Well, this algorithm performs much better than GNB in terms of accuracy. From Figure 31, I am getting a strange looking “Confusion Matrix” with ‘0’s on the False Positive and True Positive side and oddly I am also getting “zero” for “Recall”, “Precision” and “F1-split” in the second split of data for 30 samples in the classification report, and an associated warning message. I will investigate this at a later stage. Nevertheless, the bottom line is that the algorithm performs better than GNB.

```
In [113]: #Support Vector Machine
svc = SVC()
svc.fit(X_train, y_train)
svc_prediction = svc.predict(X_test)
svc_train = round(svc.score(X_train, y_train) * 100, 2)
svc_test = round(svc.score(X_test, y_test) * 100, 2)
print("SVM Train Score: ", svc_train)
print("SVM Test Score: ", svc_test)
print("Accuracy: \n", accuracy_score(y_test, svc_prediction))
print(confusion_matrix(y_test, svc_prediction))
print(classification_report(y_test, svc_prediction))
```

Figure 30: SVM implementation coding

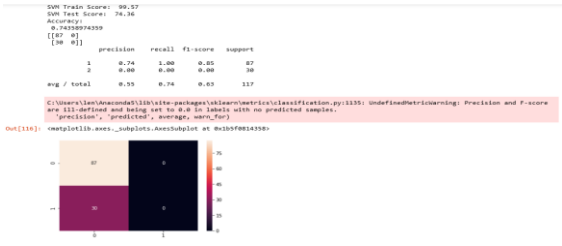

Figure 31: Results of SVM implementation

Next, I implemented k-Nearest Neighbors algorithm, as shown in Figure 32 and results for which are shown in Figure 33. It had an 83.05% accuracy for the training set but only around 68.38% accuracy in the testing phase. So, this has performed better than GNB but slightly worse than SVM. Interestingly, it has a very high “Specificity” (True Negative), where 70 samples

were predicted correctly of not having the disease, but its “Sensitivity” or “Recall” (True Positive) size is only 17, which still does not make it completely reliable.

```
In [119]: #KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
knn_predict = knn.predict(X_test)
knn_train = round(knn.score(X_train, y_train) * 100, 2)
knn_test = round(knn.score(X_test, y_test) * 100, 2)
print("KNN Train Score: ", knn_train)
print("KNN Test Score: ", knn_test)
print("Accuracy: ", accuracy_score(y_test, knn_predict))
print(confusion_matrix(y_test, knn_predict))
print(classification_report(y_test, knn_predict))
sns.heatmap(confusion_matrix(y_test, knn_predict), annot=True, fmt= "d")
```

Figure 32: KNN implementation coding

```
KNN Train Score:  83.05
KNN Test Score:  68.38
Accuracy:  0.683760683761
[[70 17]
 [20 10]]
              precision    recall  f1-score   support

           1       0.78      0.80      0.79        87
           2       0.37      0.33      0.35        30

avg / total       0.67      0.68      0.68       117

Out[119]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f0a10a20>
```

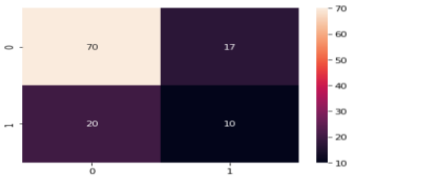

Figure 33: Results of KNN implementation

Then, I implemented Decision Tree algorithm. Figure 34 and 35 show the implementation and results, respectively. Well, surprisingly, we have 100% training score (it cannot get better than that!), but only 71.79% testing score. Just like KNN, it had a higher sample count in Specificity, rather than Recall. The Precision, Recall and F1-score were quite high in the first split, but dipped in the second split quite heavily. Something similar happened with KNN, SVM and GNB as well. What could be the cause of getting such great variation in the accuracies between two splits of data coming from the same source? It would be interesting to investigate why this is happening in further studies.

```
In [129]: #Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
dectree_predict = decision_tree.predict(X_test)
dectree_train = round(decision_tree.score(X_train, y_train)* 100, 2)
dectree_test = round(decision_tree.score(X_test, y_test)* 100, 2)
print('Decision Tree Train Score: ', dectree_train)
print('Decision Tree Test Score: ', dectree_test)
print('Accuracy: ', accuracy_score(y_test,dectree_predict))
print(confusion_matrix(y_test,dectree_predict))
print(classification_report(y_test,dectree_predict))
sns.heatmap(confusion_matrix(y_test, dectree_predict), annot=True, fmt= "d")
```

Figure 34: Decision Tree implementation coding

```
Decision Tree Train Score:  100.0
Decision Tree Test Score:  71.79
Accuracy:  0.717948717949
[[66 21]
 [12 18]]
              precision    recall  f1-score   support

           1       0.85      0.76      0.80        87
           2       0.46      0.60      0.52        30

avg / total       0.75      0.72      0.73       117

Out[129]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f0bf7748>
```

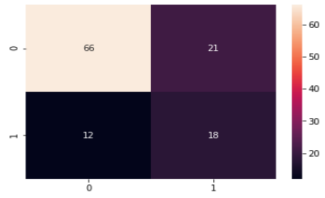

Figure 35: Results of Decision Tree implementation

Finally, I implemented RandomForestClassifier algorithm as shown in Figure 36. Results are shown in Figure 37. I got another 100% accuracy for training set, but in the testing set, got only 71.79%. Again, the Specificity was higher than Recall. The Precision, Recall and F1-Score were dropping in the second split, from the first split.

```
In [131]: # Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
randforest_predict = random_forest.predict(X_test)
randforest_train = round(random_forest.score(X_train, y_train) * 100, 2)
randforest_test = round(random_forest.score(X_test, y_test) * 100, 2)
print('Random Forest Train Score: ', randforest_train)
print('Random Forest Test Score: ', randforest_test)
print('Accuracy: ', accuracy_score(y_test,randforest_predict))
print(confusion_matrix(y_test,randforest_predict))
print(classification_report(y_test,randforest_predict))
sns.heatmap(confusion_matrix(y_test, randforest_predict), annot=True, fmt= "d")
```

Figure 36: Random Forest implementation coding

```
Random Forest Train Score:  100.0
Random Forest Test Score:  71.79
Accuracy:  0.717948717949
[[72 15]
 [18 12]]
              precision    recall  f1-score   support

           1       0.80      0.83      0.81        87
           2       0.44      0.40      0.42        30

avg / total       0.71      0.72      0.71       117

Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f0f2fbe0>
```

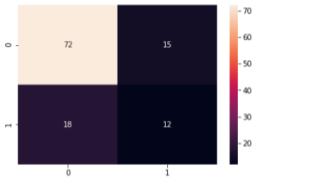

Figure 37: Results of Random Forest implementation

When I did an evaluation of all the models, as shown in Figure 38, and sorted the best testing scores in descending order, it was obvious from the results that Support Vector Machines performed the best with 74.36% accuracy in predicting liver disease. Although, not the best of the results when looking at the training score of 99.57%. Next comes Decision Tree and Random Forest Classifier tied with 71.79% accuracy and with 100% training score. Then comes KNN Classifier with 68.38% accuracy and with 83.05% accuracy. The worst performer in terms of prediction was GNB with only 52.99% accuracy and training score of only 55.36%. As it was mentioned earlier, GNB is an algorithm used for regression problems, so it was expected that it would perform poorly in this case.

```
In [137]: #Model evaluation
models = pd.DataFrame({
    'Model': ['Support Vector Machine', 'KNeighborsClassifier', 'Gaussian Naive Bayes', 'Decision Tree', 'Random Forest'],
    'Train Score': [svc_train, knn_train, gauss_train, dectree_train, randforest_train],
    'Test Score': [svc_test, knn_test, gauss_test, dectree_test, randforest_test]
})
models.sort_values(by='Test Score', ascending=False)
```

Out[137]:

	Model	Test Score	Train Score
0	Support Vector Machine	74.36	99.57
3	Decision Tree	71.79	100.00
4	Random Forest	71.79	100.00
1	KNeighborsClassifier	68.38	83.05
2	Gaussian Naive Bayes	52.99	55.36

Figure 38: Model evaluation

Alternatively, to observe if the accuracy is improving, I tried removing one or two of the features from the selected features in the “Feature Selection” process. However, I could only see that the accuracy level was decreasing and not improving.

Interpretation and explanation of the results and implications

The aim of this project was to predict the presence of liver disease in a set of 117 samples (mix of males and females) after training the model with 468 samples. As this was a second attempt at this dataset to improve prediction accuracy, it can be safely stated that this aim was achieved successfully. I was able to bring out a prediction accuracy of 74.36% from a previous best of 57%. So, there is an ~ 23% improvement in prediction accuracy between the current and last attempt. So, this means there is still scope for enhancement. Maybe it is because of the way I cleaned my data or selected the features of my data, or how I split the training and testing data (80-20 ratio), or even the choice of algorithms. All these could be reasons.

However, looking at it from a real-world perspective, nothing is good short of a 100%, especially when it comes to disease prediction. 74.36% prediction accuracy by SVM algorithm was the best I could achieve this time, which is not a great achievement as far as a patient is concerned. This means that there is still an ~ 25% chance (1 out 4) that the algorithm will make the wrong prediction of disease being present in a patient, when it is not. This can be very dangerous and can cause panic amongst families and patient themselves. This happened to one of my friends, where in a hospital he was predicted with liver disease. He then went to two other hospitals to take a second opinion and reconfirm this. He was surprisingly found to be free of disease. So, we want to avoid this situation. I personally would not want something like this to happen to me if I were in this situation. However, the success of an algorithm depends on various factors surrounding the dataset.

To answer the initially stated questions, as mentioned earlier, Support Vector Machine had the best prediction accuracy of 74.36%, followed by Decision Tree and Random Forest with 71.79%. Surprisingly, these three algorithms had nearly 100% accuracy at training level, but then dipped to the 70s during testing phase. It was also interesting to observe how the split of the testing data resulted in different Precision, Accuracy and F1-scores in each of the algorithms. Attempts to further improve accuracy by removing and adjusting features in the dataset failed as this only decreased the accuracy. GNB had the worst performance at 52.99%, which was expected as this algorithm was relevant only for regression-based algorithms.

The second question that was asked was about the features that determine the presence of the disease. Yes, there are a group of features that may determine the presence of disease as shown by the correlation chart and pair plots. These were “Direct Bilirubin” and “Total Bilirubin”; “Aspartate Aminotransferase” & “Alamine Aminotransferase”; “Total Proteins” and “Albumin”; and “Albumin and Globulin Ration” and “Albumin”. While there is no solid proof that a combination of these features will determine if a person has liver disease, and it would be incorrect to assume so. However, it would be correct to say and assume that one or more combinations of these features are surely an indication of liver disease.

Further improvement to this study to enhance prediction accuracy could be that the dataset needs to be updated and should include more rows of data for analysis. Different splits of training and testing data sets can be tried. There are many other Machine Learning algorithms, which can be implemented on the model. Different features can also be chosen for analysis.

Appendix

Bibliography

1. “Liver Patient Analysis, Prediction and Accuracy”, Kaggle, 2018. https://www.kaggle.com/sanjames/liver-patients-analysis-prediction-accuracy/notebook. Site accessed on Feb 8th, 2018
2. Stack Overflow, 2018. https://stackoverflow.com/. Site accessed several times between Feb 5th and Feb 12th, 2018
3. “Indian Liver Patient Database”, Ramana B.V., et. al., India, May 2012
4. “Analysis of Indian Liver Patient Dataset”, Moni K., Glasgow, Nov 2017

Software environment used

1. Anaconda 3 Jupyter Notebook
2. Python version 3.6.2

Screenshots from previous implementation

```
In [33]: from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()
         cols=list(features_of_data.columns)
         consistent_data=pd.DataFrame(data=features)
         consistent_data[cols]=scaler.fit_transform(features[cols])
         consistent_data.head()
```

Out[33]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Proti
0	1.252098	Female	-0.418878	-0.493964	-0.426715	-0.354665	-0.318393	0.292120
1	1.066637	Male	1.225171	1.430423	1.682629	-0.091599	-0.034333	0.937566
2	1.066637	Male	0.644919	0.931508	0.821588	-0.113522	-0.145186	0.476533
3	0.819356	Male	-0.370523	-0.387054	-0.447314	-0.365626	-0.311465	0.292120
4	1.684839	Male	0.096902	0.183135	-0.393756	-0.294379	-0.176363	0.753153

Figure 39: Scaling of features

```
In [35]: import matplotlib.pyplot as plt
         plt.style.use('ggplot')
         initial_features.hist(figsize=(14,8),alpha=1, histtype='bar', ec='black')
```

```
Out[35]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001E061F4A438>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E061FDAF60>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E062057CF8>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E0620BB748>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0620F8BE0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0620F8940>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E062199358>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E062256278>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0622B1A20>]], dtype=object)
```

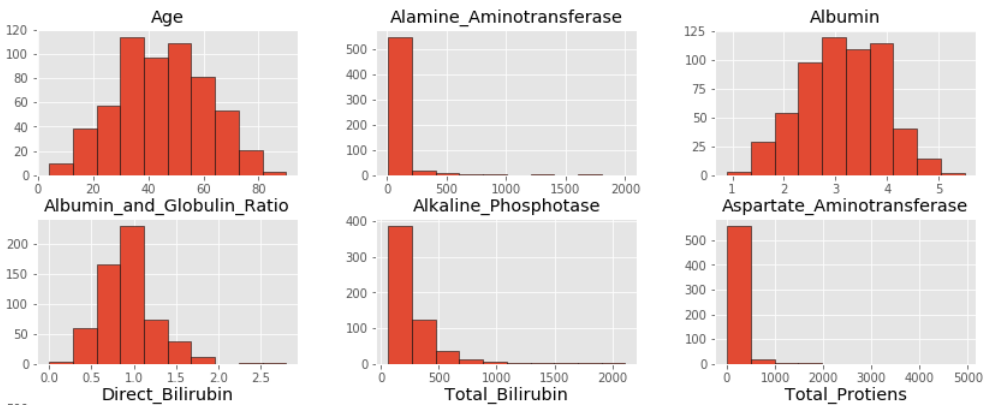

Figure 40: Skewed graphs for some features

```
In [36]: skewed = ['Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Alkaline_Phosphotase', 'Direct_Bilirubin', 'Total_Bilirubi
         n' ]
         initial_features[skewed] = df[skewed].apply(lambda x: np.log(x + 1))

In [37]: initial_features.hist(figsize=(14,8),alpha=1, histtype='bar', ec='black')

Out[37]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001E062031E48>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0628B3278>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0624D5C18>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E062577438>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E0625893C8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E062589438>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E062694EB8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E062A62630>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E062AC1A90>]], dtype=object)
```

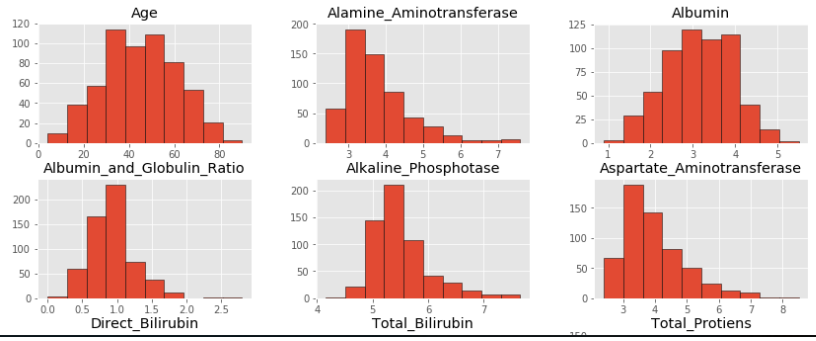

Figure 41: Fixing skewness using Log Transformations

```
In [43]: normalized = ['Age', 'Albumin','Albumin_and_Globulin_Ratio',"Total_Protiens",'Alamine_Aminotransferase', 'Aspartate_Aminotransfe
         rase', 'Alkaline_Phosphotase', 'Direct_Bilirubin', 'Total_Bilirubin' ]
         skewed = ['Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Alkaline_Phosphotase', 'Direct_Bilirubin', 'Total_Bilirubi
         n']

         initial_features[normalized]=scaler.fit_transform(df[normalized])
         display(initial_features.describe())
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protier
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
mean	0.473792	0.038858	0.070720	0.111176	0.035534	0.020311	0.548288
std	0.188254	0.083238	0.143291	0.118680	0.091769	0.058735	0.157312
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.337209	0.005362	0.005102	0.054958	0.006533	0.003049	0.449275
50%	0.476744	0.008043	0.010204	0.070835	0.012563	0.006505	0.565217
75%	0.627907	0.029491	0.061224	0.114802	0.025377	0.015654	0.652174
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Figure 42: Normalizing values for each feature

```
In [97]: features=pd.get_dummies(initial_features)
         ohe=list(features.columns)
         print ("Total features after one-hot encoding.".format(len(ohe)))
         display(features.head(n=1))
         liver_disease=pd.get_dummies(initial_disease)
         ohe=list(liver_disease.columns)
         print ("Disease columns after one-hot encoding.".format(len(ohe)))
         display(liver_disease.head(n=1))

Total features after one-hot encoding.
```

e	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Gender_Female	Gender_Male
	0.003015	0.001626	0.594203	0.521739	0.321429	1	0

```
Disease columns after one-hot encoding.
```

	1	2
0	1	0

Figure 43: Applying One Hot Encoding on Gender categorical variable