Machine Learning Engineer Nanodegree

Using Supervised Learning to predict whether a Patient has a liver disease or not.

ESVK Sri Harsha

27th June, 2018

Report

Definition

Project Overview

The number of patients for Liver disease in the state of Andhra Pradesh, India has been increasing continuously because of many factors. Some of such factors are excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs. In the case that they were admitted in the hospital, it becomes really important to know whether the patient has the liver disease or not. Hence, this dataset was used to evaluate prediction algorithms in an effort to reduce burden on doctors.

A similar thing has been done here for the classification of liver diseases:

Link:
https://www.researchgate.net/publication/319181775_Disease_Classification_Using_Machine_Learning_Algorithms-A_Comparative_Study

This dataset is similar to the one we're using:

Link:
https://www.researchgate.net/publication/309210947_Heart_Disease_prediction_using_Machine_learning_and_Data_Mining_Technique

Dataset that we're going to use: https://www.kaggle.com/uciml/indian-liver-patient-records

---

Problem Statement

Patients with Liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs. This dataset was used to evaluate prediction algorithms in an effort to reduce burden on doctors.

The doctors need us to classify the patients checking if they have the liver disease or not. Hence, this is a classification problem.

Inputs:

Different levels of the chemicals in liver - These can be used to check if the patient has the disease or not.

Age – This is an important factor as people mostly get diseases as they gradually age.

Gender – This is also an important factor.

Output:

Whether the patient has the Liver disease or not.

Metrics

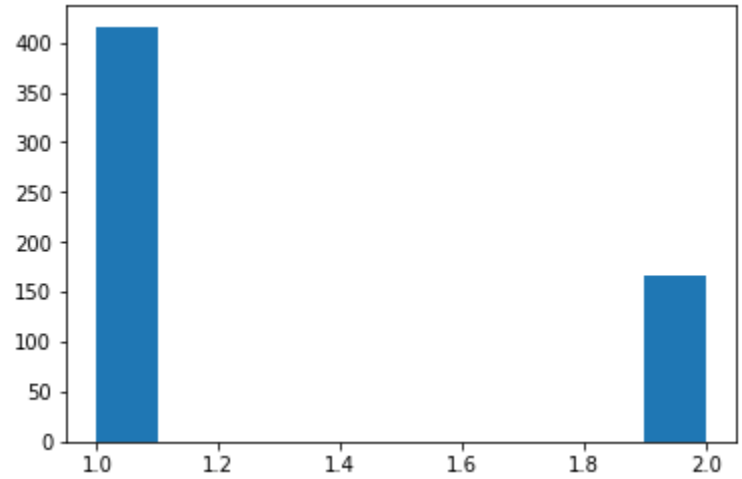
Precision tells us what proportion of messages we classified as spam, actually were spam. It is a ratio of true positives (words classified as spam, and which are actually spam) to all positives (all words classified as spam, irrespective of whether that was the correct classification), in other words it is the ratio of

[True Positives/(True Positives + False Positives)]

Recall (sensitivity) tells us what proportion of messages that actually were spam were classified by us as spam. It is a ratio of true positives (words classified as spam, and which are actually spam) to all the words that were actually spam, in other words it is the ratio of

[True Positives / (True Positives + False Negatives)]

For cases like this dataset, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average (harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score (we take the harmonic mean as we are dealing with ratios).



The above plot shows a graph between number of instances and the labels. We can see that there's a lot of imbalance in the labels. There are more 'No's than 'Yes's in the data. I don't think using Accuracy Score would give us good results. Hence, we'll use F-beta score as a benchmark evaluation metric throughout the project.

Analysis

Data Exploration

I explored the data by using the methods describe() and head(). These gave me a chance to look at how the data is distributed. The dataset has 583 rows and 11 columns. As the feature 'Dataset' in the csv file is the feature that we are trying to predict, I copied it and dropped it from the data.

```
# Load the Liver patients dataset
data = pd.read_csv("indian_liver_patient.csv")

#Copying the 'yes' or 'no' data to labels.
labels = data['Dataset']

#Dropping the feature from the dataset
features_raw = data.drop('Dataset', axis = 1)
```

Then, I used the head() and describe() functions to check the data.

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_
0	62	Male	10.9	5.5	699	64	100	7.5	3.2	
1	62	Male	7.3	4.1	490	60	68	7.0	3.3	
2	58	Male	1.0	0.4	182	14	20	6.8	3.4	
3	72	Male	3.9	2.0	195	27	59	7.3	2.4	
4	46	Male	1.8	0.7	208	19	14	7.6	4.4	
5	26	Female	0.9	0.2	154	16	12	7.0	3.5	
6	29	Female	0.9	0.3	202	14	11	6.7	3.6	
7	17	Male	0.9	0.3	202	22	19	7.4	4.1	
8	55	Male	0.7	0.2	290	53	58	6.8	3.4	
9	57	Male	0.6	0.1	210	51	59	5.9	2.7	

I converted the categorical data of ‘Gender’ into numerical data by using the ‘get_dummies()’ function in the pandas library.

```
features_raw = pd.get_dummies(features_raw)
display(features_raw.head())
```

Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Gender_Female	Gender_Male
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
208	19	14	7.6	4.4	1.30	0	1

After this step, we check for the null values by using the following command:

```
features_raw.isnull().sum()
```

```
Age                          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
Gender_Female                0
Gender_Male                  0
dtype: int64
```

As we can see that there are 4 Null values in the 'Albumin_and_Globulin_Ratio' feature, we fill them up by calculating the mean of the values in 'Albumin_and_Globulin_Ratio' and replacing the NaN with the mean.

```
mean = round(data['Albumin_and_Globulin_Ratio'].mean(), 2)
features_raw = features_raw.fillna(mean)
```

```
features_raw.isnull().sum()
```

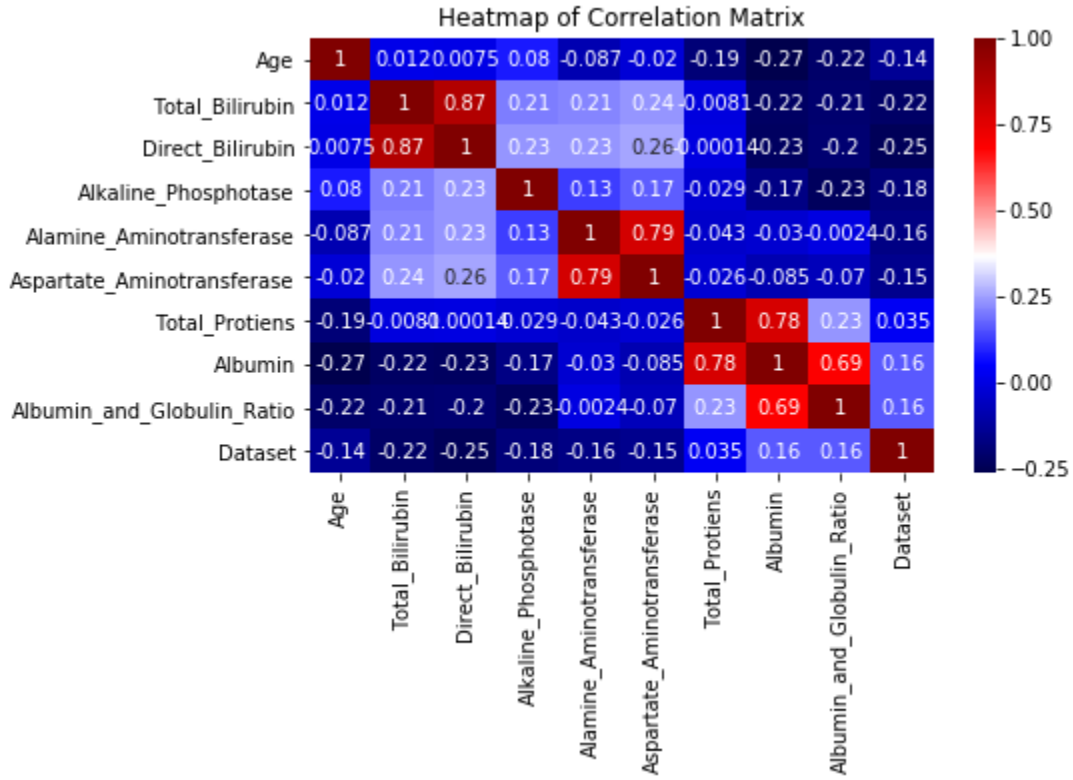
```
Age                          0
Total_Bilirubin              0
Direct_Bilirubin             0
Alkaline_Phosphotase         0
Alamine_Aminotransferase     0
Aspartate_Aminotransferase   0
Total_Protiens               0
Albumin                      0
Albumin_and_Globulin_Ratio   0
Gender_Female                0
Gender_Male                  0
dtype: int64
```

Hence, we have successfully explored, checked and corrected abnormalities in the data.

Exploratory Visualization

We now try to visualize and find any correlations between the features by using matplotlib and seaborn visualization libraries.

```
Text(0.5,1,'Heatmap of Correlation Matrix')
```



The following features seem to be highly correlated:

- Total_Bilirubin - Direct_Bilirubin
- Alamine_Aminotransferase - Aspartate_Aminotransferase
- Albumin - Total_Proteins
- Albumin - Albumin_and_Globulin_Ratio

Hence, this is what I found in the exploratory visualization.

Algorithms and Techniques

I'll first apply the benchmark model by fitting the data with the logistic regression and predicting the values. Now, I'll check with the benchmark metrics to check how my model is working.

Later, I'll apply other models from ensemble methods like Gradient Boosting, AdaBoost, XGBoost etc. and Decision Trees like Decision Tree Classifier and Random Tree Classifier, applied with the evaluation metrics to see how the data is working with different models.

Of these, I'll choose the best model based on the metrics and I plan to further improvise and optimize it by applying the GridSearchCV.

I might add some visualizations to even better the evaluation process.

Benchmark Model

I consider Logistic Regression model as the benchmark model because the data is linear. I choose fbeta score to be the benchmark evaluation metric. I'll further test with other models like ensemble methods and decision trees to see if they are performing better. Also, I'll try using other evaluation metrics like Accuracy score, Confusion matrix etc. to get the best of my model.

Methodology

Data Preprocessing

There's a need to preprocess this data because the results that we get are mostly dependent on the features with higher values. Hence, to treat every feature equally, we normalize the data.

I'll preprocess the data by applying one of the scaling methods. I've tried both MinMaxScaler and StandardScaler, StandardScaler gives me the best results.

Before preprocessing:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_
0	62	Male	10.9	5.5	699	64	100	7.5	3.2	
1	62	Male	7.3	4.1	490	60	68	7.0	3.3	
2	58	Male	1.0	0.4	182	14	20	6.8	3.4	
3	72	Male	3.9	2.0	195	27	59	7.3	2.4	
4	46	Male	1.8	0.7	208	19	14	7.6	4.4	
5	26	Female	0.9	0.2	154	16	12	7.0	3.5	
6	29	Female	0.9	0.3	202	14	11	6.7	3.6	
7	17	Male	0.9	0.3	202	22	19	7.4	4.1	
8	55	Male	0.7	0.2	290	53	58	6.8	3.4	
9	57	Male	0.6	0.1	210	51	59	5.9	2.7	

After preprocessing:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_ar
0	1.066637	1.225171	1.430423	1.682629	-0.091599	-0.034333	0.937566	0.073157	
1	1.066637	0.644919	0.931508	0.821588	-0.113522	-0.145186	0.476533	0.198969	
2	0.819356	-0.370523	-0.387054	-0.447314	-0.365626	-0.311465	0.292120	0.324781	
3	1.684839	0.096902	0.183135	-0.393756	-0.294379	-0.176363	0.753153	-0.933340	
4	0.077514	-0.241578	-0.280143	-0.340199	-0.338224	-0.332250	1.029773	1.582902	

Code Snippet:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = StandardScaler()

features_raw[:]= scaler.fit_transform(features_raw[:])

display(features_raw.head())
```

Implementation

I first checked how the Linear Regression model works on the raw data to see how it works on raw data. I worked fine and gave an Fbeta score of 0.78, which is okay. Then, I normalized the data and checked with the same model. To my surprise, the F-score has slightly decreased. It came out to be 0.779(which is only negligible).

Before Preprocessing:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,fbeta_score

clf = LogisticRegression(random_state=0)
clf = clf.fit(X_train, y_train)

pred = clf.predict(X_test)
fscore = fbeta_score(y_test, pred,beta=0.5)
acc = accuracy_score(y_test, pred)
print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.782258064516
Accuracy:  0.712328767123
```

After Preprocessing:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,fbeta_score

clf = LogisticRegression(random_state=0)
clf = clf.fit(X_train, y_train)
pred = clf.predict(X_test)

fscore = fbeta_score(y_test, pred,beta=0.5)
acc = accuracy_score(y_test, pred)
print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.779220779221
Accuracy:  0.705479452055
```

Refinement

Then, I decided to test with other models like ensemble methods and decision trees.

Testing out with different Supervised Learning models ¶

We try to fit the data into different models to see how it's doing with the other models. If these models have higher values for the metrics, we can consider these for the data.

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score,fbeta_score

clf = GradientBoostingClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
pred = clf.predict(X_test)

fscore = fbeta_score(y_test, pred,beta=0.5)
acc = accuracy_score(y_test, pred)
print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.785472972973
Accuracy:  0.705479452055
```

AdaBoost

```
In [573]: from sklearn.ensemble import AdaBoostClassifier
          from sklearn.metrics import accuracy_score,fbeta_score

          clf = AdaBoostClassifier(random_state=0)
          clf = clf.fit(X_train, y_train)
          pred = clf.predict(X_test)
          importances = clf.feature_importances_
          #print(importances)

          fscore = fbeta_score(y_test, pred,beta=0.5)
          acc = accuracy_score(y_test, pred)
          print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.804794520548
Accuracy:  0.732876712329
```

Random Forest Classifier

```
In [574]: from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import accuracy_score,fbeta_score

          clf = RandomForestClassifier(random_state=0)
          clf = clf.fit(X_train, y_train)
          pred = clf.predict(X_test)

          fscore = fbeta_score(y_test, pred,beta=0.5)
          acc = accuracy_score(y_test, pred)
          print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.80298013245
Accuracy:  0.739726027397
```

Stochastic Gradient Descent Classifier

In [575]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score,fbeta_score

clf = SGDClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
pred = clf.predict(X_test)

fscore = fbeta_score(y_test, pred,beta=0.5)
acc = accuracy_score(y_test, pred)
print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.818181818182
Accuracy:  0.678082191781
```

```
C:\Users\Administrator\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:128: FutureWarn
ing: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassi
fier'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter
defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

Decision Tree Classifier

In [576]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,fbeta_score

clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
pred = clf.predict(X_test)

fscore = fbeta_score(y_test, pred,beta=0.5)
acc = accuracy_score(y_test, pred)
print("F-score: ",fscore,"\nAccuracy: ",acc)
```

```
F-score:  0.78488372093
Accuracy:  0.671232876712
```

Looking at the above models, we can clearly see that the Stochastic Gradient Descent Classifier gives us the best result with an F-score of 0.81.

Next, I applied GridSearchCV to see if tuning the model will get us any results.

Grid Search

```
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
#import parfit.parfit as pf

# Initializing the classifier
clf = SGDClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
predictions = clf.predict(X_test)

# Creating the parameters list to tune, using a dictionary.
# parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1, value2]}
#parameters = {'n_iter':[3, 5, 7], 'warm_start':[True, False], 'shuffle':[True, False]}

"""grid = {
    'alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3], # learning rate
    'n_iter': [1000, 100, 15], # number of epochs
    'loss': ['log', 'squared_loss', 'squared_epsilon_insensitive', 'huber'], # logistic regression,
    'penalty': ['l2', 'l1', 'elasticnet', 'None'],
    'n_jobs': [-1]
}"""

grid = {
    'alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4, 1e5], # learning rate
    'n_iter': [1000, 250, 75], # number of epochs
    'loss': ['log', 'squared_loss'], # logistic regression,
    'penalty': ['l2', 'l1', 'elasticnet'],
    'n_jobs': [-1]
}

# Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score, beta=0.5)
```

```
# Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score, beta=0.5)

# TODO: Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
#grid_obj = GridSearchCV(SGDClassifier(random_state=0), scoring=scorer, param_grid=grid)

# TODO: Fit the grid search object to the training data and find the optimal parameters using fit()
grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)


# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
```

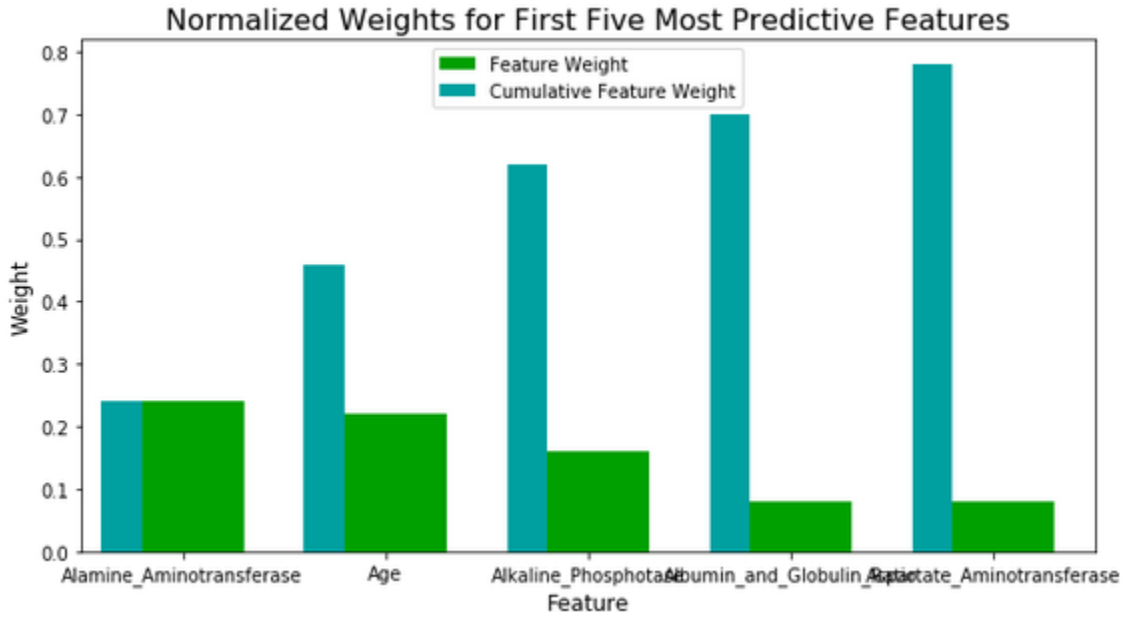
Output:

```
Unoptimized model
------
Accuracy score on testing data: 0.6781
F-score on testing data: 0.8182

Optimized Model
------
Final accuracy score on the testing data: 0.6918
Final F-score on the testing data: 0.8008
```

The initial model performs better.

Now, let's check if slicing features and keeping only the best features will do any good.




```
# Reduce the feature space
X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:5]]]
X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:5]]]

#clf = SGDClassifier()
clf = SGDClassifier()
clf = clf.fit(X_train_reduced, y_train)

# Make new predictions
reduced_predictions = clf.predict(X_test_reduced)

# Report scores from the final model using both versions of data
print("Final Model trained on full data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, pred)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, pred, beta = 0.5)))
print("\nFinal Model trained on reduced data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_predictions, beta = 0.5)))
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.6781
F-score on testing data: 0.8182

Final Model trained on reduced data
------
Accuracy on testing data: 0.7397
F-score on testing data: 0.7803
```

The performance actually got worse by cutting down features. The F-score has decreased when we cut down the features. This indicates that not only these, but there are also other useful features.

Results

Model Evaluation and Justification

The data performs very well on the model Stochastic Gradient Descent Classifier. This model has outperformed our benchmark model and all the other ensemble methods and decision tree classifiers. This model got the highest f-beta score with 0.81, which can be considered a pretty good value.

Stochastic Gradient Descent Classifier

```
In [575]: from sklearn.linear_model import SGDClassifier
          from sklearn.metrics import accuracy_score,fbeta_score

          clf = SGDClassifier(random_state=0)
          clf = clf.fit(X_train, y_train)
          pred = clf.predict(X_test)

          fscore = fbeta_score(y_test, pred,beta=0.5)
          acc = accuracy_score(y_test, pred)
          print("F-score: ",fscore,"\nAccuracy: ",acc)

          F-score:  0.818181818182
          Accuracy:  0.678082191781

          C:\Users\Administrator\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:128: FutureWarn
          ing: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassi
          fier'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter
          defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
            "and default tol will be 1e-3." % type(self), FutureWarning)
```

I'd like to conclude that the SGDClassifier is the best model for this dataset.

Conclusion

Reflection

I first loaded the data from the csv into a variable. Then, I copied the 'Dataset'(Which has the information on whether the patient has the disease or not) into 'labels'. After that, we clean the data by filling up any null values in the data set by the respective sets mean. Then, I'll apply one-hot encoding on the gender feature, which will divide the males and females into two

different features. Then, I applied normalization on the data by using one of the normalization techniques like MinMaxScaler() or StandardScaler(). Then, I used the train_test_split with the input of transformed_features and labels to get the X_train, X_test, y_train, y_test.

I first applied the benchmark model by fitting the data with the logistic regression and predicting the values. Then, I'll check with the benchmark metrics to check how my model is working.

Later, I applied other models from ensemble methods like Gradient Boosting, AdaBoost, Decision Trees like Decision Tree Classifier and Random Tree Classifier and Stochastic Gradient Descent and checked with the evaluation metrics to see how the data is working with different models.

Of these, I chose the best model based on the metrics, i.e. Stochastic Gradient Descent and I planned to further improvise and optimize it by applying the GridSearchCV.

The results were almost the same, so I went with the same model.

I tried to visualize the fscores of different models using ROC but I found it difficult.

Improvement

I think the model could be further improved by adding more data so that the model can be trained more on the data. So, increasing the instances will definitely help. Also, adding more features would certainly help us.

References

- https://werlabs.co.uk/liver-function/bilirubin/
- https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b
- http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- http://scikit-learn.org/stable/modules/tree.html
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html