Machine Learning in Bioinformatics

Midterm Report

Ayhan Yüksek
Engineering Faculty, Computer Engineering
170709041
ayhannyuksek@outlook.com

1. Introduction

In this project, I found a dataset on the diabetes patients of 130 hospitals in the USA between 1999-2008. I got my dataset on kaggle. The original data are submitted on behalf of the Center for Clinical and Translational Research, Virginia Commonwealth University.

This dataset includes the following features:

- **Encounter ID** → Unique identifier of an encounter.
- **Patient number** → Unique identifier of a patient.
- **Race** → Values: Caucasian, Asian, African American, Hispanic, and other.
- **Gender** → Values: male, female, and unknown/invalid.
- **Age** → Grouped in 10-year intervals: 0, 10), 10, 20), …, 90, 100).
- **Weight** → Weight in pounds.
- **Admission type id** → Integer identifier corresponding to 9 distinct values, for example, emergency, urgent, elective, newborn, and not available.
- **Discharge disposition id** → Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available.
- **Admission source id** → Integer identifier corresponding to 21 distinct values, for example, physician referral, emergency room, and transfer from a hospital.
- **Time in hospital** → Integer number of days between admission and discharge.
- **Payer code** → Integer identifier corresponding to 23 distinct values, for example, Blue Cross/Blue Shield, Medicare, and self-pay Medical.
- **Medical specialty** → Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values, for example, cardiology, internal medicine, family/general practice, and surgeon.
- **Number of lab procedures** → Number of lab tests performed during the encounter.
- **Number of procedures** → Numeric Number of procedures (other than lab tests) performed during the encounter.
- **Number of medications** → Number of distinct generic names administered during the encounter.
- **Number of outpatient visits** → Number of outpatient visits of the patient in the year preceding the encounter.
- **Number of emergency visits** → Number of emergency visits of the patient in the year preceding the encounter.
- **Number of inpatient visits** → Number of inpatient visits of the patient in the year preceding the encounter.

- **Diagnosis 1, Diagnosis 2, Diagnosis 3 →** The primary diagnosis (coded as first three digits of ICD9); 848 distinct values.
- **Number of diagnoses →** Number of diagnoses entered to the system 0%.
- **Glucose serum test result →** Indicates the range of the result or if the test was not taken. Values: ">200," ">300," "normal," and "none" if not measured.
- **A1c test result →** Indicates the range of the result or if the test was not taken. Values: ">8" if the result was greater than 8%, ">7" if the result was greater than 7% but less than 8%, "normal" if the result was less than 7%, and "none" if not measured.
- **Change of medications →** Indicates if there was a change in diabetic medications (either dosage or generic name). Values: "change" and "no change".
- **Diabetes medications →** Indicates if there was any diabetic medication prescribed. Values: "yes" and "no.

- 24 features for medications for the generic names: **metformin, repaglinide, nateglinide, chlorpropamide, glimepiride, acetohexamide, glipizide, glyburide, tolbutamide, pioglitazone, rosiglitazone, acarbose, miglitol, troglitazone, tolazamide, examide, sitagliptin, insulin, glyburide-metformin, glipizide-metformin, glimepiride- pioglitazone, metformin-rosiglitazone, and metformin-pioglitazone**, the feature indicates whether the drug was prescribed or there was a change in the dosage. Values: "up" if the dosage was increased during the encounter, "down" if the dosage was decreased, "steady" if the dosage did not change, and "no" if the drug was not prescribed
- **Readmitted →** Days to inpatient readmission. Values: "<30" if the patient was readmitted in less than 30 days, ">30" if the patient was readmitted in more than 30 days, and "No" for no record of readmission.

2. Our Problem:

It is important to know if a patient will be readmitted in some hospital. The reason is that you can change the treatment, in order to avoid a readmission. Our classification problem is this. Will the patient be not readmission to the hospital or will he be readmission within 30 days?

We have 3 different values in our data for this problem:

1. No readmission,
2. A readmission in less than 30 days (this situation is not good, because maybe your treatment was not appropriate),
3. A readmission in more than 30 days (this one is not so good as well the last one, however, the reason can be the state of the patient.

To solve this problem, we will first clean our data and apply appropriate classification algorithms.

3. Data Pre-processing:

Data pre-processing is a data mining technique which is used to transform the raw data in a useful and efficient format .

3.1. Loading Dataset:

We have loaded our ".csv" type dataset by importing the pandas package. Then we printed the shape and features names of our dataset as shown in Figure 1.

```
Data shape: (101766, 50)

 ['encounter_id', 'patient_nbr', 'race', 'gender', 'age', 'weight', 'admission_type_id',
'discharge_disposition_id', 'admission_source_id', 'time_in_hospital', 'payer_code', 'medical_specialty',
'num_lab_procedures', 'num_procedures', 'num_medications', 'number_outpatient', 'number_emergency',
'number_inpatient', 'diag_1', 'diag_2', 'diag_3', 'number_diagnoses', 'max_glu_serum', 'A1Cresult', 'metformin',
'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'acetohexamide', 'glipizide', 'glyburide',
'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide', 'examide',
'citoglipton', 'insulin', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-pioglitazone', 'metformin-
rosiglitazone', 'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitted']
**********************************************************************************************
```

Figure 1

3.2. Cleaning Our Dataset:

We deleted the columns in the dataset that we did not think would affect the result and corrected the unassigned, uncategorized and illogical values.

Part 1: Deleting some of the features that we don't want to use:

First, we removed "encounter_id" and "patient_nbr" columns because this didn't contribute to our model.

```
#drop 'encounter_id','patient_nbr' column
dataset.drop(["encounter_id","patient_nbr"],axis=1,inplace=True)
```

Part 2: We deleted patients whose Race was unknown:

First of all, we have 2273 unknown ("?") values in our Race feature. Since our total data is 101766, we dropped it because we thought it would not be a problem to delete 2273 data.

```
dataset = dataset.loc[dataset["race"] != "?"]
```

We performed the drop operation with our "loc[]" command. We used this process in our other columns.

Part 3: We cleared the unknown value from the Gender feature:

There was only one "unknown / valid" value in our Gender feature. Therefore, we dropped this unknown value as before.

```
dataset = dataset.loc[dataset["gender"] != "Unknown/Invalid"]
```

Part 4: We dropped our columns containing 90% missing values:

Our columns containing 90% missing value would not have a significant effect on our data

```
dataset = dataset.drop(["weight","payer_code","medical_specialty"], axis = 1)
```

Part 5: We dropped our columns containing only one value:

We drop it because our "examide" and "citoglipton" columns containing only one value will not have any effect.

```
dataset = dataset.drop(["examide","citoglipton"], axis = 1)
```

Part 6: We dropped it to avoid using unnecessary treatment methods:

We applied 3 treatments. We decided to remove the remaining two, as they show substantially similar characteristics.

```
dataset = dataset.drop(["diag_2","diag_3"], axis = 1)
```

Part 7: We dropped NAN values in our dataset:

Since there are not too many missing values left in the rest of our dataset, we first made our missing values to NAN value with the command "np.nan ()". Then we dropped the NAN values with the "dropna ()" command.

```
dataset.replace('?', np.nan, inplace = True)
dataset= dataset.dropna()
```

Part 8: Editing the "age" feature:

The values in the age column are given as ranges. To recover from this situation, we replaced all our values with a certain number from the range.

```
cleanup_age = {"age": {"[0-10)":5, "[10-20)":15, "[20-30)":25, "[30-40)":35, "[40-50)":45,
                       "[50-60)":55, "[60-70)":65, "[70-80)":75, "[80-90)":85, "[90-100)":95}}


dataset.replace(cleanup_age, inplace=True)
```

Part 9: Editing the "admission_type_id" feature:

The admission_type_id feature was taking a value between 1-8. The meanings of these values represent "emergency", "elective", "new born" and "trauma center" values in hospitals.[1]

Due to this situation, we re-filled the values in our data according to 4 categories: "emergency", "elective", "new born" and "trauma center".

```
mapped = {1:"Emergency",
          2:"Emergency",
          3:"Elective",
          4:"New Born",
          5:np.nan,
          6:np.nan,
          7:"Trauma Center",
          8:np.nan}

dataset["admission_type_id"] = dataset["admission_type_id"].replace(mapped)
dataset= dataset.dropna()
```

Part 10: Editing the "discharge_disposition_id" feature:

Similarly, in this column, their discharge status was represented by numerical values. We regarded other situations other than sending home as a single case.[2]

```
mapped_discharge = {1:"Discharged to Home",
                    6:"Discharged to Home",
                    8:"Discharged to Home",
                    13:"Discharged to Home",
                    19:"Discharged to Home",
                    18:np.nan,25:np.nan,26:np.nan,
                    2:"Other",3:"Other",4:"Other",
                    5:"Other",7:"Other",9:"Other",
                    10:"Other",11:"Other",12:"Other",
                    14:"Other",15:"Other",16:"Other",
                    17:"Other",20:"Other",21:"Other",
                    22:"Other",23:"Other",24:"Other",
                    27:"Other",28:"Other",29:"Other",30:"Other"}

dataset["discharge_disposition_id"] = dataset["discharge_disposition_id"].replace(mapped_discharge)
```

Part 11: Editing the "admission_source_id" feature:

As the same situation applies to other id columns within this column, we replaced it with the appropriate data again. [3]

```
mapped_adm = {1:"Referral",2:"Referral",3:"Referral",
              4:"Other",5:"Other",6:"Other",10:"Other",22:"Other",25:"Other",
              9:"Other",8:"Other",14:"Other",13:"Other",11:"Other",
              15:np.nan,17:np.nan,20:np.nan,21:np.nan,
              7:"Emergency"}

dataset["admission_source_id"] = dataset["admission_source_id"].replace(mapped_adm)
```

3.3. Data Encoding:

Part 1: One hot encoding for medications:

Drugs, which constitute a significant part of the columns of our data, received 4 values each. Since 24 different drug columns take the same 4 values ("no", "up", "down", "steady"), we created a separate column for each value with one hot encoding.

In this way, we changed the values of our columns to binary.

```
dataset = pd.get_dummies(dataset,columns = [dataset.columns.values[i]
                                            for i in range(17,38) ], prefix=[dataset.columns.values[i]
                                            for i in range(17,38)], prefix_sep='_',drop_first=True)
```

Part 2: Encoding for string features:

race:

```
mapped_race = {"Caucasian":0,"AfricanAmerican":1, "Hispanic":2, "Other":3, "Asian":4 }
dataset["race"] = dataset["race"].replace(mapped_race)
```

Gender:

```
mapped_gender = {"Male":0,"Female":1}
dataset["gender"] = dataset["gender"].replace(mapped_gender)
```

Admission_type_id:

```
mapped_admission_id = {"Emergency":0, "Elective":1, "New Born":2, "Trauma Center":3}
dataset["admission_type_id"] = dataset["admission_type_id"].replace(mapped_admission_id)
```

Discharged_dispoint_id:

```
mapped_discharge = {"Discharged to Home":0, "Other":1}
dataset["discharge_disposition_id"] = dataset["discharge_disposition_id"].replace(mapped_discharge)
```

Admission_source_id:

```
mapped_source = {"Referral":0, "Emergency":1, "Other":2}
dataset["admission_source_id"] = dataset["admission_source_id"].replace(mapped_source)
```

Part 3: Label encoding technic for not ordinal numeric values:

Some of our features were taking values between certain number values.

For example, "num_lab_procedures" column takes values between 1-132, but since it has 120 different unique values, I wanted to reduce the value range to 1-120 with label encoding to be ordinal. You can see examples below:

Num_lab_procedures:

```
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(dataset["num_lab_procedures"])
dataset["num_lab_procedures"] = label_encoder.transform(dataset["num_lab_procedures"])
```

Num_medications:

```
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(dataset["num_medications"])
dataset["num_medications"] = label_encoder.transform(dataset["num_medications"])
```

***** We performed the same process for "diag_1", "max_glu_serum", "A1Cresult", "change", "diabetesMed" features.

Part 4: Encoding and editing ground truth vector "readmitted":

Our readmitted feature took 3 different values. No readmission("No"), a readmission in more than 30 days(">30"), a readmission in less than 30 days("<30").

We assumed "> 30" and "NO" values as similar cases and converted it to binary class by encoding it as 0-1.

```
mapped_readmitted = {"<30":0, ">30":1, "NO":1}
dataset["readmitted"] = dataset["readmitted"].replace(mapped_readmitted)
```

4. Evaluating Models:

In this section, before implementing the algorithms, we split our data set into 80 percent for training and 20 percent for testing.

4.1. Imbalanced Class Problem:

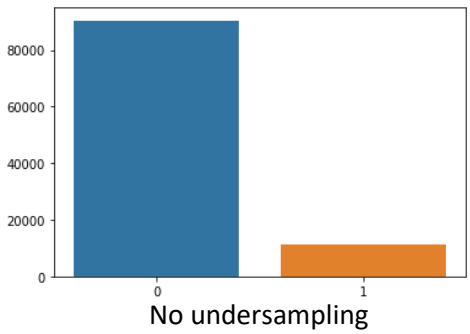
Unbalanced classification problems pose a challenge for predictive modeling, as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of samples for each class. Because imbalanced classes have a disproportionate rate of observation in each class.

We used the undersampling technique to solve this problem.

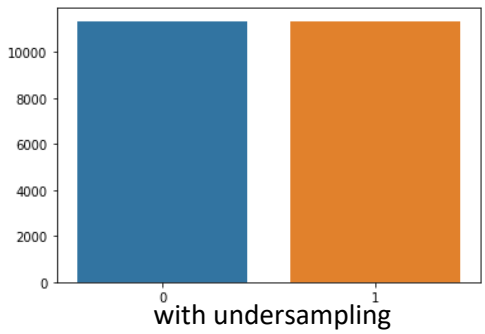
Undersampling:

It is the process where you randomly delete some of the observations from the majority class in order to match the numbers with the minority class. You can see distribution of readmitted value with under sampling and without undersampling

```
ros = RandomUnderSampler(random_state=0)
X, y = ros.fit_resample(X, y)
```



No undersampling



with undersampling

4.2. Logistic Regression*:*

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is binary. As a first task of applying algorithms part, we fitted our dataset into Logistic Regression and predicted.

```
logisticRegression
ROC : % 86.89679468985128
f1_score :  0.8735521235521235
recall :  0.8855185909980431
precision :  0.861904761904762
```

4.3. KNN Classifier:

The K in the name of this classifier represents the k nearest neighbors, where k is an integer value specified by the user. Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data.

```
KNN Classifier
ROC : % 76.10099440801102
f1_score :  0.7740714454696003
recall :  0.8003913894324853
precision :  0.74942739349519
```

4.4. Random Forest Classifier:

As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

```
RFC Classifier
ROC : % 89.45865264436841
f1_score :  0.8988223984619083
recall :  0.9148727984344422
precision :  0.8833254605573926
```

4.5. Gradient Boosting Classifier:

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

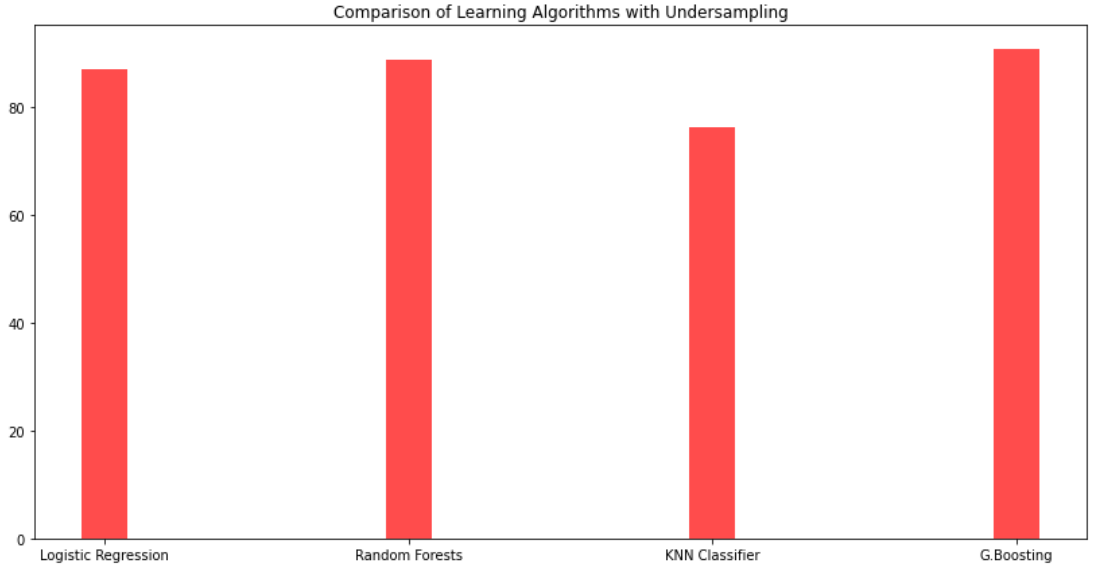
```
GBC classifier
ROC : % 90.34321267982253
f1_score :  0.9080926235378372
recall :  0.9305283757338552
precision :  0.8867132867132868
```

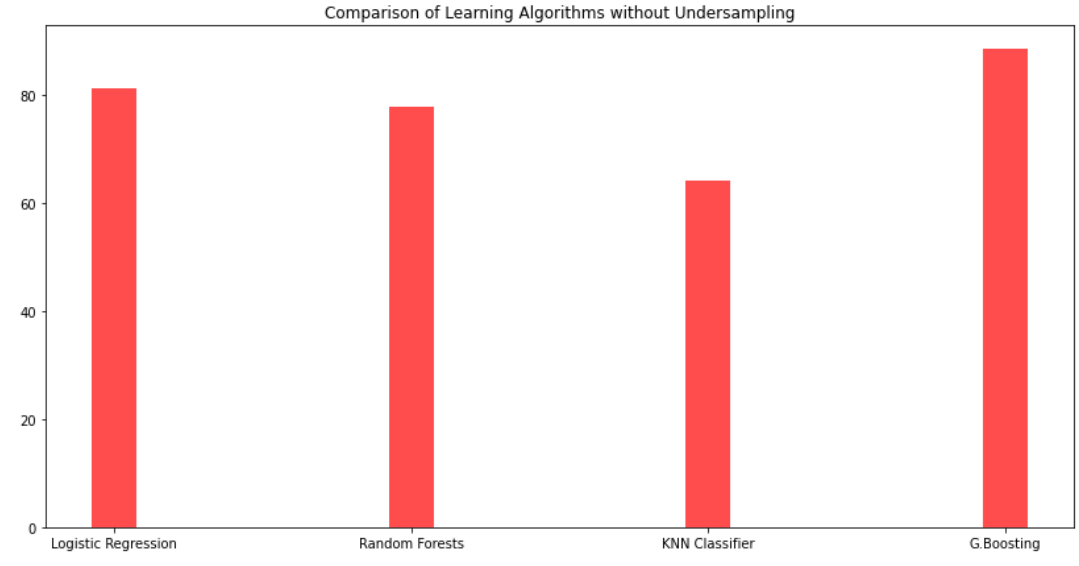
5. Conclusion:

To conclude, in this project, we tried to solve a simple real world problem by using supervisedlearning algorithms. In order to analyze our dataset, we had to perform some operations (datacleaning, encoding, scaling etc.) because our dataset contained illogical data and missing values. These flaws could cause our analysis results to be different than expected.

After completing the preprocessing processes in our dataset, we saw that our target class is an unbalanced class. As explained in section 4.1, we undersampling to solve the imbalanced class problem. After the undersampling process, our dataset is ready to implement the learning algorithms. Before undersampling, our ROC score was lower. We increased our roc value after undersampling. In these results, we see that while the gradient boosting classifier worked better in both cases, the knn classifier did worse in both cases.

Since our data was not distributed properly, using only the accuracy score would have misled us. We calculated the recall and precision scores to avoid this situation and for a better analysis. Finally, we calculated the f1 score value to see the common result of recall and precision values. Because F1 score is the harmonic average of recall and precision values.





6. References:

- Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach, ThirdEdition, Prentice Hall ISBN 9780136042594.
- https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/
- https://www.statisticssolutions.com/what-is-logistic-regression/
- https://medium.com/@sertacozker/boosting-algoritmalar
- https://www.tutorialspoint.com/machinelearningwithpython/machinelearningwithpythonclassificationalgorithmsrandomforest.htm
- https://www.geeksforgeeks.org/ml-bagging-classifier/[7] Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of NeuralComputation. MIT Press. ISBN 978-0262581684