# Predicting Hospital Readmission
# Using
# Mimic Database

HAP 880

Final Project Report

By,

Akhil Anto
G01103742

# Contents

## Introduction

As the healthcare system moves toward value-based care, CMS has created many programs to improve the quality of care of patients. One of these programs is called the Hospital Readmission Reduction Program (HRRP), which reduces reimbursement to hospitals with above average readmissions. For those hospitals which are currently penalized under this program, one solution is to create interventions to provide additional assistance to patients with increased risk of readmission. But how do we identify these patients? We can use predictive modeling from data science to help prioritize patients.[1]

The project describes the development of model for predicting whether a patient discharged from a hospital is likely to readmitted within 30 days. Used MIMIC-III database, which is a collection of hospital database has around 50000 patients who were admitted to Beth Israel Deaconess Medical center in Boston Center, Massachusetts from 2001 to 2012. Project mainly concentrates on different sampling methods and improving recall rather than accuracy

## Problem Statement

Implement a machine learning model to predict the probability of readmission within 30 days of discharge, based on the patient's release records upon discharge.

The solution outline will be as follows:

1. Preliminary research: Consult the relevant literature to assemble a list of relevant features to use.

2. Data wrangling: Use SQLite to obtain the relevant subset of data from the MIMIC-III database.

3. Exploratory data analysis: Produce relevant statistics and visualizations to characterize the data

4. Data preprocessing: Prepare the dataset for application of a machine learning model – clean missing or invalid values, calculate the time between subsequent admissions to produce label data frame. Used different NLP techniques to extract entities from discharge notes.

5. ML model implementation and refinement: Define model metrics, implement relevant code and perform hyper parameter optimization. Once optimized values are found, estimate model performance

## Metrics

Problem can be described as a binary classification problem on an unbalanced dataset (only ~7 % of the observations fall into the "positive" label category). For this reason, we cannot rely on a simplistic metric such as prediction accuracy, and must look at more robust metrics

A standard approach in such cases is to look at the precision, sensitivity, specificity, F1 score and the Receiver Operating Characteristic curve, all defined in the following:

- Precision is defined as:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

- Recall, also known as Sensitivity or true positive rate (TPR), is defined as:

$$Recall = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

- Specificity is defined as:

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative + False Positive}}$$

- F1 score, the harmonic mean of precision and recall:

$$F1 = \frac{2 * (\text{True Positive})}{(\text{True Positive + False Negative}) + (\text{True Positive + False Positive})}$$

ROC curve: A plot of the True Positive Rate (or sensitivity) as a function of the False Positive Rate (FPR, equal to 1-specificity), which implicitly sets the discrimination threshold. Specifically, the area under the ROC curve (commonly referred to as the AUC) is a good nominal metric for the classifier performance, where an AUC of 0.5 represent a classification that is no better than chance, and the closer the AUC get to 1, the better the classifier.

## Data Base

MIMIC-III database is a collection of datasets and from that we used mainly four data sets

- ADMISSIONS- Containing admission and discharge dates (has a unique identifier HADM_ID for each admission)
- DIAGNOSES_ICD - Hospital assigned diagnoses, coded using the International Statistical Classification of Diseases and Related Health Problems (ICD) system
- PATIENTS – Every unique patient in the database (defines SUBJECT_ID) consists of import attributes like GENDER AND DATE OF BIRTH
- NOTEEVENTS—contains all notes for each hospitalization (links with HADM_ID)

## Data Preprocessing

### 1.Dataset – ADMISSIONS

- Converted all dates to Date Time format

```
# convert to dates
df_adm.ADMITTIME = pd.to_datetime(df_adm.ADMITTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coer
ce')
df_adm.DISCHTIME = pd.to_datetime(df_adm.DISCHTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coer
ce')
df_adm.DEATHTIME = pd.to_datetime(df_adm.DEATHTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coer
ce')
```

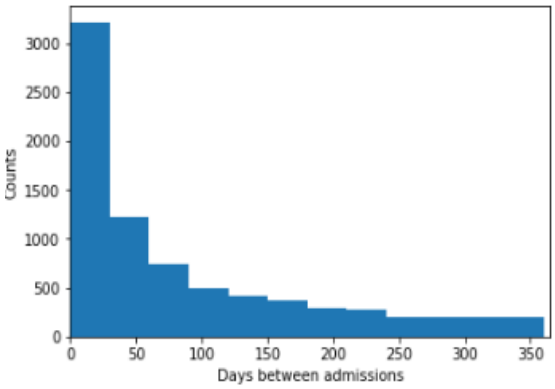
[5]

- Created a column named NEXT_ADMITIME by using group by shift operator to get next admission (if it exists) for each SUBJECT_ID

```
# sort by subject_ID and admission date
df_adm = df_adm.sort_values(['SUBJECT_ID','ADMITTIME'])
df_adm = df_adm.reset_index(drop = True)
# add the next admission date and type for each subject using groupby
# we have to use groupby otherwise the dates will be from different subjects
df_adm['NEXT_ADMITTIME'] = df_adm.groupby('SUBJECT_ID').ADMITTIME.shift(-1)
# get the next admission type
df_adm['NEXT_ADMISSION_TYPE'] = df_adm.groupby('SUBJECT_ID').ADMISSION_TYPE.shift(-1)
```

[5]

- Similarly, NEXT_ADMISSION_TYPE by following the same step as above
- Getting unplanned admissions by filtering out NEXT_ADMISSION_TYPE = ELECTIVE
- Lastly backfilled the values
- Now calculated the days until next admission
- Histogram of days between readmissions is shown here
- Compressed the ETHNICITY categories




```
# Compress the number of ethnicity categories
data['ETHNICITY'].replace(regex=r'^ASIAN\D*', value='ASIAN', inplace=True)
data['ETHNICITY'].replace(regex=r'^WHITE\D*', value='WHITE', inplace=True)
data['ETHNICITY'].replace(regex=r'^HISPANIC\D*', value='HISPANIC/LATINO', inplace=True)
data['ETHNICITY'].replace(regex=r'^BLACK\D*', value='BLACK/AFRICAN AMERICAN', inplace=True)
data['ETHNICITY'].replace(['UNABLE TO OBTAIN', 'OTHER', 'PATIENT DECLINED TO ANSWER','UNKNOWN/NO
T SPECIFIED'], value='OTHER/UNKNOWN', inplace=True)
data['ETHNICITY'].loc[~data['ETHNICITY'].isin(data['ETHNICITY'].value_counts().nlargest(5).inde
x.tolist())] = 'OTHER/UNKNOWN'
```

[5]

## 2.Dataset – DIAGNOSES_ICD

- There are 6985 unique ICD9 codes in this dataset
- Reduced the diagnosis into more general categories
- Converted diagnoses list into hospital admission-item matrix

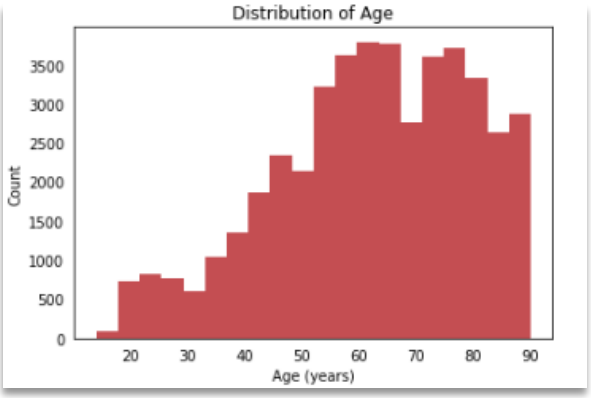
| ICD-9 | Categories |
|---|---|
| 1- 140 | infectious |
| 140 - 240 | neoplasms |
| 240 - 280 | endocrine |
| 280 - 290 | blood |
| 290 - 320 | mental |
| 320 - 390 | nervous |
| 390 - 460 | circulatory |
| 460 - 520 | respiratory |
| 520 - 580 | digestive |
| 580 - 630 | genitourinary |
| 630 - 680 | pregnancy |
| 680 - 710 | skin |
| 710 - 740 | muscular |
| 740 - 760 | congenital |
| 760 - 780 | prenatal |
| 780 - 800 | misc |
| 800 - 1000 | injury |
| 1000 - 2000 | misc |

```
# ICD-9 Main Category ranges
icd9_ranges = [(1, 140), (140, 240), (240, 280), (280, 290), (290, 320), (320, 390),
               (390, 460), (460, 520), (520, 580), (580, 630), (630, 680), (680, 710),
               (710, 740), (740, 760), (760, 780), (780, 800), (800, 1000), (1000, 2000)]

# Associated category names
diag_dict = {0: 'infectious', 1: 'neoplasms', 2: 'endocrine', 3: 'blood',
             4: 'mental', 5: 'nervous', 6: 'circulatory', 7: 'respiratory',
             8: 'digestive', 9: 'genitourinary', 10: 'pregnancy', 11: 'skin',
             12: 'muscular', 13: 'congenital', 14: 'prenatal', 15: 'misc',
             16: 'injury', 17: 'misc'}

# Re-code in terms of integer
for num, cat_range in enumerate(icd9_ranges):
    df_daig['recode'] = np.where(df_daig['recode'].between(cat_range[0],cat_range[1]),
            num, df_daig['recode'])

# Convert integer to category name using diag_dict
df_daig['recode'] = df_daig['recode']
df_daig['cat'] = df_daig['recode'].replace(diag_dict)

# Create list of diagnoses for each admission
hadm_list = df_daig.groupby('HADM_ID')['cat'].apply(list).reset_index()
hadm_list.head()

# Convert diagnoses list into hospital admission-item matrix
hadm_item = pd.get_dummies(hadm_list['cat'].apply(pd.Series).stack()).sum(level=0)
hadm_item.head()

# Join back with HADM_ID, will merge with main admissions DF later
hadm_item = hadm_item.join(hadm_list['HADM_ID'], how="outer")
hadm_item.head()

# Merge with main admissions
data = data.merge(hadm_item, how='inner', on='HADM_ID')
```

[5]

## 3.Dataset – PATIENTS

- Gender feature is binarized
- Converted date of birth to date time format
- Calculated Age by finding difference between date of birth and first admission date



## 4.Dataset – Note Events

- The main columns of interest are: SUBJECT_ID, HADM_ID, CATEGORY, TEXT
- The dataset has 2,083,180 rows, indicating that there are multiple notes per hospitalization.
- Used only discharge summary by filtering out CATEGORY like 'Discharge summary'
- Since there were multiple Discharge Summary, used only the last one

### 4.1Natural Language Processing

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages. [4]

The following steps where applied

- **Transformed all text to lower case**
- **Removed punctuations**
- **Removed numbers**
- **Tokenized each word**

Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words etc.

- **Removed stop words using**
- **Normalization – Applied lemmatization**

Normalization puts all words on equal footing, and allows processing to proceed uniformly. Normalizing text can mean performing a number of tasks, but for our framework we will approach normalization in 2 distinct steps.

**Stemming**

Stemming is the process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

running → run

**Lemmatization**

Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma.

better →good

Here I used Lemmatization for this project.

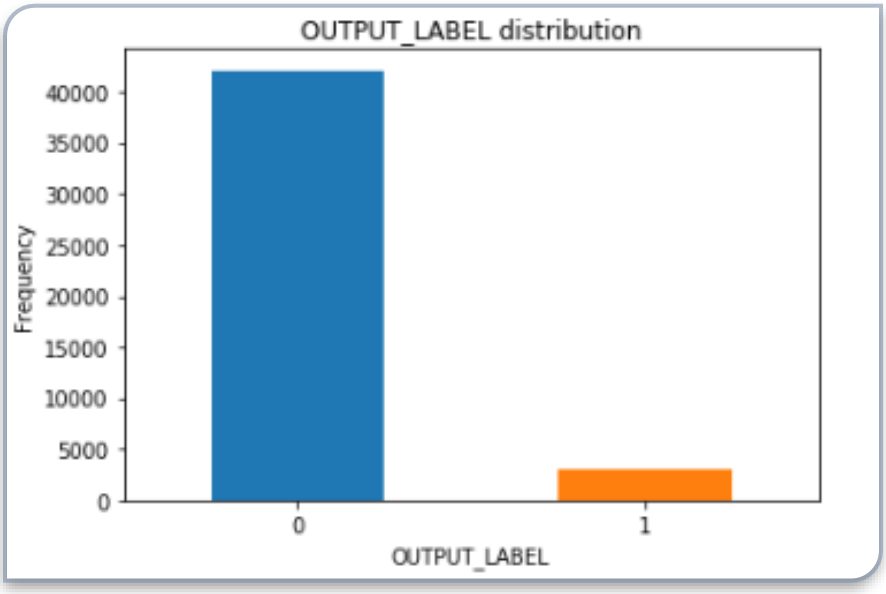
- **Count Vectorization**

Count Vectorizer works on Terms Frequency, i.e. counting the occurrences of tokens and building a sparse matrix of documents x tokens Vectorized each word using count vectorizer (1's and 0's) and considered 3000 features

```
def preprocess_text(df):
    # This function preprocesses the text by filling not a number and replacing new lines ('\n')
and carriage returns ('\r')
    df.TEXT = df.TEXT.fillna(' ')
    df.TEXT =df.TEXT.str.replace('\n',' ')
    df.TEXT =df.TEXT.str.replace('\r',' ')
    return df

df = preprocess_text(df)

stopwords = nltk.corpus.stopwords.words('english')
ps = nltk.PorterStemmer()
wn = nltk.WordNetLemmatizer()

def clean_textmain(text):
    text = text.lower()
    text ="".join([char for char in text if char not in string.punctuation ])
    #text=re.sub(r'\b\w{1,4}\b', '',text)
    result = re.sub(r'[0-9]+','',text)
    tokens = re.split('\W+', result)
    words = [word for word in tokens if word.isalpha()]
    text1=[word for word in words if word not in string.digits]
    #text = [ps.stem(word) for word in text1 if word not in stopwords]
    text = [wn.lemmatize(word) for word in text1 if word not in stopwords]
    return text

from sklearn.feature_extraction.text import CountVectorizer
count_vect_sample = CountVectorizer(max_features = 3000,analyzer = clean_textmain)
X_counts_sample = count_vect_sample.fit_transform(df['TEXT'])
print(X_counts_sample.shape)
#print(count_vect_sample.get_feature_names())


X_counts_df =pd.DataFrame(X_counts_sample.toarray())
#X_counts_df =pd.DataFrame(count_vect_sample.get_feature_names())
X_counts_df.columns = count_vect_sample.get_feature_names()
#print(X_counts_df.head())
#type(X_counts_df)
```

## Final Dataset

After all the processing the final dataset has in total 45059 records with 3018 attributes. Class 1 which is patients readmitted with in less than 30 days are 2957 and class 0 has 42012 patients. The diagram below shows the distribution of the output class.




So the final dataset is imbalanced with respect with target variable **Class** which is the patients admitted with in less than 30 days

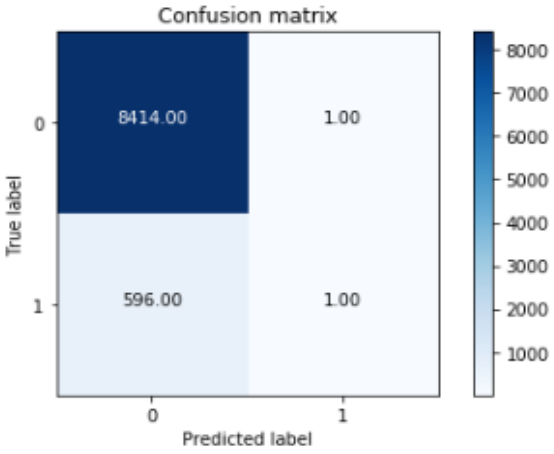
## Simple Modeling

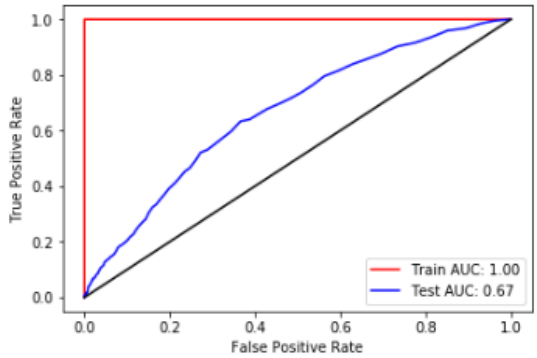
The data split into training and testing set in which 80 % is training and 20 % is testing. Using the random forest classifier with 300 estimators a model was designed. And here are the results.

Test Accuracy is 93.4%

Test Precision is 50 %

Test recall is 2%



## Dealing with Imbalanced Data

Over sampling and under sampling are techniques used in data mining and data analytics to modify unequal data classes to create balanced data sets. Over sampling and under sampling are also known as resampling. These data analysis techniques are often used to be more representative of real world data. For example, data adjustments can be made in order to provide balanced training materials for AI and machine learning algorithms.

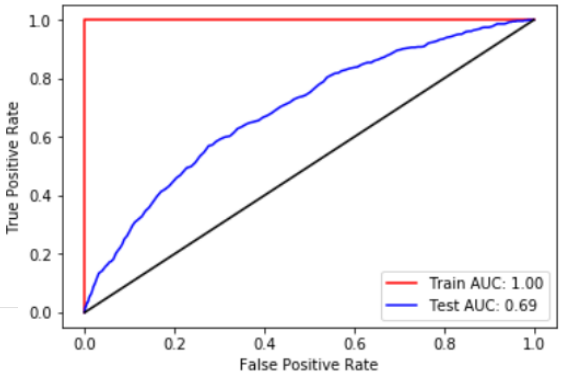
When one class of data is the underrepresented minority class in the data sample, over sampling techniques maybe used to duplicate these results for a more balanced amount of positive results in training. Over sampling is used when the amount of data collected is insufficient. A popular over sampling technique is SMOTE (Synthetic Minority Over-sampling Technique), which creates synthetic samples by randomly sampling the characteristics from occurrences in the minority class.

Conversely, if a class of data is the overrepresented majority class, under sampling may be used to balance it with the minority class. Under sampling is used when the amount of collected data is sufficient. Common methods of under sampling include cluster centroids and Tomek links, both of which target potential overlapping characteristics within the collected data sets to reduce the amount of majority data.

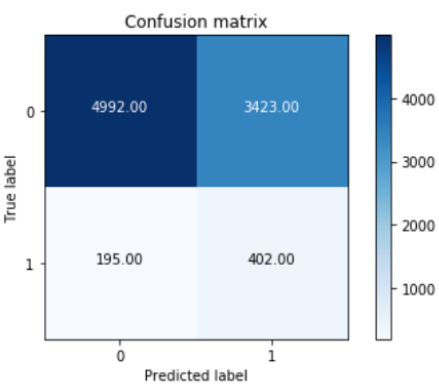
In both over sampling and under sampling, simple data duplication is rarely suggested. Generally, over sampling is preferable as under sampling can result in the loss of important data. Under sampling is suggested when the amount of data collected is larger than ideal and can help data mining tools to stay within the limits of what they can effectively process[2]

## Under sampled data - Modeling

The training set was reduced to 4720 records with class 1 and 0 both equal to 2360 records. Random sampling was used to reduce the majority class. Modeled using Random Forest with 300 estimators. The results are



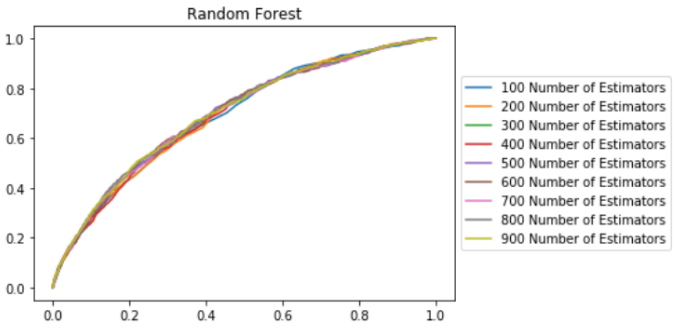

Train AUC:1.000
Test AUC:0.693
Train accuracy:1.000
Test accuracy:0.590
Train recall:1.000
Test recall:0.673
Train precision:1.000
Test precision:0.105
Train specificity:1.000
Test specificity:0.584
Train prevalence:0.500
Test prevalence:0.066

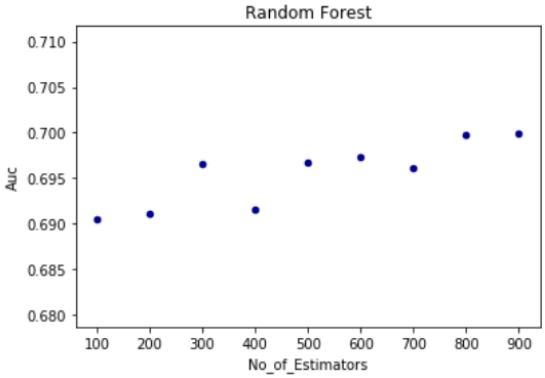


## Effect of Estimators

To calculate the effect of estimators in AUC used the following code

```
rf = dict()
probs_rf = dict()
fpr_rf = dict()
tpr_rf = dict()
thresholds_rf = dict()
auc_rf = dict()
Auc =[]
Number_of_Estimators =[]
for n in range(100, 1000, 100):
    rf[n]=RandomForestClassifier(n_estimators =n)
    rf[n].fit(X_train_tf, y_train)
    probs_rf[n]=rf[n].predict_proba(X_test_tf)
    fpr_rf[n], tpr_rf[n], thresholds_rf[n] = roc_curve(y_test,probs_rf[n][:,1])
    auc_rf[n]=auc(fpr_rf[n],tpr_rf[n])
    Auc.append(auc_rf[n])
    Number_of_Estimators.append(n)
    plt.title('Random Forest')
    plt.plot(fpr_rf[n], tpr_rf[n],label = "{} Number of Estimators".format(n))
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```




We can see that 300 estimators have given almost equal performance as 800 and 900 estimators. So it better to go with 300 estimators to reduce the computational cost and time.

## Feature Selection

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance. Benefits of performing feature selection before modeling your data are

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: Fewer data points reduce algorithm complexity and algorithms train faster[3]

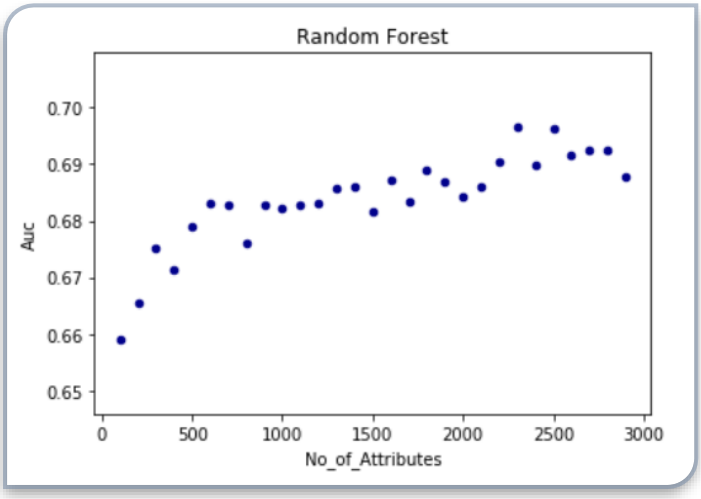
The feature selection method used is chi – Square. The chi-square test is a statistical test of independence to determine the dependency of two variables. It shares similarities with coefficient of determination, $R^2$. However, chi-square test is only applicable to categorical or nominal data while $R^2$ is only applicable to numeric data. From the definition, of chi-square we can easily deduce the application of chi-square technique in feature selection. Suppose you have a target variable (i.e., the class label) and some other features (feature variables) that describes each sample of the data. Now, we calculate chi-square statistics between every feature variable and the target variable and observe the existence of a relationship between the variables and the target. If the target variable is independent of the feature variable, we can discard that feature variable. If they are dependent, the feature variable is very important[

```
rf200=RandomForestClassifier(n_estimators=200)
Auc =[]
No_of_Attributes =[]
for n in range(100, 3000, 100):
    cols_sel_mic=s.sort_values('mic', ascending=False)['att'][:n]
    rf200.fit(df_train[cols_sel_mic],df_train['OUTPUT_LABEL'])
    probs_rf200=rf200.predict_proba(df_test[cols_sel_mic])
    fpr_rf200, tpr_rf200, thresholds_rf200 = roc_curve(df_test['OUTPUT_LABEL'],probs_rf200[:,1])
    auc_rf200=auc(fpr_rf200,tpr_rf200)
    Auc.append(auc_rf200)
    No_of_Attributes.append(n)
    plt.title('Random Forest')
    plt.plot(fpr_rf200, tpr_rf200,label =" {} Number of attributes".format(n))
    plt.legend()
```

```
aucperf= pd.DataFrame({'No_of_Attributes':No_of_Attributes,'Auc':Auc})
aucperf.plot.scatter(x='No_of_Attributes',y='Auc',c='DarkBlue',title = 'Random Forest')
```

Model - Random Forest

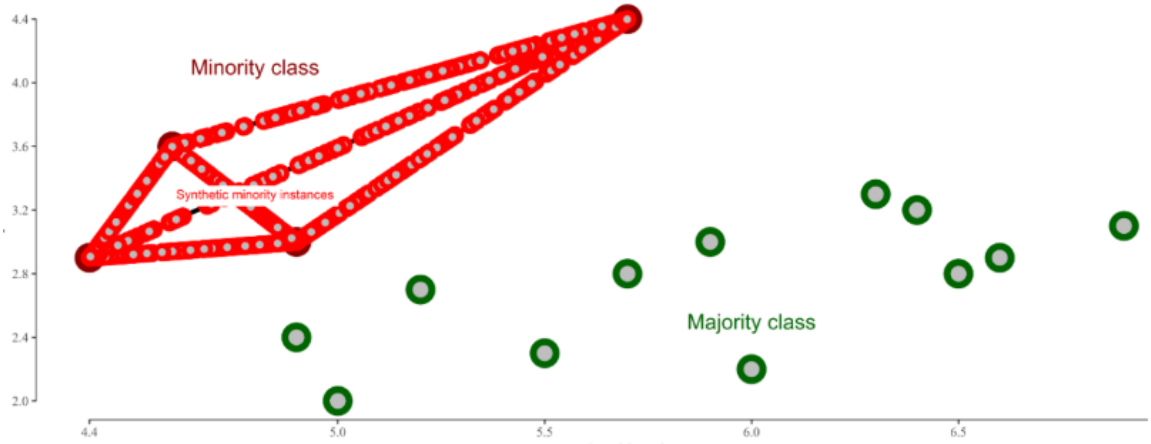
Estimators 200



## Over sampling and Under sampling Methods

Different over sampling and under sampling techniques used are

- Synthetic Minority Over-sampling Technique (SMOTE) - Over Sampling Technique
  SMOTE takes the entire dataset as an input, but it increases the percentage of only the minority cases.




- Random Over sampler - Over Sampling Technique
  Object to over-sample the minority class(es) by picking samples at random with replacement
- Near Miss – Under Sampling Technique
  Near Miss is an under-sampling technique. Instead of resampling the Minority class, using a distance, this will make the majority class equal to minority class.
- Random Under Sampler – Under Sampling Technique
  Under-sample the majority class(es) by randomly picking samples with or without replacement

## Sampling with Grid Search

- Used Grid Search found the optimal hyperparameters of a model which results in the most accurate predictions
- Using grid search selected optimal hyperparameters for the Logistics Regression model

| Sampling Type | F1 Score | Precision | Recall | Accuracy | AUC ROC | Confusion Matrix |
|---|---|---|---|---|---|---|
| Base | 0.003317 | 0.166667 | 0.001675 | 0.933311 | 0.50054 | [[8410, 5], [596, 1]] |

| | | | | | | |
|---|---|---|---|---|---|---|
| SMOTE | 0.17307 | 0.110146 | 0.403685 | 0.744452 | 0.586156 | [[6468, 1947], [356, 241]] |
| Random Over Sampler | 0.172563 | 0.109986 | 0.400335 | 0.745672 | 0.585254 | [[6481, 1934], [358, 239]] |
| Near Miss | 0.126126 | 0.067608 | 0.938023 | 0.138926 | 0.510129 | [[692, 7723], [37, 560]] |
| Random Under Sampler | 0.183113 | 0.107946 | 0.603015 | 0.643586 | 0.62474 | [[5440, 2975], [237, 360]] |

## Conclusion

- Since the data was imbalanced the running different models and hyper tuning just increased the accuracy instead of recall
- Under sampling helped the model to perform much better for this imbalanced data.
- After feature engineering using Chi- square top 2400 features gives the best AUC
- Under sampling Method Near Miss with the application of grid search on the Logistic Regression model performed the best with 94% recall

## Future Research

In this project while vectorizing the word I considered only unigrams . Instead of that if I can use bigrams or trigrams even though it increases the number of features ,it can help to improve the model performance. Also while implementing the count vectorizer I limited the features to 3000. Instead if we can use the all the features it can also help in improving the model performance.

Dandelion is an API which can extract the top entities form a text . So using this API to extract top entities from the discharge summary can improve the accuracy of the text vectorization. Also there are many for data sets in MIMIC data base like lab events and ICU stay and if we include these attributed to the final data set before applying the models then I believe that can make a big change in the prediction

## Reference

[1] Predicting Hospital Readmission for Patients with Diabetes Using Scikit-Learn | Andrew Long |Oct 21, 2018 |

https://towardsdatascience.com/predicting-hospital-readmission-for-patients-with-diabetes-using-scikit-learn-a2e359b15f0

[2] over sampling and under sampling | Matthew Haughn

https://whatis.techtarget.com/definition/over-sampling-and-under-sampling

[3] Feature Selection Techniques in Machine Learning with Python | Raheel Shaikh | Oct 28, 2018

https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

[4] A Simple Introduction to Natural Language Processing | Dr. Michael J. Garbade | Oct 15, 2018

https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32

[5] Predicting hospital length-of-stay at time of admission | Daniel -codes |

https://github.com/daniel-codes/hospital-los-predictor