Prediction of hospital readmission in diabetic inpatients

Data analytics and Data driven decision, 2017/2018

Università degli studi di L'Aquila

Matricola ID № 256921, Stoil Yasenov Yanchev, stoilyanchev@gmail.com
Matricola ID № 256915, Suleyman Erim
Matricola ID № 256916, Sena Er, iamotley7@gmail.com

Prof. Fabrizio Rossi

Prof. Giovanni Felici

2017/2018

Contents

1 Introduction

A hospital readmission is when a patient who is discharged from the hospital, gets re-admitted again within a certain period of time. Hospital readmission rates for certain conditions are now considered an indicator of hospital quality, and also affect the cost of care adversely. Although diabetes is not yet included in the penalty measures, this is a big problem. In 2011, American hospitals spent over 41 billion dollars on diabetic patients who got readmitted within 30 days of discharge. Being able to determine factors that lead to higher readmission in such patients, and correspondingly being able to predict which patients will get readmitted can help hospitals save millions of dollars while improving quality of care. So, with that background in mind, we used a medical claims dataset (description below), to answer these questions:

What factors are the strongest predictors of hospital readmission in diabetic patients?

How well can we predict hospital readmission in this dataset with limited features?

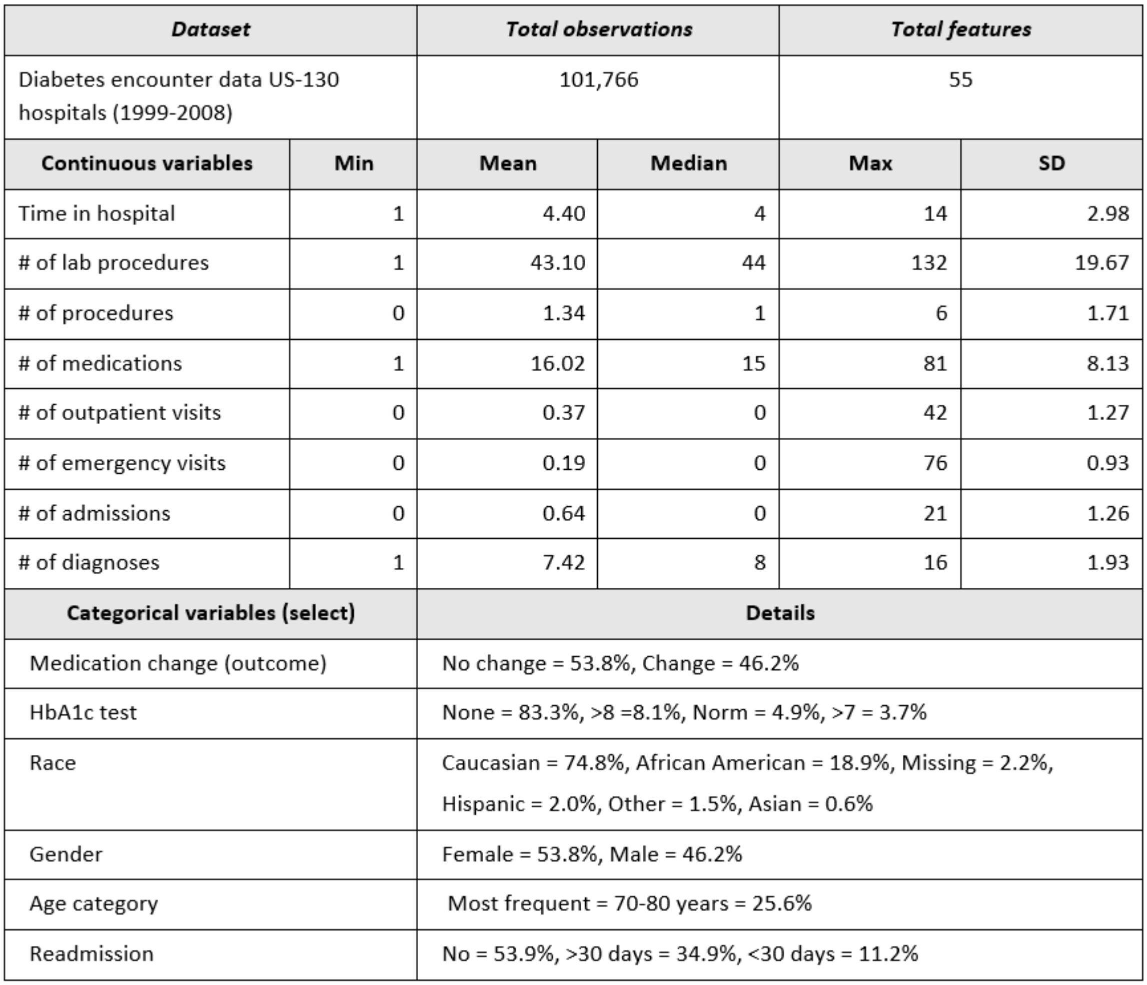
Choosing a dataset

Finding a good dataset is one of the first challenges (besides defining a meaningful question), when trying out machine learning methods. The current state of the healthcare world is such that we can easily find datasets that rich (full of useful information) but dirty (unstructured content or messy schemas) or datasets that are very clean but otherwise sterile in terms of information contained.

With this limitation, we picked a publicly available dataset from UCI repository, containing de-identified diabetes patient encounter data for 130 US hospitals containing 101,766 observations over 10 years. The dataset has over 50 features including patient characteristics, conditions, tests and 23 medications. Only diabetic encounters are included (i.e. at least one of three primary diagnosis was diabetes). This dataset has been used by Strack et al. in 2014 for an interesting analysis on the same topic. So we begin by loading the dataset (csv file downloaded from the link above) as a pandas dataframe.

2 Preprocessing

The first thing to do when picking a dataset is to do some data profiling and look at key features, as we have done in the table below:



Dataset		*Total observations*		*Total features*	
Diabetes encounter data US-130 hospitals (1999-2008)		101,766		55	
Continuous variables	**Min**	**Mean**	**Median**	**Max**	**SD**
Time in hospital	1	4.40	4	14	2.98
# of lab procedures	1	43.10	44	132	19.67
# of procedures	0	1.34	1	6	1.71
# of medications	1	16.02	15	81	8.13
# of outpatient visits	0	0.37	0	42	1.27
# of emergency visits	0	0.19	0	76	0.93
# of admissions	0	0.64	0	21	1.26
# of diagnoses	1	7.42	8	16	1.93
Categorical variables (select)		**Details**			
Medication change (outcome)		No change = 53.8%, Change = 46.2%			
HbA1c test		None = 83.3%, >8 =8.1%, Norm = 4.9%, >7 = 3.7%			
Race		Caucasian = 74.8%, African American = 18.9%, Missing = 2.2%, Hispanic = 2.0%, Other = 1.5%, Asian = 0.6%			
Gender		Female = 53.8%, Male = 46.2%			
Age category		Most frequent = 70-80 years = 25.6%			
Readmission		No = 53.9%, >30 days = 34.9%, <30 days = 11.2%			

What preprocessing and feature engineering techniques should be applied?

Before we can get to actual modeling, some wrangling with the data is almost always needed. We applied three types of methods here:

1. Cleaning tasks such as dropping bad data, dealing with missing values.

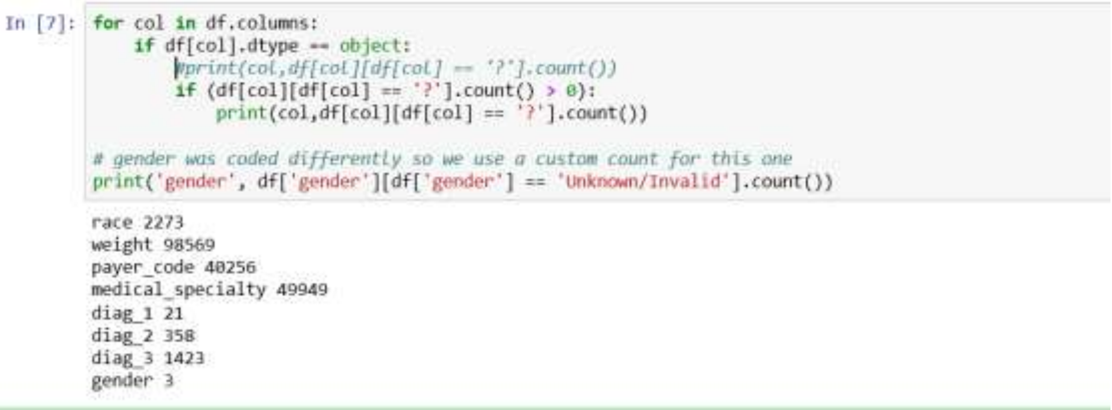
2. Modification of existing features e.g. standardization, log transforms etc.

3. Creation or derivation of new features, usually from existing ones.

The individual steps are described in detail below. However, note that what looks like a nice sequence of steps now, was a result of many trial and error attempts to see what works well for getting our data into best shape.

2.1 Dealing with missing values

This gives us a long list but the following variables had missing values:



```
In [7]: for col in df.columns:
            if df[col].dtype == object:
                #print(col,df[col][df[col] == '?'].count())
                if (df[col][df[col] == '?'].count() > 0):
                    print(col,df[col][df[col] == '?'].count())

        # gender was coded differently so we use a custom count for this one
        print('gender', df['gender'][df['gender'] == 'Unknown/Invalid'].count())
```

```
race 2273
weight 98569
payer_code 40256
medical_specialty 49949
diag_1 21
diag_2 358
diag_3 1423
gender 3
```

Now the important part—deciding what to do:

Weight is missing in over 98% records. Owing to the poor interpretability of missing values and little predictive generalizability to other patients, best thing is to just drop it.

Medical Specialty of treating physician also have 40–50% missing values. We decided to drop this, but there are other ways too to deal with such missing values.

Primary (diag_1), Secondary (diag_2) and Additional (diag_3) diagnoses were have very few missing values. Technically, if all three are missing, that’s bad data. So we only drop those records where all three diagnoses are missing.

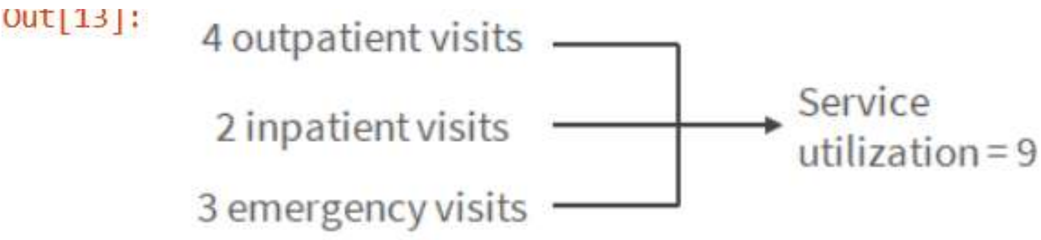
Gender has only 3 missing or invalid values so we decided to drop these records.

Also, one more cleaning step that depends on understanding the data and some common sense: since we are trying to predict readmissions, those patients who died during this hospital admission, have zero probability of readmission. So we should remove those records (discharge_disposition = 11).

2.2 Creating and/or Recoding New Features

This is highly subjective, and partly depends on a knowledge of healthcare services, and making sense of the potential relationships between features. There are perhaps thousands of ways to try here. We tried some (none are perfect) and here's why.

Service utilization: The data contains variables for number of inpatient (admissions), emergency room visits and outpatient visits for a given patient in the previous one year. These are (crude) measures of how much hospital/clinic services a person has used in the past year. We added these three to create a new variable called service utilization (see figure below). The idea was to see which version gives us better results. Granted, we did not apply any special weighting to the three ingredients of service utilization but we wanted to try something simple at this stage.



Number of medication changes: The dataset contains 23 features for 23 drugs (or combos) which indicate for each of these, whether a change in that medication was made or not during the current hospital stay of patient. Medication change for diabetics upon admission has been shown by previous research to be associated with lower readmission rates. We decided to count how many changes were made in total for each patient, and declared that a new feature. The reasoning here was to both simplify the model and possibly discover a relationship with number of changes regardless of which drug was changed.

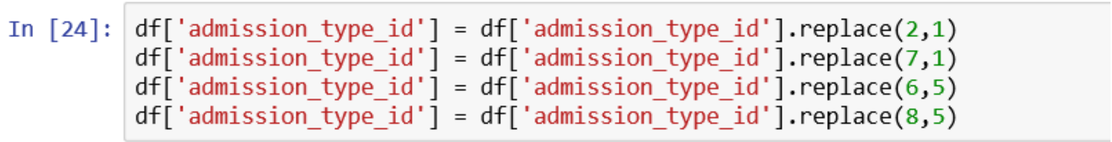
Number of medication used: Another possibly related factor could be the total number of medications used by the patient (which may indicate severity of their condition and/or the intensity of care). So we created another feature by counting the medications used during the encounter.

2.3 Categorization of diagnoses

The dataset contained up to three diagnoses for a given patient (primary, secondary and additional). However, each of these had 700–900 unique ICD codes and it is extremely difficult to include them in the model and interpret meaningfully. Therefore, we collapsed these diagnosis codes into 9 disease categories in an almost similar fashion to that done in the original publication using this dataset. These 9 categories include Circulatory, Respiratory, Digestive, Diabetes, Injury, Musculoskeletal, Genitourinary, Neoplasms, and Others. Although we did this for primary, secondary and additional diagnoses, we eventually decided to use only the primary diagnosis in our model.

2.4 Collapsing some other variables

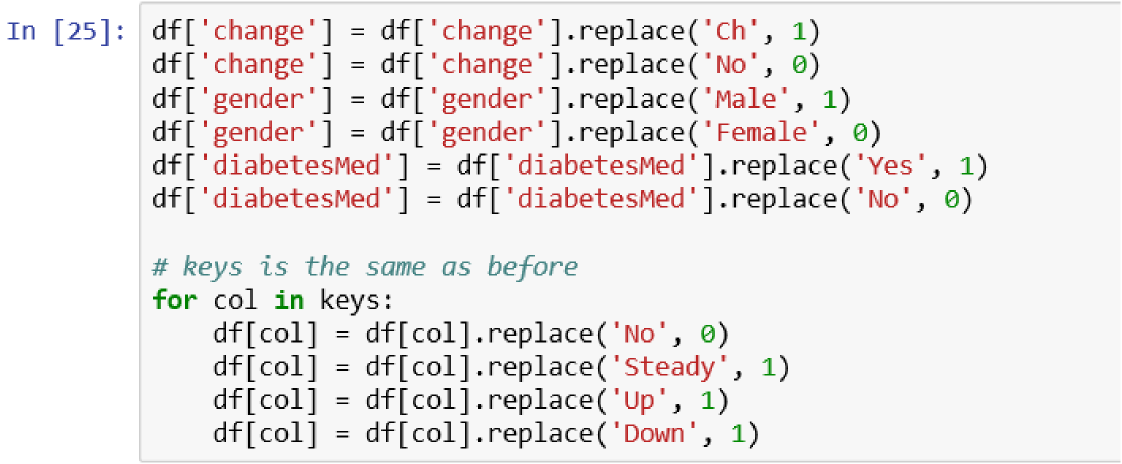
Just like diagnoses, there were quite a few categories for admission source, admission type and discharge disposition. We collapsed these variables into fewer categories where it made sense. For example, admission types 1, 2 and 7 correspond to Emergency, Urgent Care and Trauma, and thus were combined into a single category as these are all non-elective situations.



```
In [24]: df['admission_type_id'] = df['admission_type_id'].replace(2,1)
         df['admission_type_id'] = df['admission_type_id'].replace(7,1)
         df['admission_type_id'] = df['admission_type_id'].replace(6,5)
         df['admission_type_id'] = df['admission_type_id'].replace(8,5)
```

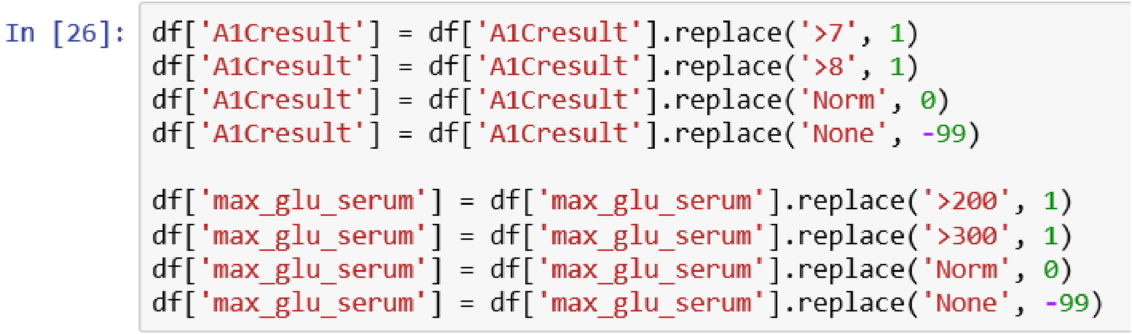
2.5 Recoding some variables

The original dataset used string values for gender, race, medication change, and each of the 23 drugs used. To better fit those variables into our model, we interpret the variables to numeric binary variables to reflect their nature. For example, we encoded the “medication change” feature from “No” (no change) and “Ch” (changed) into 0 and 1.



```
In [25]: df['change'] = df['change'].replace('Ch', 1)
         df['change'] = df['change'].replace('No', 0)
         df['gender'] = df['gender'].replace('Male', 1)
         df['gender'] = df['gender'].replace('Female', 0)
         df['diabetesMed'] = df['diabetesMed'].replace('Yes', 1)
         df['diabetesMed'] = df['diabetesMed'].replace('No', 0)

         # keys is the same as before
         for col in keys:
             df[col] = df[col].replace('No', 0)
             df[col] = df[col].replace('Steady', 1)
             df[col] = df[col].replace('Up', 1)
             df[col] = df[col].replace('Down', 1)
```

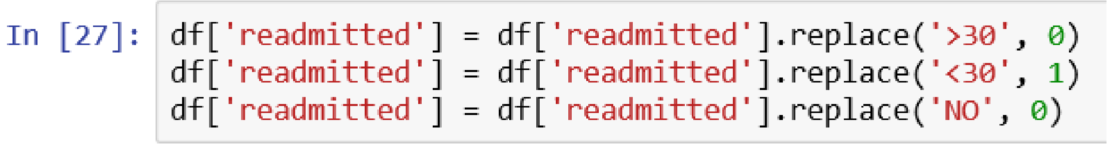
We also reduced both A1C test result and Glucose serum test result into categories of Normal, Abnormal and Not tested.



```
In [26]: df['A1Cresult'] = df['A1Cresult'].replace('>7', 1)
         df['A1Cresult'] = df['A1Cresult'].replace('>8', 1)
         df['A1Cresult'] = df['A1Cresult'].replace('Norm', 0)
         df['A1Cresult'] = df['A1Cresult'].replace('None', -99)

         df['max_glu_serum'] = df['max_glu_serum'].replace('>200', 1)
         df['max_glu_serum'] = df['max_glu_serum'].replace('>300', 1)
         df['max_glu_serum'] = df['max_glu_serum'].replace('Norm', 0)
         df['max_glu_serum'] = df['max_glu_serum'].replace('None', -99)
```

2.6 Recoding the outcome variable

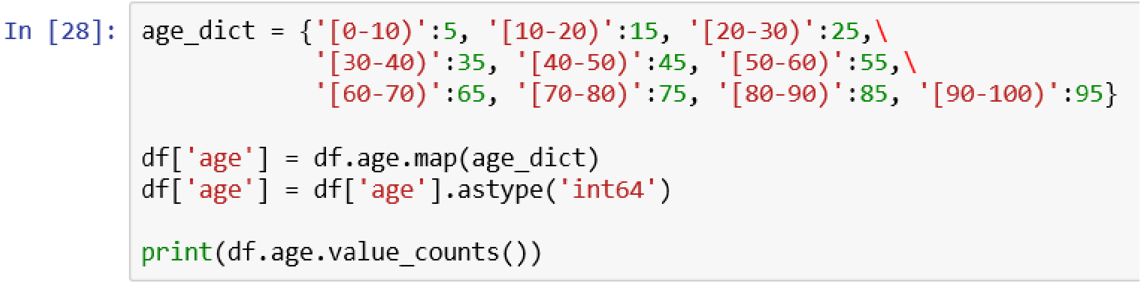
The outcome we are looking at is whether the patient gets readmitted to the hospital within 30 days or not. The variable actually has < 30, > 30 and No Readmission categories. To reduce our problem to a binary classification, we combined the readmission after 30 days and no readmission into a single category:



```
In [27]: df['readmitted'] = df['readmitted'].replace('>30', 0)
         df['readmitted'] = df['readmitted'].replace('<30', 1)
         df['readmitted'] = df['readmitted'].replace('NO', 0)
```

2.7 Dealing with age

There are different ways to deal with this. The dataset only gives us age as 10 year categories, so we don't know the exact age of each patient. The previous study on this dataset used age categories as nominal variables, but we wanted to be able to see the effect of increasing age on readmission, even if in a crude way. To do that, we assume that age of the patient on average lies at the midpoint of the age category. For example, if the patient's age category is 20–30 years, then we assume the age = 25 years. So we converted age categories to midpoints, resulting in a numeric variable:



```
In [28]: age_dict = {'[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}

         df['age'] = df.age.map(age_dict)
         df['age'] = df['age'].astype('int64')

         print(df.age.value_counts())
```

2.8 Collapsing of Multiple Encounters for same patient

Some patients in the dataset had more than one encounter. We could not count them as independent encounters because that bias the results towards those patients who had multiple encounters. Thus we tried multiple techniques to collapse and consolidate multiple encounters for same patient such as:

1. Considering more than 2 readmissions across multiple encounters as readmission for collapsed record.

2. Considering average stay at hospital across multiple encounters.

3. Considering the percentage of the medication changes across multiple encounters

4. Considering the total number of the encounters to replace the encounter unique ID

5. Considering the combination of diagnoses across multiple encounters as a list

2.9 Log Transformation

A preliminary analysis of our numerical features revealed that many of these were highly skewed and had high kurtosis. As a reference, the skew of a normal distribution is 0 and the excess kurtosis (difference of actual kurtosis from ideal normal distribution value of 3), as returned by the kurtosis() function for a normal distribution is 0, which would impact standardization. Features such as number of emergency visits, service utilization, number of inpatient admissions and number of outpatient visits had high skew and kurtosis. Thus, we performed log transformation where a skew or kurtosis beyond the limits of -2 ≤ skew and kurtosis ≤ 2. Also, since log (0) is not defined, we decided to use the following rule:

1. Compute log(x) for any feature x if percentage of 0s in x ≤ 2%, after removing the zeros. This ensured that we didn’t bulk-remove records that hold predictive power for other columns.

2. Compute log1p(x) otherwise (log1p(x) means log(x+1), while retaining the zeros.

2.10 Standardization

Since we had used log transformation to ensure that the numeric variables had a Gaussian-like or normal distribution (before the log transformation) or were log transformed to ensure a normal distribution, we decided to standardize our numerical features using the formula:

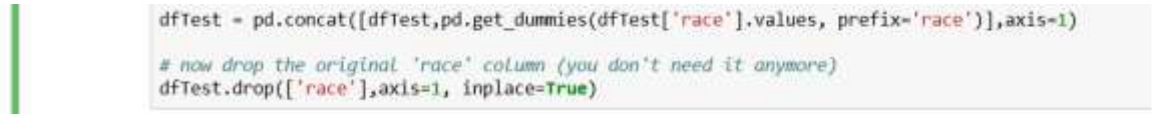
$$New\ value = \frac{Value - Mean(Values)}{Standard\ Deviation\ (Values)}$$

2.11 Using one hot encoding columns

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

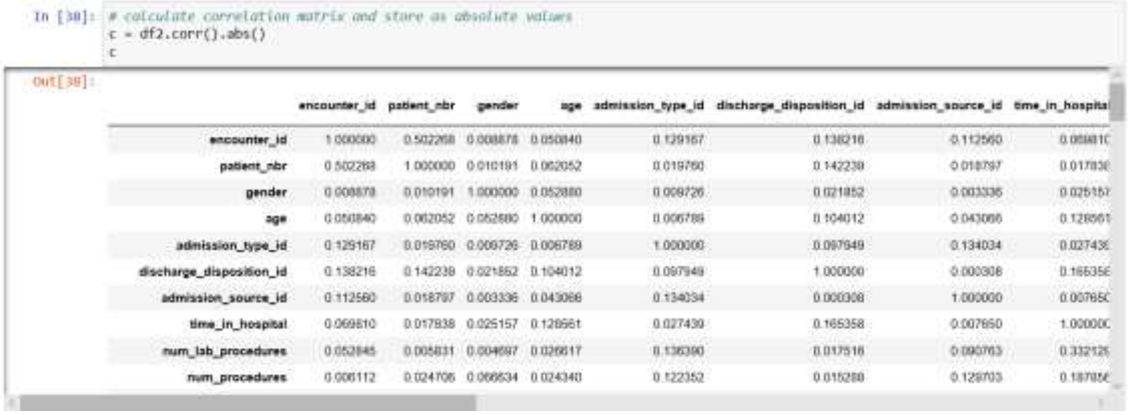
In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.



```
dfTest = pd.concat([dfTest,pd.get_dummies(dfTest['race'].values, prefix='race')],axis=1)

# now drop the original 'race' column (you don't need it anymore)
dfTest.drop(['race'],axis=1, inplace=True)
```

3 Interaction Terms

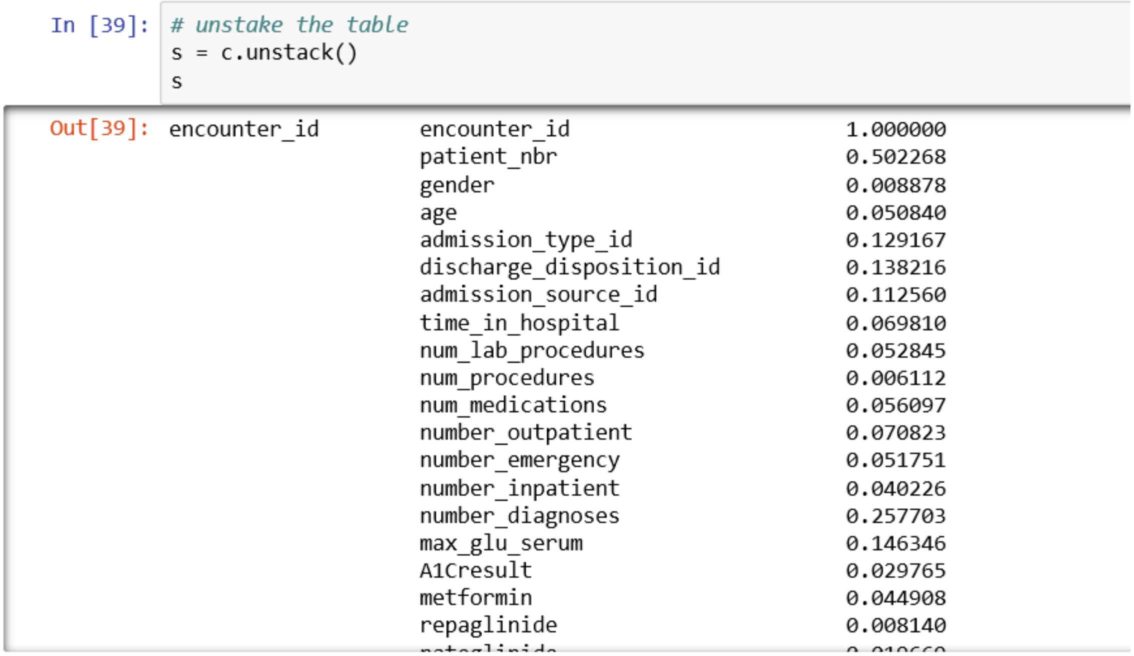
Variables can have interdependent effects on readmission, called interactions. We can identify possible candidates for interaction terms by looking at what makes theoretical sense and by observing a correlation matrix of the predictor variables to see which ones seem highly correlated.



```
In [38]: # calculate correlation matrix and store as absolute values
         c = df2.corr().abs()
         c
```

Out[38]:

	encounter_id	patient_nbr	gender	age	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospita
encounter_id	1.000000	0.502268	0.008878	0.050840	0.129167	0.138216	0.112560	0.06981
patient_nbr	0.502268	1.000000	0.010191	0.062052	0.019760	0.142239	0.018797	0.01783
gender	0.008878	0.010191	1.000000	0.052880	0.009726	0.021852	0.003336	0.02515
age	0.050840	0.062052	0.052880	1.000000	0.006789	0.104012	0.043066	0.12856
admission_type_id	0.129167	0.019760	0.009726	0.006789	1.000000	0.097949	0.134034	0.02743
discharge_disposition_id	0.138216	0.142239	0.021852	0.104012	0.097949	1.000000	0.000308	0.16535
admission_source_id	0.112560	0.018797	0.003336	0.043066	0.134034	0.000308	1.000000	0.00765
time_in_hospital	0.069810	0.017838	0.025157	0.128561	0.027439	0.165358	0.007650	1.00000
num_lab_procedures	0.052845	0.005831	0.004697	0.026617	0.136390	0.017516	0.090763	0.33212
num_procedures	0.006112	0.024706	0.066634	0.024340	0.122352	0.015288	0.129703	0.18785



```
In [39]: # unstake the table
         s = c.unstack()
         s

Out[39]: encounter_id        encounter_id                        1.000000
                             patient_nbr                         0.502268
                             gender                              0.008878
                             age                                 0.050840
                             admission_type_id                   0.129167
                             discharge_disposition_id            0.138216
                             admission_source_id                 0.112560
                             time_in_hospital                    0.069810
                             num_lab_procedures                  0.052845
                             num_procedures                      0.006112
                             num_medications                     0.056097
                             number_outpatient                   0.070823
                             number_emergency                    0.051751
                             number_inpatient                    0.040226
                             number_diagnoses                    0.257703
                             max_glu_serum                       0.146346
                             A1Cresult                           0.029765
                             metformin                           0.044908
                             repaglinide                         0.008140
                             nateglinide                         0.010660
```

In this list, there are two kinds of situations:

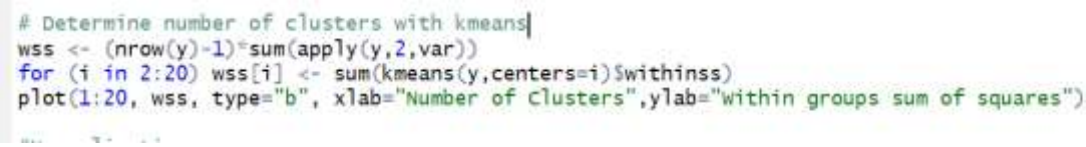
1. One variable is contained in/derivative of another: In the above examples, number of outpatient visits is part of service utilization. However, we created this feature ourselves, so in this case our decision is to not put these in same feature set (we used two feature sets described later). Similarly, diabetesMed (any diabetic medication prescribed) is contained in the number of medications used. We decided to drop diabetesMed from analysis in this case because, well, it is understood that all of these patients are getting some diabetic medication.

2. Possible actual co-variance: The other situation is an actual co-variance between two variables. This seems to be the case for number of medication and whether a change was made or not, and it also makes some intuitive sense. So we created interaction terms for such cases.

4 Unsupervised Learning

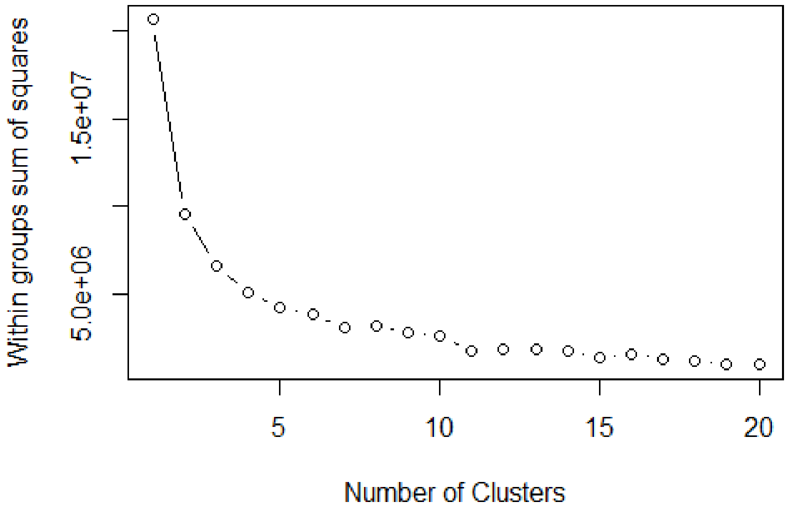
As an unsupervised learning method, K means clustering is chosen. K means clustering is easier to implement, cheaper and easier to understand compared to other unsupervised learning methods. It can work with huge datasets. However, it is sensitive to outliers. In our dataset, K means is used to give an understanding of data and to see how we can group the dataset. To use K means clustering we need an unlabeled data, and aim is to find groups in the data. The data is clustered according to feature similarities. K means is done by calculating local sum of squares and putting each data point one-by-one.

K means clustering is only used with number, therefore we only implemented K means with numerical variables. Not all the data should be grouped, there are some data which cannot be grouped well. Therefore, we only choose the variables which can be grouped better. To make it PCA (principle component analysis) is used to decide the number of clusters within sum of squares graph is used.

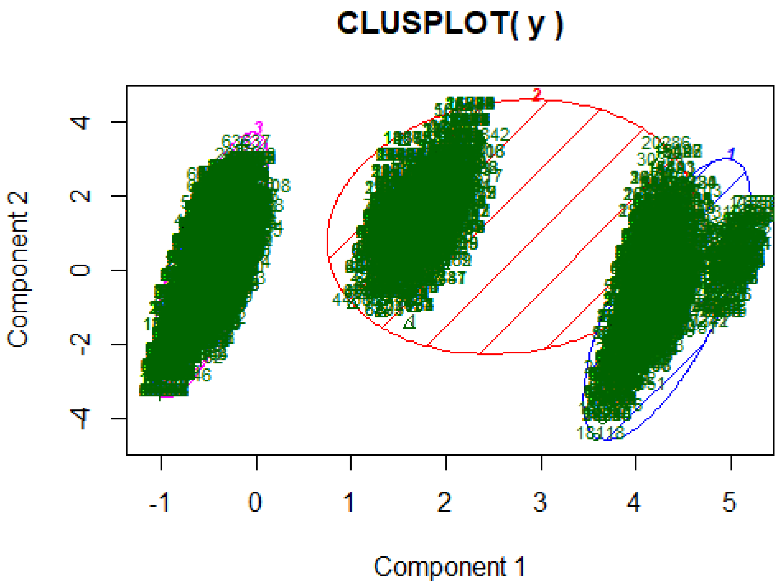


```
# Determine number of clusters with kmeans
wss <- (nrow(y)-1)*sum(apply(y,2,var))
for (i in 2:20) wss[i] <- sum(kmeans(y,centers=i)$withinss)
plot(1:20, wss, type="b", xlab="Number of Clusters",ylab="Within groups sum of squares")
```

y is the chosen variable set.



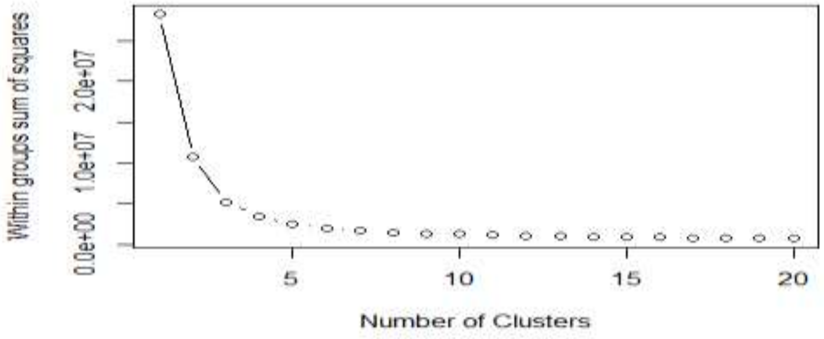
The graph above includes the variables: age, discharge_disposition_id, admission_source_id and readmitted. By looking at graph, we can use 3 clusters. Which explains 61% of point variability. Clusters and points seem to be well distributed.



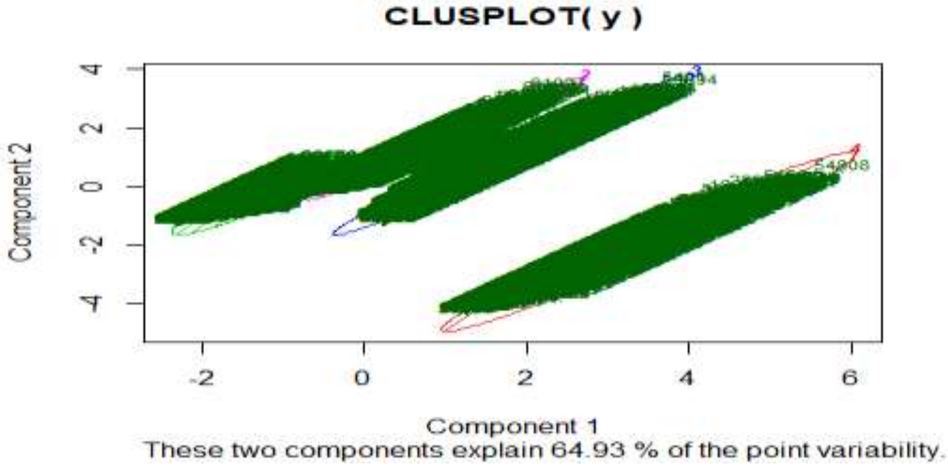
These two components explain 61.58 % of the point variability.

According to clusplot, age variable has a huge impact on clusters such as lower ages, middle ages and higher ages. With the increase in age, the possibility of readmission increases.

The next variables are: time_in_hospital, number_lab_procedures, number_procedures and readmitted. According to graph, 4 clusters are chosen.



Here in the second graph with 4 clusters, the components explain 64.93%-point variability.



According to clusplot, the likelihood of readmission increases with inclement of time spent in hospital and lab procedures.

5 Supervised Learning

We used logistic regression and decision trees as supervised learning methods. We have found that decision trees work better in this setting (problem, dataset etc.)

5.1 Motivations on choosing the methods

The hint is that reducing the readmission rate is a present concern and one of the reasons why we have chosen our problem. Therefore, we did not want to just focus on the accuracy but also what insights can these methods give us.

Hence, we have chosen logistic regression and decision trees. Logistic regression can help us understand the relative impact and statistical significance of each factor on the probability of readmission. On the other hand, decision trees is a method that comes to mind easily since it is intuitively similar to human-like decision making processes that are used in similar problems. With decision trees, we can observe the level of certainty iteratively and hierarchically. These observations give us valuable insights.

5.2 What we have tried? – A brief summary

5.2.1 LOGISTIC REGRESSION

First, we have used logistic regression. We used 80% of our data as training sample, and 20% as test sample and we shuffle the samples as seen in the below code.

```
X_train, X_dev, Y_train, Y_dev = train_test_split(train_input_new,
train_output_new, test_size=0.20, random_state=0)
```

We used sklearn's train_test_split method. Since shuffle parameter's default value is True it is not seen in the above code.

We have used 10-fold cross validation

```
logreg = LogisticRegression(fit_intercept=True, penalty='l1')

logreg.fit(X_train, Y_train)
```

The output is

Cross Validation Score: 61.10%

Dev Set score: 60.62%

5.2.2 DECISION TREES

First, we needed to do data balancing. Data was highly imbalanced with respect to readmissions (only 10% records for 30-day readmissions), leading to high accuracy. We used synthetic minority over-sampling technique (SMOTE) to oversample our underrepresented class of readmissions and obtain equal representation of our overrepresented and underrepresented classes.

We used the decision tree implementation of scikit-learn. The gain function we used is “entropy”, maximum depth of the tree is 28. We used 10-fold cross validation

```
dte = DecisionTreeClassifier(max_depth=28, criterion = "entropy", min_samples_split=10)
```

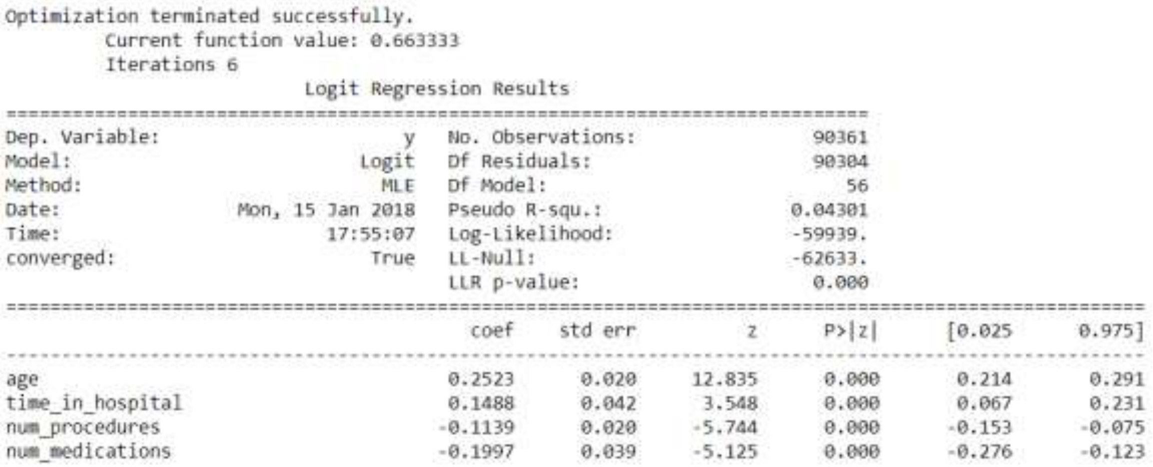
The cross validation score is found 91.03%. Accuracy is %91.

5.3 Digging deeper – Understanding the models

5.3.1 LOGISTIC REGRESSION

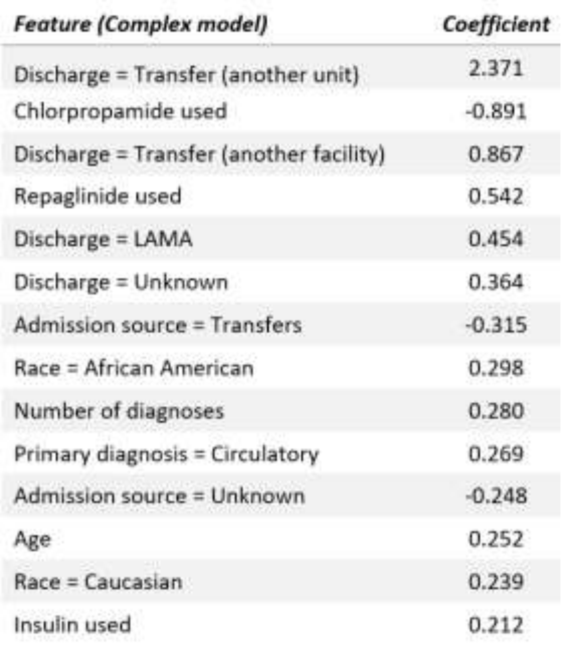
Below, we see the code that is responsible for the summary of logistic regression and its output. The table gives insights to the impact of factors and their interactions.

print(result.summary())



```
Optimization terminated successfully.
         Current function value: 0.663333
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                90361
Model:                          Logit   Df Residuals:                    90304
Method:                           MLE   Df Model:                           56
Date:                Mon, 15 Jan 2018   Pseudo R-squ.:                 0.04301
Time:                        17:55:07   Log-Likelihood:                -59939.
converged:                       True   LL-Null:                       -62633.
                                        LLR p-value:                     0.000
=======================================================================================================
                                          coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------------
age                                     0.2523      0.020     12.835      0.000       0.214       0.291
time_in_hospital                        0.1488      0.042      3.548      0.000       0.067       0.231
num_procedures                         -0.1139      0.020     -5.744      0.000      -0.153      -0.075
num_medications                        -0.1997      0.039     -5.125      0.000      -0.276      -0.123
```

Since there are too many variables and coefficients to look at, we can make it a little easier by picking only those coefficients that have p-value < 0.01 (i.e. statistically significant) and have at least 0.2 magnitude.



Feature (Complex model)	*Coefficient*
Discharge = Transfer (another unit)	2.371
Chlorpropamide used	-0.891
Discharge = Transfer (another facility)	0.867
Repaglinide used	0.542
Discharge = LAMA	0.454
Discharge = Unknown	0.364
Admission source = Transfers	-0.315
Race = African American	0.298
Number of diagnoses	0.280
Primary diagnosis = Circulatory	0.269
Admission source = Unknown	-0.248
Age	0.252
Race = Caucasian	0.239
Insulin used	0.212

As a general note for logistic regression coefficients reported, the interpretation is to be done while considering both the nature of logistic prediction (in terms of odds) and the transformations we have applied to the data before modeling. For example, in interpreting the effect of age on readmissions, we may follow these steps:

EXP (Coefficient of age) = EXP (Log of unit odds change)

= EXP (0.25) = 1.28

But remember Age was standardized and 1 SD of Age = 15 years

Therefore:

For every 15 years increase in age, there is 28% increase in Odds of being readmitted versus not being readmitted!

Some of the insights we have from these coefficients:

- The strongest predictors of readmission within 30 days appear to be four types of discharge conditions in both versions. Intuitively, these make sense — transfer to another unit in a hospital or another hospital may indicate higher severity/complexity of disease and make readmission likely.

- Interestingly, patients who Left Against Medical Advice (LAMA) are also likely to be readmitted, perhaps because their condition was not fit for discharge in the first place.

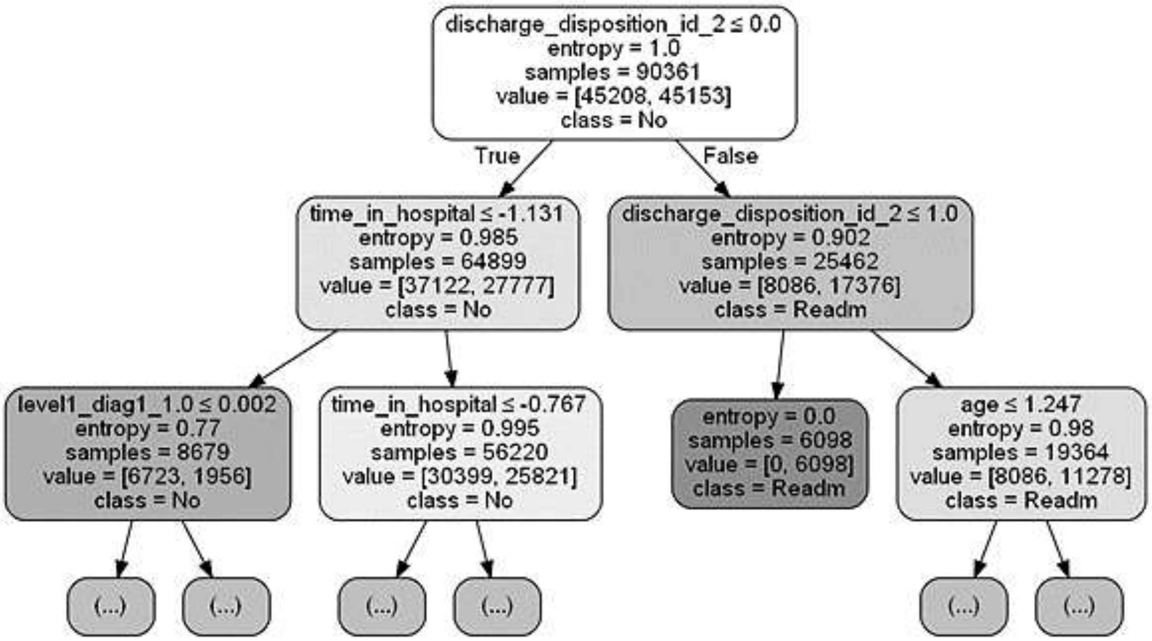
- The effect of race being African American is another example where caution must be exercised in interpretation and conclusion. It would be not appropriate to assume differential treatment based on race because it could be very well due to environmental, genetic or other factors that we have not even measured here.

- Use of Repaglinide and Insulin appear to increase the odds of readmission while Chlorpropamide usage decreases the odds. But, the usage of these drugs can be very situation specific so it is hard to make conclusions about specific drugs here.

5.3.2 DECISION TREE

We can visualize the decision tree to understand the ordinality of features used in our decision making. Below, we see the output of the code we used to visualize our decision tree with the help of the library called GraphViz. (Here only 2 levels are showed)

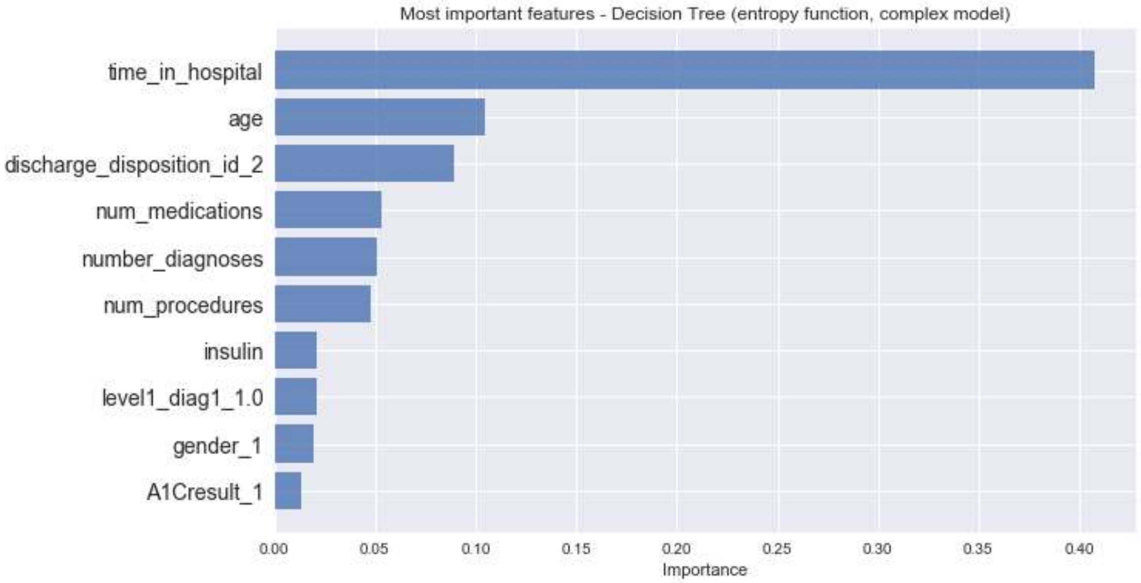
```
dot_dt_q2 = tree.export_graphviz(dte, out_file="dt_q2.dot", feature_names=X_train.columns, max_depth=2, class_names=["No","Readm"], filled=True, rounded=True, special_characters=True)

graph_dt_q2 = pydotplus.graph_from_dot_file('dt_q2.dot')

Image(graph_dt_q2.create_png())
```



Looking at this, we can tell that the first feature used in deciding whether a patient will get readmitted or not, is whether the patient is discharged to another hospital or facility (discharge_disposition_id_2… category). Next level of the tree shows this same feature repeated on one hand and days spent in hospital on the other.

An easier alternative way is to arrange the features by their “importance”.

```
# Shot top most features based on importance

feature_names = X_train.columns

feature_imports = dte.feature_importances_

most_imp_features = pd.DataFrame([f for f in zip(feature_names,feature_imports)], columns=["Feature", "Importance"]).nlargest(10, "Importance")

most_imp_features.sort_values(by="Importance", inplace=True)

plt.figure(figsize=(10,6))

plt.barh(range(len(most_imp_features)), most_imp_features.Importance, align='center', alpha=0.8)

plt.yticks(range(len(most_imp_features)), most_imp_features.Feature, fontsize=14)

plt.xlabel('Importance')

plt.title('Most important features - Decision Tree (entropy) (Question 2 - complex model)')

plt.show()
```

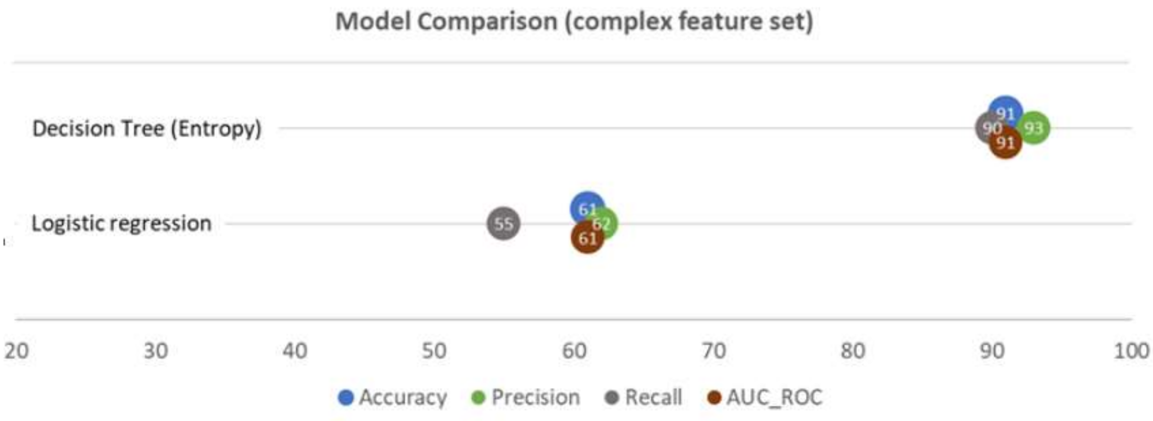


We see that the feature time_in_hospital is the biggest clue we have in guessing whether a patient is getting readmitted or not.

6 Conclusions

Our decision tree model indicates highest importance of time spent in hospital, age and discharge to another hospital for both simple and complex versions. If we plot these against the coefficients from logistic regression, they do not correlate well. Since we used cross-validation and achieved similar test and train accuracy to avoid overfitting, this may suggest a different correlation structure of the variables in the model or lower explained variance in logistic model.

To compare overall model performance metrics for different models, we collected all the metrics and created a chart shown below. As you can tell, the difference between performances of logistic versus tree based models is remarkable. Besides accuracy, recall is important here since hospitals get penalized and incur additional costs both for the patient and the insurance agencies if a patient expected not to be readmitted shows up in 30 days.



Machine learning model comparison for hospital readmission prediction

We can say something about the overall process:

Considering accuracy of predicting outcomes, tree based models are clearly outperforming logistic regression, perhaps because decision boundaries are non-linear or there are complex time-sequence dependent interactions between the things happen to a patient during a hospital stay.

Machine learning techniques can allow nuanced analysis of predictors. For example, we see that the regression coefficient is highest for a certain type of discharge, but time spent in hospital is the most important feature when using decision tree. However, visualizing the tree shows that both of these features appear above/below each other, which indicates high interaction. One might then include this interaction term in the logistic model and see improvement. For a hospital manager, this means that patients who are likely to stay longer and then get discharged to another unit, are highly likely to get readmitted.