Data Science PNP 2019 — Final Assignment

Simeon Stoykov

***Abstract*— This report will examine a sample of the "DIABETES 130-US HOSPITALS FOR YEARS 1999-2008" multivariate data set. It will try to predict whether and when patients are *readmitted*.**

I. DATA EXPLORATION

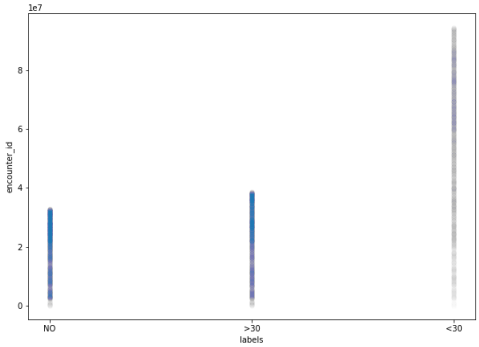
A. Overview

The sample dataset contains 9999 rows with 50 columns each, including the response variable *readmitted*. Description of the features could be found in the references [1]. It contains 37 categorical and 13 numerical features. Their according distributions are presented in Appendix I.

B. Representativity of the sample used

For the purpose of this assignment, a smaller sample of the original data set is used. The full one contains 101766 rows, from which about 10% are selected. In order to make coherent conclusions, the following facts should be noted:

- The rows in the sample were chosen so that the response variable values are of equal proportions (a third)
- Features like *acetohexamide*, *glipizide-metformin* and others have only 1 unique value in the sample
- The *encounter_id* feature is, surprisingly, quite a strong indicator of the readmission outcome. The reason is that in the sampled dataset, a lot of the not readmitted *encounter_id*s, together with those readmitted after more than 30 days, come before *encounter_id*s of patients readmitted within less than 30 days. This could be seen in the plot below:



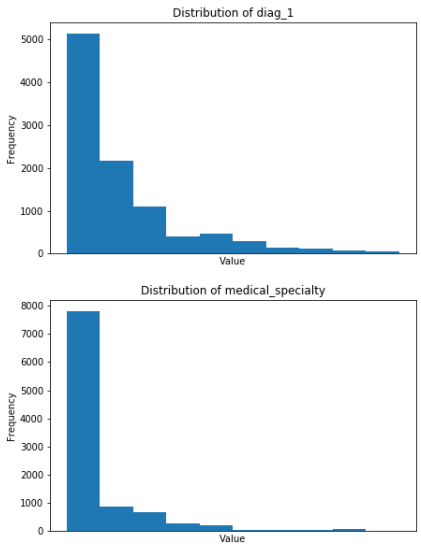
A possible reason is that the sample has been obtained by the following algorithm:

1) Agree on a balanced number of rows for each readmission outcome (3333 in this case)
2) Examine the original rows sorted by *encounter_id*, and include each one in the sample, as long as the count of the readmission value doesn't exceed the target

Due to the fact that in the original dataset 53% of the samples have readmission value *NO*, this group was quickly filled with smaller *encounter_id*s.

C. Categorical features with high cardinality

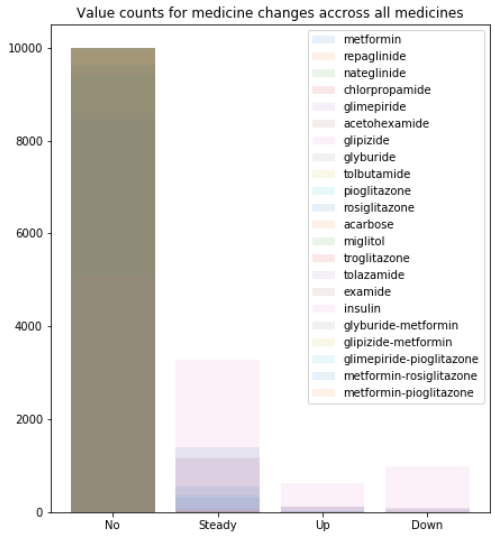
Features with a large number of different values are *medical_specialty* (about 50) and *diag_1*, *diag_2*, *diag_3* (more than 400 each). As categorical values are often encoded in an one-hot manner, that would introduce a lot of new binary features, which make model training and prediction slower. Therefore, it's worth examining the distributions of those features:



Both exhibit high **kurtosis**, hence most of the low-frequency values in the tails could be combined into a single "other" category

D. Binary features with high cardinality

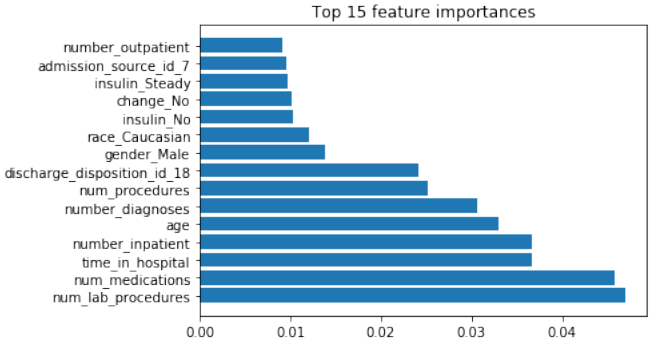
Features that signify changes in medicine dosage are also worth examining. The following plot shows the proportion of the values they have:



Darker shades correspond to bigger density. As it can be seen, most of the time the drug was not taken (value *NO*), followed by a significantly smaller number of cases in which the dosage was kept the same. Those properties were further explored in section II.

E. Important features

In order to get a basic understanding of which features matter, it's worth making some basic data transformations and running a simple Random Forest model in order to get the feature importances. The results below were obtained by dropping *weight, payer_code, encounter_id, patient_nbr*, putting an *other* category in the place of missing values for features *medical_specialty, race, diag_1, diag_2, diag_3*. Furthermore, all categorical values, together with *admission_type_id, discharge_disposition_id, admission_source_id* were encoded with a one-hot approach. These tranformations are rather intuitive; they have been elaborated on in section II:



II. FEATURE ENGINEERING

A. Handling of missing and redundant variables

As mentioned before, some of the features have only 1 unique value in the sampled dataset. The complete list of those features are: *acetohexamide, examide, citoglipton, glipizide-metformin, glimepiride-pioglitazone, metformin-rosiglitazone, metformin-pioglitazone*. Because they carry no information, they should be removed.

Another case of redundant variables are *encounter_id* and *patient_nbr*. They are supposed to be unique identifiers, having nothing to do with our prediction. The best we could use them for, in the case of *patient_nbr*, is to analyse consecutive visits by the same patient. I only kept the first visit of each patient, because consecutive ones are not independent. After doing that, I removed both features.

Examining the values of the *discharge_disposition_id* feature, we observe that ids

equal to one of *11, 13, 14, 19, 20, 21* are related to death or discharged to hospice. In those cases further readmission is not possible, so our models might learn to give them a lot of significance. Even though this might increase the accuracy on paper, we are not expecting it would find any practical application in actual real world predictions, as it is trivial to guess that. Hence, rows having those values for *discharge_disposition_id* should be removed.

Furthermore, there are some features that have a lot of missing values - *weight* and *payer code* are missing in more than two-thirds of the cases. That's too much to make use of them, so they were removed from the feature set. What's more, we don't expect *payer code* to have a lot of significance for our predictions, whereas *weight* could have been useful. For that reason, a new *has_weight* binary feature was introduced, which shows whether *weight* is present or not. Keeping track of the weight might correspond to the hospital being more thorough, and hence exhibiting smaller number of readmissions.

Features with smaller proportion of missing values are *medical_specialty* and *race*. We expect them to be important in our predictions of readmission, so we wouldn't want to discard them. The approach that was taken is to create a new *unknown* category for each. In the case of *race*, just 2% are missing, so that wouldn't create a lot of distortion. For *medical_specialty*, missing values could even be a clue when making a prediction.

B. One-hot encoding of categorical variables

Most of the categorical values in the data set are **nominal** — there is no ordering between different values (e.g male and female). The best way to encode them is by using a one-hot approach. That is, we create a new feature for each possible value, being 1 if the current row is of this category. This is the case for: *race, gender, medical_specialty, max_glu_serum, A1Cresult, metformin, repaglinide, nateglinide, chlorpropamide, glimepiride, glipizide, glyburide, tolbutamide, pioglitazone, rosiglitazone, acarbose, miglitol, troglitazone, tolazamide, insulin, glyburide-metformin, change, diabetesMed*.

Even though *admission_type_id*, *discharge_disposition_id'* and *admission_source_id* are not categorical, their meaning is — there is no ordering between ids that we know of. Hence, they were also encoded with the one-hot approach.

With *age* we observe the opposite — even though it is categorical in the data set, we could use its numerical properties. The approach that was taken, is to convert each age category of the form [X, Y) to the mean of X and Y (e. g [0, 10) becomes 0.5).

C. Binary medicine-change features

As discussed in I-D, it's worth experimenting with binary medicine features. A few approaches were tested by running a 5-fold cross validation with a toy Logistic Regression model on the current state of the data set:

- Keep them as they are (190 features, accuracy 0.520)
- Remove all binary medicine features, since most of them have a *NO* value (157 features, accuracy 0.519)
- Introduce counts for how many types of medicine were kept *Steady*, went *Up* or *Down*, or were not prescribed at all, and discard the original ones (161 features, accuracy 0.520)
- Keep the old and the new features, giving the models the freedom to decide which ones to use (194 features, accuracy 0.520)

I decided to keep both the old and the new features, because the increase in their number is not too big, whilst still giving some more flexibility to our model.

D. Diagnosis category features

The issue with *diag_1, diag_2, diag_3* is similar and was mentioned in I-C. Four approaches were considered, and their feature set sizes and cross validation accuracy analysed:

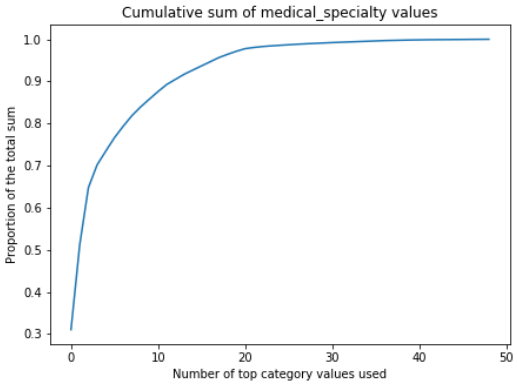
- Leave them as they are, and let the models handle them (1461 features, 0.500 accuracy)
- Remove them completely (137 features, 0.506 accuracy)
- Keep the top 95% of the features as described in I-C (707 features, 0.503 accuracy)
- The values are ICD-9 codes of medical diagnoses, which could be arranged into 18 groups of related diagnoses [2]. This brings the number of features down, whilst preserving the

information they carry (190 features, 0.503 accuracy)

I chose to compress them using their ICD-9 codes, because it results in less features carrying similar amount of information.

E. Compression of the medical specialty feature

As discussed in I-C, *medical_specialty* has 50 unique values, but by using the first 30 of them, and grouping the others, we could still use the original values in 99% of the cases. This is visualised below:



By doing this we reduce the number of the features, whilst preserving the meaning they carry. This ensures that our models run faster and give better results.

F. Summary

Performing those transformations results in a data set containing 7831 rows of 170 features, including the response variable.

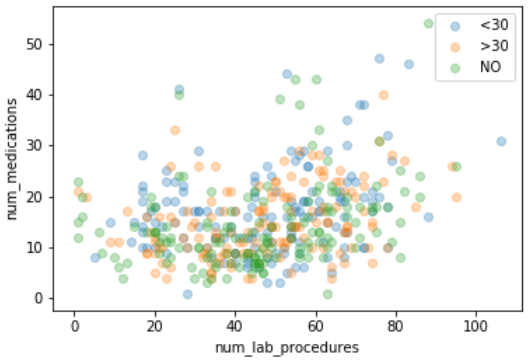
III. MACHINE LEARNING ALGORITHMS

The response variable has a discrete domain, therefore the problem posed is an example of **classification**, as opposed to regression. There are three possible values — *NO, >30, <30*. Therefore a baseline estimator that guesses randomly would achieve an accuracy of a third — anything above that is a success.

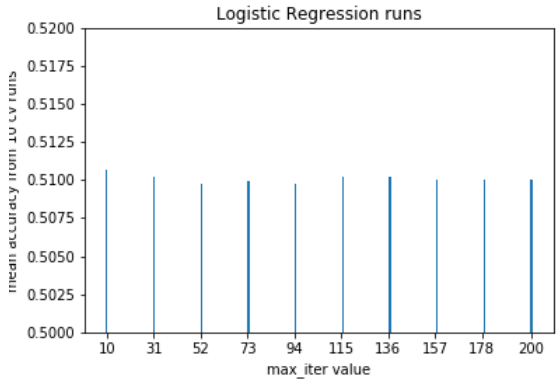
Since many models are sensitive to differences in feature variance, standard scaling was applied to all independent variables, using sklearn's *StandardScaler*. The data was split into a training (80%) and testing (20%) parts with balanced number of each response category, using *StratifiedShuffleSplit*.

A. Logistic Regression

Plain Logistic Regression works best in cases in which the data is close to being linearly separable. That is, there is a hyper-plane of dimensionality one less than the number of features, that divides the response classes. It's difficult to judge in our case, when the number of features is in the hundreds. Plotting the top 2 most important features, given in I-E, against each other, yields:



There is no visible separation — linear or non-linear. Nevertheless, Logistic Regression still gives relatively good results:



Even though it gives no-convergence warnings for values of *m_iter* less than 150, it produces similar results when varying the maximum number of iterations allowed, below or above that number.

Logistic Regression tries to find a linear combination of the features. A way to introduce non-linearity is by combining existing features into polynomial expressions of some maximum degree. I tried using *PolynomialFeatures* of degree 2, but this gave worse accuracy than before. As it often happens, this approach resulted in overfitting, (high variance). A proof is that the accuracy when testing on the training data increased, whilst the accuracy on fresh unseen data decreased.

Another way to introduce non-linearity in the predictions is by the "kernel trick", which aims to find a linearly separable expression of the current features in a higher dimensional space. I introduced a *RBFSampler* before the Logistic Regression in the pipeline, but this made the predictions' success close to a random guess (accuracy 0.35 with 10-fold cross validation).

Logistic Regression comes with a range of parameters that can be modified. I ran a *GridSearchCV* search on *solver, class_weight* and *multi_class* parameters. The best parameters achieved an accuracy of **0.510** (with *max_iter* set to 150), and were as follows:

- *solver* set to *saga*
- *multi_class* set to *multinomial* (as opposed to *ovr*, which is a One Versus All multiclass classification)
- *class_weight* set to default (the options tested were all possible combinations of proportions 1:1:2 for the three classes)

Logistic Regression is usually performed on a binary classification problem. Extensions to multi class classifications provided by the *sklearn* library include using a one-versus-all or multinomial generalisation, as shown above. I also tried using a one-versus-one classifier, with a Logistic Regression estimator, but this did not improve the results.

B. Support Vector Classifier

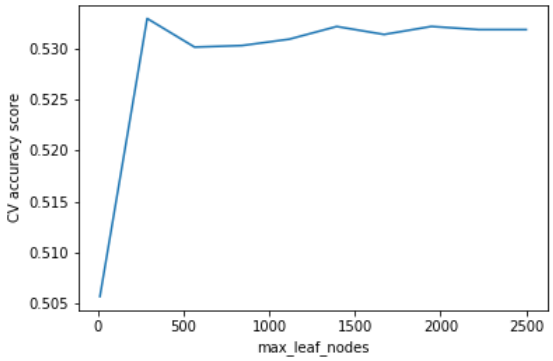
The Support Vector Classifier is similar to Logistic Regression in the sense that it too tries to find a hyper-plane that separates the different classes. By using *GridSearchCV*, I tried the following parameters and values:

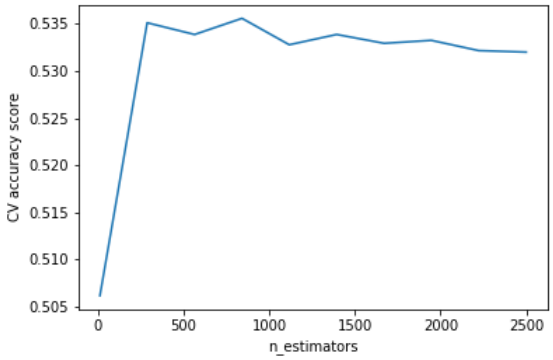
- *kernel*, ranging over values *poly, rbf, sigmoid*
- *degree* (for the *poly* kernel), ranging over values 1 and 2
- *shrinking*, ranging over values *True, False*

The best parameters were kernel *poly* with degree 2 and *shrinking* set to *False*. The mean accuracy of the 10-fold cross validation for this configuration is **0.511**.

C. Random Forest Classifier

Random Forests are an ensemble of decision trees. They performed better than the linear models above, achieving accuracy of **0.535**. I tried different values for the *max_leaf_nodes* and *n_estimators*, and plotted the results:





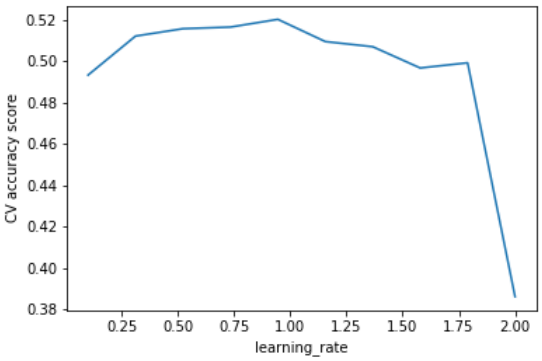
Increasing them beyond some limit doesn't seem to show improvement in prediction. It is also worth nothng that large values add more overhead — it took 1s per fit and prediction for 300 estimators versus 7s for 2500 in the case of *n_estimators*. Therefore I chose a value of 300 for both.

D. Extremely Randomised Trees

Keeping the same parameters as in the Random Forests, the Extremely Randomised Trees achieved similar accuracy — **0.531**.

E. Adaptive Boost Classifier

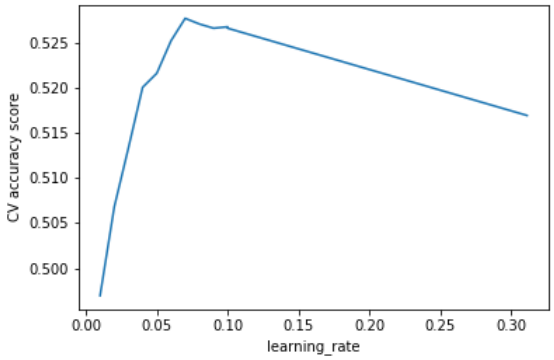
Boosting strategies train ensembles of classifiers one after the other, so that subsequent models perform better on tests that previous ones got wrong. I tried AdaBoost with a decision tree as the underlying model, and tried different values for the *learning_rate*, as it can be seen below:



The best score was achieved by setting the learning rate to 0.9. The reported mean cross validation accuracy was **0.519**.

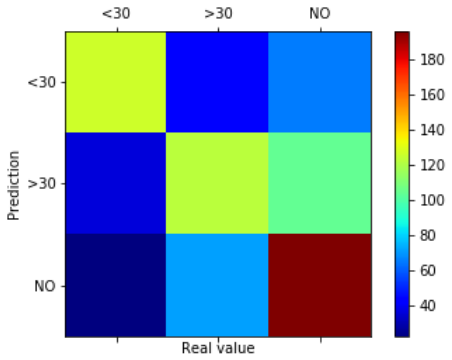
F. Gradient Boosting Classifier

The best score when trying out different *learning_rate* values, was **0.527** (*learning_rate* = 0.7), as it can be seen below:



IV. EVALUATION

The model I have chosen is the Random Forest Classifier. It achieves accuracy of **0.566**. Beyond that, another useful evaluation metric is the confusion matrix:

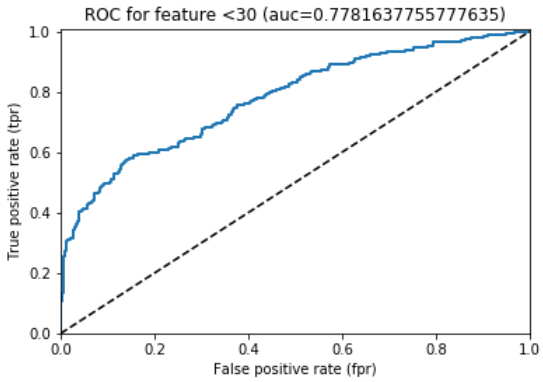


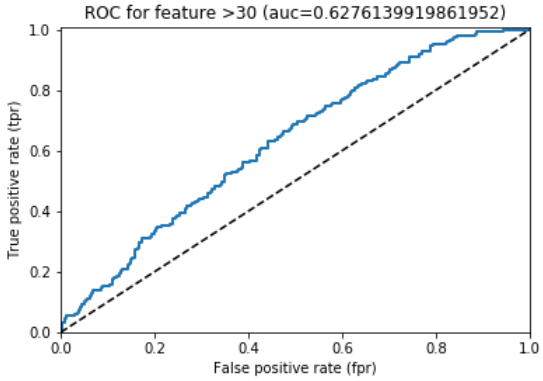
Most of the time, the model does well classifying *<30* and *NO* cases. However, it has major problems differentiating between *NO* and *>30*.

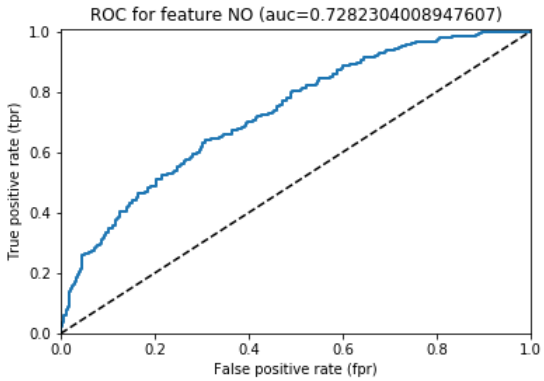
The same thing is depicted by the precision, recall and the f1 score — the f1-score, which is the harmonic mean of the precision and the recall values, is greater in the *<30* and *NO* cases than the *>30* one:

	<30	>30	NO
precision	0.684783	0.519149	0.536986
recall	0.540773	0.465649	0.678201
f1-score	0.604317	0.490946	0.599388

Another useful metric is the Receiver Operator Curve and its Area Under Curve. In the multi-class classification case, an approach similar to one-versus-rest should be taken, which is to plot a separate ROC curve for each feature, comparing it against the rest:





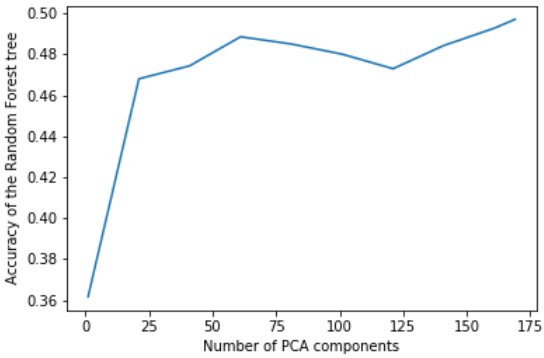


Again, there are no surprises and the *>30* category has its curve the furthest away from point (0,1) and hence the smallest AUC.

V. PCA DIMENSIONALITY REDUCTION

Dimensionality reduction is a good way of extracting the meaning out of the data. In some cases, less features might be able to describe what's needed by the algorithm to perform the classification.

I tried a range of number of PCA components to be used, and plotted their accuracy (again, using the Random Forest classifier) below:



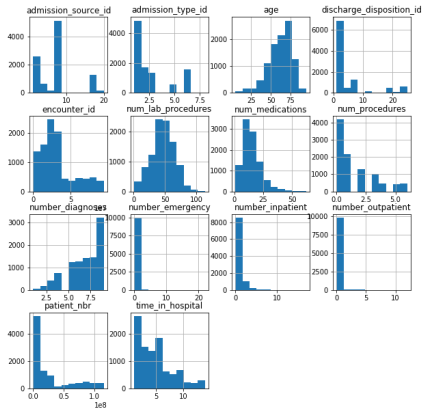
We observe that around 60 components were able to perform as good as using all 169 of them, achieving accuracy of **0.488**. This is, however, less than the original one. It's worth nothing that less components will make the model run faster.

Using just the first two PCA components (with the highest variance between the points), we could make a scatter plot and colour it by class:

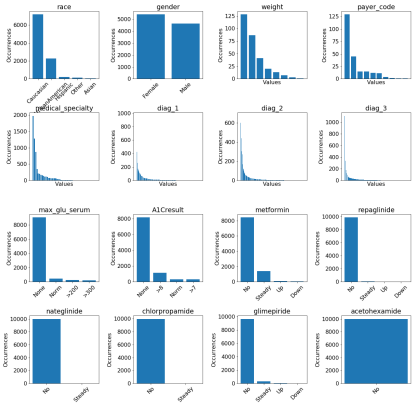
Unfortunately, no clear separation or other insight seems to be present.

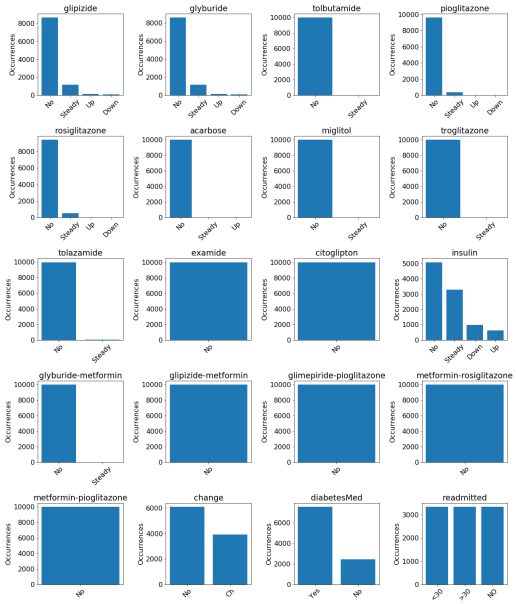
APPENDIX I
FEATURE DISTRIBUTIONS

Distribution of the numerical features (with *age* transformed to a numerical feature)



Distribution of the categorical features (without *age* which is presented above)





REFERENCES

[1] List of features and their descriptions in the initial dataset
[2] Wikipedia: ICD-9 Codes