CSI5155 Final Project

December 16 2020 Oliver Benning

Abstract

In this report I analyse the accompanying dataset for the study Impact of HbA1c Measurement on Hospital Readmission Rates. The dataset contains 47 features, most of which are without missing values. Each row is a patient, with data about their blood glucose levels, as well as information about their medication, health insurance provider, diagnosises, visit rates and more.

I first perform feature engineering, with the goal of turning all features numerical through one-hot and categorical encoding, while keeping dimensionality to a minimum. I then tackle the multiclass problem of predicting readmission rates using single-learners and boosting. I then tackle predicting whether metformin was administered to patients using two more single-learners as well as bagging. Finally, I revisit the readmissions class and attempt to use semi-supervised learning to make predictions using various degrees of unlabeled data, notably sets with 0%, 10%, 20%, 50%, 90% and 95% of labels missing. Algorithms are evaluated using ROC scores, a confusion matrix and associated metrics including recall and precision. All predictions are made using 10-fold cross-validation.

General remarks

Coding

The code and results for this project are in the accompanying notebook, project.ipynb. The notebook was made to be as clear and concise as possible, and leaves the verbosity for this document.

Tools used to run this project include: sklearn, numpy, pandas, matplotlib and semisupervised (containing keras and torch). The project was created in the Anaconda Python 3.8 environment. The notebook takes about 20 hours to run on a modern machine with GPU acceleration.

Computational Metheodology

Every algorithm configuration here is analysed in the following manner.

The algorithm is executed and evaluated on all combinations of size two in a 10 fold cross validation manner, to draw roc curves of the performance and assess its performance on the binary problem subsets in the data. The mean auc score from these attemps is calculated and the visual curves show both the mean auc line as well as the 10 individual folds and the standard deviation.

Next, 10 fold cross-validation is run on to compute a score on the algorithm on all classes, which is used later to determine statistical difference.

Next, 10 fold cross-validation is once again used on the multi-class setting to produce a confusion matrix. The confusion matrix is also analysed and the metrics of recall, specificity, precision, F-score, accuracy and balanced accuracy are computed using 0 as the negative class and the others as positive.

Finally, once a group of algorithms is analysed, the pairwise T-test is used on the 10-fold scores for each pair of algorithms to compute statistical difference and the table is shown. We also create tables of the confusion matrix derived metrics to compare performance.

Part A: Supervised learning

Feature engineering

The goal here is to ensure all features are numerical so the data is compatible with more ML algorithms. We utilize categorical and one-hot encodings to encode categorical data, paying attention not to overload our data with dimensionality from one-hot encoding, nor to lose too much information either.

Features that did not provide qualitative information were dropped, this included encounter id and patient number. This is because they're assigned arbitrarily, and provide nothing qualitative to learn from.

The only qualitative column that was dropped was weight. It was chosen to drop the column as 98% of observations was missing and I did not see a clear path to impute.

Columns that were categorical but not ordinal, such as admission type (which has numerical values but only because they are encoded with ids) were one-hot encoded. Some categorical columns contained multiple no information values, like "NULL", "not available", and "not mapped", so these were fused into one category.

The 3 diagnosis columns presented a risk of making the data too sparse one-hot encoded as-is, due to the large amount of features, i.e. medical diagnosis ids. Within the associated paper from the data, the authors outline reasonable category groupings for these ids, so they were grouped according to the paper in order to reduce dimensionality.

Categorical columns with an ordering were mapped to numerical values representing the ordering. Binary columns were mapped to a -1, 0, 1 mapping, where 0 represented unknown if present. Ordinal columns with more than two features were mapped to 0, 1, 2... with 0 representing an unknown or null case if present.

With the exception of weight, the dataset barely had any missing values so no imputation was required. Only categorical features had unknown values, which just assumed the 0 category.

Assumptions: - I decided not to group the medical_speciality column despite it containing 73 categories as I was not familiar with the different categories of medicine, the size of the tail set of features with <10 observations also was not unreasonable. In a real case I would do more research or ask a domain expert for a grouping suggestion. - I decided not to regularize the domain of features as none of the numerical features had too large a domain, I also read up and found that most regularization sensitive algorithms do regularization internally. - I made a judgement call dropping weight but I would revisit it more in depth to see if the 2% could in fact generalize the 98% well.

Task 1

The goal here is to predict re-admission rates. The algorithms chosen were Decision Tree (DT), Naive Bais (NB) and k Nearest Neighbours (kNN). Boosting (BO) was also tried using the GradientBoostingClassifier. These algorithms were chosen due to their native multi-class support, as well as to get a good mix of variety in the

learners. Boosting was chosen to see if it can improve on the single learner results.

The n for kNN was chosen by making repeated predictions on a 10% holdout set for the first 100 k values, and seeing which had the minimum error.

The prediction here is more of two nested binary predictions. We care most about 0 vs. {1,2}, i.e. whether or not they were readmitted at all, and then 1 vs 2 to show how long it took. We observe O.K. results for the 0-1 and 0-2 paired ROC curves, but notable very low scores for 1-2 curve in the single learner algorithms. This suggests that readmittal vs no readmittal performance may be passable, but differentiating over or under 30 days is likely not possible.

Let's first understand the problem, these are patient readmittals, and a hospital would likely want to know this in order to predict whether it should prepare for re-admittal or not. In a real case I would verify with a client which one is worse, being too prepared and wasting resources (choosing high recall), or being underprepared and wasting resources (choosing high precision/specificity).

In this case this is indeed a tradeoff in this case as all models are statistically different and no model excels in both. For high recall DT looks to ourperform NB. For high precision/specifity BO looks to outperform kNN with higher precision and F-score, it also had the highest accuracy.

I would therefore recommend to a client DT if recall is a priority, and BO if not, i.e if specificty and/or precision are more important. That being said, if it were the case that specificity or precision was priority AND that it was an online or training time sensitive setting, kNN would be best as it trains much faster.

I'm concerned about the high number of features in the data so I try and see if cutting that number in half would help, by selecting the k=120 best features out of 239.

DT got a slight boost in all metrics except specificity, and BO got a slight decrease. The performance change was so minor I do not think feature selection is worth it on this data and would avoid doing so in production.

Task 2

I opted to predict the 'change' column. This was conveniently already a balanced binary class, but also looked like a useful feature to predict for the medical domain, in the case of a patient who has insuifficient records. I got near perfect results and realized it had perfect correlation with a value of 1 or 3 in the medication column (a change) so I realized I had to switch to something more meaningful.

Instead I looked into 'metformin', which is a diabetes medication taken commonly by individuals with diabetes, but not all. This is also a variable that wasn't analysed much in the paper so I'm hoping to provide some new insight into its predictability. In this dataset, 1/4 of individuals who take diabetes medications take metformin. Let's see if we can predict whether an individual is taking metformin from the data, we will turn this into a binary problem by treating an increase, steady or decrease as the same postitive class label.

I used Random Forest bagging technique and LinearSVM models to round out the remaining model classes left, and to use models that are optimal in binary classification settings on a binary classification problem. I also use bagging with the default linear kernel to see whether performance can be increased.

Results here are extremely interesting. First, all 3 algorithms are statistically different. Next, RF completely ourperforms SV. SV is subject to overfitting, which I believe is what happened here.

When comparing BA to RF, it has much better recall and F-score and accuracy, it's much better at predicting the positive class. But, the balanced accuracy metrics are the same. This suggests to me that class imbalance could be the reason RF is performing poorly, and BA by nature is better suited at overcoming this problem. I like the performance of BA here and would suggest this as the best algorithm.

Overall I find it interesting how much of an improvement bagging can give over a single learner. Furthermore, I am suprised as to the difference in performance that can be achieved between different bagging algorithms.

Part B: Semi-Supervised learning

In this part we test different semi-supervised learning algorithms on Task 1 in Part A, to predict readmittal from the data. We will use the percentages 0% (baseline), 10%, 20%, 50%, 90% and 95% of missing labels for each of 3 algorithms for a total of 18 analyses. The algorithms are LabelPropagation (LP), LabelSpreading (LS) and StackedAutoEncoder (SA). For LP and LS, a knn kernel was used with the optimal k as found in part A as the linear kernel was computationally prohibitive. We note that knn performed poorly in part A, so expectations are not high.

The same procedure as Part A was followed as well, with the addition of unlabelling a certain percentage of observations from the feature column.

First we will analyse each algorithm individually.

For LP we have overall performance similar to kNN. All percentages perform differently here with 10% actually providing the best recall and F score. 20% and 50% decrease the performance, whereas 90% and 95% perform so poorly they crash the algorithm as seen in this GitHub issue.

For LS all percentages are statistically different predictors and in the baseline we have similar results to kNN(39) from part A, with a slight decrease in recall and F score and an increase in precision. Recall and F score decreases up to 50% but actually increases at 90% and 95%!

The SA algorithm is a much more complicated one, using neural nets internally - layers of 50 and 100 with ReLU. 10 iterations were selected for computation speed, and because it produced better ROC scores than 100 in baseline. We note however that predictions are all over the place, roc curve variance is very high. Baseline predicts all 0, whereas at 20% and 50% it predicts all postive cases, but shifting from mostly 1 to 2. m_auc scores stay comparable with agorithms in part A. When we get to 90 and 95% the results are more interesting, in the 90% case we predict mostly 2, and in the 95% case we predict mostly 1, but it is slightly less dominant in one category. Overall it's an extremely volatile predictor, and tends to predict one category entirely. It's also extremely complicated, so I am unable to identify which parameters to tweak in order to make the predictor more meaningful.

We'll now compare each algorithm by percent removed and see which is best in each category.

- At baseline, LP and LS are statistically similar while SA differs. SA predicts all zeros at this level, while the other two show low recall - high specificity negatively biased predictions.
- At 10% LP and LS are again statistically similar and SA different. Here SA is predicting all positive class while LP and LS show slightly less recall when compared to baseline.
- At 20% we observe the same statistical difference. SA is all positive while LP and LS decrease further in recall.
- At 50% we observe the same statistical difference. SA is all positive while LP and LS decrease even further in recall.
- At 90%, all 3 algorithms are now different due to LP producing all negative results. LS now increases in recall even higher than baseline and SA makes mostly positive, but not all positive predictions.
- AT 95%, all 3 algorithms differ statistically. LP does not produce meaningful results, while recall in LS increases even further, SA also becomes even more balanced here.

Overall the prediction quality is low, but the increase at 90% and 95% for LS is very interesting, we are able to beat the baseline model and get relatively close to the kNN single learner.

Other Lessons learned

In general, I learned to start early in machine learning, and I'm glad I did with this project. This would have been physically impossible to do last minute as the notebook takes around 20 hours to execute from start to finish. Related, I learned even further how useful jupyter notebooks were, by saving past variables we don't need to re-run everything from scratch and can instead re-iterate on a cell.

I learned a lot about open source. When I was looking for a third SSL algorithm I came across the semisupervised project for python. Unfortunately the S3VM and SAE algorithms didn't work in a plug and play fashion. What was really cool was that I was able to fix them and actually commit my changes back to the project via the two pull requests below, which the author actually accepted! I originally wanted to use S3VM in the project, but didn't have enough memory to accomodate the sparse matrix.

https://github.com/rosefun/SemiSupervised/pull/1

https://github.com/rosefun/SemiSupervised/pull/2

I learned that semi-supervised learning can be extremely powerful, the results in the SP algorithm for part B were suprising and very interesting.

I learned that neural network based algorithms are extremely complex to tune and take many unintuitive parameters. They take a long time to train with a lot of compute power, so it takes a long time to optimize them properly.

Conclusions and future work

In conclusion, we performed 3 different tasks with 3 different degrees of success. Predicting hospital readmissions using the data worked relatively well, likely helping confirm the hypothesis of the authors that measures of HbA1c measurements help predict readmissions rate. Future work could include incorporating more of the qualitative analysis of the authors into the models to try to improve predictions even further, as well as doing the task without the HbA1c measurements to see whether results are significantly worse or not. Next, I predict whether patients are taking metformin medication with very strong results. Future work in that regard could include considering all medications together as a vector and training a probabilistic model to show confidence scores for each patient in terms of which medications they may be taking. In the last section, we investigated semi-supervised learning on the data with varied results. Future work would include how the algorithms could be optimized to provide better results, or finding a new potential technique that could perform better - Label Propagation and Label Spreading with linear kernel as well as S3VM is a notable candidate that would have been evaluated in this work if memory was not a constraint.