Katharyn Jia and Sean Yanik

6-25-2021

ROBAI 240

Final Project Documentation and Written Summary

For our final project, we developed a binary classification model to determine whether or not a patient who suffered a Myocardial Infarction (heart attack) would survive. This prediction would be based on the patient's health data at admission.

We used a dataset of 1700 rows, each row containing 124 features. Each row represented a patient admitted to a hospital for a heart attack, and contained information regarding whether or not they survived, and, if they died, their cause of death. Individual features contained information such as age, sex, whether or not the patient suffered from heart disease, presence and degree of hypertension, etc. Several of these features were unrelated to our problem (such as patient ID), and were removed. This dataset contained missing values.

We chose this dataset because of our interest in the medical field on the whole. Heart attacks are a matter of life and death; an effective predictor of whether or not a patient is at high risk to die could save lives and allow resources and medical staff to be more efficiently deployed.

After downloading this dataset, we thoroughly inspected it, and graphed different features in different ways to get a better understanding of skews, key features, etc. Then, using the understanding we had gained, we began cleaning the data.

Cleaning and preparing this dataset was a multistep process. We first dropped all duplicate rows (by patient ID), separated the target columns (and removed unnecessary ones like patient ID), and then carefully treated the missing values. If a feature had more than 50% missing values, it was removed. Otherwise, values were imputed (in a variety of ways, depending on whether the variable was cardinal, ordinal, etc.).

Our goal was to replace the missing values without affecting the distribution of features. One of the interesting aspects noted in the data description was that the data was missing randomly—that is, there was no trend in the types of missing values, or in the relationship between missing values and outcomes. This allowed us to use random, or non-stratified methods to replace these values.

There were three categories of values: nominal, ordinal, and real. The real values were the easiest to impute. By plotting the histograms, we could see which ones had a skewed versus a normal distribution. For normal (or near normal) distributions, the bulk of the data is centered at the mean, and therefore we replaced missing values with means for the column. For skewed distributions, we used the median, which is where the weight of the distribution sat.

For binary missing values, we wanted to retain the proportions of the data that we had (this is especially important for our naïve bayes modeler), so we set a random value generator with the probabilities of the column proportions. This allowed for some randomness without significantly impacting the distributions.

Finally, for ordinal values, we used the modes of the data. This works well for data where there is one overwhelming node, which was the case for all our columns, since the additional values should not significantly skew the results.

Last, we transformed the data with a MinMax transform, which moved our data into the range [-1,1], and reduced the data by 5% with PCA, getting rid of 52 components. We did this to increase training speed.

We split the data into two sets (80% training, 20% testing), and then we performed Grid Search with 4-fold Cross validation on 6 different model types. We compared K-nearest neighbors, Logistic Regression, Kernel Support Vector Machines, Naïve Bayes, Random Forest, and Gradient Boosted trees. In effect, we assembled a collection of every type of model that we were familiar with that we new could support binary classification.

Initially, we compared a large number of different hyperparameters; our grid search was quite exhaustive, and took about an hour to run even on a distributed cloud-computing network. Eventually, we had to reduce the number of checked hyperparameters to make training more efficient. Our final GridSearch, while not extremely exhaustive, did a fair job. Through this search we evaluated and compared our different model types. Hyperparameter groups were changed and adjusted several times in our coding process to get the best results while keeping runtimes low.

Our best model was the Logistic model. Because this is medical data, and because there are real and terrible consequences to falsely identifying someone as doomed to die, we wanted to minimize false positives without completely flattening our detection rate (aka, maximize precision). While our Naïve Bayes had fewer false positives, the logistic model had the highest accuracy, and excellent precision and recall—that is, there was not only a reduction of false positives, but a high rate of true positives.

From our confusion matrix plots, we can see that the highest number of true positives was the Kernel SVM (the highest detection rate), but Logistic Regression was only a single true positive behind. The highest number of true negatives was the Naïve Bayes model (though most of the models were very close for true negatives), and the best recall was close between the logistic (2nd) and the SVC (1st) models.

The training times for the SVC was the longest, next to the logistic. However, for run times, the fastest was logistic regression, followed by Naïve bayes, followed by Gradient Boosted Trees. This makes sense, since for these models, the complexity limitation lays in the training of the optimum model—once trained, only a single instance needs to be run. If resources are a concern, such as with very large patient classification tasks, the Naïve Bayes offers a good balance of high detection of negative, combined with low computing cost and fast training.

Either Naïve Bayes or Logistic offer a good compromise on features. Both had high accuracy, and they represent a tradeoff between low system resource requirements, and a higher positive detection rate.

If we wanted to identify patients with a high chance of death in order to save them, Logistic is a good choice, since it had a very high true positive. Kernel SVM would also be a decent choice. For large patient sets, or in a situation where it is more important to ration medical care (triage so we don’t waste time on patients who will die anyway), Naïve bayes runs and trains quickly, and had a decent identification of negatives.

Overall, we think that logistic regression, coming in 1st for accuracy, 2nd for recall, and third for precision, is the best model.

The applications of our project are manifold. Being able to predict if a patient is at a higher risk for death can help families prepare for the worst, but can also help medical professionals make more informed decisions regarding the distribution of resources and staff. Patients who are at risk can be more carefully and closely watched. We think that patients who are guessed to be at low risk, however, should not be treated more lightly. All models are imperfect, and neither our recall nor precision can be relied upon to be perfect. The protection of human life always comes first.

One of the challenges was in the run times for the models. Due to processing power restraints, we were unable to run multiple gammas in the GridSearchCV for SVC. We were also unable to cross validate this set due to those limitations. We were able to overcome this somewhat by artificially partitioning the dataset using a non-resampled bagging model, which allowed us to run more SVC’s in parallel, but since Python does not have native multicore support, we continually hit limits on the number of different fits we could accommodate. In order to speed this up, we layered the min\_max scaler on top of the standard scaled data to reduce the inner products and speed up run times, but this could have introduced distortion to our data, especially for our ordinal values.

A surprisingly complex topic was data imputation—choosing how to replace missing values was critical for many of the ScikitLearn models, which cannot accommodate missing values, but because there were such large swathes of missing data, we may have compromised our own results during data preparation.

Another issue was the large amount of crossover in the complications dataset—in real life, patient outcomes are messy—people might have many complications, and then die of multiple causes. We originally desired to perform multiclass separation, to predict specific complications and causes of death. Ultimately, we decided to focus on a simple lethality model. If we had more time, or more computing power, a clustering algorithm to group these complications into co-occurring sets, or perhaps running multiple sets of *One v. All* classifiers for all the different types of complications would be a good topic to explore.

All in all, this project was an excellent and highly informative experience.

*Source Information:*

*Link to dataset:* [*https://archive.ics.uci.edu/ml/datasets/Myocardial+infarction+complications#*](https://archive.ics.uci.edu/ml/datasets/Myocardial+infarction+complications)