Hospital Occupancy Prediction: Classification

Justin Ledford
Eric Smith

November, 2018

Abstract

In this report, we use several machine learning models to perform a classification task. That task was to predict whether or not a person will visit a hospital next year. We tested five different models and used a sensitivity metric to conclude that a random forest worked best for this dataset. We argue that this model could be deployed to aid hospitals in predicting the number of returning patients.

Contents

1 Data Preparation

Our classification task is to predict if a member will visit the hospital at all the next year given that member's claim history of the current year. This gives us a binary class prediction task, with the positive class being that the member does visit the hospital.

We have flattened the claims into summary statistics, for example age, sex, pay delay, and a member's maximum Charlson Index. For sex, which is a binary value, we used a 0 for female and 1 for male. We also used the number of lab visits per year, and number of unique prescriptions filled. The number of unique prescriptions, or drug count, is in the range 1-7+. We chose these features as we thought they are most indicative of the number of days spent in the hospital the next year.

For any feature that contained more than one number, like 7+ for drug count, we replaced it with that number, because these values were chosen as they are the 95th percentile for that feature.

We used data from year 1 and year 2 claims, and days in hospital from year 2 and year 3. For members with claims in both years we considered them as separate samples, so our model only considers the claim history of a single year.

1.1 Categorical Feature Counts

In order to create more features and use the categorical features, we have created columns containing the counts of unique values for Primary Condition Group, Specialty and Place of Service. For example, given a member, some of these features would be number of claims with a metastatic cancer, number of claims where the member visited an diagnostic imaging lab, and number of claims where the member visited an internal medicine specialist.

1.2 Class Imbalance

After labeling the data with our binary classes, there is a large class imbalance between the two, with 88% of the data having 0 days. To handle this large class imbalance we downsample the data by removing some rows of the 0 days class to have an even amount with the 1+ days class. We then evaluate models using a split taken from the data with the original class distribution.

2 Modeling

We created several models and used stratified k-fold cross validation to select the parameters that gave each model the best metric. We split our data with a 95/5 split for training and evaluation sets. Taking the train split, we divide it into 10 folds (while keeping class balance the same throughout each fold), and then for each fold, we train a model with the remaining 90% of the training data after taking the fold out, and then use the fold to evaluate the model's performance on data it hasn't seen before. This is repeated for each fold and the average metric is taken across folds to compare models.

After selecting the best parameters for each model with k-fold cross validation we then evaluate each model using the 5% test split.

2.1 Evaluation Metric

Our use case is for insurance companies to estimate their budget for hospital expenditures. Once a budget is created, it is easier for businesses to spend below that budget, than to exceed it and request more money. Since the estimated number of people visiting the hospital will determine this budget, it is best to minimize the amount of false negatives in our model. False negatives in this case are members that are predicted not to visit the hospital but actually do. False positives are less concerning because those will just increase the budget, whereas the false negatives will cause the insurance companies to exceed their budget.

The goal to minimize false negatives determines our evaluation metric, which is sensitivity, also known as recall, or true positive rate. Sensitivity is calculated as the true positives divided by the sum of the true positives and false negatives. So out of all positive cases, it calculates the number of cases truly labeled as positive. The higher this metric is, the less false negatives our model has, so we aim to maximize it.

We treat the 1+ days class as the positive class. For each model we inspect the ROC curve, which plots sensitivity (true positives) by specificity (true negatives) as the discrimination threshold (which is the threshold probability at which a sample is classified as positive). With the ROC curve, the best result is a curve reaching the top left corner. This is measured with area under the curve. A model that randomly guesses will just have a straight line from the bottom left corner to top right corner.

We also inspect the confusion matrix as a heat map, with the relative frequencies of classifications within the true class. Since we aim to decrease false negatives, we want the top left square to be as light as possible.

2.2 *k*-nearest neighbor

K-Nearest Neighbor is a simple model which classifies an input sample by the k closest data points from the training. The majority class of the k neighbors determines the class. With k at 1, 5, 10, 15 and 25, we see in Figure 8 that the sensitivity gets worse the higher the k. This is most likely related to the fact that the data is extremely dense, with samples of the positive class distributed among the samples of the negative class without much clustering. Because of this, our final KNN model has a k of 1.

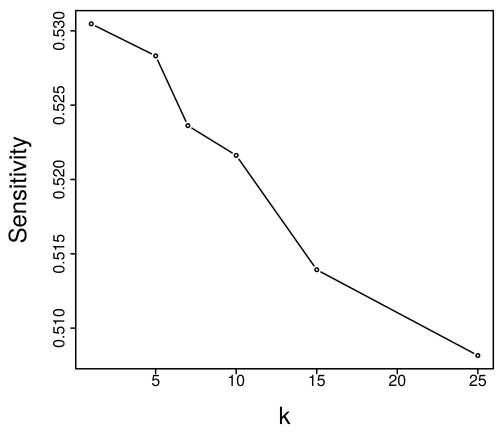
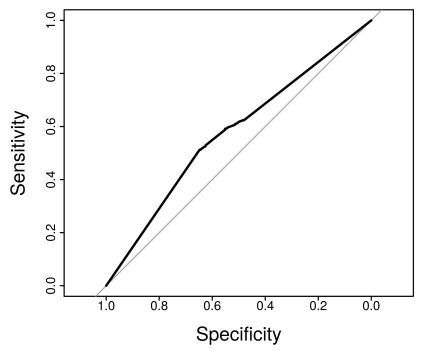


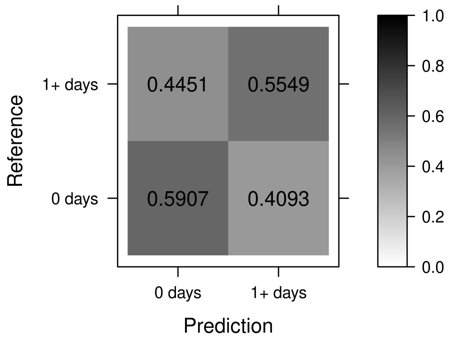
Figure 1: Mean Fold Sensitivity by k for KNN model

Looking at Table 1 we see the performance on the test split. The accuracy and sensitivity are only slightly better than average. However, the negative predictive value shows that our model is very good at predicting the negative class. This is expected, as the class imbalance is high, but our model still predicts better than random guessing.

Looking at Figure 9 we can see the ROC curve is only slightly better than random guessing. The heatmap shows that 44% of positive samples are classified as negative, which is pretty poor.



(a) ROC Curve on Test Split



(b) Contingency Matrix on Test Split

Figure 2: KNN Results

2.3 Random Forest

Random forests are a collection of random decision trees that collaborate to make a prediction. The algorithm works by randomly selecting N samples from the training set (with replacement) and splitting a decision tree based off each sample's attributes. This process is repeated multiple times and the classification of any given sample is decided by majority voting.

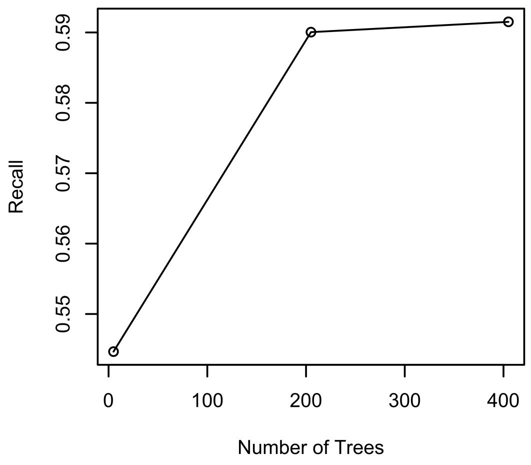
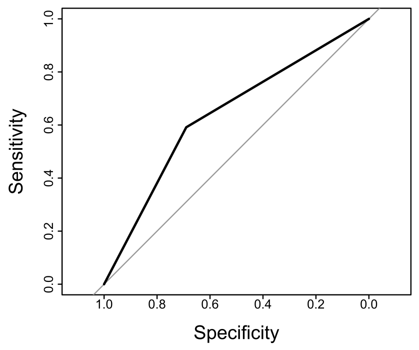
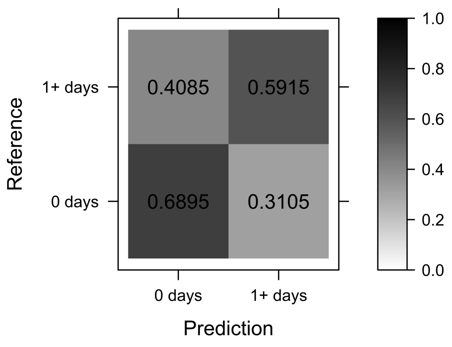


Figure 3: Sensitivity for Number of Trees in a Random Forest

The accuracy (or any given metric) of a random forest usually increases with the number of trees used to make the initial split. In our case, setting the number of decision trees to 400 provided the best results: a sensitivity score of 59%. We would have likely seen better results with more decision trees, however, we found that sensitivity stop increasing significantly past 200 trees. Furthermore, adding more trees increases the time it takes for training a random forest to complete.



(a) ROC Curve on Test Split



(b) Contingency Matrix on Test Split

Figure 4: Random Forest Results

2.4 Gradient Boosting Machine

Gradient boosting machines are similar to random forests, in that they use an ensemble of decision trees, but the process for building the model is different. We start with a weak learner, then train a new learner to estimate the errors from the first learner, add it to the ensemble and repeat. For each learner we also use bagging, to take a subsample of 50% of the data. This is known as stochastic gradient boosting. We performed a grid search on interaction depth from 1-5 and number of trees in the ensemble of 50, 100, 150, 200 and 250 to find the combination of the two that gives the best sensitivity. Interaction depth is the number of splits at each node. In Figure 5 we can see the sensitivity increases with interaction depth. Since we have 70+ features, it makes sense that an interaction depth greater than 1 is preferred. The parameters chosen from the grid search gave an interaction depth of 5 and number of trees of 100. It is possible that more than 100 trees resulted in overfitting.

Overall the GBM performed better than KNN, which is expected given how simple KNN is.

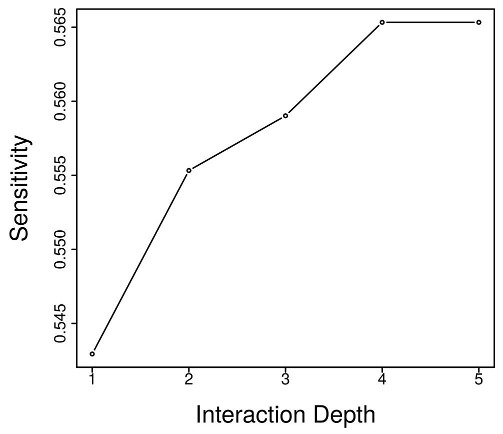
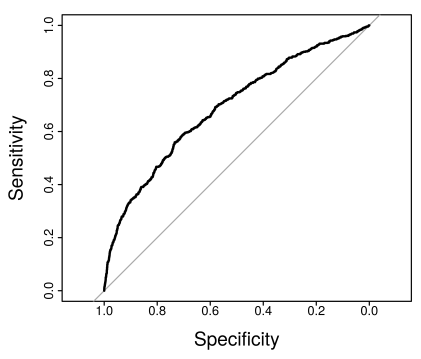
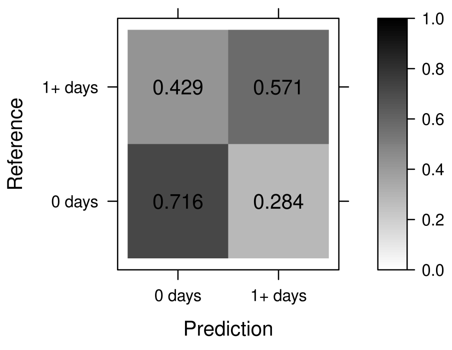


Figure 5: Mean Fold Sensitivity by Interaction Depth for GBM model



(a) ROC Curve on Test Split

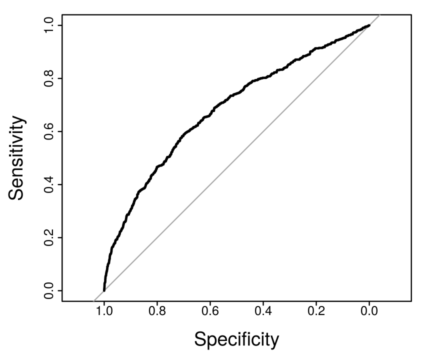


(b) Contingency Matrix on Test Split

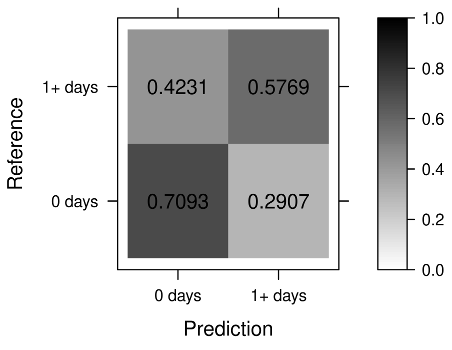
Figure 6: GBM Results

2.5 XGBoost

XGBoost (eXtreme Gradient Boosting) is a popular implementation of gradient boosting with additional features, such as regularization penalties in the boosting equations. We also performed a grid search with XGBoost with the number of trees, depth, feature sample, subsample and learning rate. Our final XGBoost model had 250 trees, a depth of 4, learning rate of 0.3, feature sample of 0.8, and subsample of 0.875. This model performed only slightly better than the GBM, with an increase of sensitivity by about 0.6%. Because XGBoost has many parameters, it is possible that with additional tuning we could increase this further.



(a) ROC Curve on Test Split



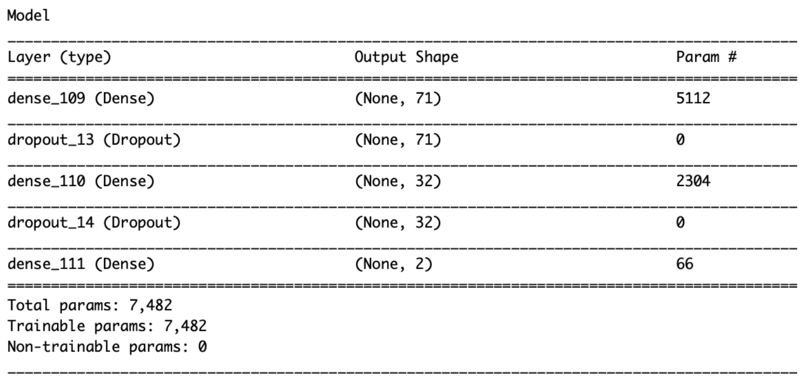
(b) Contingency Matrix on Test Split

Figure 7: XGBoost Results

2.6 Neural Network

Perhaps the most complicated model was the neural network that we developed. An artificial neural network consists of several artificial neurons used to learn attributes associated with a class. The weights of the individual neurons are updated with each training epoch, and when grouped into layers, offer a powerful way to make a prediction on new data.

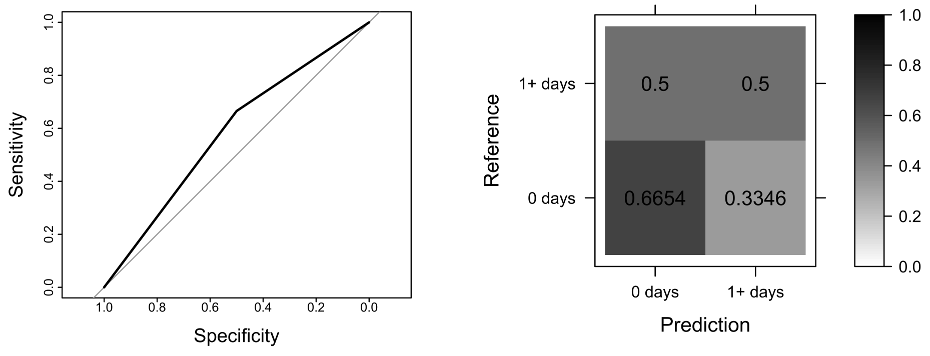
The model that we created consisted of three dense layers of three different sizes. The input layer consisted of 71 nodes, which is the number of one-hot-encoded attributes in the training data. The network's middle or hidden layer has 32 nodes, and the output layer has two nodes (the number of classes from which to choose). Between the three layers are two dropout layers which ignore a random 20% of neurons when updating the weights of the network - this prevents the network from overfitting.



```
Model
_________________________________________________________________________________________
Layer (type)                         Output Shape                        Param #
=========================================================================================
dense_109 (Dense)                    (None, 71)                          5112
_________________________________________________________________________________________
dropout_13 (Dropout)                 (None, 71)                          0
_________________________________________________________________________________________
dense_110 (Dense)                    (None, 32)                          2304
_________________________________________________________________________________________
dropout_14 (Dropout)                 (None, 32)                          0
_________________________________________________________________________________________
dense_111 (Dense)                    (None, 2)                           66
=========================================================================================
Total params: 7,482
Trainable params: 7,482
Non-trainable params: 0
_________________________________________________________________________________________
```

Figure 8: Architecture of Artificial Neural Network

A neural network has many parameters which can be fine-tuned to improve results. We tuned these parameters using a mixture of hardcoded changes as well as using simple programmtic tuning methods like grid search. Even with the fine tuning, results for the neural network were subpar: with a 0.5 sensitivity score. Furthermore, training this network for 200 epochs took substantially longer than the training for other models.



(a) ROC Curve on Test Split (b) Contingency Matrix on Test Split

Figure 9: Neural Network Results

3 Evaluation

Our best model, a random forest, was able to get a sensitivity about 10% higher than random guessing. It is likely that this data is not very predictive, and with additional features per person we could make better predictions. It is also possible that given a larger dataset with millions of members, a deep neural network could be created to give a more accurate prediction.

Although our model may not be useful for budgeting purposes, it may be useful for a loose prediction on a member visiting a hospital. This could be used for marketing purposes, for example informing a member that based on their claim history, they may be visiting the hospital the next year. This could be used to encourage members to visit doctors to potentially avoid visiting the hospital later on. This could then save insurance companies money since hospitals are more costly than doctor's offices.

	Accuracy	**Kappa**	**Sensitivity**	**Specificity**	**PPV**	**NPV**	**AUC**
KNN	0.5844	0.0689	0.5593	0.5878	0.1545	0.9083	0.5763
RF	0.6779	0.1544	0.59151	0.68955	0.20425	0.92609	0.6405
GBM	0.6988	0.1662	0.5710	0.7159	0.2131	0.9253	0.6896
XGB	0.6936	0.1634	0.5769	0.7093	0.2109	0.9256	0.6827
NN	0.6469	0.0851	0.50000	0.66535	0.15842	0.91351	0.5827

Table 1: Model Performance Statistics

4 Deployment

Given the marketing use case above, this model could provide value to insurance companies. In a production setting, it is likely much more data will be available. With our current data set we have claims for about 100,000 members, but insurance companies tend to have millions of members. This could definitely improve our current model, but a new model could be developed to better perform with this volume.

Our model is also currently trained to predict based on one year of claim history. Because of this, the model should be retrained every month by removing the claims from the earliest month, and adding the claims from the current month.