pager, inc.

NEISS Survey Data Model

Brendon Villalobos
ML Engineer Application Project

Abstract

The National Electronic Injury Surveillance System (NEISS) is a weighted sample of U.S. hospitals aimed at collecting data to characterize the nature of injuries that are associated with consumer products. This classification model aims to predict "disposition" (coded outcomes) for these injuries given other variables collected in the NEISS survey. The chosen model was a Random Forest trained on a feature-engineered random sample (weighted by NEISS survey weights). The model had an out-of-sample predictive accuracy of ~95.6% on a randomly held-out 20% of the data. Since the data have unbalanced outcomes (91% outcomes are of class 1), a better measure of performance is the conditional accuracy given the outcome. The model performs very well on this measure, with a greater than 50% chance of correctly predicting any disposition category - even disposition 9, which was only observed 32 times out of the entire NEISS survey of ~2.7 million records. Performance dramatically increases with the slightly more frequent categories of 2 and 4 (each < 6% of training data), and is virtually perfect for disposition category 1.

Exploratory Data Analysis

Implementation

A good amount of time was spent on simply getting to know the survey design. I read through the coding schemes in the Coding Manual and inspected summary statistics and exploratory visualizations of correlations and distributions.

As mentioned in my email, I initially suspected that product code information would have little bearing on outcome when conditioned on *diagnosis* and *body_part*, but after conducting a chi-squared independence test conditioned on these variables, I discovered that they were indeed informative. Therefore I ultimately included it in the set of features to be engineered and fed into the model.

With some more time I would have liked to…

…make some informative heat maps

Calculate and plot covariance matrix heat maps between several important variables to get a sense of the multicollinarity among predictors and conditional relationships between the predictors and outcome classes.

Feature Extraction

Implementation

Extracting month information from dates

I felt that if any information would be useful in the date variable, it would be the month of the year. You could imagine that the quality/type of medical changes cyclicly with new graduating classes of medical students, residents, nursing students, etc. It might be interesting to leverage day-of-the-week information, but that was not implemented here.

Standardizing age

NEISS codes *age* in months as 2XX, while age in years is simply encoded as XX. This requires a transformation of *age* in months / 12 to put the entire column on the same scale.

Dummifying categorical variables

Sex, race, stratum, diag, body_part, and location were decomposed into binary indicator variables of all possible values for each of these features.

Extracting Bayesian likelihood from *prod1*

Prod1 could not be reasonably transformed into dummy variables, since it has so many possible outcomes that it would radically increase the dimensionality of the problem. Instead, I captured some information about the *prod1* codes by estimating the conditional likelihood: P(*disposition* | *prod1*). This captured significant information about *prod1* while only adding 7 dimensions (one for each disposition) to the set of engineered features.

A more robust technique would've been to use an NLP measure of vocabulary similarity. However, considering that model productionization was also a requirement, employing a proper NLP method on *prod1* infeasible in the time allotted (see next section).

Although I didn't use *prod2* in this model, it might bump performance to create duplicate records for every prod2 value, setting *prod1=prod2*.

With some more time I would have liked to…

…use NLP methods on *narr* and *prod* features

In my email, I discussed implementing NLP techniques to extract features from the *narr* field. Unfortunately, the need to productionize the model in such a short time encouraged me to drop this computationally nontrivial implementation. The implementation would be nontrivial because all of the useful methods (bag-of-words, topic similarity metrics) require handling very large sparse matricies, which would either require an online training algorithm or a non-trivial data structure/store solution (i.e. Spark cluster). This also applies to the *prod* field, since it can also be thought of as a vocabulary.

…implement smoothing to assign nonzero probabilities to unobserved feature values
Fortunately, since the survey includes "other" codes for almost all of it's variables, this isn't a real-world necessity. But, in principal, we should use a discounting method to assign nonzero probabilities to an "unseen/new" value for each feature, and in scoring on an unseen value, replace it with this generic code. This increases the generalizability of models.

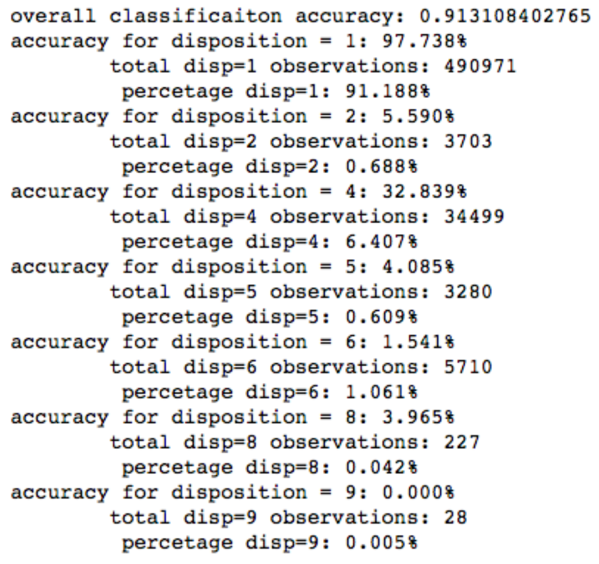
Modeling & Validation

Implementation

Random Forest crushed it
I was going to try three methods: RF with hyperparameter tuning (i.e. over tree depth, min_split_count, etc.), KNN with a dimensionality reduction technique like PCA, and a weighted Multinomial Bayes. However, when I first tried the random forest, the out-of-sample predictive accuracy on the test set was so good that I decided to forge ahead with productionization in the interest of time. See README.md at https://github.com/bkvillalobos/ct-neiss-ml/tree/master/ml-coding-assignment.

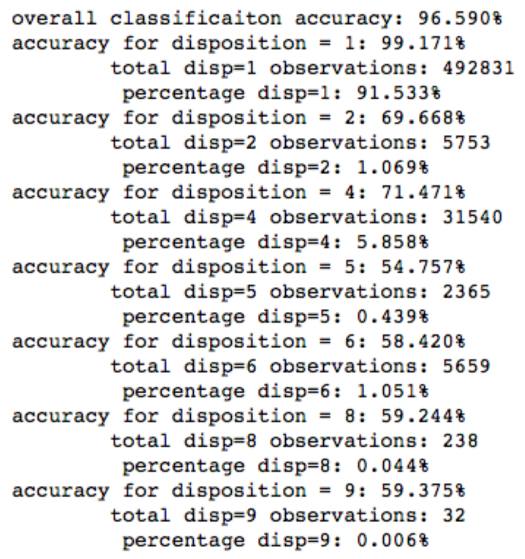
Dramatically improved performance by resampling based on NEISS survey weights
At first, as expected with these uneven outcome data, the RF dramatically overfitted to the *disposition* value of 1 (which accounted for ~91% of training observations). However, leveraging the fact that this survey was designed to be a weighted representative sample of US hospiatls, I decided to leverage these weights in reseampling (with replacement) from the training data before training the model. This **dramatically** improved performance on the rarer outcomes, as shown in the comparison below:

Results Prior to Weighted Resampling



```
overall classificaiton accuracy: 0.913108402765
accuracy for disposition = 1: 97.738%
        total disp=1 observations: 490971
         percetage disp=1: 91.188%
accuracy for disposition = 2: 5.590%
        total disp=2 observations: 3703
         percetage disp=2: 0.688%
accuracy for disposition = 4: 32.839%
        total disp=4 observations: 34499
         percetage disp=4: 6.407%
accuracy for disposition = 5: 4.085%
        total disp=5 observations: 3280
         percetage disp=5: 0.609%
accuracy for disposition = 6: 1.541%
        total disp=6 observations: 5710
         percetage disp=6: 1.061%
accuracy for disposition = 8: 3.965%
        total disp=8 observations: 227
         percetage disp=8: 0.042%
accuracy for disposition = 9: 0.000%
        total disp=9 observations: 28
         percetage disp=9: 0.005%
```

After Weighted Resampling



```
overall classificaiton accuracy: 96.590%
accuracy for disposition = 1: 99.171%
        total disp=1 observations: 492831
         percentage disp=1: 91.533%
accuracy for disposition = 2: 69.668%
        total disp=2 observations: 5753
         percentage disp=2: 1.069%
accuracy for disposition = 4: 71.471%
        total disp=4 observations: 31540
         percentage disp=4: 5.858%
accuracy for disposition = 5: 54.757%
        total disp=5 observations: 2365
         percentage disp=5: 0.439%
accuracy for disposition = 6: 58.420%
        total disp=6 observations: 5659
         percentage disp=6: 1.051%
accuracy for disposition = 8: 59.244%
        total disp=8 observations: 238
         percentage disp=8: 0.044%
accuracy for disposition = 9: 59.375%
        total disp=9 observations: 32
         percentage disp=9: 0.006%
```

With some more time I would have liked to…

…optimize RF hyperparameters with 10-fold cross validation

This is really important in almost every other case – I would've liked to tune the RF hyperparameters with 10-fold cross-validation over the training data. However, the results without cross-validation were so good that I decided to move on to productionization in order to finish the assignment on time.

Productionization

see README.md at https://github.com/bkvillalobos/ct-neiss-ml/tree/master/ml-coding-assignment