Rory Creedon

Predicting Life Insurance Risk Class

A Gradient Boosting Machine Approach

The Problem: Buying Insurance is Hard

The Customer

- There is a trillion dollar life insurance gap in the USA
    - Application process requires “fluids” as well as survey responses
    - Extremely time consuming
    - Invasive
    - Not in line with current purchase journeys (e.g. online, rapid, quick delivery)
- Customer purchase experience could be significantly improved if applications were based on predictive models of a parsimonious set of data points.

The Insurer

- Insurers categorize people into risk classes based upon their expected mortality (how long they will live).
- The risk class tells the insurer how much to charge in premium.
- This is often supplemented by physical examinations.
- The process is very costly for the insurer (up to $200).
- The insurer could save money and attract more business if risk class could be predicted using a small set of data points.

The Data: Prudential Application Data

The Dataset

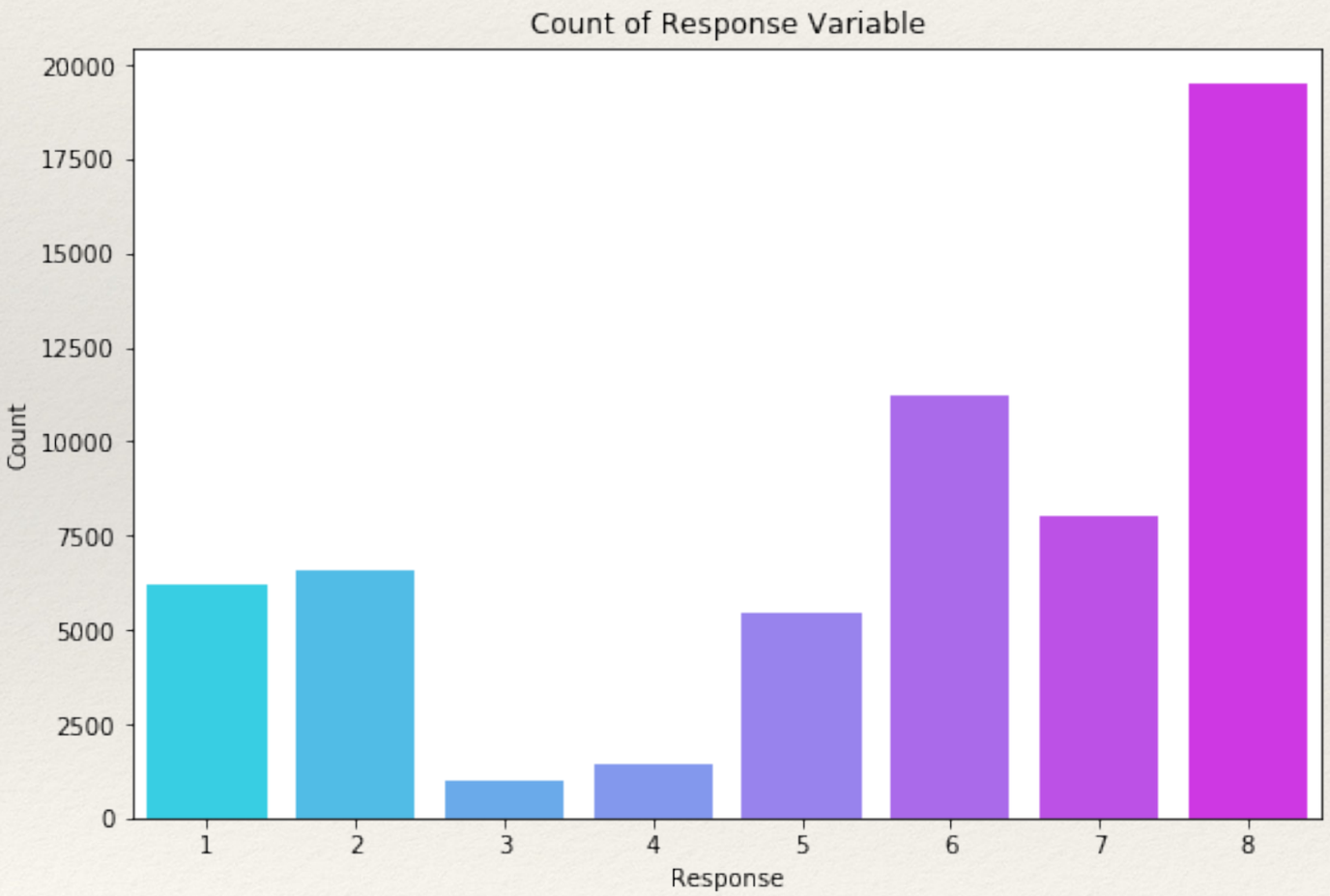
- In 2015 the Prudential life insurance company released a dataset to the Kaggle community.
- The response variable is an ordinal variable (0-8) which represents the risk classes.
- The predictors are applicant attributes that have been drawn from a self-reported questionnaire that asks information about applicant health and lifestyle.
- There are 60,000 rows of applicant data with response values.
- All of the features are reported anonymously so there was very little scope for feature engineering

Features

Variable	Description
Id	A unique identifier associated with an application.
Product_Info_1-7	A set of normalized variables relating to the product applied for
Ins_Age	Normalized age of applicant
Ht	Normalized height of applicant
Wt	Normalized weight of applicant
BMI	Normalized BMI of applicant
Employment_Info_1-6	A set of normalized variables relating to the employment history of the applicant.
InsuredInfo_1-6	A set of normalized variables providing information about the applicant.
Insurance_History_1-9	A set of normalized variables relating to the insurance history of the applicant.
Family_Hist_1-5	A set of normalized variables relating to the family history of the applicant.
Medical_History_1-41	A set of normalized variables relating to the medical history of the applicant.
Medical_Keyword_1-48	A set of dummy variables relating to the presence of/absence of a medical keyword being associated with the application.
Response	This is the target variable, an ordinal variable relating to the final decision associated with an application

The Response & Defining Success

Understanding the Response



Defining Success

- The key metric for the success of the model is the mean per class error. This is the loss function that I seek to minimize.
- Pure chance would mean that we could guess the risk class 1/8 of the time meaning that the mean per class error that I need to beat to be better than random is 0.875.
- For incorrect predictions the distance they are from the true rate class is important as this costs insurers money. So the root mean square error will also be important.

The Method: Gradient Boosting

Implementing GBM

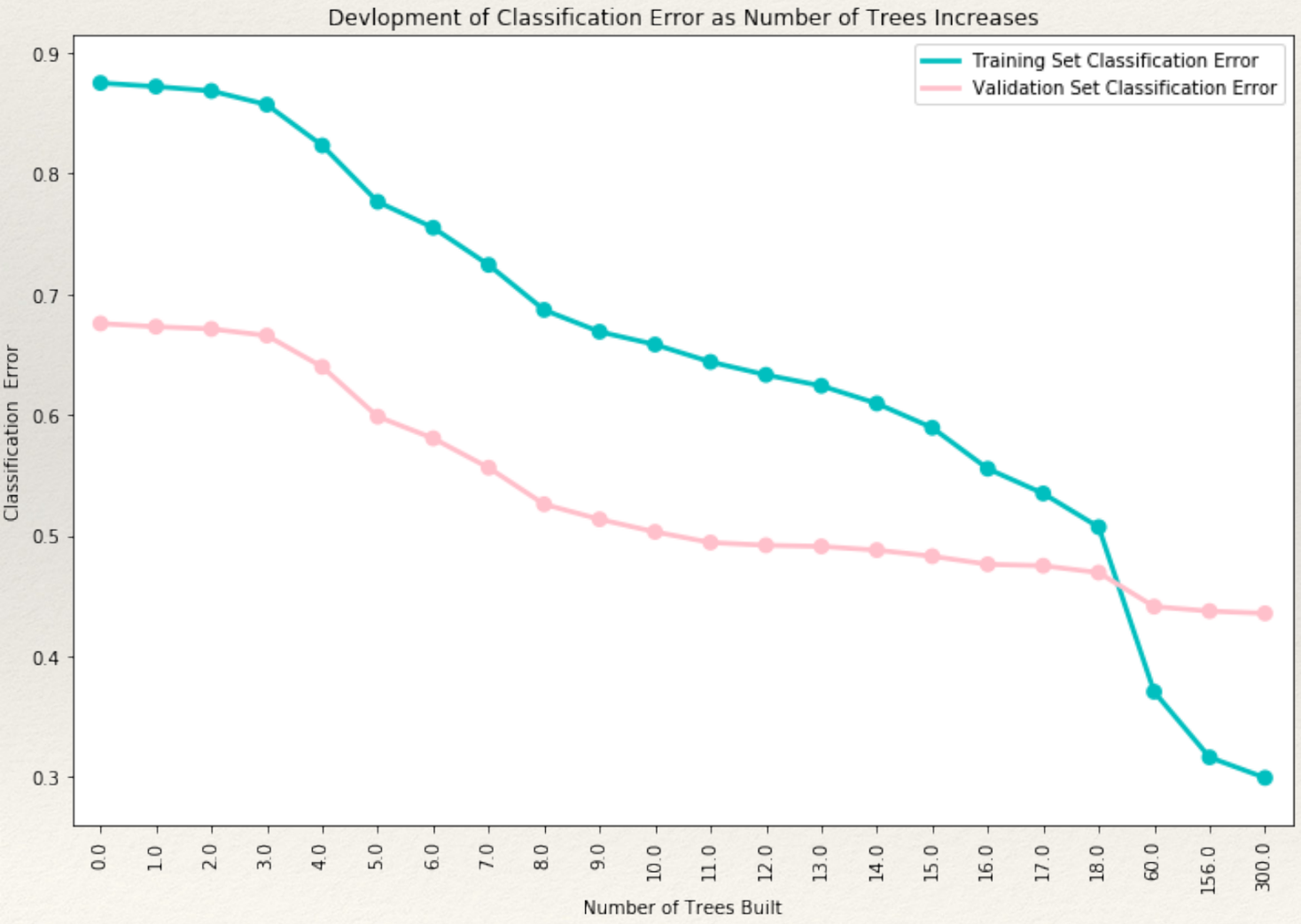
- Boosting is a forward learning decision tree model
- A weak learner (i.e. a tree that is only slightly better than chance) generates predictions. Another tree is built on the residuals (errors) from this first tree. The first tree is updated based on the second, and so on.
- I used the H2O implementation of the GBM algorithm.
- I conducted model tuning on an Amazon EC2 Linux machine with 64 cores (8 times more powerful than my mac pro) for added speed.

Tuning Method

1. Get baseline mean per square error scores by running model with default parameters and 5 fold cross validation
2. Conduct grid search for learning_rate (how quickly the error is corrected from each tree to the next) and max_depth (number of nodes for each tree) parameters to narrow the search field. Make decisions based on validation scores
3. Conduct gird search for other relevant parameters (particularly those that lessen overfitting which was an issue). Select model with highest validation scores.
4. Generate scores using best model with test data.

Results of the Final Model

Development of the Final Model



Scoring the Final Model

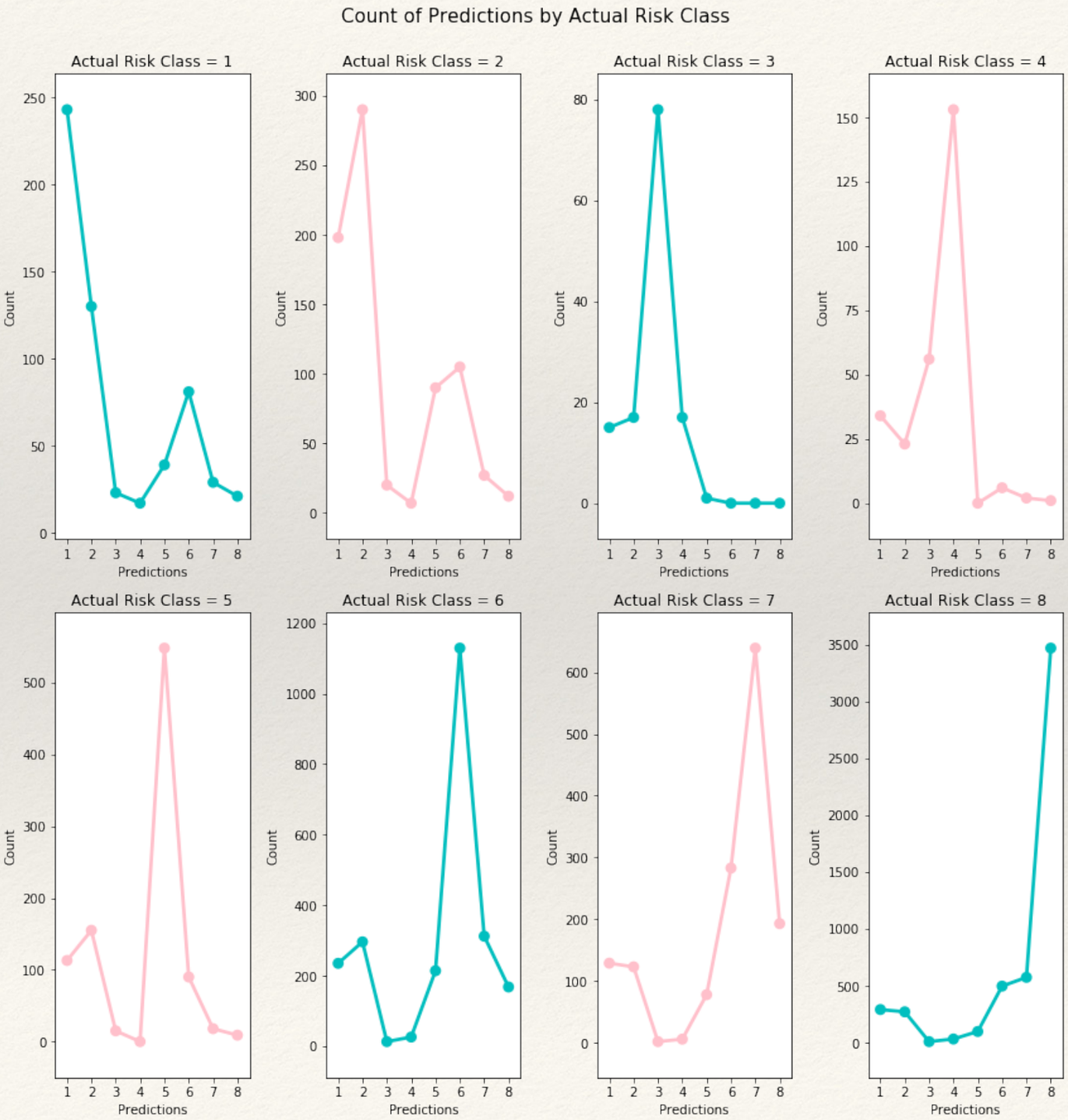
- Training Mean Per Class Error: 0.299
- Validation Mean Per Class Error: 0.523

- Training Root Mean Square Error: 0.568
- Validation Root Mean Square Error: 0.644

- **Test Mean Per Class Error: 0.538**
- **Test Root Mean Square Error: 0.653**

```
Prediction 1: 0.476809287062006 5
Prediction 2: 0.484684563124787 36
Prediction 3: 0.518363834857830 6
Prediction 4: 0.547740841147318 4
Prediction 5: 0.570328307825510 6
Prediction 6: 0.501774000656548 1
Prediction 7: 0.443025014230276 57
Prediction 8: 0.676190696669620 3
```

Results: How Serious Are the Errors?



Analysis

- The root mean square error is less than 1, so for each observation that is wrong the prediction is likely to be very close to the actual risk class. This lowers the risk of using a less than perfect model.
- The peak of each graph is where the actual risk class and the predicted risk class is the same. What we want to see is a downward slope in both directions.
- This we do see in most cases. Predicting risk class 1 and 2 seems problematic.

Conclusions and Next Steps

Conclusions

- Using a GBM model I am able to make predictions that are correct around 50% of the time.
- For those predictions that are incorrect the loss that would be incurred due to the error is manageable due to the low root mean square error.
- A major limitation of the data was not being able to engineer my own features. I have access to much higher quality data so I am hopeful that what I have presented will be capable of being deployed once more work is done.

Next Steps

1. Work on real data
2. Explore other H2O techniques and particularly the ensemble methods.
3. Explore options for increasing computational power through using GPU enabled machines.