Capstone Project

Machine Learning Engineer Nanodegree

Hamdy Tawfeek

January 26th, 2019

Project Overview

In a one-click shopping world with on-demand everything, the life insurance application process is antiquated. Customers provide extensive information to identify risk classification and eligibility, including scheduling medical exams, a process that takes an average of 30 days.The result? People are turned off. That’s why only 40% of U.S. households own individual life insurance. Prudential wants to make it quicker and less labor intensive for new and existing customers to get a quote while maintaining privacy boundaries.

In this project, I developed a predictive model that accurately classifies risk using a more automated approach, and that will greatly impact public perception of the industry. The results will help insurance companies better understand the predictive power of the data points in the existing assessment, enabling them to significantly streamline the process.

Problem Statement

Can you make buying life insurance easier?
By developing a predictive model that accurately classifies risk using a more automated approach, you can greatly impact public perception of the industry.

In machine learning terms, we have a supervised problem of type classification. Which means we have a set of data points about features and response variable. And the response variable is a categorical one (i.e it has different categories). Given the data points, our job is to find the relation between the features and the response variable so we will be able to predict the behavior of new data points.
The features we will use in our problem are numerical description of the insurance applicant like his/her employment information, family history, BMI, medical history, and other relevant features. The response variable is the final decision associated with an application which designates the risk of accepting this applicant.

Solution Statement

In this project, I would like to use compare different predictive classification models and see which is better. I will implement logistic regression, XGBoost, Neural Network.

Benchmark Model

In this project I will use **Logistic regression** as my benchmark model.

Evaluation Metrics

I will use the quadratic weighted kappa, which measures the agreement between two ratings as my evaluation metric, so I could submit and compare my solutions with other kagglers . This metric typically varies from 0 (random agreement) to 1 (complete agreement). In the event that there is less agreement between the raters than expected by chance, this metric may go below 0.

Kaggle describes the metric as follows:
The response variable has 8 possible ratings. Each application is characterized by a tuple (ea,eb), which corresponds to its scores by Rater A (actual risk) and Rater B (predicted risk). The quadratic weighted kappa is calculated as follows.

First, an N x N histogram matrix O is constructed, such that Oi,j corresponds to the number of applications that received a rating i by A and a rating j by B. An N-by-N matrix of weights, w, is calculated based on the difference between raters' scores:

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

An N-by-N histogram matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater's histogram vector of ratings, normalized such that E and O have the same sum.

From these three matrices, the quadratic weighted kappa is calculated

as:

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}.$$

I will aslo use mean absolute error to tune my hyperparameters in xgboost. MAE is a common and simple metric that has the advantage of being in the same unit as our target, which means it can be compared to target values and easily interpreted. You can compute MAE by summing the absolute errors between your predictions and the true values of the target and averaging over all observations,

---

Data Exploration

The dataset provided by Prudential on Kaggle.
In this dataset, a hundred variables describing attributes of life insurance applicants. The task is to predict the “Response” variable for each Id in the test set. “Response” is an ordinal measure of risk that has 8 levels.

File descriptions

- train.csv - the training set, contains the Response values
- test.csv - the test set, you must predict the Response variable for all rows in this file

Data fields as described by Kaggle

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.
Response	This is the target variable, an ordinal variable relating to the final decision associated with an application

The following variables are all categorical (nominal):

Product_Info_1, Product_Info_2, Product_Info_3, Product_Info_5, Product_Info_6, Product_Info_7, Employment_Info_2, Employment_Info_3, Employment_Info_5, InsuredInfo_1, InsuredInfo_2, InsuredInfo_3, InsuredInfo_4, InsuredInfo_5, InsuredInfo_6,

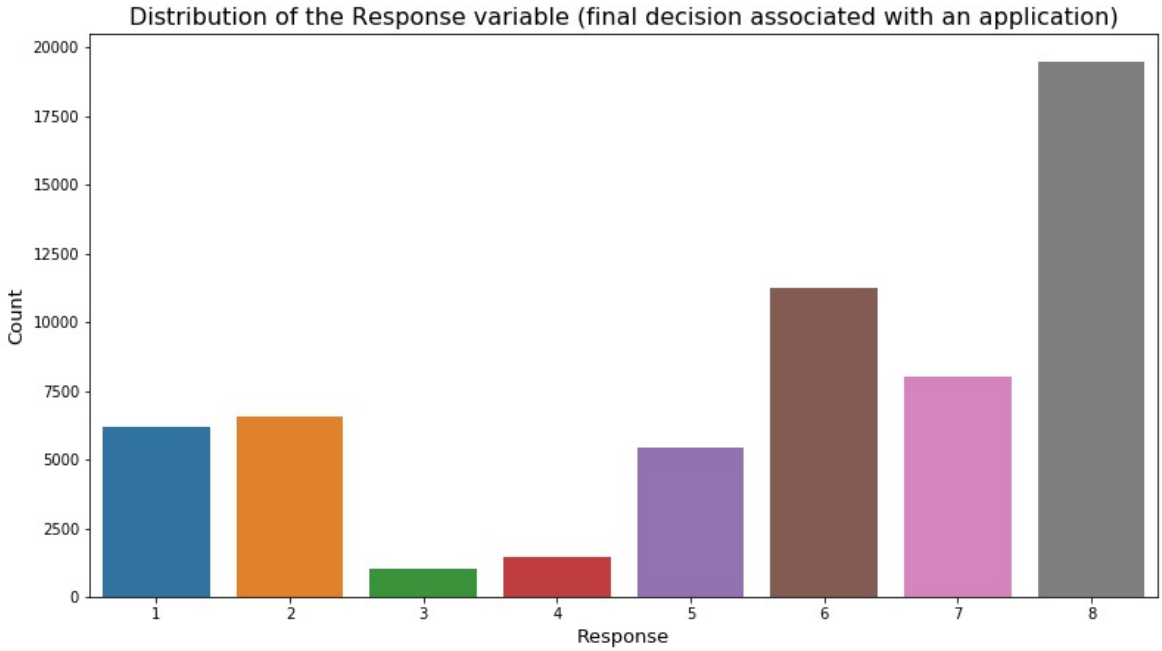
InsuredInfo_7, Insurance_History_1, Insurance_History_2, Insurance_History_3, Insurance_History_4, Insurance_History_7, Insurance_History_8, Insurance_History_9, Family_Hist_1, Medical_History_2, Medical_History_3, Medical_History_4, Medical_History_5, Medical_History_6, Medical_History_7, Medical_History_8, Medical_History_9, Medical_History_11, Medical_History_12, Medical_History_13, Medical_History_14, Medical_History_16, Medical_History_17, Medical_History_18, Medical_History_19, Medical_History_20, Medical_History_21, Medical_History_22, Medical_History_23, Medical_History_25, Medical_History_26, Medical_History_27, Medical_History_28, Medical_History_29, Medical_History_30
edical_History_31, Medical_History_33, Medical_History_34, Medical_History_35, Medical_History_36, Medical_History_37, Medical_History_38, Medical_History_39, Medical_History_40, Medical_History_41

The following variables are continuous:
Product_Info_4, Ins_Age, Ht, Wt, BMI, Employment_Info_1, Employment_Info_4, Employment_Info_6, Insurance_History_5, Family_Hist_2, Family_Hist_3, Family_Hist_4, Family_Hist_5

The following variables are discrete:
Medical_History_1, Medical_History_10, Medical_History_15, Medical_History_24, Medical_History_32
Medical_Keyword_1-48 are dummy variables.

Exploratory Visualization

I started exploring the data by first looking at the distribution of the response variable. It turned out that cateegory 8 of the response variable is the most common
value between other categories. Both categories 3 and 4 are least common categories in the response variable.



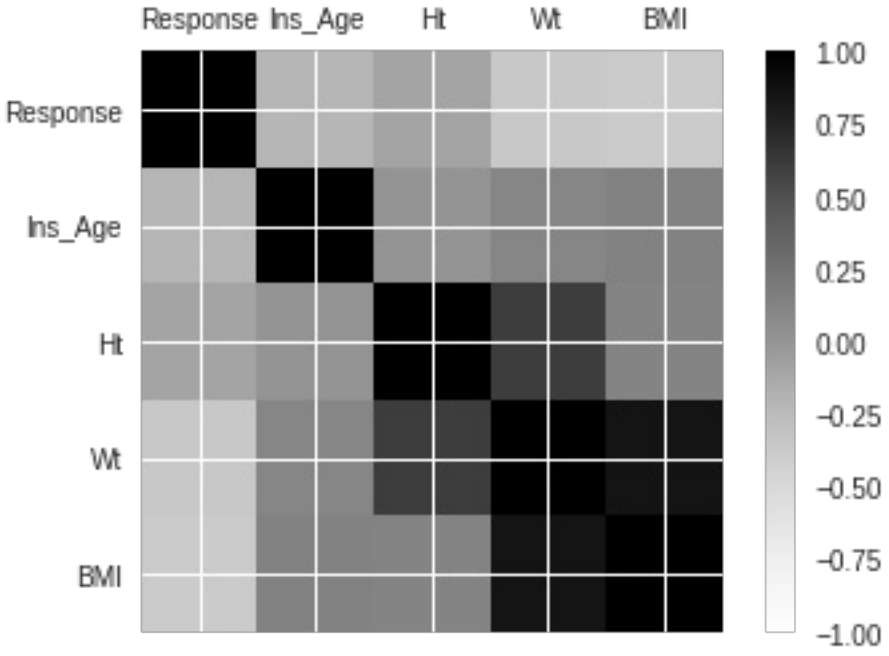
Then I took a look at the proportion of the missing values across all the variables in the training data. I find that variable “Medical_History_32” is almost missing, also other variables as you can see from the figure. I used this information later in the data preprocessing part to drop columns with too many missing values.

```
print('proportion of nan values in train set : ')
prop = train.isnull().sum(axis = 0)/len(train)
prop[prop != 0.0]
```

```
proportion of nan values in train set :

Employment_Info_1     0.000319968
Employment_Info_4     0.114161095
Employment_Info_6     0.182785740
Insurance_History_5   0.427678887
Family_Hist_2         0.482578603
Family_Hist_3         0.576632256
Family_Hist_4         0.323066301
Family_Hist_5         0.704114111
Medical_History_1     0.149694347
Medical_History_10    0.990619895
Medical_History_15    0.751014634
Medical_History_24    0.935989626
Medical_History_32    0.981357673
```

Then I explored physcial caracteristics of the applicants by visualizing variables
like Height, Weight and BMI in relation to the response using a correlation matrix.



Data preprocessing

In the data preprocessing step, I did four things. First I dropped all the mostly missing features like ‘Medical_History_10’,‘Medical_History_15’,‘Medical_History_24’,and ‘Medical_History_32’. I also dropped ‘Id’ feature as it dosn’t convey information.

Secondly, I filled in the missing values by the mode if the variable was cateorical, and by the median if the variable was continuous.

Then, I did one-hot encoding to the categorical variables so that I can train my models on the data.Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric. In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

Finally, I scaled the numerical variables to standardize the range of the variables. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the

features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.

Tools Explanation

I this project, I used python as the main tool to do the analysis and model training. I used packages like:

- matplotlib: for plotting
- pandas: for dealing with structured data
- numpy: for computations
- sklearn: for preprocessing the data and logistic regression training
- keras: for neural netwrok training
- xgboost: for xgb training, and hyperparameters tuning

Models and Techniques

I choosed to train and compare between two very successful classifiers in kaggle competitions recently relative to the benchmark model **Logistic regression**:

- **XGBoost**
- **Neural Network**

- **Logistic regression** makes predictions using probability. It’s a classification algorithm, that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

    - 0 = you are absolutely sure there is no risk in accepting this applicant insurance request.
    - 1 = you are absolutely sure that it’s very risky to accept this applicant insurance request.
    - Any value between 0 and 1 convey a corresponding degree of risk about an applicant.
      Logistic regression is simple and efficient. It provides probability score for observations.

- **XGBoost** stands for eXtreme Gradient Boosting. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict.Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

I used grid search to tune the main parameters of your XGBoost model. In an ideal world, with infinite resources and where time is not an issue, you could run a giant grid search with all the parameters together and find the optimal solution. Here I tuned only a subset of the hyperparameters that are usually having a big impact on performance.

I started by tuning max_depth and min_child_weight hyperparameters, Those parameters add constraints on the architecture of the trees. max_depth is the maximum number of nodes allowed from the root to the farthest leaf of a tree. Deeper trees can model more complex relationships by adding more nodes, but as we go deeper, splits become less relevant and are sometimes only due to noise, causing the model to overfit. min_child_weight is the minimum weight (or number of samples if all samples have a weight of 1) required in order to create a new node in the tree. A smaller min_child_weight allows the algorithm to create children that correspond to fewer samples, thus allowing for more complex trees, but again, more likely to overfit.

Then I tuned subsample and colsample_bytree. Those parameters control the sampling of the dataset that is done at each boosting round. subsample corresponds to the fraction of observations (the rows) to subsample at each step. By default it is set to 1 meaning that we use all rows. colsample_bytree corresponds to the fraction of features (the columns) to use. By default it is set to 1 meaning that we will use all features.

- **Neural Network** are based on how biological brains work. They

are a copy cat of biological brains but obviously simpler in every imaginable way. Neural nets have artificial neurons which are counterparts of biological neurons, they behave in a similar manner. They all learn from examples and become better at a particular task at hand just from examples, they can learn to recognize faces for example just by being presented with faces.

For comparing the results of my models, I used kaggle leaderboard to score my models and the results were as following.

Submission and Description	Private Score	Public Score
nn_submission.csv 5 hours ago by hamdymostafa	0.57424	0.57065
xgb_submission.csv 6 hours ago by hamdymostafa	0.57224	0.56787
lr_submission.csv 6 hours ago by hamdymostafa	0.49694	0.48729

I beleive we can reach to a better solution if we used a wide range for hyperparameters and tune more of them but that needs more time.

Conclusion

I will use Neural Netwrok model to make buying life insurance easier by accurately classifying risk. Now we can greatly impact public perception of the industry.

Reflection

The process used for this project can be summarized as following:

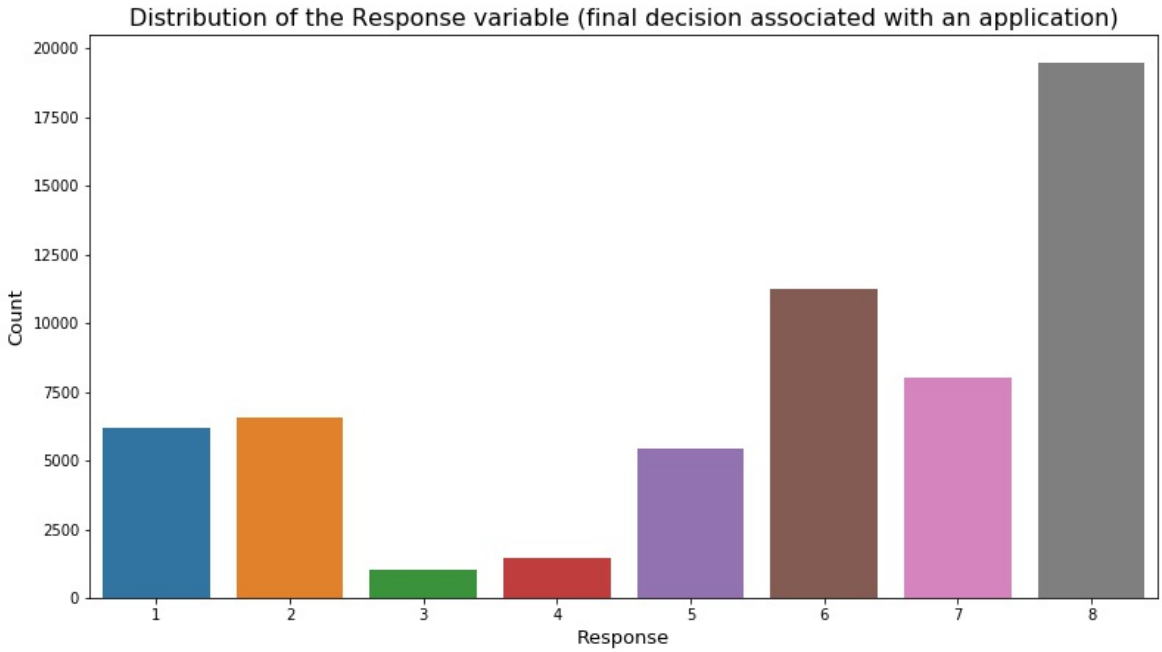
1. I choosed a kaggle problem so I could work on a real-life problem.
2. The problem setup with easy as it was mainly made by kaggle.
3. I used python as my tool for carrying out the analysis. I used also google colab
   for heavy computations.
4. Visualing the data and understanding it was a challenge as the variables are not intuitive to understand.
5. I choosed a simple algorithm like logistic regression as my benchmark model to compare my results to.
6. I trainded logistic regression, xgboost, and neural network.
7. I comapred between the different classifiers.

For me the most difficult part is when tunning xgboost model and I don’t know when it will stop. Except that, the whole experience was great.Now I feel more confident to work with real-life problems and compete in kaggle competitions.

Improvements

One of the areas where we can improve our model is to consider the issue of the imbalanced categories in the response variable. Imbalanced data typically refers to a problem with classification problems where the

classes are not represented equally. As you see in the figure below, Category 3 and 4 are less represented in the training data which make it harder for the model to be trained on these categories.



One of the solution to this problem is resampling the dataset. You can change the dataset that you use to build your predictive model to have more balanced data. This change is called sampling your dataset and there are two main methods that you can use to even-up the classes:

- You can add copies of instances from the under-represented class called over-sampling (or more formally sampling with replacement), or
- You can delete instances from the over-represented class, called under-sampling.

These approaches are often very easy to implement and fast to run. They are an excellent starting point.

References

- https://en.wikipedia.org/wiki/Feature_scaling
- https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
- https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
- https://www.quora.com/What-is-logistic-regression
- https://www.quora.com/How-would-explain-neural-network-theory-to-a-non-technical-person
- https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f