Capstone Project

Machine Learning Engineer Nanodegree

Kevin Palm
August, 2016

I. Definition

Project Overview

The official project overview can be found on the competition description page. The following overview is just my own paraphrasing and interpretation of the competition goals/context.

Red Hat is a software company which specializes in open-source, linux-based enterprise solutions. They're interested in improving their models which classify potential clients as future clients, which is understandable because once that a company knows who their serious prospects are they can devote extra resources to winning them over.

They've just released three anonymized data sets for Kagglers to use as they compete on creating that improved model.

The first data set is a one pertaining to the people in the data set - potential and past clients - and what kind of characteristics those people have. All of the characteristics are anonymized, in the sense that the column headers are named things like "char_1" and "char_2", rather than "Gender" or "Location". Most of the characteristics are booleans. There is also an unlabeled date column and a feature called "group_1". The people data set can be joined to the other two files using a unique identifier field they provide.

The second data set is the competition training data, which similarly contains a date column, a feature called "activity category", and 10 anonymized characteristic columns - this time presumably pertaining to the characteristics of each activity. In addition, there is a final column called "outcome", which has two potential values of 0 or 1, and which is the feature which the competition has us attempting to predict.

The third data set is the testing data, which is exactly the same format as the training data except that it lacks the "outcome" column.

The problem domain of this challenge definitely includes supervised machine learning. Also, I think this competition will be hugely a problem of exploratory data analysis and observation. The anonymized features introduce a lot of challenge in the sense that they remove most elements of business intuition and make the contestants almost totally relent on raw data analysis. But at the same time, that reliance on data analysis can provide the benefit or eliminating researcher bias, in the sense that I might give each feature a more thorough look into its applications than if I were to come in with preconceptions about each feature's usefulness.

I think this project has special application to business development and marketing departments for companies which tailor to enterprises. The framing of the challenge matches the overall theme of how many businesses are attempting to automate and make data driven processes for "qualifying" their marketing and sales leads.

Problem Statement

Ultimately, the goal of this project is to create a list of true/false predictions which append to the testing data set. The competition guidelines don't specifically explain what it is that competitors are predicting, but I would guess the general gist of it is something like "Will this prospect become a client in the next thirty days?" So, given that the training/testing data actually represents activities, I think a good problem statement for this project would be this:

As potential customers interact with Red Hat in the future, some of their activities will be of interest and some will not, and we want to know which so Red Hat can better use their selling resources.

My expected tasks towards a solution to this problem statement are:

1. **Exploratory data analysis and data joining -** there's going to be a lot of EDA required for this project. Understanding how that these data sets were created, split up, and set up will be critical to creating the right model in the end. I don't have the history or any inside knowledge about the quirks of the data, and I'll need to know as much as I can to create the right model. It will be an investigation. During this phase, I'll need to join the people data set to the other data sets.
2. **Feature preparation -** I'll format the data in an appropriate manner for my final model. Exactly how I go about this will be hugely reliant on the EDA step, as how I condition features and which algorithm I intend to use will be dependant on what I've learned so far. Because of the relatively large size of this data set, I will likely need to use some method of dimensionality reduction, such as PCA.
3. **Early Modelling -** I'll create a model that outputs predictions, using a subset of the training set so that I can use classification metrics on the leftover training data. At this point my top two expected algorithms that I might use to address this problem are linear SVMs or gradient boosting. I'll use the metric scores to tune my model.
4. **Model -** I'll apply my model testing data which outputs a predictions file in the correct format for submission to kaggle.com, and submit the entry.
5. **Repeat -** I'll go back, learn more, and improve.

Metrics

This Kaggle competition is scored on area under receiver operating characteristic curve (AUC). AUC is a bit less intuitive of a classification metric, but in the words of the top reply to **this excellent blog post on the subject**:

> Pick a random negative and a random positive example; The AUC gives you the probability that your classifier assigns a higher score to the positive example (ie, ranks the positive higher than the negative).

– *Directly copied from Peter Prettenhofer, but also the same explanation is cited on Wikipedia from "Fawcett, Tom (2006); An introduction to ROC analysis, Pattern Recognition Letters, 27, 861–874"*

I think probably the reason this metric was chosen over accuracy is that the ratio of each output label is not equal, and one of the benefits of AUC is that it isn't negatively affected by imbalances in the frequency of labels.

Understanding why that this competition is scored on AUC over, say, F1 score (which is also able to deal with imbalanced label frequencies) is less clear. The best discussion I've found comparing F1 and AUC to each other is from **this stackoverflow thread**. My best guess right now is that Red Hat must be interested in a model that will perform well as conditions and label frequencies experience structural change over time, and that the changing thresholds comprising AUC could simulate that change. Perhaps we're making a model to last through the good times and the bad, the times of plenty and the days of tough sells.

Anyways, that's my best guess. I think left to my own devices I would prefer to use F1 as a metric for this project, but in the spirit of competition I'll be optimizing for AUC.

II. Analysis

Data Exploration

There are 55 total columns after merging the people data to the training data. One of which is the output label feature to predict. Two are id columns, two are date columns, and two are group/type columns (one of each for people and one of each for activities). The the rest are characteristics, 10 of which pertain to activity characteristics and 38 of which pertain to person characteristics.

Output Labels

There are 975,497 rows of data with labels of 1, and 1,221,794 rows with labels of 0 (a grand total of 2,197,291 rows of training data).

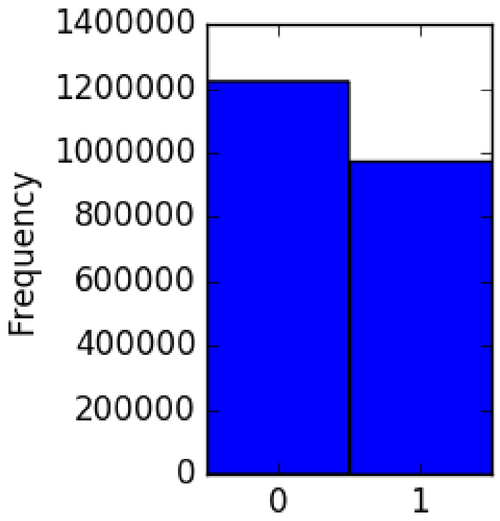

Figure 1: Histogram of Output Label Frequency

So the output labels are binary and somewhat close to equally distributed. Exploring these labels' relationships with the other features is the whole point of this project, but as a first step it's nice to know that there's a somewhat balanced amount of examples for each label in our training data.

Date Columns

The date field for the activities ranges from 2022-07-17 through 2023-08-31, so we have about one year of anonymized activities data. The date field for people ranges from 2020-05-18 through 2023-08-31, so whatever people date represents we have about 3 years worth of it.

A **really fantastic exploration of the relationship between the frequency of these dates and the output labels that I really can't outdo** was created by Kaggle user anokas. The big take home observations of the analysis where:

- There's a strong weekend/weekday trend to the activities date, in which more activates occur during weekdays and the likelihood of positive labels on weekends drops.
- There's also a weekend/weekday trend to the people date frequencies, but on a considerably lesser scale. Any trends between the likelihood of a given label based on people date is less apparent.
- The training and the testing sets have similar frequency distributions for activity date, suggesting that the train test split was not based on time.

- The training and testing sets have a lesser similarity between people date, but which grows more similar each progressive year.

I think a reasonable hypothesis concerning the dates is that the activity date pertains to the date of the actual activity, and the people date pertains to some significant event that every person has. An example of what the people date might be could be the date that they signed up for an account on the website, or the date that they first contacted Red Hat.

Group/type Columns

There are only seven activity types, and most of them don't look to be that great of indicators for the output labels on their own.

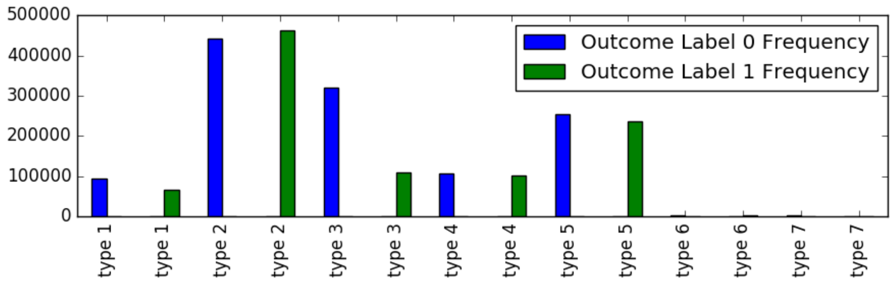

Figure 2: Bar Chart of Output Label Frequency by Activity Type

Of all the activity types, type 3 looks like it has the most information pertaining to the labels on its own. However, recall that the overall training set has more 0 labels than 1s, so some of the increased likelihood for label 0 is exaggerated. It's certainly still possible that this feature will be more useful to us in conjunction with other features.

For the people group columns, there are 29,899 unique groups in the training data and 11,640 unique groups in the testing data. There's a huge focus on this feature right now on the Kaggle competition page, because the feature in conjunction with the action date **was recently demonstrated by Kagglers loiso and team as able to be used to achieve ~0.987 AUC with just some logic and simple statistics**. A really great **explanation of the technique was written by dmi3kno**. The whole thing is getting called a hack or a leak, but the overall consensus is that the discovery doesn't necessarily break the competition. It just makes the competition a small numbers game about perfecting predictions for the relatively few rows of data in which their label cannot be directly inferred from the original features.

So what is group_1? It appears to be clusters of people who all had the same dates for when their activities switched labels. So for instance, for a given cluster, we might be able to infer that anything before January 3rd was labeled 0 and anything after is labeled 1. Very powerful stuff considering that those labels are the whole point of the competition, and a good example of how powerful EDA can be.

Characteristics Columns

There are 10 characteristic columns pertaining to activities, and 38 pertaining to people.

All of the activities characteristics are groupings, with values such as “type 1” and “type 2”. Most of those columns appear to have less than 50 possible values, but one does include more than 7,000 types.

Nine of the people characteristics are also groupings, and they’re all less than 50 types. Twenty-eight are booleans. And one is an integer class, with values from 0 to 100.

I published **my own kernel exploring the people characteristics exclusively, and whether or not it makes sense to use decomposition techniques on them prior to joining them into the data set**. I found that all of the characteristics in the people data set were interrelated. After expanding all the features out to be one-hot encoded (and for the case of the integer column, min-max scaled) there were 160 resulting columns. PCA worked pretty well, with 28 components explaining 80% of the variance in the data, and 50 components explaining 90% of the variance in the data. So I think using activities characteristics while keeping my dimensions under control will work out.

Exploratory Visualization

Because the data leak is going to be an important part of this competition, and because it’s a little tricky to understand exactly how the leak happened, I’m devoting this section to my benchmark model **which is inspired by and very similar to the loisso team’s leak model**.

The following graphic is five separate scatter plots, each of a randomly selected people group from the “group_1” feature, with predictions from my benchmark model. The graph was definitely inspired by one that **dmi3kno created originally**, but this one pertains specifically to **my implementation of the leak/exploit model**.

On the x axis we have the days ongoing since the group first appeared, and on the y axis is the output label. Hopefully looking at this graph, how that the benchmark model inferred the labels that it is a little more intuitive.

To walk through the logic, though, the model basically looks at each point that it needs to estimate. It looks at the nearest output from the training set on the right, and it looks at the nearest output on the left. If they both agree, it

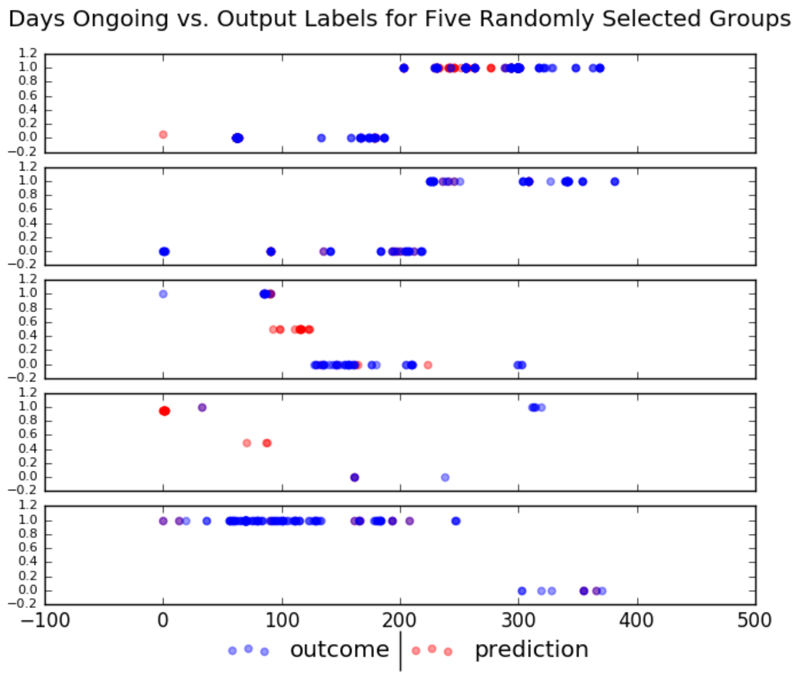

Figure 3: Multiple Scatter Plots of the Benchmark/Leak Model Predictions

assigns the same label to the point it is currently estimating. If they disagree, it assigns a label of 0.5. If the point happens to be on the rightmost or leftmost extreme, it assumes the closest output label is correct. Finally, it adds a little cushioning to any of the less certain values, such as the values on the rightmost and leftmost extremes.

No doubt a lot of kagglers are going to use an inferential leak model as their first pass, and then estimate the remaining data points with a more sophisticated model. I think that's a very valid approach, as there are still 70,000 remaining points that have disagreeing labels to each side of them. I think the other dominant approach will be to create a model that accounts for the leak by "translating" it into the input features. Whichever way the problem is approached, the relationship between activity dates and people groups is very important!

Algorithms and Techniques

Given that this project is for a machine learning nanodegree, I'm going to use the approach of translating the leak (and other relevant features) into a machine learning model!

So what will my final regressor be? I think this is a classically good use case for gradient boosting. Gradient boosting works by starting off with a weak learner, say a short decision tree. It does its best to apply that little decision tree and starts off with a broad stroke that correctly classifies the most data points that it can. But inevitably there will be points that don't fit the major trend. So next, the algorithm focuses in on those points that it classified wrong by calculating what's called a "residual", which is just a measurement of how wrong it was for each point. Then it does its best to apply another weak learner, but this time it fits to the residual. It adds up the first weak learner and the second weak learner, and calculates a residual for their combined explanatory power. Then if fits a third weak learner on the current residual. It repeats this again and again until it reaches some maximum allowed number if iterations, or it explains the training data perfectly.

So for the problem at hand, I think it'll be very doable to set up a bunch of decent predictor features like normal, but also include a feature suggesting which label the leak would classify and some measure(s) of that label's dependability. Say that I created a feature that was a scale of 1-100 for how dependable the leak approach was for each point - a gradient boosting regressor might train individual decision trees for each step along that scale of 1-100. At least that would be a simplified possible outcome of using gradient boosting on this problem.

Some specific examples of what could constitute those measures of the leak feature's dependability might be:

- How many units away the right and left nearest neighboring labels are
- The density of the neighbors within some range, and their label uniformity

- If the right and left nearest neighbors even have the same label

In addition to the features that translate the data leak to my final estimator, I want to include information pertaining to:

- Date seasonality - like what day of the week, or which month. I'll try creating these kinds of features for both the activity date and the people date, then they'll need to be one hot encoded.
- Boolean characteristics - the characteristics that are true/false will be easiest to include in the model, they just need to be converted to 0s and 1s.
- Categorical characteristics and activity type - I want to include information from the characteristics and activity type features, but how that I can do that depends on how many categories are in each feature. I definitely can't just one-hot encode the features that have 1,000+ categories, that will be way too many dimensions. So it'll be a combination of one hot encoding, and for the big ones I'll be using feature hashing or **leave one out encoding**.
- Integer characteristics - there's only one of these. I expect to just end up scaling this feature, but I may experiment with one-hot encoding if I end up deviating from the gradient boosting regressor.

Then finally, on top of all this, I'm expecting to have some PCA or RCA in the mix just to try and keep my dimensions under control. Gradient boosting is a computationally expensive algorithm and this is a big data set, so keeping my running time and memory requirements reasonable is going to be a major concern.

Benchmark

For this project, my benchmark is going to be my exploit/leak inference model that's sort of a simplified version of lasio and team's. The exploit model is mostly about filling NaNs in a tricky way, so I figure that my final model should be able to beat that with the right feature engineering, as the target information is pretty plainly contained in the data.

It may turn out to be a challenging benchmark. For context, the exploit model scores 0.987028 AUC (1.0 is the maximum possible) and at the time of writing this scoring anything above that puts you in the top 15% of the leaderboard.

Not to say that this project will be evaluated entirely off the Kaggle leaderboard, though. When cross validating locally using a split of the training data file, I'll be evaluating both my final model and the leak model. I hope to have a final model that beats the leak model for every random split of the data.

III. Methodology

Data Preprocessing

To start off my data preprocessing, I needed to translate the data leak into my estimator. I also needed to resplit my training and testing data so that the leaked rows from the testing data were instead included in the training data.

Ultimately, I created four features to translate the leak. One is a boolean denoting the closest known label from the same group looking backwards in time, and one is the same looking forward. Then I created a feature which is equal to the distance to that backward point as a proportion of the overall group density (number of known labels from the group divided by the date range for the group). And again, a similar feature for looking forward. The units on those latter two features are a bit strange - technically it's days squared over activities - but it seems to work out for the model. Finally, for all the missing values I filled with -1, because those four features are otherwise always positive and gradient boosting should be fine to handle the -1s as their own case. The code for this stage of the preprocessing is located in utilities.py under extract_leak_features().

Next, I tried to create some real features. For activity date I created one-hots for the day, weekday, and month. For people date I created one-hots for the month and the year. I got to copy over all the features that were already booleans, I scaled the one ordinal feature to be between 0 and 1 as well, and then I one-hot encoded the remaining categorical features that didn't contain more than 100 unique possible values. At this point, I'm dropping all the columns that contain NAs and too many categories to one-hot encode - I may revisit the issue later to see if I want to salvage anything.

The features data set is very large at this point of the script. It contains 227 columns, all with values between 0 and 1. The column names are somewhat descriptive (considering they're anonymized variables…), so I've listed them below.

['act_day_1' 'act_day_10' 'act_day_11' 'act_day_12' 'act_day_13' 'act_day_14' 'act_day_15' 'act_day_16' 'act_day_17' 'act_day_18' 'act_day_19' 'act_day_2' 'act_day_20' 'act_day_21' 'act_day_22' 'act_day_23' 'act_day_24' 'act_day_25' 'act_day_26' 'act_day_27' 'act_day_28' 'act_day_29' 'act_day_3' 'act_day_30' 'act_day_31' 'act_day_4' 'act_day_5' 'act_day_6' 'act_day_7' 'act_day_8' 'act_day_9' 'act_month_1' 'act_month_10' 'act_month_11' 'act_month_12' 'act_month_2' 'act_month_3' 'act_month_4' 'act_month_5' 'act_month_6' 'act_month_7' 'act_month_8' 'act_month_9' 'act_weekday_0' 'act_weekday_1' 'act_weekday_2' 'act_weekday_3' 'act_weekday_4' 'act_weekday_5' 'act_weekday_6' 'people_month_1' 'people_month_10' 'people_month_11' 'people_month_12' 'people_month_2' 'people_month_3' 'people_month_4' 'people_month_5' 'people_month_6' 'people_month_7' 'people_month_8' 'people_month_9' 'people_year_2020' 'people_year_2021' 'people_year_2022' 'people_year_2023' 'char_38' 'activity_category_type 1' 'activity_category_type 2' 'activity_category_type 3' 'activity_category_type 4' 'activity_category_type 5' 'activity_category_type 6' 'activity_category_type 7' 'char_1_type 1' 'char_1_type 2' 'char_10' 'char_11' 'char_12' 'char_13' 'char_14' 'char_15' 'char_16' 'char_17' 'char_18' 'char_19' 'char_2_type 1' 'char_2_type 3' 'char_20' 'char_21' 'char_22' 'char_23' 'char_24' 'char_25' 'char_26' 'char_27' 'char_28' 'char_29' 'char_3_type 1' 'char_3_type 10' 'char_3_type 11' 'char_3_type 12' 'char_3_type 13' 'char_3_type 14' 'char_3_type 15' 'char_3_type 16' 'char_3_type 17' 'char_3_type 18' 'char_3_type 19' 'char_3_type 2' 'char_3_type 20' 'char_3_type 21' 'char_3_type 22' 'char_3_type 23' 'char_3_type 24' 'char_3_type 25' 'char_3_type 26' 'char_3_type 27' 'char_3_type 28' 'char_3_type 29' 'char_3_type 3' 'char_3_type 30' 'char_3_type 31' 'char_3_type 32' 'char_3_type 33' 'char_3_type 34' 'char_3_type 36' 'char_3_type 38' 'char_3_type 39' 'char_3_type 4' 'char_3_type 40' 'char_3_type 41' 'char_3_type 5' 'char_3_type 6' 'char_3_type 7' 'char_3_type 8' 'char_3_type 9' 'char_30' 'char_31' 'char_32' 'char_33' 'char_34' 'char_35' 'char_36' 'char_37' 'char_4_type 1' 'char_4_type 10' 'char_4_type 11' 'char_4_type 12' 'char_4_type 13' 'char_4_type 14' 'char_4_type 15' 'char_4_type 16' 'char_4_type 17' 'char_4_type 18' 'char_4_type 19' 'char_4_type 2' 'char_4_type 20' 'char_4_type 21' 'char_4_type 22' 'char_4_type 23' 'char_4_type 24' 'char_4_type 25' 'char_4_type 3' 'char_4_type 4' 'char_4_type 5' 'char_4_type 6' 'char_4_type 7' 'char_4_type 8' 'char_4_type 9' 'char_5_type 1' 'char_5_type 2' 'char_5_type 3' 'char_5_type 4' 'char_5_type 5' 'char_5_type 6' 'char_5_type 7' 'char_5_type 8' 'char_5_type 9' 'char_6_type 1' 'char_6_type 2' 'char_6_type 3' 'char_6_type 4' 'char_6_type 5' 'char_6_type 6' 'char_7_type 1' 'char_7_type 10' 'char_7_type 11' 'char_7_type 12' 'char_7_type 13' 'char_7_type 14' 'char_7_type 15' 'char_7_type 16' 'char_7_type 17' 'char_7_type 18' 'char_7_type 19' 'char_7_type 2' 'char_7_type 20' 'char_7_type 21' 'char_7_type 22' 'char_7_type 23' 'char_7_type 24' 'char_7_type 25' 'char_7_type 3' 'char_7_type 4' 'char_7_type 5' 'char_7_type 6' 'char_7_type 7' 'char_7_type 8' 'char_7_type 9' 'char_8_type 1' 'char_8_type 2' 'char_8_type 3' 'char_8_type 4' 'char_8_type 5' 'char_8_type 6' 'char_8_type 7' 'char_8_type 8' 'char_9_type 1' 'char_9_type 2' 'char_9_type 3' 'char_9_type 4' 'char_9_type 5' 'char_9_type 6' 'char_9_type 7' 'char_9_type 8' 'char_9_type 9']

Finally, on this big giant data set, I run a PCA and only keep the 20 components of greatest variance. The code for this stage of the preprocessing is located under utilities.py under prep_features().

So I end up with a total of 24 features in my final model.

Implementation

I used the scikit-learn gradient boosting regressor implementation for my final model. Because sklearn handles all the heavy lifting, all my implementation required was:

1. I extracted the leak features and resplit the data based on the leaks
2. I extracted the principle components and joined them with the leak features
3. I trained the gradient boost regressor on the outcomes and features, and appended the predictions to the test data frame
4. I returned predictions and the leaked labels to the original testing index.

I had a little difficulty with memory management while I was setting up the PCA stage of this project, but otherwise I had pretty smooth sailing. By the time that I was feature engineering, I already had a pretty clear idea of what I wanted to try first because of all the EDA that I did at the beginning of the project.

Refinement

The out of the box gradient boosting regressor did beat the benchmark score when submitted to Kaggle at this point. It scored 0.987882 AUC, which is 0.000854 AUC higher than the benchmark. At the time of writing, that translates to 0.004842 AUC under the leading model, and in the top 20% of the leaderboard.

So next I started local testing with different parameter tunings. I set up my local test to split by people ID, with 5,000 people IDs and their corresponding activities going to each the training and the testing sets. I did not worry about keeping the output labels proportionally stratified in my local tests, on the reasoning that the gradient boosting model and the leak model are fundamentally quite similar and so my model and my comparison point would be similarly affected by each sampling.

I focused on three parameters - learning rate, n estimators, and max depth - but I also dabbled in whitening my PCA outputs and changing the gradient boosting loss function. The latter two had significantly negative effects on my final output, so I abandoned them by the time I was conducting structured tests. The table below is a complete matrix of the combinations of:

- learning rate = [0.1, 0.2, 0.3, 0.4, 0.5]
- n estimators = [50, 100, 150]
- max depth = [2, 3, 4]

In the table below, the local score is actually an average of three runs per tuning.

Learning Rate	N Estimators	Max Depth	Local Score (Delta Benchmark)	Kaggle Score (AUC)
0.3	50	2	0.040443315	0.987842

Learning Rate	N Estimators	Max Depth	Local Score (Delta Benchmark)	Kaggle Score (AUC)
0.3	100	2	0.008453154	0.987832
0.3	100	3	0.008119399	0.987881
0.3	150	4	0.00497335	0.987732
0.1	100	4	0.004393546	0.987882
0.2	150	3	0.003847023	
0.5	150	3	0.003387742	
0.4	50	3	0.00311012	
0.2	100	4	0.002753689	
0.5	100	2	0.002540337	
0.4	100	2	0.00240208	
0.2	50	3	0.002017758	
0.4	50	2	0.001837789	
0.3	150	2	0.001612978	
0.3	50	4	0.001266027	
0.5	100	3	0.00101742	
0.4	50	4	0.000894402	
0.5	100	4	0.000522531	
0.1	50	2	0.00012683	
0.1	100	2	0.000115795	
0.1	50	3	2.90E-05	
0.2	150	4	-0.000111787	
0.5	50	3	-0.000138638	
0.3	150	3	-0.000510806	
0.4	150	2	-0.000605365	
0.2	50	4	-0.000685423	
0.5	150	2	-0.000743299	
0.2	100	3	-0.000752007	
0.5	150	4	-0.000770367	
0.1	50	4	-0.001132329	
0.4	150	4	-0.001658237	
0.2	150	2	-0.001841178	
0.4	100	3	-0.002059669	
0.4	100	4	-0.002239869	
0.1	100	3	-0.00224634	0.987882
0.5	50	4	-0.002260407	
0.5	50	2	-0.003087298	
0.1	150	3	-0.00349696	
0.1	150	4	-0.005492377	
0.2	100	2	-0.006507676	
0.3	100	4	-0.009030611	
0.4	150	3	-0.010184013	
0.1	150	2	-0.012296184	
0.3	50	3	-0.012841561	

Learning Rate	N Estimators	Max Depth	Local Score (Delta Benchmark)	Kaggle Score (AUC)
0.2	50	2	-0.015455808	

IV. Results

Model Evaluation and Validation

In my local tests I decided to optimize for the difference from the benchmark model, because the small size of my samples introduced a lot of fluctuation in AUC scores but the benchmark model fluctuated in parallel. The best ones I did try submitting to Kaggle, and while they do all beat the benchmark, none of them improve beyond the out of the box model. So at this point, my final model is the sklearn out of the box gradient boosting regressor.

On the local miniature tests, the final model seems to beat the benchmark model about 2/3rds of the time. So at least it is overall better. That beats the benchmark model while using the full training set is more compelling.

Switching to the big picture here, just looking at the model performance 0.987882 AUC is a fantastic ROC AUC score. There's less than 2% of improvement left to fight for in this model. It all depends on context, but I think generally in real world applications this would be past the point where most machine learning engineers stop investing time because the return on investment is so little.

At least it would be an excellent AUC score if it were a real model.

It's unfortunate for Red Hat that their competition contains a data leak, because the models in this competition are all built using information that won't be available to their actual work. So while my model seems very dependable and would be highly valuable, it can't be used in any sort of real work setting.

In terms of the robustness of the model, it's not too bad. When training on only 5,000 people's data, the AUC is seems pretty constantly plus or minus 0.025 of 0.9 AUC. This seems reasonable for such a reduction of training data.

Justification

The benchmark model scores 0.987028 AUC and my model scores 0.987882 AUC when using all of the training data, so while my model isn't the best it does successfully perform better than some tricky NaN filling. It's not a huge difference, but in terms of eliminating the degree of error remaining that's 6.6%. At the beginning of this project I defined beating that benchmark as my goal, and I did manage to successfully do that.

V. Conclusion

Free-Form Visualization

I started working on this project before I realized there was a data leak, and for a while after learning about the problem I was a bit bummed that my final project wouldn't be on a real machine learning problem as a result of the leak. I stuck with the project because I'd already invested some time, and I'm very glad that I did for one specific reason in particular.

The data leak provided a window into visualizing my model results that I could make a lot of sense looking at - meaning when I was able to look at a visualization and I have an intuition towards what that I would classify each point at if I were doing them by hand. The result was that I prepared my leak features with very specific intentions - I felt like I was "translating" the leak to my gradient boosting regressor - and then opening up my visualizations to see if my efforts had the intended effect. I also opened up the same visualization while I was tuning my algorithm parameters, and having that concrete view into the effects was really great for learning.

The visualization that has been my primary window throughout this project is the same one that we used above to look at the benchmark model. Again, each separate graph is one randomly selected group, with the y axis being the outcome label probability, and the x axis being the days ongoing.

I'm at a point on this project where I'm pretty happy with how my model is placing the predictions in this visualization. I think it agrees pretty well with how I would expect them to be, yet I also expect them to be better than anything I could do by hand because it's now accounting for extra features in addition to just the leak features.

Reflection

For this project, the start had a huge component of EDA. The leak features were the most critical to understand because some 97% of the performance of the final models in this competition comes directly from those features, but in addition there was a big challenge to understand the other features well enough to engineer meaningful, efficient features that don't overwhelm your model in terms of dimensions. After I felt that I had a grasp on the data, I went about engineering features to represent the leaked data, and then I engineered additional features which I used PCA to reduce down to 20. I then trained a sci-kit learn gradient boosting regressor model to create final predictions.

The most challenging aspect of this project was understanding and dealing with the data leak. The reason I ended up reimplementing the benchmark model from loiso was to make sure that I was understanding the nature of the leak (and to speed up the python implementation). Once I felt like I had a good

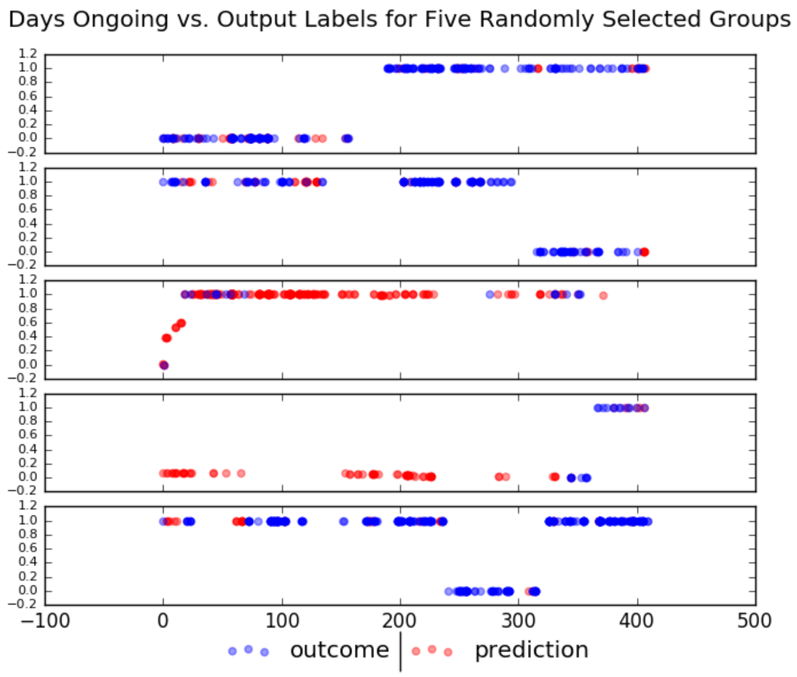

Figure 4: Multiple Scatter Plots of the Final Model Predictions

handle on what was wrong with this data set, I was able to section apart my feature engineering into into leak features and regular features. This helped me make better sense of the problem, and hopefully to section away my learnings into what's useful generally and what I can only apply to this specific kind of problem.

My model doesn't have any general, real-world applications. Because of the data leak, this model was built using information that won't be available in a real-world setting. Otherwise, I think my model is pretty decent! My model does fit my expectations and I did meet my goals for this project. I'm glad I stuck through it despite the data leak, because it offered a very unique learning experience and really ended up reinforcing the value of exploratory data analysis.

Improvement

I think gradient boosting really is the way to go with this problem, but I'm less certain about my decision to use PCA as my method for dimensionality reduction on those additional features. Simple feature selection might have been a better decision. My reasoning is that the bulk of my data points are "covered" by the leak features (their tree building probably doesn't use any of the additional features), so the emphasis that PCA puts on explaining the most amount of variance is probably wasting precious space in my final data set. Another possibility is that I should only fit my PCA component using rows of data which are difficult for my model to predict using just the leak features.

Another possibility is switching from the sklearn gradient boost implimentation to xgboost. When I finish with this project for my nanodegree purposes, I probably will convert the project to python 3 and experiment further.