Mystery Binary Classification Challenge Report

Spencer Yue (sty223)

Kaggle Username: SpencerYue

Overview

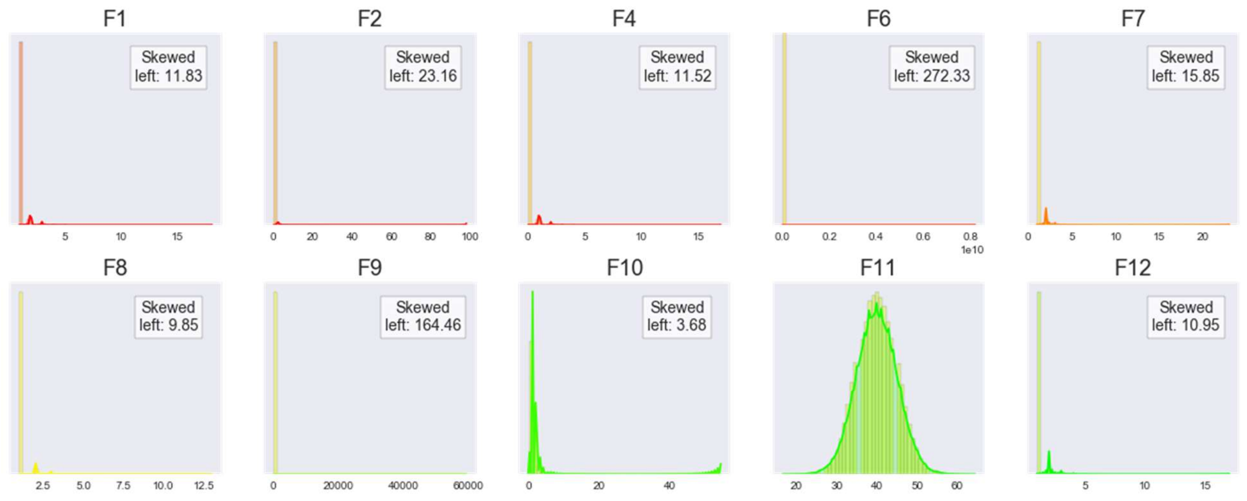
I focused my efforts primarily on investigating how feature engineering would affect xgboost models. After experimenting and getting feedback from the public and private Kaggle scores, I have gained several practical lessons. These include the need to use (and believe the results of) cross validation, to establish a baseline with the simplest models without preprocessing first, and to spend time building a smooth pipeline for grid-searching and testing ideas. I've also observed many small insights about decision trees and possibly other models in general. The most significant of these observations was that the "curse of dimensionality" can be a major tradeoff to consider when adding newly engineered features (such as with one-hot encoding).

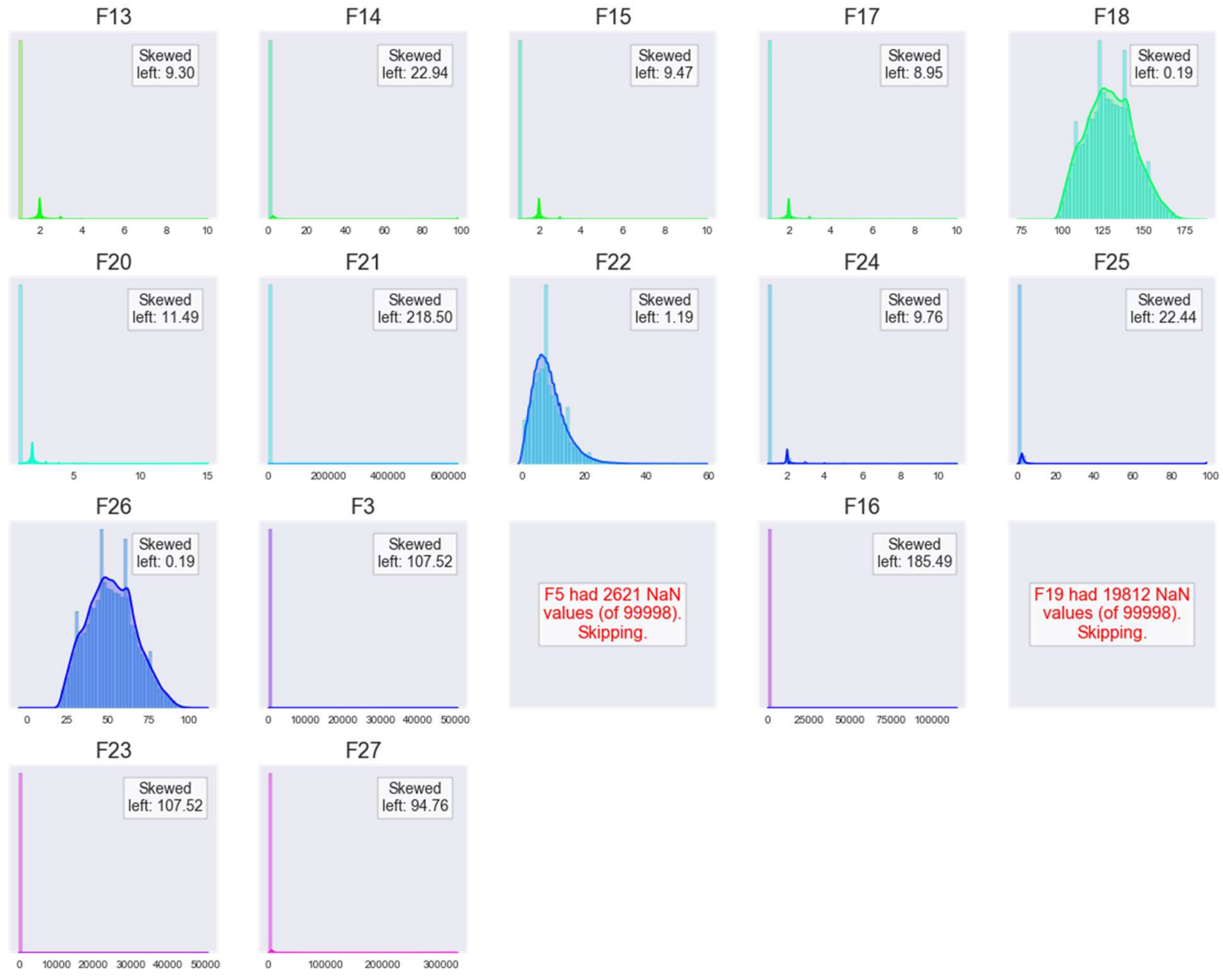
Investigations and Experiments

1. Unskew Transforms

My first question for this data set, and as a general curiosity, was whether the distributions of feature values could negatively affect a model's performance. I had been investigating in the recent homework labs whether certain feature transformations that would result in a target distribution would help a given model, but I did not reach any conclusive results. (This had been mostly in the context of linear regression models. The initial idea of measuring skew as a feature transformation step was from Alexandru Papiu's Housing Prices notebook.) Even though decision trees are not linear, I wanted to investigate this as a feature engineering step, because it is one of the few knobs that I as opposed to the model have direct control of and can graph to hopefully understand.

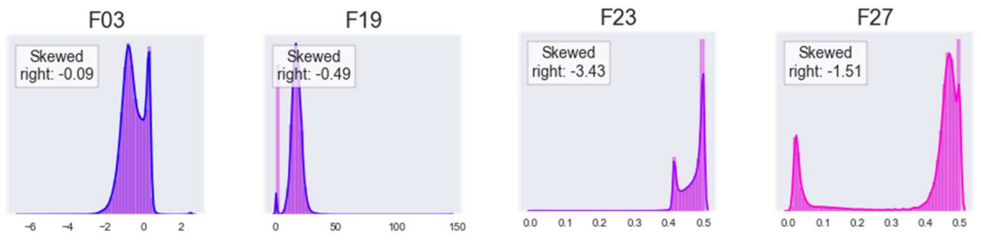
Here are the histograms of the initial vanilla data set with skew statistics included as labels. (F11 was not skewed according to scipy's skewtest function so I omitted its label.)





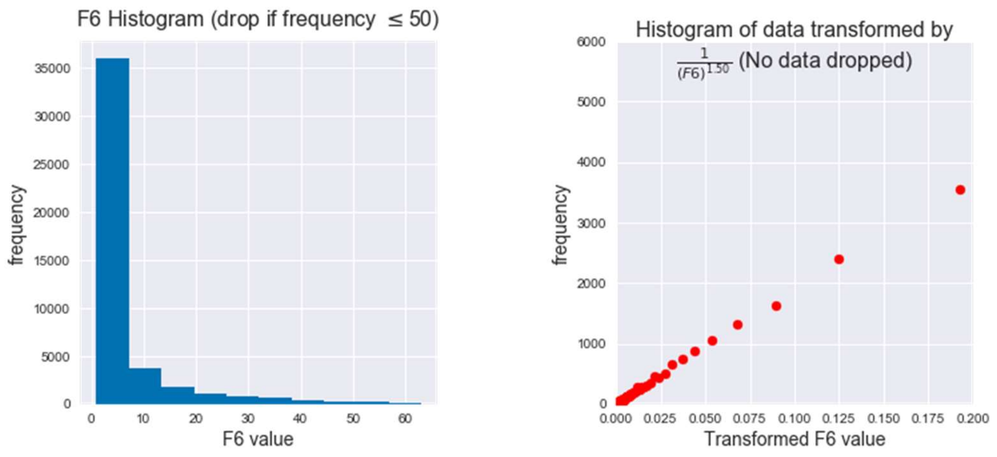
I wanted to target the normal distribution. To do this, I measured the "deviation from normal" using scipy's skewtest and skew statistic functions. I applied several transformations to minimize this statistic for each feature. These transformations include log, square root, cube root, reciprocal, scipy's Box-Cox function, and my custom implementation of the histogram flattening algorithm (a technique from the image processing domain).

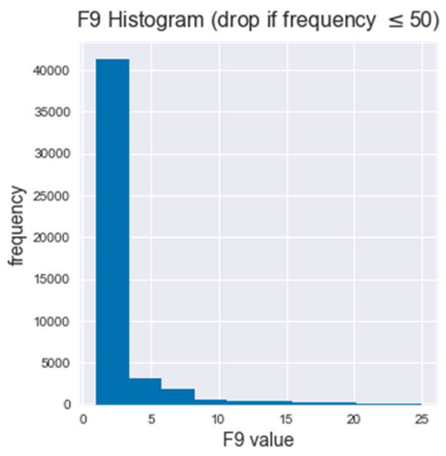
The following are unskewed histograms of the continuous variables (“continuous” guessed by the number of unique values in the feature and the data type).

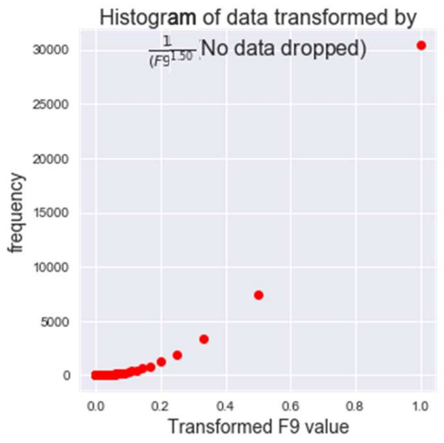


The transforms chosen to minimize the skew for these features were F3: Box-Cox, F19: histogram flatten, F23: Box-Cox, and F27: inverse of square-plus-one. These four numerical features contributed roughly 0.08 to the overall AUC-score included in the xgboost models versus being omitted. I could not, however, conclude if they helped xgboost more than their original counterparts however.

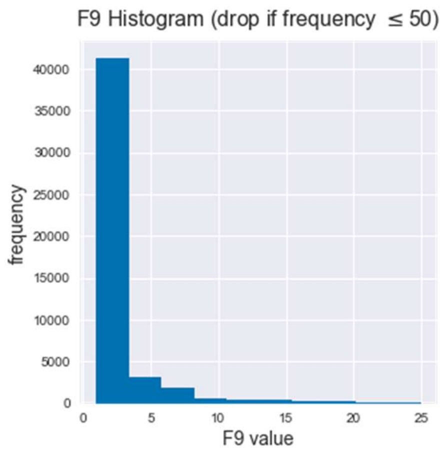
I also investigated feature distributions individually in more detail to try to learn something about the nature of the data. From the following transformed distributions, I saw that features F6 and F9 almost exactly followed an inverse power distribution, which suggested to me this data set was most likely artificial, and I did not pursue any further to try to find any meaning in the data.

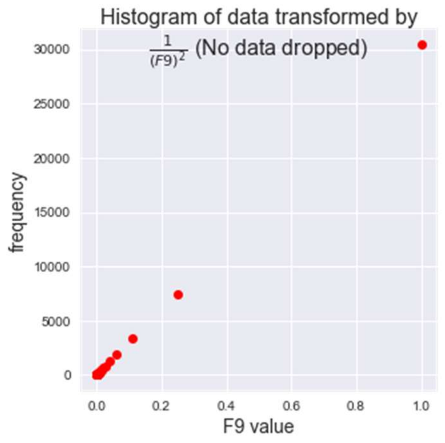






(Compare this 1.50 power above to the squared power below.)





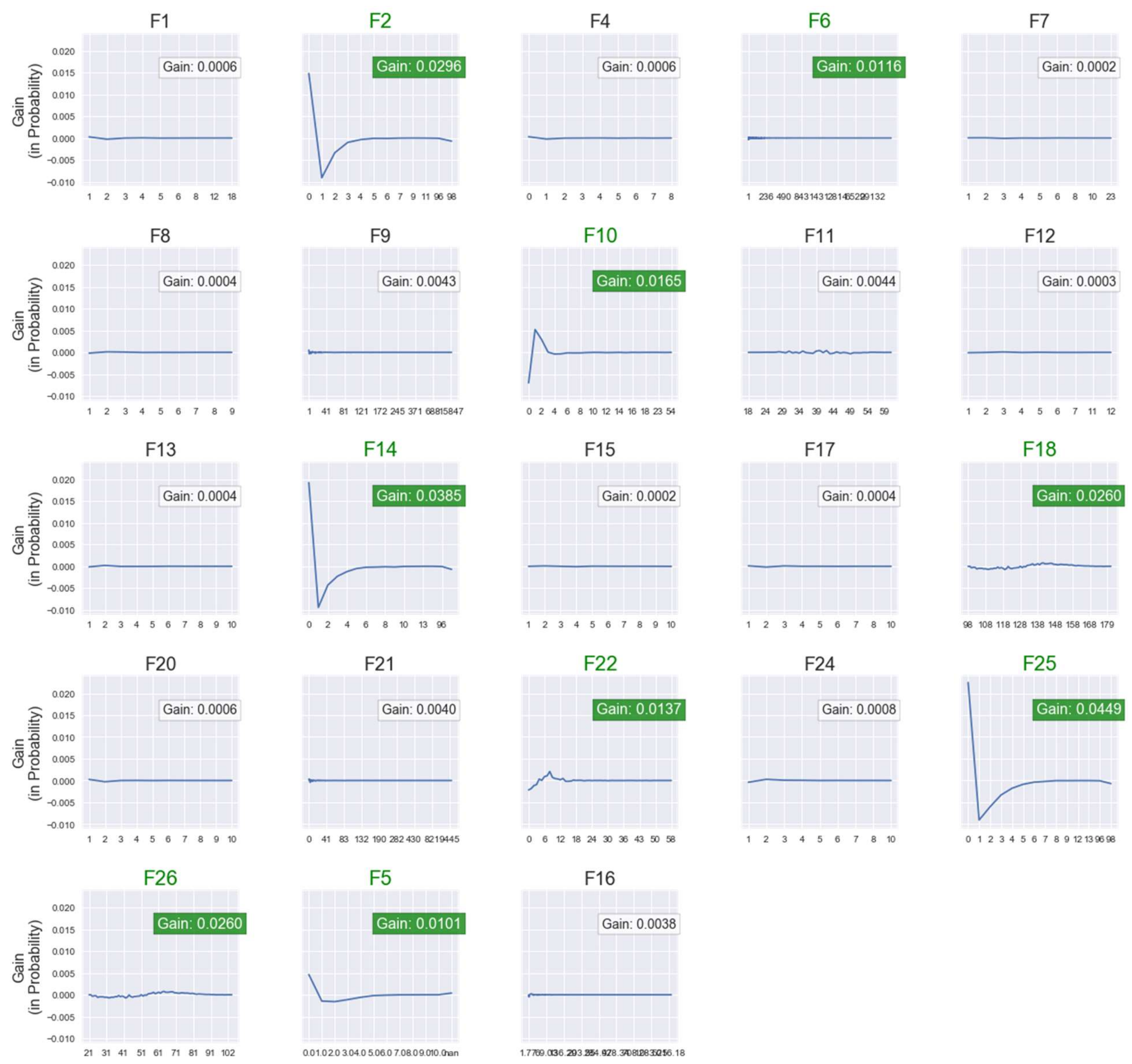
2. Feature engineering

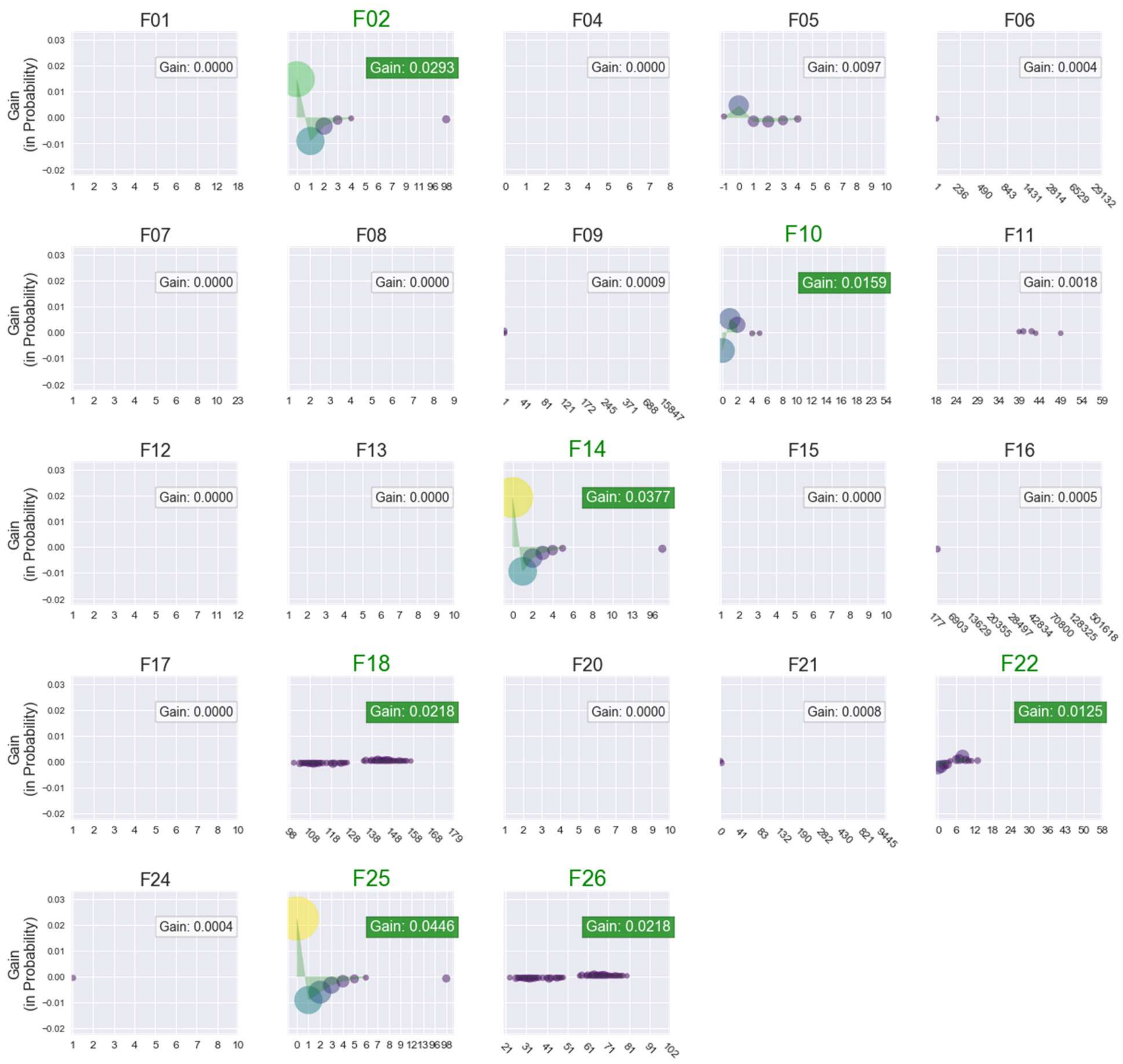
I noticed that many of the features in the data set had very a relatively low number of unique values and were of integer type. This suggested the idea of treating them as categorical variables and one-hot encoding them. To do this efficiently, I found that using a SparseDataFrame and encoding the 0's in the one-hot encoding as missing values saved considerably on space. I also read about the GPU implementation of xgboost to see if this would be efficient (https://peerj.com/articles/cs-127.pdf). In order to reduce the number of features generated by one-hot encoding, I grouped certain low-occuring feature values together. I measured the predicted effect this would have on the prediction using a custom measure I called *gain*. The gain of feature F1=1 for instance is defined as

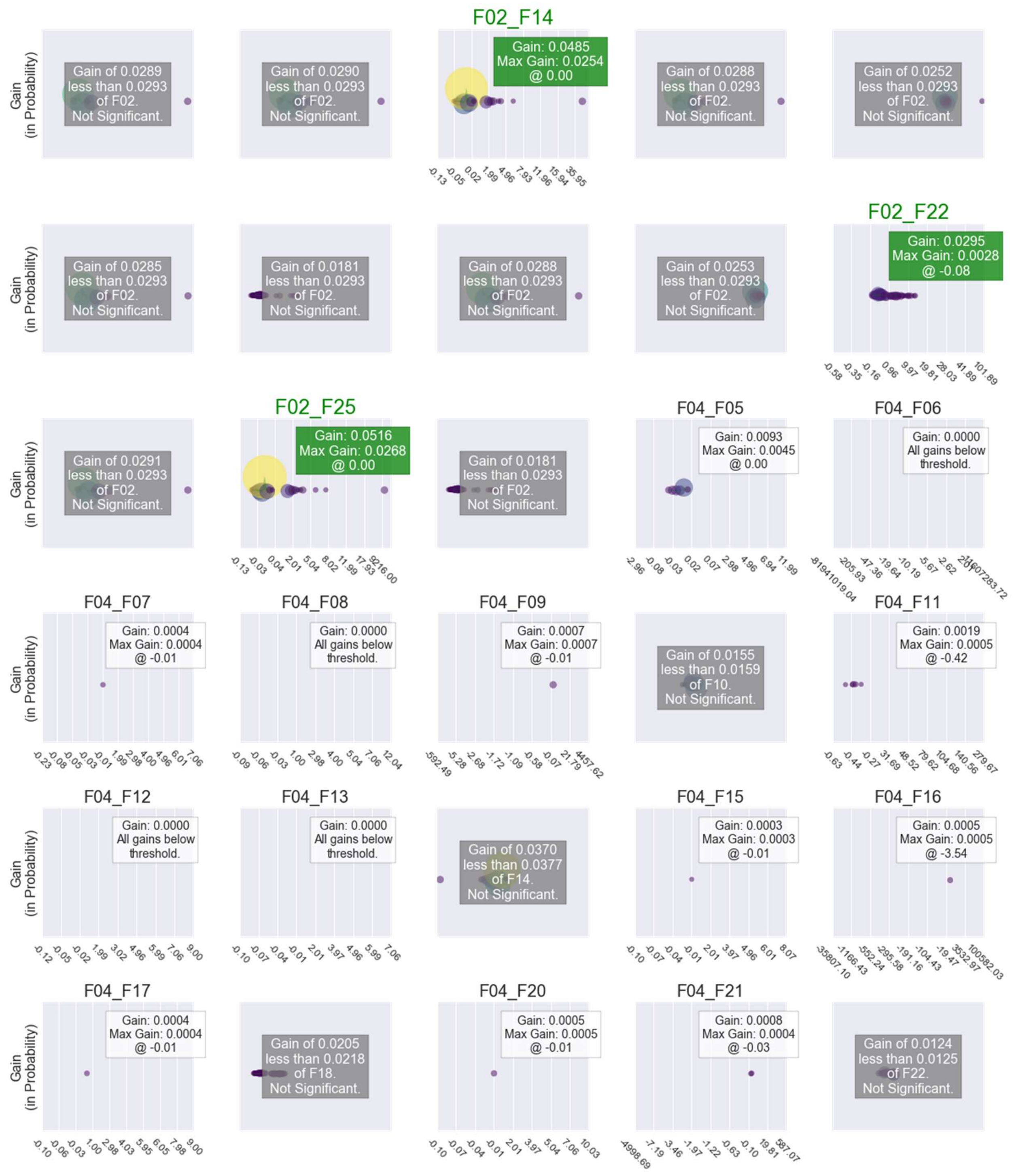
$$\text{gain}(F1=1) = P(F1=1 \text{ and } Y=0) - P(Y=0).$$

The absolute value of the gain is used to measure how independent the feature value is from the target class. This was also used to in a separate investigation after the contest ended to try feature selection on interaction pairs.

Here are some of the plots produced in the feature engineering process:


Gain (in Probability)
F1
Gain: 0.0006
F2
Gain: 0.0296
F4
Gain: 0.0006
F6
Gain: 0.0116
F7
Gain: 0.0002
F8
Gain: 0.0004
F9
Gain: 0.0043
F10
Gain: 0.0165
F11
Gain: 0.0044
F12
Gain: 0.0003
F13
Gain: 0.0004
F14
Gain: 0.0385
F15
Gain: 0.0002
F17
Gain: 0.0004
F18
Gain: 0.0260
F20
Gain: 0.0006
F21
Gain: 0.0040
F22
Gain: 0.0137
F24
Gain: 0.0008
F25
Gain: 0.0449
F26
Gain: 0.0260
F5
Gain: 0.0101
F16
Gain: 0.0038


F01
Gain: 0.0000
F02
Gain: 0.0293
F04
Gain: 0.0000
F05
Gain: 0.0097
F06
Gain: 0.0004
F07
Gain: 0.0000
F08
Gain: 0.0000
F09
Gain: 0.0009
F10
Gain: 0.0159
F11
Gain: 0.0018
F12
Gain: 0.0000
F13
Gain: 0.0000
F14
Gain: 0.0377
F15
Gain: 0.0000
F16
Gain: 0.0005
F17
Gain: 0.0000
F18
Gain: 0.0218
F20
Gain: 0.0000
F21
Gain: 0.0008
F22
Gain: 0.0125
F24
Gain: 0.0004
F25
Gain: 0.0446
F26
Gain: 0.0218
Gain
(in Probability)
0.03
0.02
0.01
0.00
-0.01
-0.02



Conclusions

I was ranked 20 on the public leaderboards but dropped 20 ranks in the private leaderboards. I realize I should have trusted my local cross validation scores over the public leaderboard scores now. I also realized that it would have been better to set up a smoother pipeline that would facilitate more rapid preprocessing-testing cycles. These are lessons I will carry going forward.