

Machine Learning Engineer Nanodegree

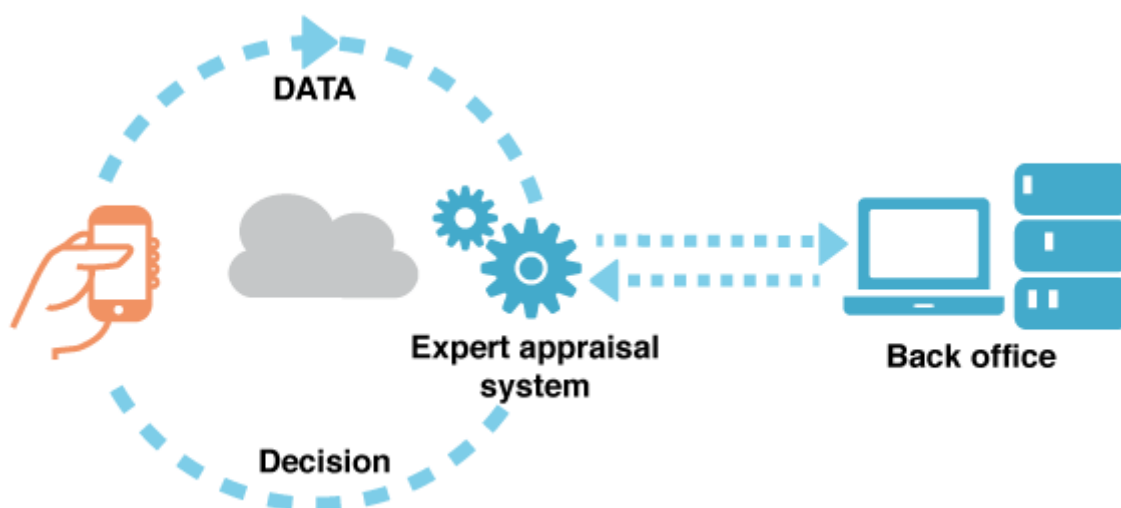
Alexandre Rousseau

February 23, 2017

I. Definition

Project Overview

The Kaggle competition I pick for my capstone project is the BNP Paribas Cardif Claims Management. The duration of this competition was from February 3rd to April 18th 2016. This competition is supported by an insurance company called BNP Paribas Cardif. The company receives many claims from its customers every day, and the claims need different levels of check. So that the company wants to build a system which can automatize the process of determining to which level a claim belongs.



<https://www.kaggle.com/c/bnp-paribas-cardif-claims-management>

I choose this for a project because it was my first contact with Kaggle and my struggles were my motivators to start the MLND. I want to see how much I have improved.

Problem Statement

The data in this challenge was health care data related to claims. It is an anonymized dataset with two categories of claims is provided by BNP Paribas Cardif. The objective was to use the 133 predictors to determine if a claim needed further work or not. The set was much larger than anything I had ever used with over 160,000 observations. The predictors were unlabeled. Additionally, there were thousands of missing values in the data set as well. The output is the probability for a sample to be in the first category. The training set is consisted of 114322 samples, and the test set has 114394 samples.

To get experience, my plan is to use ensemble techniques to try a top 10%. Ensemble methods seems mandatory to work with in Kaggle competitions nowadays and I need to understand the concepts. I will do a quick data exploration and try to create at least 2 distinct datasets. Then each of these dataset will be use with a series of few different models with Kfold. Predictions obtained will be gathered and use in a final model to create the final submission file. I will use the test set given by Kaggle to benchmark my models with the logloss metric.

Metrics

The evaluation metric for this competition is **Log Loss**

$$\text{logloss} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

where N is the number of observations, log is the natural logarithm, y_i is the binary target, and p_i is the predicted probability that y_i equals 1.

Note: the actual submitted predicted probabilities are replaced with $\max(\min(p, 1-10^{-15}), 10^{-15})$.

The submission file should contain two columns: ID and PredictedProb. The file should contain a header and have the following format:

ID, PredictedProb

0,0.5

1,0.5

2,0.5

7,0.5

etc.

This error metric suits well our problem because it is used when we have to predict a binary class with a probability (likelihood) ranging from definitely true (1) to equally true (0.5) to definitely false(0). The use of log on the error provides extreme punishments for being both confident and wrong.

Source: <https://www.kaggle.com/wiki/LogarithmicLoss>

II. Analysis

Data exploration

Our dataset look likes that :

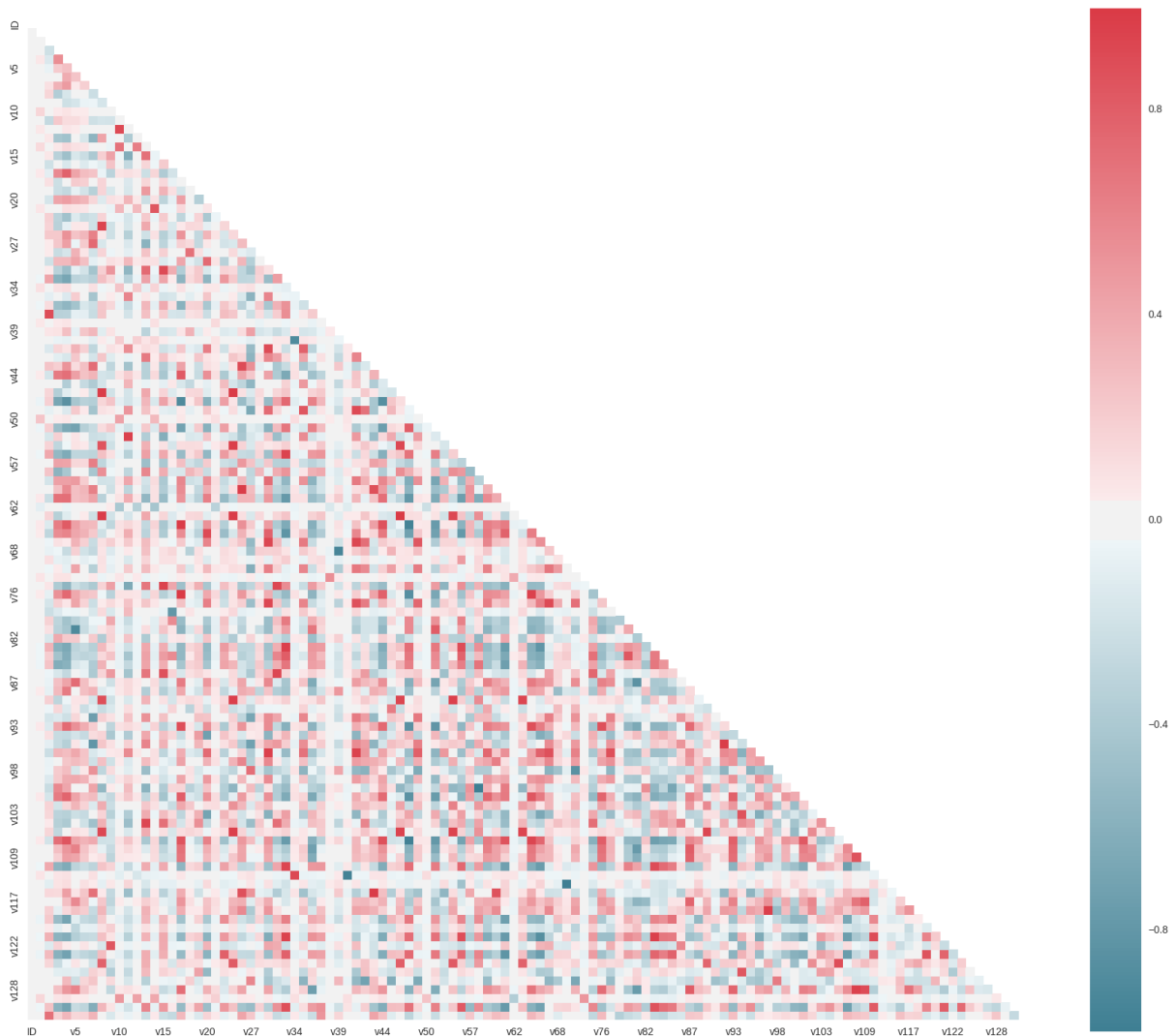
	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127
0	3	1	1.335739	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	AU	1.804126	3.1137
1	4	1	NaN	NaN	C	NaN	9.191265	NaN	NaN	2.301630	...	NaN	NaN	0.598896	AF	NaN	NaN
2	5	1	0.943877	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	AE	1.773709	3.9221
3	6	1	0.797415	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	CJ	1.415230	2.9543
4	8	1	NaN	NaN	C	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Z	NaN	NaN

These are the basic description of the training dataset:

- Total: 114,393 observations (rows) x 133 features (columns)
- Data Types: float, integer, string
- Column "ID": the ID of each row, not being used as predictor
- Column "target": the target of each row, not being used as predictor
- Numbers of Text-based Predictors: 112 Numbers of Numeric
- Predictors: 19
- Numbers of Columns with missing value: 119

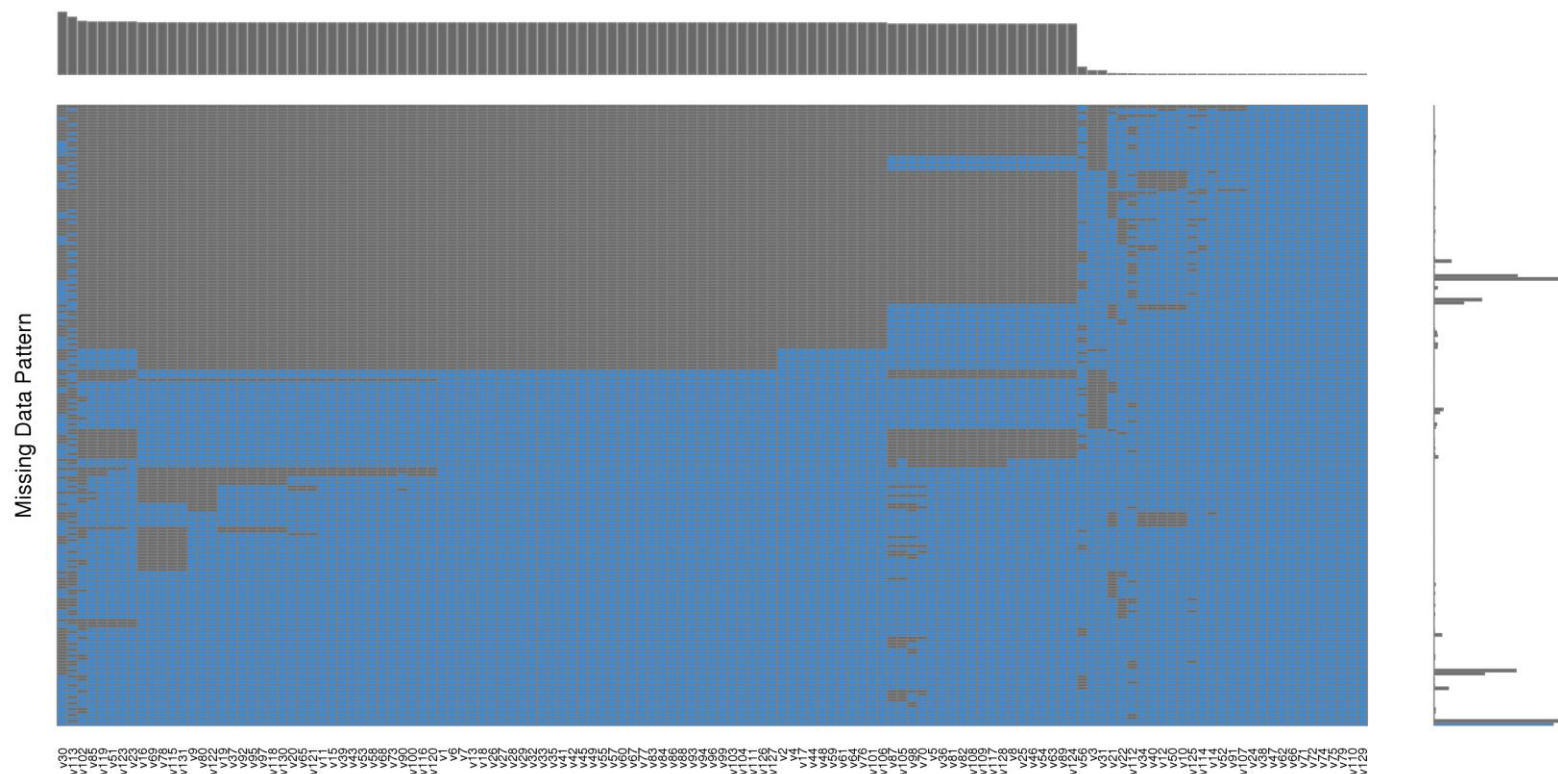
Exploratory Visualization

As shown in the correlation matrix plot, some variables are highly correlated (not just 1-to-many but also many-to-many). Correlation values are between -1 and 1. Red color means positive correlation while blue color means negative correlation.



Correlation Matrix Plot

We can notice that there are a lot of missing values. 96565 rows contain at least one. Here is a map showing the missingness in the dataset:



We can see that approximately 40% of data is missing.

Source : <https://www.kaggle.com/jpmiller/bnp-paribas-cardif-claims-management/visualizing-the-nas/code>

Algorithms and Techniques

After testing and fine-tuning all this set of models, we decide to move forward with the common Kaggle technique of models ensembling (see this very good [article](#) for more info).

My goal is to use the common Kaggle technique of models ensembling (see mlwave). After testing and tuning a set of models, I will the method called “blending”. I choose this method it seems to be the more elegant to me and I want to be able to use in different context.

Blending has been introduced during the Netflix prize, and has since then spread out to many of the winning solutions in Kaggle competitions. The concept is as follow:

- * Split the training dataset in k folds
- * For each of the k folds, use the (k-1) other folds to output probabilities on the k-th fold. Stack these to output probabilities on the whole training dataset without predicting the training data.

- * Iterate over all models and concatenate the probabilities provided by each model together. In doing so, we recreate a set of p features, where p is this time the number of models we trained and cross-validated.
- * Use these new features to train a new model.

Source: <http://mlwave.com/kaggle-ensembling-guide/>

I went through these steps and built a logistic regression, an XGBOOST classifier and a Gradient Boosting classifier on top of our 8 previous models. Then I choose the one giving the best results in regards of the loss function.

All the well-known algorithms I used can be described as follow:

Random forest classifier

Random forests are an ensemble learning method that use random subsets of observations and features from the training data to create multiple decision trees. Then, those separate decision trees fit the data separately. The classifier chooses the classification by voting the better class.

Pros:

- Prevent from overfitting (by random subset of trees)
- Good with very large data set (also due to random subset of trees)
- No transformation needed
- Robust against outliers
- Can work in parallel

Cons:

- Slow due to construction of multiple classifier for subset trees.
- Difficult to interpret
- Biased in multiclass problems toward more frequent classes

eXtreme Gradient Boosting (XGBoost)

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. It has definitely boosting capabilities and additional components overcoming the weakness of GBM.

Pros:

- Regularization: it reduces overfitting (
- Parallel processing: XGBoost is also said a parallelized gradient boosting model. This makes the model faster than GBM (sequential)

- XGBoost do not required feature engineering (handle missing values, scaling and normalization)
- Can be used for classification, ranking or regression
- Built-in Cross Validation

Cons:

- Only work with numeric features
- Hyperparameters must be tuned properly or it will lead to overfitting

For XGB, we were tuning the following parameters.

- `learning_rate`: learning rate (not going to be tune at the beginning)
- `max_depth`: depth of the trees
- `n_estimators`: Number of boosted trees to fit.

Extra Tree Classifier

Similarly, to Random Forest, a Extra Tree Classifier fits a number of randomized decision trees on random subsets of observations and features. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the `max_features` randomly selected features and the best split among those is chosen. When `max_features` is set 1, this amounts to building a totally random decision tree.

Pros:

- The same as Random Forest
- More random thus lower variance

Cons:

- The same as Random Forest
- Computation can grow much bigger

Logistic Regression

Use regression to solve binary classification problems by fitting a sigmoid function to calculate the probabilities

Pros:

- Intrinsically simple and fast (runs in constant time)
- Easy to interpret

- Less prone to overfitting
- Low variance

Cons:

- Unable to model complex relationships (non linear ?)
- Doesn't perform well with large number of features

My understanding of pros and cons is largely due to <http://www.kdnuggets.com/2015/07/good-data-science-machine-learning-cheat-sheets.html>

Benchmark

To evaluate the performance of my solution I will use the public leaderboard of the competition. Kaggle rankings are decided by the model's performance on a test data set. Only Kaggle has the answers for this data set. The Public score is what you receive back upon each submission, calculated using the evaluation metric described earlier. BUT: The Public Score is being determined from only a fraction of the test data set -- 33% in our case. The other 67% of the test set is used for the private (and final) leaderboard.

I will use this submission process to score on the private leaderboard.

III. Methodology

Data Preprocessing

I decided to create multiple datasets from the one given. Each of these datasets is the product of several transformations. These transformations are:

- Label encoding for categorical variables
- One hot encoding
- Transform NaNs to 999
- Transform NaNs to the mean of each column
- Only keep continuous variables and label encoding on categorical variables
- ...

I tried several techniques. Considering my computer I only kept 4 datasets with these combinations of transformation: label encoding + NaN means, label encoding + NaN at 999, only categorical + NaN means and only categorical variables with label encoding.

Label encoding is a technique consisting of encoding labels with value between 0 and $n_classes-1$. `LabelEncoder` can be used to normalize labels. It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

One hot encoding on the other hand consists of encoding categorical integer features using a one-hot aka one-of-K scheme. The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature. It is assumed that input features take on values in the range $[0, n_values)$.

Source : <http://scikit-learn.org>

Transforming NaNs is a common technique to use the maximum of data available. To give a “high score” like 999 is a trick to labelize missing values. Choosing the mean is just a way to extrapolate and ignore missing values in our datasets.

Name ▾	Type	Size	Value
D1	DataFrame	(228714, 134)	Column names: ID, target, v1, v10, v100, v101, v102, v103, v104, v...
D2	DataFrame	(228714, 134)	Column names: ID, target, v1, v10, v100, v101, v102, v103, v104, v...
D5	DataFrame	(228714, 116)	Column names: ID, target, v1, v10, v100, v101, v102, v103, v104, v...
D6	DataFrame	(228714, 20)	Column names: v110, v112, v113, v125, v22, v24, v3, v30, v31, v47,...
cat_var	list	18	['v110', 'v112', 'v113', 'v125', 'v22', 'v24', 'v3', 'v30', 'v31', ...]
cont_var	list	114	['ID', 'target', 'v1', 'v10', 'v100', 'v101', 'v102', 'v103', 'v10...
df_all	DataFrame	(228714, 134)	Column names: ID, target, v1, v10, v100, v101, v102, v103, v104, v...
target	Series	(114321,)	0 1 1 1
test	DataFrame	(114393, 133)	Column names: ID, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v1...
train	DataFrame	(114321, 133)	Column names: ID, target, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10,...

Summary of datasets created

Implementation

After creating the datasets, my next step is to train and test several models, and save the prediction in a proper format (the same as the submission form given by Kaggle). These predictions will be used later.

To do I create a script called `models.py`. I define 2 functions, one to load the datasets created before and prepare them for the models with basic splitting. For instance the train set is splitted into a train set and a test set. I can now load each datasets. I use the `StandardScaler` (thanks to `sklearn.preprocessing`) on the whole train set. Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

The purpose of the second written function is to create a list of pair dataset / classifier to train on. With this list I can now, with a loop, select each model of the list, train with `StratifiedKfold`, test it, evaluate it and save the results in a CSV file. The log loss function is coming from the `sklearn.metrics`.

Same methodology is for the blending method. I load blend data, outputs of the first script and follow the same step to finally get my submission file.

Refinement

First improvements I made were on the datasets. You can see on my code that I originally created 7 datasets. Unfortunately One Hot Encoder transform was too costly for my computer. Then I moved to the algorithms. I tuned a lot the parameters of the first list of models. This tuning was done empirically. I mainly choose to adjust the number of estimators and the maximum depth to find the ideal compromise between execution time and results.

The trick is that in a real context (competition), each result, even the mediocre, is saved and will be used later for the blending. So every trial will be rewarded in a way, and that is probably the major advantage of ensembling methods.

Regarding of the blending algorithms, I choose to use a simple logistic regression at first because it is the most recommended in the literature. In addition, often simpler is the better. For the sake of experience, I also tried a gradient boosting and a XGBoost classifier.

Here are my results on the kaggle leaderboard:

Logistic regression => 0.45865.

Gradient Boosting Classifier => 0.45851

XGB Classifier => 0.45710

For the record, without ensembling method, my best model (extra tree) scores 0.45877.

IV. Results

Model Evaluation and Validation

I finally performed predictions. If we look at the test score for each models for the round we have:

```
dataBLEND_ExtraTreesClassifier_tr-val_0.0806-0.4578
dataBLEND_ExtraTreesClassifier_tr-val_0.0806-0.4578
dataBLEND_RandomForestClassifier_tr-val_0.1625-0.4645
dataBLEND_RandomForestClassifier_tr-val_0.1714-0.4648
dataBLEND_XGBClassifier_tr-val_0.2299-0.4636
dataBLEND_XGBClassifier_tr-val_0.2299-0.5381
dataBLEND_XGBClassifier_tr-val_0.2299-0.5425
```

Therefore, the Extra Tree Classifier is surprisingly our best model here. I was expecting that the XGBC to score better. I think it is probably due to my data preparation. My method was a bit brutal and with a proper matrix, score should be higher.

Anyway, with these probabilities, I ready to launch my blender. I tried 3 different algorithms for blending: simple logistic regression, a XGBC and a GBM. Here are the results:

```
dataBLENDED_LogisticRegression_tr-val_0.4580-0.4581
dataBLENDED_XGBClassifier_tr-val_0.3010-0.4589
dataBLENDED_GradientBoostingClassifier_tr-val_0.4532-0.4574
```

So, the blending was successful only with the GBM, we went from 0.4578 to 0.4574. It is not a lot but it is enough to win several ranks in a Kaggle competition. Plus, more importantly, it validates the method.

Justification

So I compare my score to the public leaderboard, my rank is 1187/2926. Now if I use the submission process on Kaggle, results slightly change. As seen earlier, XGB gives us a score of 0.45710 on the private leaderboard, which means a ranking of $\sim 1330/2926$. The difference is of course due to test set being twice as large. Without blending, my score is 0.45877, and that is a ranking of 1560. Therefore, I am beating 230 people with my method. I think it is a proper reward and with just a few more not-too-much correlated models, the gap should be a lot higher.

Anyway, my results are good enough because I consider my code to be a «starter kit» for future competitions. Of course, my score is certainly beaten by a single model with proper features engineering. Next time I will use this starter code and focus directly on variables.

V. Conclusion

Free-Form Visualization

To summarize some of my results:

Classifiers	Training Score	Test Score
Random Forest	0.1625	0.4625
ExtraTrees	0.0999	0.4778
XGB	0.2299	0.4636
Logistic Regression as Blender	0.4580	0.4681
Gradient Boosting as Blender	0.4532	0.4574
XGB as Blender	0.3010	0.4589

Table with different results

Many models with different parameters have been used, and without surprise with a kaggle competition, ensembling method coupled with multiple XGBoost give the best results.

Looking at this table, I ask myself why I did not used another blender to blend the blended probabilities? Maybe it is time to move on. :)

Reflection

The BNP Paribas Cardif Claims Management competition is challenging due to its anonymous variables and high amount of missing data. To answer this challenge I used different method of data cleaning and imputation. Different machine learning classifiers such as Logistics Regression, ExtraTrees, XGBoost, Random Forest and GBM were used for training and predictions. Finally I used a model ensembling method called blending, which slightly improve the score from one of combinations.

More importantly I wrote the code as a starter code for my next competitions. I also have identified what could be done to improve my result on this specific competition.

Many of the Kagglers in the competition have used the similar methods that I did. The best kaggler used ensembling method with many models and really advanced features engineering. My future work in an incoming competition will certainly be related to look deeper on the data and do more feature engineering.

All-in-all, a lot of efforts for a little gain, which surely makes sense in the context of Kaggle competitions, and probably less in real time contexts. I am still really happy with the experiment.

Improvement

What can have be done to score higher?

By taking a more classical approach, I could have done a lot better at transforming variables. For instance I have checked the correlation factor between variables but I have not removes any. For instance I can remove a variable of a pair when the factor is superior is close to 1.

In addition, many tree-based methods provides feature importance, which can be used to discard the irrelevant features. I used ExtraTrees classifier and I could have coupled it with «SelectFromModel» in the scikit-learn package "feature_selection". SelectFromModel can be used along with any model that has a coef_ or feature_importances_ attribute after fitting. It also allows us to set the threshold for feature selection. Features whose importance is greater or equal are kept while the others are discarded. A similar method

can be used with XGBoost estimate the «feature importance». These techniques can be considered easy to set up.

See: <https://www.kaggle.com/mmueller/liberty-mutual-group-property-inspection-prediction/xgb-feature-importance-python>

I could have also use all the work shared on the forum by the kaggler to improve drastically my score. For instance this post seems to had a large impact on the leaderboard:

<https://www.kaggle.com/trottefox/bnp-paribas-cardif-claims-management/nearest-neighbour-linear-features/comments>

Moreover, the blending method, to be truly effective, needs to be used with a lot more models. Ideally I could have created a few more datasets and adapt it to a lot more models. Competition winners claim to use thousands of different models.

Lastly, I could have done some grid-search to select some of the parameters. I did not because it was too costly in time and resource for me, but I used this technique in another project.

