# Solution for Santander Customer Satisfaction competition, 3rd place

Lucas Silva[1], Gilberto Titericz[2], Dmitry Efimov[3], Ikki Tanaka[4], Darius Barušauskas[5], Marios Michailidis[6], Mathias Müller[7], Davut Polat[8], Stanislav Semenov[9], and Dmitry Altukhov[10]

[1]lucas.eustaquio@gmail.com
[2]titericz@yahoo.com
[3]diefimov@gmail.com
[4]ikki0407@gmail.com
[5]darius.barusauskas@gmail.com
[6]kazanovassoftware@gmail.com
[7]muellermat84@gmail.com
[8]davutpolat87@gmail.com
[9]stasg7@gmail.com
[10]alt.dmitry@gmail.com

## 1 Approach summary

The goal of this competition was to predict if customers are satisfied or dissatisfied with their banking experience. Our solution mostly based on the ensembling of different models. The document is constructed in the following way: first sections describe the models of each team member, the last section describes the final ensemlbing and the way to obtain the predictions.

# 2 Dmitry Efimov:

## 2.1 Preprocessing steps

We have applied several manual preprocessing steps such as replacing some values by NA, dropping sparse and duplicated features, normalization. the methods process_base, drop_sparse, drop_duplicated, normalize_features in the file `dmitry/santander_preprocess.py` contains all preprocessing steps.

## 2.2 Feature engineering

The following types of features have been generated before the models trained:

- **Sum of zeros feature**: how many zeros for each training and test example.
- **t-SNE features**: t-Distributed Stochastic Neighbor Embedding [6] is the dimensionality reduction technique that maps the feature space to lower dimensional space such that distribution of distances between training examples stays the same.
- **PCA features**: Principal Component Analysis is another dimensionality reduction procedure that maps the feature space on the lower subspace by minimizing the distances to that subspace.
- **K-means features**: we applied k-means algorithm to cluster the data and took number of cluster assigned to each training example as a feature (we used from 2 up to 10 clusters).
- **Likelihood features**: likelihood features are calculated using out-of-fold predictions with the formula:

  $$LL = \frac{30 \cdot \bar{y} + \sum\limits_{i \in G} y_i}{30 + |G|},$$

  where $G$ is the set of indices for the training examples with constant value of some chosen raw feature (for example, saldo_var13), $|G|$ is a size of group $G$, $\bar{y}$ is an average of output for all training examples.

## 2.3 Models

- **FTRL2**

  The idea of Follow The (Proximally) Regularized Leader (FTRL-Proximal) model is described in the [5, 4].

  Assuming that our purpose is to minimize the loss function

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \tag{1}$$

  where $\hat{y}_i$ is a prediction for the instance $i$, we transform the design matrix $X$ to the sparse binary design matrix such that one column corresponds to one value of each feature (the hashing trick ([3, Chapter 3] is applied to transform feature values to indices).

  The prediction for sample $i$ is constructed as $\hat{y}_i = \sigma\left(w_i \cdot x_i\right)$, where $w_i$ is a weight vector of size $n$ on $i$-th iteration.

  The algorithm updates the weights of the sample $i+1$ based on the previous samples $\{1, \ldots, i\}$ by finding

$$w_{i+1} = \arg\min_{w}\left(\sum_{r=1}^{i} g_r \cdot w + \frac{1}{2}\sum_{r=1}^{i}\tau_r||w - w_r||_2^2 + \lambda_1||w||_1\right) =$$
$$= \arg\min_{w}\left(w \cdot \sum_{r=1}^{i}(g_r - \tau_r w_r) + \frac{1}{2}||w||_2^2\sum_{r=1}^{i}\tau_r + \lambda_1||w||_1 + \text{const}\right)$$

  and

$$\sum_{r=1}^{i}\tau_{rj} = \frac{\beta + \sqrt{\sum_{r=1}^{i}\left(g_{rj}\right)^2}}{\alpha} + \lambda_2,\ j \in \{1, \ldots, N\},$$

  where $\lambda_1$, $\lambda_2$ are regularization parameters, $\alpha$, $\beta$ are parameters of the learning rates schedule, $\tau_r = (\tau_{r1}, \ldots, \tau_{rN})$ is a vector of learning rates for the step $r$, $g_r = \left(\frac{\partial L}{\partial w_{r1}}, \ldots, \frac{\partial L}{\partial w_{rN}}\right)$ is a gradient vector of logarithmic loss $L$ in the form (1) for step $r$.

  In this model we used raw, SumZeros and Likelihood features.

- **RGF3**

  Regularized Greedy Forest algorithm is described in [7]. In this model we used raw, SumZeros, PCA and Likelihood features.

- **RGF5**

  In this model we used raw, SumZeros, tSNE and Likelihood features.

- **RGF6**

  In this model we used raw, SumZeros, K-means and Likelihood features.

- **AdaboostClassifier**

  We have used implemented in scikit-learn [2] class AdaboostClassifier with raw, SumZeros, PCA and Likelihood features. The description of the Adaboost model can be found in [8].

- **XGBOOST**

  Xgboost [1] is one of the most effective implementation of the famous trees gradient boosting algorithm. We used raw, SumZeros, PCA and Likelihood features. We applied 5 iteration bagging procedure with different seed to smooth the noise in the predictions.

All parameters for the models have been found using cross validation procedure and can be found in the file `dmitry/santander_models.py`.

## 3 Ikki Tanaka:

### 3.1 Preprocessing and feature engineering

Following data cleaning and feature engineering tricks were used:

1. Calculating zero-count per ID.
2. Calculating count of integer variables.
3. Creating dummy variables of var3 in the threshold (var3 $\geqslant$ 4 or 5).
4. Creating boolean variable if var38 is peak value (117310.979016494).
5. Calculating log of var38 and replace log(117310.979016494) with zero.
6. Removing constant variables.
7. Removing identical variables.

8. For neural nets, removing some one-hot-encoded categorical variables.
9. For neural nets, limiting variables in test based on min/max of train.
10. For neural nets, applying log transformation to all variables.
11. One-hot encoding of specific variables (some variables are duplicated to the variables created in the item 3, in this case they are removed in the model training).

This procedure can be found in `ikki/ikki_feat_verXX`.

### 3.2 Models

I used XGBOOST [1] and neural networks. Each model was trained 20 times with 20 fold sets and averaged using rank transformation. Slightly different parameters have been used in different models. These parameters have been chosen based on cross validation procedure.

In the neural nets the number of layers were 3 or 4, BatchNormalization has been utilized, LeakyReLU and Parameterized ReLU were used as activation functions. Stochastic Gradient Decent (SGD) was the basic optimization algorithm for all models. The data have been also standartized by substracting the mean value of each feature and dividing by it standard deviation.

The XGBOOST models can be found in `ikki/scripts/ikki_XGB_X.py` and the neural net model can be found in `ikki/scripts/ikki_NN_1.py`.

My own class of models for stacking has been used, it can be found in the file `ikki/base_fixed_fold.py`. This script helps to implement stacking easily as follows:

1. Specify the parameters and model to variable "m = ModelV1(∼)"
2. Run "m.run()"

### 3.3 Important remarks

Finally, I decided to use simple feature engineerings as shown above because I would like to trust in local CV and public LB at almost the same weights, although plan of our team was to trust in CV. In fact, the weights of CV are higher than the ones of public LB.

# 4 Darius Barušauskas:

## 4.1 Preprocessing and models

Following data cleaning and feature engineering tricks were used:

1. Removing almost all indicator variables (ind_*); they cover same info provided by num_* variables.
2. Removing constant variables.
3. Removing variables which non-0 attributes have 0,1,2,3 TARGET=1 instances.
4. Setting var38==117310.979016494 ⇒ var38=NA (missing at random); assumption that Santander mean imputed NA's.
5. Summarizing how many 0's, 3's, 6's, 9's, $\{X \mod 3 == 0\}$'s appear in each row.
6. Calculating var38 percentile rank within same var15 for each instance.
7. Calculating var38 percentile rank within +/-1 var15 for each instance.
8. Calculating var38 percentile rank within +/-2 var15 for each instance.
9. Calculating ratios of ult1/ult3 variables.
10. Calculating ratios of hace2/hace3 variables.
11. Calculating if $X \mod 3 == 0$ (for money type variables).
12. Special feature seperating 2 distint population segments within data (file `data/features/population_split.csv`).

The XGBoost, lasso and SVM models have been trained on the feature set described above. Features in lasso and SVM models are transformed using log transformations. XGBoost models are identical, except feature from the item 12 was removed in xgb2 version.

## 4.2 Important remarks

The most interesting feature from the item 12 has brought big discussions within our team about its predictive performance. It showed high CV AUC gains for our models, however, public LB did not give us same feedback. But in the end, this single feature provided highest AUC gain in private LB among our ensemble's models.

# 5 Marios Michailidis:

## 5.1 Preprocessing

This step included the removal of 63 variables that were either completely non-significant because they were taking only 1 value or very highly correlated with other variables, hence there were not yielding new information. One extra feature was created that was capturing the number of zeros per row. Additionally, var38 has its 117310.979016494 value was deemed as null and further replaced with "-1" in order to separate it from the rest of the values. Moreover, different preprocessing steps were applied in various model including:

1. Standard scaling for linear and NN models.
2. Counts of each categorical variables.
3. Likelihood of categorical variables.
4. t-SNE, PCA and KNN features.
5. Different combinations of the above.

## 5.2 Models

Built 15 different models with different input data (as specified in the Preprocessing section) with different algorithms that implemented bagging and different hyper parameters optimized (manually) via 10 times 5-fold cv. These model included:

- 6 xgboost

- 3 neural networks (package keras in Python)
- 2 Random Forests
- 2 Ada-boosted Trees
- 1 ExtraTreesClassifier
- 1 KNN

### 5.3 Remarks

This competition was a lot about avoiding overfitting. To counter that I developed many different models with high number of bags, employing multiple CV schemas to make my ensemble as generalizable as possible, minimize variance while being conservative in many aspects of the modelling process (like in feature engineering).

## 6 Mathias Müller:

### 6.1 Preprocessing

Only basic preprocessing has been applied:

1. Removing duplicated columns
2. Removing columns with zero variance
3. Adding a feature with count of zeros per row

Based on this, two datasets called *mm1* and *mm2* were created with different amounts of finally selected features. Feature selection for *mm1* based on repeated (different seeded) cross validated noise injection with XGBoost as wrapper to rank the features. Additionally, holdout sets were used to avoid overfitting due to too greedy feature subset selection regarding CV-AUC improvements. This dataset consists of around 150 features. Dataset *mm2* consists of around 300 features, where only a few features based on the findings of other team members have been removed.

## 6.2 Models

Two XGBoost models have been trained. The first one on dataset *mm1* and the second one on dataset *mm2*. Both models are 15 times bagged per fold with random undersampling of the majority class (ratio: 2:1).

# 7 Gilberto Titericz:

## 7.1 Ensembling

The ensembling of all models has been done with the R method `optim` with the direct optimization of AUC metric. To get the same scale for all predictions we have run the optimization on the ranked predictions (the R method `rank`). The parameters of the optimization can be found in the file `final_ensemble.R` in the root directory.

# 8 How to generate the predictions

Important: all predictions should be generated in the order they appear in this document.

## 8.1 Dmitry Efimov:

| **Task** | **File name** |
|---|---|
| **PCA features** | `pca_features.py` |
| **K-means features** | `kmeans_features.py` |
| **tSNE features** | `tsne_features.py` |
| **Main file** | `main_20sets.py` |
| **Train models** | `santander_models.py` |
| **Helper functions** | `santander_preprocess.py` |
| **RGF wrapper** | `rgf.py` |
| **Train FTRL model** | `ftrl.py` |

1. Upload all input files to the folder `data/input`
2. Open command line and change path to `dmitry` folder
3. Run the script `run.sh`

4. The output files will be saved in the folder `data/output`

**Dependencies:**
The model can be run on Linux Ubuntu 12.04 or Mac OS.
**Python:** pandas, numpy, sklearn [2], xgboost, ml_metrics, scipy, tsne, os, sys, getopt, re, glob, tempfile
**Other:** pypy, rgf 1.2
The Python version used 2.7.3.

## 8.2 Ikki Tanaka:

| **Task** | **File name** |
|---|---|
| **Feature engineering for XGBOOST** | `ikki_feat_verXXX.py` |
| **Feature engineering for Neural nets** | `ikki_feat_ver3.py` |
| **Train XGBOOST** | `scripts/ikki_xgb_XXX.py` |
| **Train Neural nets** | `scripts/ikki_NN_1.py` |
| **Combine prediction** | `combine_pred.py` |
| **Model wrapper class** | `base_fixed_fold.py` |

1. Upload all input files to the folder `data/input`

2. Open command line and change path to `ikki`

3. Run the script `run_ikki.sh`

The final output files will be saved in the folder `data/output/train` and `data/output/test`. The created features will be saved in `ikki/data/output/features`. The temporary files in of each models will be saved in `ikki/data/output/temp`. It will generate files of individual models under `/ikki/data/output/temp/` ( 2(train,test) * 20(fold sets) * 3(models) = 120 files), including 2 final files under `/data/output/train/` and `/data/output/test/` with predictions of 3 models - 2 xgboosts and 1 neural net model.
**Dependencies:**
The model can be run on Linux Ubuntu 14.04 or Mac OS.
**Python:** os, sys, pandas, numpy, xgboost, keras, sklearn

## 8.3 Darius Barušauskas:

| Task | File name |
|---|---|
| **Main script** | `build.R` |
| **Train xgb1** | `scripts/xgb_raddar_1.R` |
| **Train xgb2** | `scripts/xgb_raddar_2.R` |
| **Train lasso** | `scripts/lasso_radar.R` |
| **Train svm** | `scripts/svm_radar.R` |

1. Upload all input files to the folder `data/input`
2. Open R and change working directory to `darius`
3. Run `darius/build.R` file

It will generate several files, including 2 final files under `/data/output/train/` and `/data/output/test/` with predictions of 4 models - 2 xgboosts, 1 svm and 1 lasso model.

**Dependencies:**

Models were trained using Ubuntu 12.04 with 32 thread cpu.

**R:** doSNOW, foreach, dplyr, xgboost, glmnet, LiblineaR, Metrics

## 8.4 Marios Michailidis:

| Task | File name |
|---|---|
| **Blending script** | `blender.py` |
| **Train KNN model** | `knn_marios_1.py` |
| **Train Neural Net models** | `nn_marios_XXX.py` |
| **Train Random Forest models** | `rf_marios_XXX.py` |
| **Train XGBOOST models** | `xgboost_marios_XXX.py` |
| **Train Adaboost models** | `ada_marios_XXX.py` |
| **Train Extra Trees model** | `et_marios_1_2.py` |

1. Upload all input files to the folder `data/input`
2. Open command line and change path to `marios`
3. Run the script `run.sh`

**Dependencies:**

The model can be run on Linux Ubuntu 12.04 or Mac OS.

**Python:** pandas, numpy, sklearn [2], collections, keras, XGBoostClassifier

### 8.5 Mathias Müller:

| Task | File name |
|---|---|
| **Create dataset *mm1*** | `create_dataset_mm1.py` |
| **Create dataset *mm2*** | `create_dataset_mm2.py` |
| **Train XGB model 1** | `faron_xgb_01.py` |
| **Train XGB model 2** | `faron_xgb_02.py` |
| **Helper functions** | `san_utils.py, kaggletils.py, combine_oof_sets.py` |
| **Solution runner script** | `_runme.py` |

1. Upload all input files to the folder `data/input`
2. Open command line and change path to `mathias` folder
3. Run the python script `_runme.py`
4. The output files will be saved in the folder `data/output`

**Dependencies:**
**Python 2.7:** pandas, numpy, scipy, sklearn [2], xgboost [1], ntpath

### 8.6 Gilberto Titericz:

| Task | File name |
|---|---|
| **Final ensembling** | `final_ensemble.r` |

1. Run models of all team members as described above
2. Open R and change working directory to the root directory
3. Run `final_ensemble.r` file

It will generate one csv file with predictions in the folder `/data/submission/`.
**Dependencies:**
**R:** data.table

## References

[1] https://github.com/dmlc/xgboost

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay *Scikit-learn: Machine Learning in Python.* 1991, Journal of Machine Learning Research, 12, pp. 2825-2830.

[3] A. Rajaraman and J. D. Ullman *Mining of massive datasets.* 2011, Cambridge University Press.

[4] H.B. McMahan, G. Hold, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos and J. Kubica *Ad Click Prediction: a View from the Trenches.* August, 2013, KDD, Chicago, Illinois, USA.

[5] H. B. McMahan *Follow-the-Regularized-Leader and mirror descent: Equivalence theorems and L1 regularization.* 2011, 14th International Conference on Artificial Intelligence and Statistics (AISTATS), 15.

[6] L.J.P. van der Maaten and G.E. Hinton *Visualizing High-Dimensional Data Using t-SNE.* 2008, Journal of Machine Learning Research, 9(Nov), pp. 2579-2605, 2008.

[7] R. Johnson and T. Zhang *Learning nonlinear functions using regularized greedy forest.* 2011, Technical report, Tech Report: arXiv:1109.0887.

[8] Y. Freund and R. Schapire *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting.* 1995.