Neural Networks for Santander Customer Satisfaction

Nick Steele, Sam Schwager, Scott Morris
nsteele@stanford.edu, sams95@stanford.edu, swmorris@stanford.edu

Abstract

Large financial institutions such as banks must have a good understanding of customer satisfaction in order to maintain loyal customers. In 2016, Santander bank created a Kaggle competition to identify satisfied and dissatisfied customers (https://www.kaggle.com/c/santander-customer-satisfaction). To address this problem, we present a multi-hidden-layer, fully-connected neural network that classifies a customer as satisfied or unsatisfied based on the anonymized features given in the Kaggle competition dataset. Our model classifies customers with .7940 AUC (area under the receiver operating curve), the metric for the competition, on the Kaggle test dataset.

1. Introduction

Customer satisfaction is a key measure of success in the banking industry. Unhappy customers don't stay long and are difficult to detect since they often don't voice dissatisfaction before leaving. Two years ago, Santander Bank launched a Kaggle competition to help with the problem of identifying dissatisfied customers, with a $60,000 prize pool. We use deep learning to create a competitive algorithm for the Kaggle competition. The input to our algorithm is anonymized customer information provided in the Kaggle dataset. We then use a neural network to output a 1 or 0 for whether the customer is unsatisfied or satisfied. Finally, to make predictions we average over the outputs of an ensemble of neural network models with randomly initialized weights.

2. Related Work

Since we address a problem posed by a Kaggle competition, the relevant related work comes from other submissions to the competition, and papers written about the competition after the fact. The competition uses an AUC evaluation function, which compares the predicted confidence and actual value of each example and scores according to the area under the curve. The most common approach is XGBoost, which is explained in the next paragraph [1]. For reference, the top leaderboard score was .829072. Here we outline several past approaches.

A. van den Elsen [2]

AUC Score: 821777. van den Elsen utilized an XGBoost model, which is a method formed by a combination of multiple trees, which are built iteratively. This contrasts our neural network approach. In XGBoost, when a new tree is built, the model assigns higher weights to the sections where previous trees made mistakes. This model led to a relatively high AUC score, but the score is not near a winning score on the leaderboard. One drawback of this approach is the

lack of a model ensemble, which creates a voting system with multiple models, leading to a higher AUC score.

B. Silva et al (2016) [3]

AUC score: .828530. This team placed 3rd in the competition, and as such their work represents a successful, state-of-the-art approach to the problem at hand. The team (consisting of 10 members) did data pre-processing, feature engineering, and model creation independently. They then combine their model's predictions into a model ensemble using the R optim package. Our team also utilizes a model ensemble. Silva el al.'s model was strong in that it used a wide variety of pre-processing techniques (including getting rid of duplicated features, normalization, and log transforming features), feature engineering methods (including principal component analysis, K-means, and t-Distributed Stochastic Neighbor embedding), and model choices (including Regularized Greedy Forest, Adaboost, XGboost, and Neural Networks). Alternatively, our model ensemble utilizes only neural networks with different numbers and depths of layers.

C. Wang (2016) [4]

AUC Score: .8249. Wang's approach was notable in that he did very little data preprocessing and utilized only decision trees. He selected features based on their importance in a Gradient Boosting Classifier. He also trained parameters using a coarse to fine approach, one at a time. Similarly to Silva et al., Wang ensemble multiple models, including Adaboost, Random Forest, and XGBoost. Unlike Silva et al., he did not include Neural Networks in the ensemble.

D. Yooyen and Ma (2016) [5]

This team removed duplicate observations in the train set, chose features using Pearson's Correlation Coefficient with the target and crossvalidation, and added additional heuristic rules (for example, if var15 <23, then the target is 0). These techniques led to a relatively high AUC score using decision tree models, but did not include a model ensemble, which may have improved results.

3. Dataset and Features

We obtained the dataset from the Kaggle competition. [6]

Data Analysis

The dataset has a total of 68418 training examples. Each example has 370 anonymized features, all numerical. Of the 68418 examples, only 2735 or 4.0% are positive (i.e. reflect an unhappy customer). Thus, we were careful evaluating our models, since measuring loss as a very basic classifier could do very well by traditional evaluation methods by simply outputting 0 (happy customer) for every test case. Some features are duplicates of others, and other features contain the same value for every example (standard deviation of 0.00).

Data Preprocessing

We split the training data into train/test/dev subsets. We have approximately 70,000 examples and initially thought of doing a 60,000 / 5,000 / 5,000 split, but later decided to amend that to a 50,000 / 10,000 / 10,000 breakdown given the relative sparsity of positive examples.

Once we had these sets, we normalized each using scikit-learn's StandardScaler, which performs a basic subtract-mean divide-by-variance process [7]. We used the same mean and variance for all sets to ensure they come from the same distribution. We did not derive any features, as in theory the network will automatically do so.

4. Methods

Given the anonymized numerical feature data and single output, we felt that a fully connected multi-layer neural network would be the best bet for our algorithm. However, before creating that model, we created 1. a logistic regression model and 2. a multi-layer perceptron neural network with a single hidden layer of 100 nodes. We choose for our initial loss function a standard logistic loss: Loss (y^{truth}, y^{pred}) = $y^{truth}\log(y^{pred})$ + (1- y^{truth})(1-log(y^{pred})).

Multi Layer Neural Network

We used tensorflow to implement this network [8]. The results of this network are discussed in the next section.

We also decided to use a Xavier weight initialization, which is designed to keep the scale of the gradients roughly the same in all layers. This initializes the weighs in the range x = sqrt(6. / (in + out)); [-x, x], where in is the number of input neurons and out is the number of ouput neurons [9].

5. Experiments/Results/Discussion

We performed three experiments to improve our multi layer model's AUC score: tweaking the loss function, ensemble methods, and hyper-parameter tuning. We will discuss these individually here.

Tweaking the Loss Function

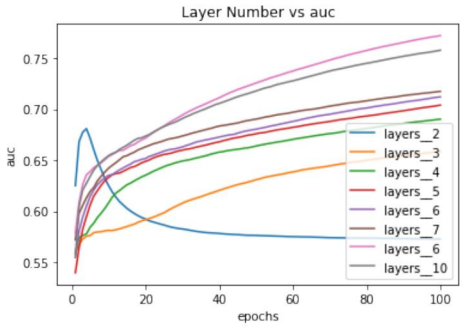
We changed the loss function of the network to Loss (y^{truth}, y^{pred}) = $\lambda * y^{truth}\log(y^{pred})$ + (1- y^{truth})(1-log(y^{pred})). Since more than 95% of output values in the data were 0's, our model initially had a hard time predicting 1s. We wanted to incentivize the network to weight these cases more heavily. We adjusted the lambda value until we found one that outputs 1's with the same frequency as they appear in the underlying data. That lambda value is 5. We added this lambda by simply multiplying all y values by that number.

Hyper-Parameter Tuning

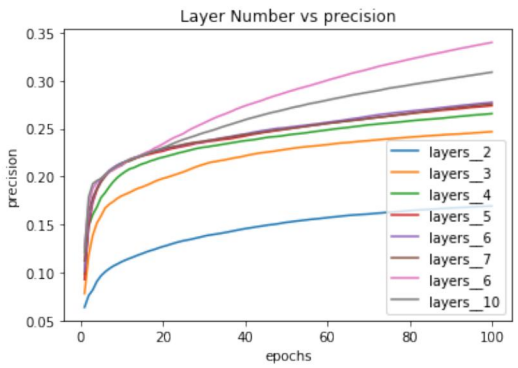
We tuned the number of layers of the network (from 2 to 10 layers), and the number of nodes in each layer. To do so, we ran 8 networks with various numbers of layers and numbers nodes per layer, and settled on the network which generated the highest AUC (a network with 6 layers

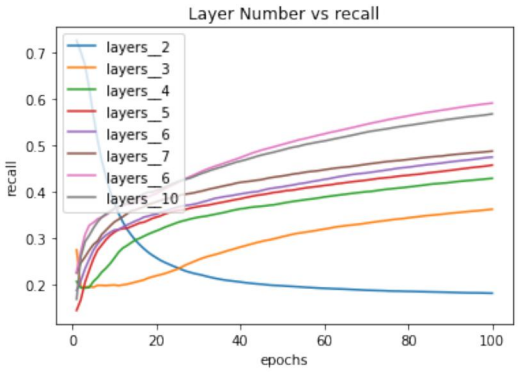
with [370, 400,1000,500,100,1], nodes per layer). This model had an AUC score of .7720. This process is shown here:

Model	AUC Score
layers__2	0.6808
layers__3	0.6586
layers__4	0.6901
layers__5	0.7039
layers__6	0.7120
layers__7	0.7174
layers__6(#2)	0.7720
layers__10	0.7577



This model also preformed the best in terms of precision and recall:

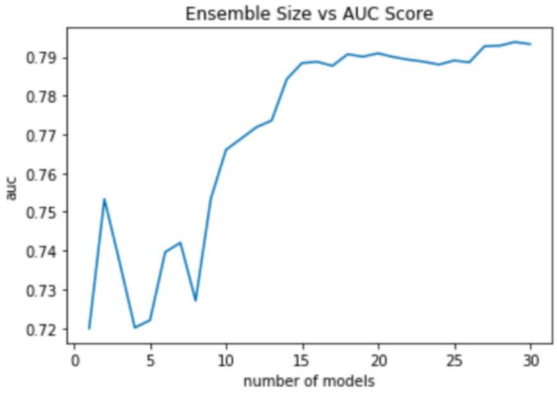




Ensemble Methods

Using various models all with different (random) initializations, we were able to get better results by using model ensemble to reduce the noise in our data. We ran ensembles of the highest performing model after hyper-parameter tuning of various sizes from 1 to 30, and

averaged their outputs to generate predictions by ensemble instead of by individual models. Ultimately, the best-performing ensemble (of 29 models) yielded an AUC of .7940 on the test set.



6. Conclusion/Future Work

With a final AUC of 0.794 on our test set, we feel that our model performed well given that an AUC of 0.83 won the competition. For future work consider the following:

1. We would like to get a better sense of the relative importance of our features. In order to do this it would be interesting to tweak the feature values one by one and see how doing so affects our model's outputs.

2. Additionally, after some inspection it appears that the data may be distributed according to two different Gaussians. As such, it would be interesting to fit a mixture of Gaussians model to the data, which would give us the mean vectors and covariance matrices of the two Gaussians. After fitting the mixture of Gaussians, we would be able to assign each point in the our training, dev, and test datasets to one of the two Gaussians. After this, we would create two separate neural networks, potentially with different architectures but both with sigmoid outputs. The first network would be trained with all points in the train set corresponding to the first Gaussian, and the second would be trained with all points in the train set corresponding to the second Gaussian. After training the networks, predictions would be made on each point in the dev and test sets. Concretely, this process would involve the following: for each point in the dev or test set, if the point corresponds to the first Gaussian, run it through the trained neural network corresponding to the first Gaussian, and predict class 0 if the output of sigmoid is less than or equal to 0.5 and class 1 otherwise; for all points corresponding to the second Gaussian, perform the same process but using the neural network corresponding to the second Gaussian.

Contributions:

Sam Schwager - logistic regression baseline, initial TensorFlow model, loss function tuning, final paper writing and editing
Scott Morris - MLP baseline, loss function tuning, project poster, final paper writing and editing
Nick Steele - hyperparameter tuning, model ensemble, project poster, final paper writing and editing

References

[1] Guo, Cheng and Berkhahn, Felix. Entity embeddings of categorical variables. arXiv:1604.06737v1, 2016.

[2] van den Elsen, D. (2017). Santander Customer Satisfaction, Vrije Universiteit Amsterdam Faculty of Economics and Business Administration research paper. https://science.vu.nl/en/Images/werkstuk-elsen_tcm296-865964.pdf

[3] Silva, L., G. Titericz, D. Efimov, I. Tanaka, D. Barusauskas, M. Michailidis, M. Muller, D. Polat, S. Semenov, and D. Altukhov (2016). Solution for santander customer satisfaction competition, 3rd place. https://github.com/diefimov/santander_2016/blob/master/ README.pdf.

[4] Wang, S. (2016). Predicting banking customer satisfaction. https://shuaiw.github.io/assets/data-science-project-workflow/santander-customer-satisfaction.pdf.

[5] Yooyen, T. and K.-C. Ma (2016). Csci 567 spring 2016 mini-project. https://markcsie. github.io/documents/CSCI567Spring2016Project.pdf.

[6] https://www.kaggle.com/c/santander-customer-satisfaction/data

[7] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer