Binary Classification of Rare Events in Consumer Advertising Data Using Machine Learning Methods

1 Introduction

The data being considered comes from Valassis, an advertising and marketing analytics company, and represents a combination of the buying and browsing behavior of consumers. The behavior for each consumer is split up to represent their relative interest in each category, and each consumer is labeled according to whether or not they were 'converted'. Converted refers to when a consumer completes a valuable action such as signing up, registering, or purchasing, after viewing an advertisement.

Given the nature of the data, two natural questions arise: What features correspond to consumers converting, and can we predict if a given user will convert given their user data. Mathematically, these questions can be addressed as dimensionality reduction and prediction, respectively. For dimensionality reduction, we seek to establish which features hold the most representation in the data. For prediction, various machine learning algorithms are employed by training the methods on the provided training data and evaluating their effectiveness at predicting customer conversion in the validation data set.

2 Data Selection and Preparation

The data was provided as .csv files, which we converted to numpy matrices. When a user did not have any data for a category, we added a zero in its place. This allowed classical matrix methods such as SVD to be applied to the problem directly. The matrices for each set of data were $N \times M$ where N is the number of users, and M is the number of features (following conventional notation). One challenge we encountered was an extra feature (topic ID number not present in the list of interests) in the validation set that was not present in the training set. Ultimately, we decided to simply not include that feature in our analysis by deleting that column since we had no way of interpreting that feature's inclusion in the algorithms and results.

Each data set contains both long term and short term data. The short term data is considerably sparser than the long term data, and many users do not have any short term data at all. Our plan was to start by running methods on the long terms data as a baseline, and potentially using short term data to further refine those results. For the sake of the analyses in this project, we only ended up using the long term data.

We also noticed there were more category labels in the STI data and we had to go in and add columns of zeros in the appropriate places.

3 Algorithms

3.1 Feature Selection

For feature selection, we decided to focus on using the sklearn implementation of PCA (with the full SVD solver) to analyze the training data. The PCA algorithm calculated that we needed the first 453 singular vectors to capture 99% of the variance in the data, so we used a truncated SVD

with the cutoff being $p = 453$ singular vectors. Upon fitting our PCA model with the training data and projecting it onto its two leading singular vectors, we produced the representation in Figure 1.

The PCA representation can often be used to find a lower dimensional space where the data can be split in an intuitive way for classification purposes. However, our PCA projection shows that the data for both the converted and non-converted customers is clearly reliant on far more than just its two most prominent singular vectors. The second part of Figure 1 is all of the singular values of the training set. One would usually hope to see some 'obvious' cutoff point for truncation of the singular values (made even more obvious with a semi-log plot), but in this example that dropoff point does not happen until almost all of the singular values have been included. Therefore, we decided to rely on the 453 singular vectors that sklearn told us to use, as discussed above.

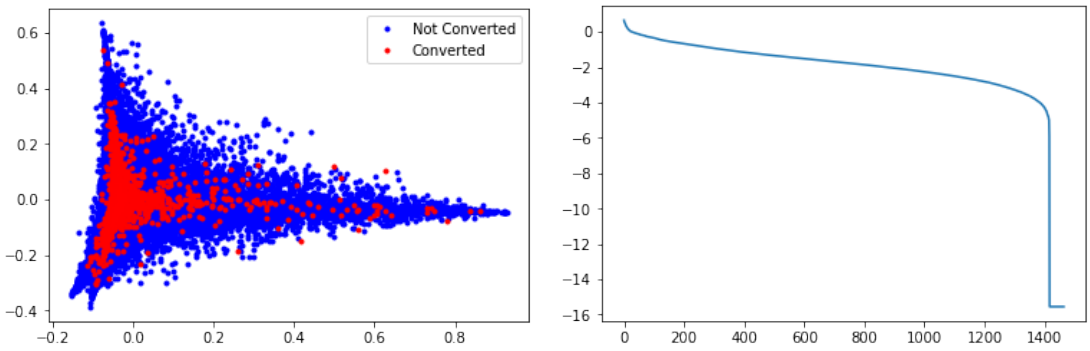

Figure 1: (Left) Training data transformed into its PCA space.
(Right) The semi-log plot of singular values of the training data set.

3.2 Prediction

One method that is often employed to deal with data with rare events is ensemble models. Employing multiple machine learning models and using the average or weighted average of the results to obtain the final prediction is a simple example of an ensemble method. We employed a simple voting classifier from sci-kit learn to obtain final predictions. The models that voted within this ensemble method were simple Naïve Bayes and Random Forest models, as well as a logistic regression model. We began by performing dimension reduction on the training and test data using PCA. Results for this method can be found in Table **??**.

When dealing with machine learning models, we normally judge performance of the model according to accuracy. However, when attempting to predict rare events, this metric is not appropriate. Therefore, we instead consider the ROC AUC score, which is defined the area under the ROC curve plotted with one minus specificity versus sensitivity in the prediction. This metric takes into account the true positive rate as well as the true negatives and false positives in evaluating the performance of the model. Using this metric, we train a logistic regression model using the sci-kit learn package in python. Logistic regression employs the sigmoid function within the regression algorithm to deal with outliers in the data [1]. We compile results for this method in Table **??**.

We explored using deep neural networks with large number of nodes in each layer for prediction. We considered a 4 layer neural network with decreasing number nodes per layer, starting with

10,000 nodes and decreasing by an order of magnitude in each layer. We expand upon this simple feed-forward network by considering an autoencoder.

3.3 Autoencoder

Because of the rareness of the successful conversions, we consider this event an anomaly in the dataset. Therefore, we apply techniques for anomaly detection. Autoencoders are used for image denoising, dimensionality reduction, and anomaly detection (among others). This unsupervised machine learning technique is, in simplest terms, a deep neural network consisting of two parts, encoding and decoding. Autoencoders take an input and pass it through a series of layers with monotonically decreasing nodes (the encoder) and then passes it through another set of layers which are monotonically increasing in the number of nodes (the decoder). The final layer the size of the original input. The goal of the autoencoder is be able to reconstruct the original input with respect to the mean squared error.

We use autoencoders to detect anomalies. This is done by training the neural network with only the "normal" data points. In our case, "normal" refers to the data points where the customer did not convert. Once this is done, the network should have a fairly good understanding of what a "normal" data point looks like. So when a datapoint when the customer converted is predicted by the autoencoder (an anomaly), the mean squared error should be large. This means we need to find a threshold for this mean squared error.

The architecture we use for the autoencoder is shown in Figure 3. It consists of 9 layers with varying nodes. After training the autoencoder, we feed the entire training dataset into it and obtain the mean squared error as shown in Figure 2. This figure shows that some of the anomalies will be found with a threshold of over 150.

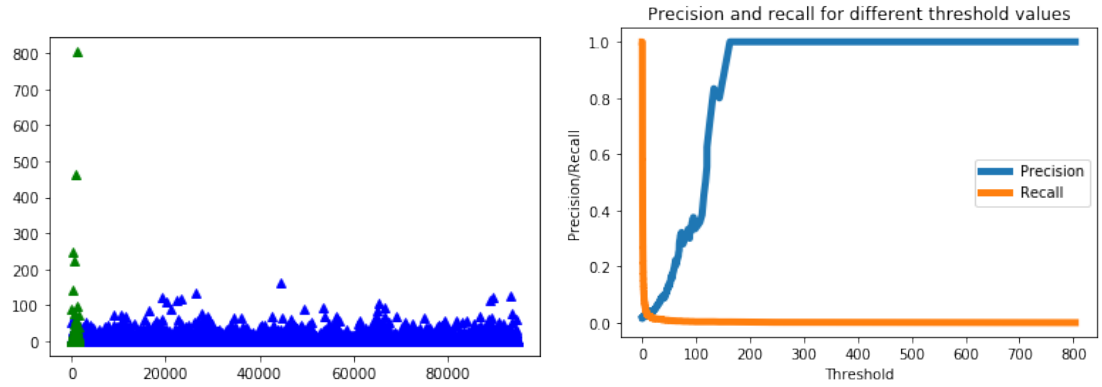

Figure 2: Mean squared error (left) and precision plot (left).

The results can be found in Table **??**.

3.4 Poor Predictions

Among the methods that we employed, some were employed naively, and their results will only be informally summarized here. We discovered that Support Vector Machines (SVM) did not work at

all, likely due to the combination of highly nonlinear classification with an imbalance in the number of data points in each class. SVM attempted to classify all of the test data points as not converted, which we took to mean there were too many not converted data points (see Figure 1.

Another method that did not work well was naively using a voting ensemble without performing PCA first. This created predictions with very few (if any) correct predictions, and many false negatives. As discussed earlier, this did lead to doing predictions on the data after running PCA, which significantly improved the results.

Gradient boosting was attempted, but only with the default settings. We are not prepared to say that it cannot be used to help analyze this data, but more work is needed to determine which solver and tolerances are most appropriate for this problem.

4 Conclusion

While modern machine learning methods are very powerful, they are ill-equipped for handling sparse and rare-event data. We tried implementing many of these methods in a straight-forward, out-of-the-box kind of way, but it often proved to be ill-suited to the situation. Dimensionality reduction helped ensemble methods perform better, as well as auto-encoder, but using PCA to understand the underlying behavior of the data proved unfruitful. Other methods could still work, but they would likely need to exploit the asymmetry in the classification.

References

[1] D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression*. Wiley-Interscience Publication, 2 ed., 2000.

[2] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, and e. al, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[3] T. Oliphant, "NumPy: A guide to NumPy." USA: Trelgol Publishing, 2006–. [Online].

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] P. Virtanen, R. Gommers, and j. . P. y. . . . i. . . Ian T Jolliffe and Jorge Cadima, title = "Principal component analysis: a review and recent developments"

[6] *Random Decision Forests*, (Montreal, QC), 1995.

[7] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. Academic Press, 4 ed., 2009.

[8] 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," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] "sklearn.ensemble.votingclassifier." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassi Accessed: 2019-11-2.

5 Appendix

Table 1: Log Regression with PCA

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	1	79390	P′
	n′	9	608	N′
	total	P	N	

Table 2: Ensemble with PCA

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	1	79386	P′
	n′	2	619	N′
	total	P	N	

Table 3: AutoEncoder with PCA

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	19	77478	P′
	n′	2442	69	N′
	total	P	N	

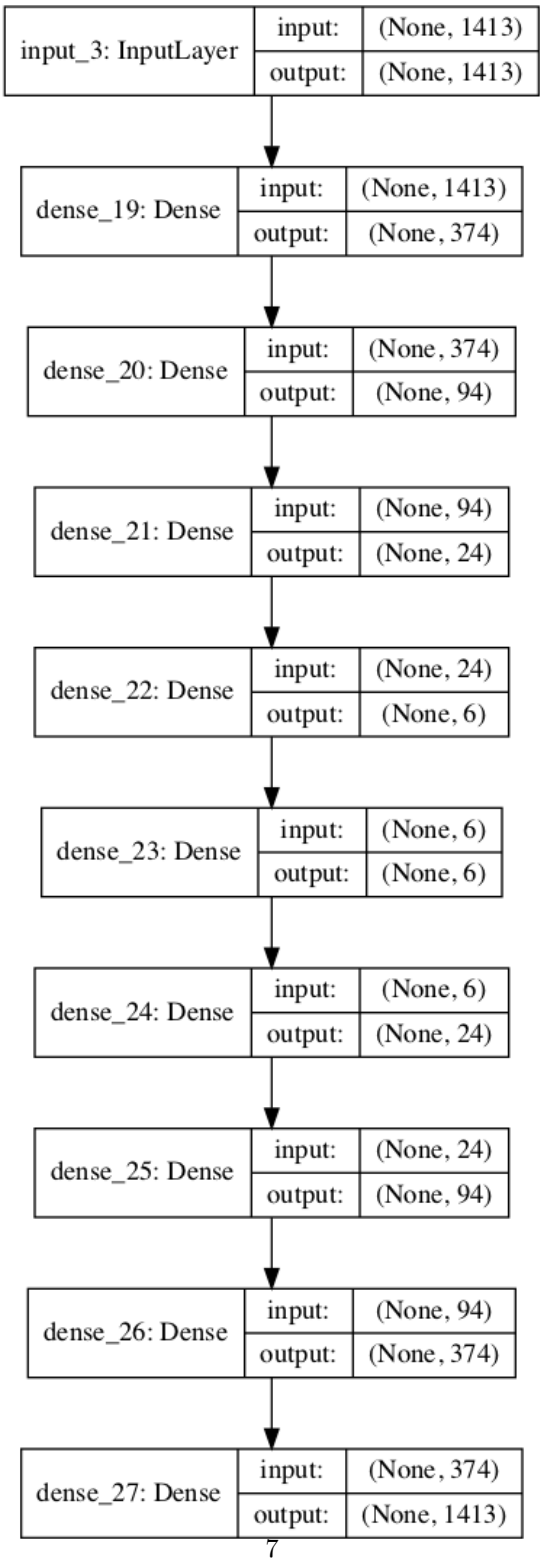

Figure 3: Architecture for the autoencoder.