Prediction Churn Reduction

Bhantana H., Bizhanov D., Harutyunyan T., Kishore S.

Motivation

It is hard to imagine the modern world without machine learning algorithms, which are used ubiquitously in many industries. For instance, banks use predictive algorithms to score their clients and assess risks. IT giants such as Google and Amazon use complex Deep Neural Networks as essential components of their services - Google Translate, Alexa, Google Photos, self-driving cars, and others. As our lives become more digital, cybersecurity becomes more necessary. An extremely flexible tool, machine learning is used in many different contexts, from simple spam filters to sophisticated methods to detect suspicious activity, spotting attacks and blocking any malicious intent. There are many reasons why machine learning and deep learning are successful in such a dynamic field: automated algorithms are better adapted for rapid changes than heuristic expert-driven approaches, scalability, machine learning is more adaptable to new conditions than expert-driven approaches.

Despite the obvious usefulness of machine learning algorithms, many challenges persist. One such challenge is prediction churn. Prediction churn is the discrepancy between the predictions made by the same model trained on different data samples and is defined as a proportion of mismatched data points to the number of overall predictions. Churn inevitably occurs during the iterative lifecycle of any predictive algorithm. However, mismatches may create many problems for the end users and can be detrimental for businesses that use such algorithms. As an illustration, Google faced backlash when their image classification system first identified photos as trees, but then classified them as cars after retraining the model on an extended dataset. In a different context, this undesired behavior could lead to serious negative consequences.

Prediction churn can be further split into two categories. The first category is desirable or good churn. It happens when an updated model version makes correct predictions in cases where the old model makes mistakes. The second category is undesirable or bad churn. This churn is the most problematic since an updated model makes mistakes in cases where the old model’s predictions are correct. Our goal is to reduce undesirable churn, and at the same time, increase accuracy.

Despite its significant importance, prediction churn is a highly overlooked research area. However, two approaches have been suggested to address prediction churn. The first approach reduces randomness in the training methodology to improve training stability. Sources of randomness in typical machine learning models include initialization, different training runs, choice of hyperparameters, and order of GPU computation. When randomness is reduced, prediction stability increases and prediction churn decreases. However, this approach requires solutions unique to the underlying model type. An alternative approach frames the problem from a label modification perspective and is model-agnostic. Instead of training on the true labels, the target model trains on a convex combination between hard one-hot encoded labels and soft labels produced by the auxiliary model.

We pursue the following objectives within this paper: first, we will create a unified approach to conduct experiments and compare three known methods. Second, we will create a python package that generalizes our approach and is independent of underlying models. That will ensure that anyone can use the package in their environment. Finally, we will work on a novel churn reduction method.

Methodology

Overview

The goal of our work is to identify effective and easy-to-implement methods for reducing prediction churn independent of the underlying prediction model. To achieve this goal, we compare existing three methods: Label Smoothing (Bahri & Jiang, 2021), Anchor (Fard et al., 2016) and Distillation (Jiang et al., 2021). Specifically, we're looking at the steps and parameters that need to be tuned to achieve churn reduction for each method. We compare the methods quantitatively using metrics defined in those methods' papers. All three methods involve modification of training labels using the older model (also referred to as the teacher model).

Complexity of implementation

Label Smoothing: Label smoothing uses the K-Nearest Neighbors algorithm on the logit layer of the predictions from the teacher model for finding similar data points to each data point. This method combines local and global smoothing to produce modified labels, see equation (1).

$$y_{a,b}^{KNN} = (1-a) * y + a\left(b * {}^{1}/_{L} * 1_L + (1-b) * \eta_k(x)\right) \quad (1)$$

Here, a is the tradeoff between true label y and smoothed label, b is the tradeoff between global smoothing and local smoothing of teacher's k labels $\eta_k(x)$, L is the number of classes and 1_L is a vector of ones.

Anchor: The anchor method works in two stages. First, it employs a Markov Chain Monte Carlo (MCMC) method to stabilize the training to imitate future changes in data. Second, it implements stabilizing operators to mitigate actual changes from new data. In the scope of this work, we focus only on the stabilizing operator stage, using the operator: Regress to Corrected Prediction (RCP), see equation (2).

$$\tilde{y}_j = \begin{cases} \alpha * f(x_j) + (1-\alpha) * y_j, & if\ y_j * f(x_j) \geq 0 \\ \epsilon * y_j, & otherwise \end{cases} \quad (2)$$

Here, a is tradeoff between true label y_j and teacher's label $f(x_j)$ when teacher classification is accurate, and $\epsilon \in [0,1]$ is hyperparameter to smooth the true label if the teacher classification is wrong.

Distillation: The concept of distillation (Ba & Caruana, 2013; Hinton et al., 2015) was originally developed to use knowledge gained from complex networks to train smaller ones. The distillation method is essentially the training of student/downstream models using distilled labels. Distilled labels are a weighted combination of the true labels and labels from the older/teacher model, see equation (3).

$$\tilde{y}_j = \lambda * f(x_j) + (1-\lambda) * y_j \quad (3)$$

Here, λ is tradeoff between true label y_j and teacher's label $f(x_j)$.

Comparison

We choose three different types of data and three metrics to measure the effectiveness of these methods.

Data Sets: We have selected *CIFAR 10* (Canadian Institute for Advanced Research) which is collection of 60,000 images (32 by 32 pixels) across ten classes, *Online News Popularity* which is set of 40,000 samples of tabular data with 60 features and two classes (popular or not), and *IMDB* which is textual data with of 60,000 online reviews and two classes (recommend or not).

Models: We are using ResNet20 for image classification, Bidirectional LSTM for sentiment analysis (text classification), and a Feedforward network for tabular data classification. The employed models have sensible default hyperparameters in place, hence we are not conducting hyperparameter tuning.

Churn is defined as the expected number of disagreements between two models, i.e., it is the average number of the labels misclassified between student and teacher, noted as $C(f_T, f_S)$.

Metrics:

- *Churn Ratio* is calculated as $C(f_0, f_1)/C(f_0, f_2)$ where the old model is f_0, new model trained with a methodology is f_1 and the new model trained without any methodology is f_2 (also referred to as the baseline model). This is a measure of model improvement, where a lower ratio indicates that the model methodology reduces churn.
- *Good Churn* is the number of correctly classified datapoints by student's model which were misclassified by the teacher's model; *Bad Churn* is the number of misclassified datapoints by student's model which were correctly classified by the teacher's model.
- *Win-Loss Ratio* (WLR) captures tradeoff between good churn and bad churn as $Good\ Churn/Bad\ Churn$. This is a measure of model performance, where values greater than one indicate the predominancy of good churns over bad churns.

Results

In this section, we are going to present the results of experiments carried out exclusively by distillation and anchoring methods. We did not experiment with the label smoothing method, since the complexity of hyperparameter tuning is relatively higher, which does not meet easily to implement criteria. Each experiment was run multiple times and standard errors were generated.

Method	Lambda	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Distillation	0.2	0.657 (0.0)	0.045 (0.0)	0.047 (0.0)	1.042 (0.07)	0.92 (0.1)
	0.4	0.658 (0.01)	0.039 (0.0)	0.041 (0.0)	1.062 (0.06)	0.8 (0.11)
	0.6	0.658 (0.01)	0.031 (0.0)	0.035 (0.0)	**1.1 (0.07)**	0.656 (0.07)
	0.8	0.657 (0.01)	0.025 (0.0)	0.026 (0.0)	1.079 (0.09)	**0.506 (0.06)**
Baseline		0.656 (0.0)	0.05 (0.0)	0.051 (0.0)	1.019 (0.07)	

Table 1: Results of Knowledge Distillation on Tabular Dataset

Method	Alpha	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Anchor	0.2	0.657 (0.0)	0.049 (0.0)	0.051 (0.01)	1.049 (0.1)	0.985 (0.09)
	0.4	0.658 (0.0)	0.044 (0.01)	0.047 (0.01)	1.059 (0.1)	0.899 (0.1)
	0.6	0.658 (0.0)	0.039 (0.0)	0.042 (0.0)	**1.083 (0.1)**	0.796 (0.08)
	0.8	0.658 (0.01)	0.033 (0.0)	0.035 (0.0)	1.074 (0.07)	**0.676 (0.07)**
Baseline		0.656 (0.0)	0.05 (0.0)	0.051 (0.0)	1.019 (0.07)	

Table 2: Results of Anchor RCP (epsilon=1) on Tabular Dataset

Tables 1 and 2 show results from using Tabular Online News Popularity dataset using the Distillation and Anchor method, respectively. The model's accuracy seems invariant of the choice of churn reduction method. However, we do see changes in the churn metrics. Bad Churn, Good Churn, and Churn Ratio decrease with increase in Lambda and Alpha. This is because increasing these parameters pushes the model to learn more from the teacher model. Amongst the key metrics, Churn Ratio is minimized using Knowledge distillation with lambda = 0.8. Thus, we see knowledge distillation outperforms the anchor method. In WLR, we observe that it is maximized on intermediate values of Lambda and Alpha. This is because Lambda/Alpha too low Lambda/Alpha pushes the model to learn more from original labels, and thus there will be a significant amount of bad churn. When these parameters are too high, the model does not learn too much new information and thus the good churn is too low to cause a rise in WLR. Win Loss Ratio was maximized in Knowledge Distillation with lambda=0.6.

Method	Lambda	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Distillation	0.2	0.816 (0.01)	0.076 (0.01)	0.072 (0.01)	1.43 (0.19)	0.939 (0.03)
	0.4	0.814 (0.01)	0.071 (0.01)	0.068 (0.0)	**1.534 (0.16)**	0.871 (0.03)
	0.6	0.816 (0.0)	0.067 (0.0)	0.061 (0.0)	1.488 (0.15)	0.793 (0.02)
	0.8	0.818 (0.0)	0.061 (0.0)	0.054 (0.0)	1.25 (0.15)	**0.753 (0.02)**
Baseline		0.810 (0.0)	0.05 (0.0)	0.051 (0.0)	1.286 (0.09)	

Table 3: Results of Knowledge Distillation on Image Dataset

Method	Alpha	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Anchor	0.2	0.818 (0.01)	0.073 (0.01)	0.059 (0.0)	**1.068 (0.13)**	0.883 (0.03)
	0.4	0.812 (0.00)	0.074 (0.01)	0.058 (0.0)	1.042 (0.07)	0.875 (0.03)
	0.6	0.818 (0.0)	0.07 (0.0)	0.057 (0.0)	1.052 (0.1)	0.865 (0.04)
	0.8	0.818 (0.1)	0.073 (0.01)	0.056 (0.0)	1.019 (0.08)	**0.858 (0.03)**
Baseline		0.811 (0.0)	0.05 (0.0)	0.051 (0.0)	1.019 (0.07)	

Table 4: Results of Anchor RCP (epsilon=1) on Image Dataset

Tables 3 and 4 show results from the CIFAR-10 image dataset. Here, once again, we see that accuracy does not change much from change in Alpha/Lambda. We see that knowledge distillation significantly outperforms the anchor method, both on WLR and on Churn Ratio. Churn Ratio is minimized at higher lambda (lambda=0.8), whereas WLR is maximized at intermediate values of Lambda (lambda=0.4). One theory on why distillation performs much better is that the distillation method was specifically developed and tested on neural networks with multi-class classification objectives, whereas the Anchor method was developed for machine learning methods in general, primarily for binary classification.

Method	Lambda	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Distillation	0.2	0.073 (0.009)	0.02 (0.002)	0.267 (0.007)	1.054 (0.12)
	0.4	0.054 (0.006)	0.02 (0.003)	0.373 (0.082)	0.832 (0.052)
	0.6	0.045 (0.002)	0.017 (0.003)	**0.389 (0.071)**	0.704 (0.013)
	0.8	0.044 (0.002)	0.015 (0.002)	0.334 (0.035)	**0.668 (0.037)**

Table 5: Results of Distillation on Text Dataset

Method	Alpha	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio
Anchor	0.2	0.066 (0.002)	0.02 (0.002)	0.295 (0.027)	0.974 (0.034)
	0.4	0.074 (0.016)	0.021 (0.0)	0.286 (0.052)	1.077 (0.179)
	0.6	0.061 (0.017)	0.025 (0.008)	0.256 (0.116)	1.92 (1.774)
	0.8	0.064 (0.009)	0.023 (0.002)	**0.368 (0.045)**	0.994 (0.121)

Table 6: Results of Anchor RCP (epsilon=1) on Text Dataset

Tables 5 and 6 show results from the text dataset. Once again, distillation with high lambda performs best in reducing churn ratio. Similarly, within distillation, win loss ratio is minimized on an intermediate value of lambda=0.6. The anchor method fails to reduce churn ratio in this case. The behavior on win loss ratio is also different, with it being maximized on alpha=0.8. More investigation will be needed to understand why the anchor does not perform as expected here. The appendix contains more detailed results with a larger choice of hyperparameters.

These results lead us to believe that going forward, knowledge distillation should be the preferred method of choice for reducing churn. Development of a novel approach to reducing churn warrants understanding what the distillation method does well that the anchor method does not. A caveat is that our experiments were restricted in scope and not exhaustive. Additionally, given that distillation does not significantly outperform anchor in the tabular binary classification task implies that we cannot generalize our results to conclude that distillation is always better than anchor.

Other Objectives:

Package development

The main goal for this milestone is to create a python package. This package needs to be generalizable and model agnostic, meaning that the teacher and student models can be any python model without much constraint. Creating the package will require a few steps: software design, developing core methods compatible with PyTorch framework, extending the package to TensorFlow, and providing interface for further extensibility.

In the software design stage, we will deliver an architectural structure. We will use UML (Unified Modeling Language) diagrams for documenting interclass connections. During this phase, we will restrict ourselves to the PyTorch library and make sure that the package works as expected. To make that happen, we will develop a set of unit and integration tests using the Pytest framework and the PyTorch internal testing functionality.

We plan to extend the package to be compatible with TensorFlow to ensure that the solution covers more use cases. During this phase, we need to find differences and similarities between the two frameworks. It may require some refactoring in the initial version of the package.

Finally, to make our framework truly general, we need to develop an extension-based API (Application Programming Interface) and provide a code interface. This step will guarantee that new frameworks will be compatible with our code base.

Work on a novel method

In this milestone, we have to develop a novel prediction churn reduction method. Having completed the experimentation on existing methods, and after having completed package creation, we have a strong foundation to start working on a new churn reduction approach. This milestone will be divided into sprints at the beginning of the second semester since we do not have enough information to develop a plan of actions at this stage.

Citations

Ba, J., & Caruana, R. (2013). Do Deep Nets Really Need to be Deep? NIPS,

Bahri, D., & Jiang, H. (2021). Locally Adaptive Label Smoothing for Predictive Churn. *ArXiv*, *abs/2102.05140*.

Fard, M. M., Cormier, Q., Canini, K. R., & Gupta, M. R. (2016). Launch and Iterate: Reducing Prediction Churn. NIPS,

Hinton, G. E., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. *ArXiv*, *abs/1503.02531*.

Jiang, H., Narasimhan, H., Bahri, D., Cotter, A., & Rostamizadeh, A. (2021). Churn Reduction via Distillation. *ArXiv*, *abs/2106.02654*.

Appendix

Method	Alpha	Eps	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio Anchor
Anchor	0.2	0.6	0.801 (0.00)	0.072 (0.0)	0.058 (0.0)	1.025 (0.14)	0.893 (0.04)
		0.8	0.812 (0.01)	0.073 (0.01)	0.059 (0.0)	1.068 (0.13)	0.883 (0.03)
	0.4	0.6	0.803 (0.01)	0.073 (0.0)	0.057 (0.0)	1.053 (0.12)	0.871 (0.04)
		0.8	0.818 (0.0)	0.074 (0.01)	0.058 (0.0)	1.042 (0.07)	0.875 (0.03)
	0.6	0.6	0.801(0.0)	0.071 (0.01)	0.057 (0.0)	1.056 (0.1)	0.864 (0.03)
		0.8	0.818 (0.1)	0.07 (0.0)	0.057 (0.0)	1.052 (0.1)	0.865 (0.04)
	0.8	0.6	0.801(0.00)	0.068 (0.0)	0.055 (0.0)	1.053 (0.1)	0.84 (0.03)
		0.8	0.818 (0.1)	0.073 (0.01)	0.056 (0.0)	1.019 (0.08)	0.858 (0.03)
Baseline			0.811(0.0)	0.05 (0.0)	0.051 (0.0)	1.019 (0.07)	

Table 7: Anchor results on image dataset

Method	Alpha	Eps	Accuracy	Bad Churn	Good Churn	Win Loss Ratio	Churn Ratio Anchor
Anchor	0.2	0.6	0.656 (0.01)	0.034 (0.0)	0.035 (0.0)	1.035 (0.08)	0.687 (0.06)
		0.8	0.657 (0.01)	0.039 (0.0)	0.041 (0.0)	1.052 (0.07)	0.797 (0.07)
		1	0.657 (0.0)	0.049 (0.0)	0.051 (0.01)	1.049 (0.1)	0.985 (0.09)
	0.4	0.6	0.657 (0.0)	0.032 (0.0)	0.033 (0.0)	1.063 (0.07)	0.646 (0.07)
		0.8	0.659 (0.01)	0.036 (0.0)	0.039 (0.0)	1.105 (0.11)	0.743 (0.08)
		1	0.658 (0.0)	0.044 (0.01)	0.047 (0.01)	1.059 (0.1)	0.899 (0.1)
	0.6	0.6	0.657 (0.01)	0.029 (0.0)	0.032 (0.0)	1.079 (0.07)	0.605 (0.06)
		0.8	0.658 (0.01)	0.034 (0.0)	0.037 (0.0)	1.103 (0.1)	0.708 (0.06)
		1	0.658 (0.0)	0.039 (0.0)	0.042 (0.0)	1.083 (0.1)	0.796 (0.08)
	0.8	0.6	0.657 (0.01)	0.027 (0.0)	0.028 (0.0)	1.059 (0.05)	0.542 (0.04)
		0.8	0.658 (0.01)	0.029 (0.0)	0.032 (0.0)	1.084 (0.07)	0.609 (0.06)
		1	0.658 (0.01)	0.033 (0.0)	0.035 (0.0)	1.074 (0.07)	0.676 (0.07)
Baseline			0.656 (0.0)	0.05 (0.0)	0.051 (0.0)	1.019 (0.07)	

Table 8: Anchor results on tabular dataset