**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, it is easier and faster to tune a machine learning algorithm than to develop a brand new set of rules.

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. CHowever, 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.

Proofpoint is an American cybersecurity company whose services require many machine learning models. Such services offer a full range of protection, including cloud accounts, emails, web security, and private data safety. Given the delicate area in which Proofpoint operates and the large scale of production models, our client faces many challenges, including prediction churn.

Given the wide variety of machine learning models

Proofpoint uses, they are interested in a model-agnostic framework that they can use regularly. There are many different machine learning packages that Proofpoint may use, including PyTorch, TensorFlow, XGBoost, and others. Therefore, we need to develop a label modification solution and interface that Proofpoint mayt apply to all possible models with the least effort.

The capstone project consists of three objectives. First, we will compare a unified approach and compare to three known approaches. Second, we will create a python package that generalizes our approach and is independent of underlying models. That will ensure that our clients can use the package in their environment. Finally, we will work on a novel churn reduction method.

**Methods**

**Experimental Approaches**

We have identified three possible label modification methods to reduce prediction churn: knowledge distillation, anchor, and label smoothing. Each technique is independent of the underlying prediction model. We will test each of these using the same data and underlying models to compare the effectiveness of each approach.

*Knowledge Distillation*

Developed in 2015, ksAftering,,K,

*Anchor Method*

The anchor method employs Markov Chain Monte Carlo (MCMC) method and two stabilizing operators this method can be used with a wider range of machine learning models.

*Label Smoothing*

Label smoothing uses the K-Nearest Neighbors algorithm instead of a teacher model for finding similar data points using the logit layer of a neural network that is used as a helper model. Further, they combine local and global smoothing to produce modified labels.

**Data, Models, and Metrics**

We will test our approaches on different classification models with various data types to understand how we may reduce prediction churn in multiple scenarios. Namely, ResNet for image classification, Bidirectional LSTM for sentiment analysis (text classification), and TabNet for tabular data classification. The data sets we have selected are CIFAR 10 (Canadian Institute for Advanced Research), Online News Popularity, and IMDB reviews. CIFAR 10 is a collection of 60,000 images (32 by 32 pixels) across ten classes. Online news popularity is a tabular data set with 40,000 samples, 60 features, and two classes. IMDB online review is textual data with two prediction classes containing 60,000 reviews. We have chosen these particular data sets due to their popularity and their large data volume to train neural networks and satisfy our computation capacity.

The models we will employ have sensible default hyperparameters in place. We will not conduct a hyperparameter tuning because our solution is focused on model training. If we use hyperparameters tuning, we will employ the Ray framework.

To compare churn reduction methods, we have selected the following four metrics: churn, churn ratio, win-loss ratio, and good and bad churn. *Churn* is the expected amount of disagreements between two models, i.e it is the average number of the labels misclassified between student and teacher. *Churn Ratio*is calculated as C(f0,f1) / C(f0,f2) where the old model is f0, the new model trained with a methodology is f1, and the new model trained without any methodology is f2. This is a measure of model improvement, where a lower ratio indicates that the model methodology reduces churn. *Win-loss Ratio (WLR)* where the WIN is when the new model is able to correctly classify data points that the old model misclassified. LOSS is when the new model misclassified data points that were correctly classified by the old model. *Good Churn and Bad Churn***,** this metric divide churn into two categories: good churn—when the teacher's model predicts wrong, but the student's model predicts the correct classes; bad churn - when the predictions of the teacher's model are correct, but the student's model predicts the wrong classes.

The experiment will be conducted in the Colab, using GPUs. To ensure the reproducibility of the experiment, we will log the details of the experiments with MLFlow.This tool allows logging all metrics with parameters and artifacts such as models and data used for training.

**Package development**

The main goal for this milestone is to create a python package. As mentioned before, the package has to be general and model agnostic, meaning that the teacher and student models can be any python model. 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 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 and provide a code interface. This step will guarantee that new frameworks will be compatible with our code base.

**Work on a novel method**

The final milestone is the development of a novel prediction churn reduction method. After we have worked on the existing methods and package creation, we will 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.