Handling Concept Drift in Weakly Supervised Learning

Gaurav Sharma, Geoffrey Glass

gaurav_sharma1@brown.edu, geoffrey_glass@brown.edu

ADVISOR Prof. Stephen Bach

Department of Computer Science, Brown University

Abstract

As machine learning models increase in complexity, there are many applications that struggle to meet the demands for large amounts of labeled training data. This has caused researchers to turn to weak supervision models. The goal of our research is to improve performance of machine learning models in the weakly supervised online time series setting by accounting for concept drift in the data. Weakly supervised learning refers to scenarios where we have limited labeled training data, but large amounts of unlabeled training data. We write our own heuristic labeling functions that use input features of unlabeled data points to vote on a predicted output; we then train the model using the votes as labels. Machine learning in an online time series domain means that the model receives data points chronologically, and the model is tasked to make predictions on the current timestep before it can look at the next timestep. Concept drift is a phenomenon in time series data where the distribution of input features, or the relationship between inputs and labels shifts over time. Our experiments yielded mixed results with a concept drift-aware model outperforming the traditional control model in some instances.

This work and writing was done as a collaboration between Gaurav Sharma and Geoffrey Glass with advising from Professor Stephen Bach.

1 Introduction

The absence of labeled data in abundance is a major bottleneck in developing new machine learning algorithms and applications, especially because these require tens of thousands of labeled samples which is very expensive to obtain. Methods have been developed to use generative models from weak supervision sources like heuristics and domain knowledge where true label for a sample is modeled as a latent variable to generate observed labels. Since the structure of generative models affects these inferred labels, some methods (Alfonseca et al., 2012; Takamatsu et al., 2012; Roth and Klakow, 2013; Ratner et al., 2016), assume that it is user defined but because statistical dependencies cannot be ruled out, improved methods (Bach et al., 2017; Ratner et al., 2020) aim to learn the structure from weak supervision sources alone. For datasets laden with concept drift, we use a method based on the latter technique to generate probabilistic labels for unlabeled data. These labels are then used to train a downstream discriminative model.

Concept drift refers to the phenomenon where changes occur in the statistical distribution of time series data over time. This problem was first proposed in (Schlimmer and Granger, 1986) where it was stated that noise in the data may become useful information (or non-noise information) over time. We can distinguish drifts into two types based on the sources:

- **Real Drift** occurs when \( P_t(Y|X) \neq P_{t+1}(Y|X) \) with or without any change in \( P_t(X) \) and \( P_{t+1}(X) \). This will cause decision boundary change and has also been referred to as *conditional change* (Gao et al., 2007).
Virtutal Drift occurs when $P_t(X) \neq P_{t+1}(X)$ while $P(Y|X)$ does not change over time. This does not lead to change in decision boundary and has hence been referred to as temporary drift (Lazarescu et al., 2004) or feature change (Gao et al., 2007).

In addition to the types of drift, the changes in the distribution of data may occur in different forms like sudden/abrupt, incremental, gradual or cyclical. A drift may happen when the concept is switched suddenly or incrementally consisting of intermediate concepts in between. A drift can also happen when a new concept gradually replaces an old concept over time. There may also be a situation when old concepts might reoccur after sometime (Gama et al., 2014).

2 Methods
2.1 Definitions
Data Variables Used

- $X$: the input time series dataset where each timestep has certain number of samples. The dimensionality of the input is $t \times s \times f$ where $t$ is the number of timesteps, $s$ is the number of samples per timestep and $f$ is the number of features of the input data.

- $Y$: the true output labels with dimensionality $t \times s$ where each value is the true label corresponding to the input sample. Note that none of the true labels have been used in the training process. They have been used only for performance evaluation of the models implemented in this project.

- votes: predictions from the labeling functions (which are explained later). The dimensionality is $t \times s \times l$ where $t$ and $s$ are number of timesteps and samples per timestep whereas $l$ is the number of labeling functions. Each value represents the label that a particular labeling function predicts for a given input sample.

- unsupervised_labels: probabilistic labels from the label model where each value is a probability of occurrence of each class for a given input sample. The dimensionality of these probabilistic labels is $t \times s \times c$ where $c$ is the number of output classes for the given dataset.

Models and Labelers Used

- labeling_functions: These are the weak supervision sources that are used to generate labels for the input data using the generative models. These functions are rules derived out of patterns, heuristics, domain knowledge, etc. corresponding to each dataset (Ratner et al., 2020). These take a data sample as input and either assign a label to it or abstain from voting. The output of different labeling functions need not be mutually exclusive. This means that they can overlap and also at times, can conflict with each other. The label_model handles these statistical dependencies. The process of generating labeling functions usually involves an Exploratory Data Analysis (EDA) for each dataset. This helps in obtaining relevant information and patterns in the dataset which can then be used to write these rules. Sometimes, feature engineering is done to generate new features from the existing ones to generate clearer patterns to get better class separability. This is done for the Electricity dataset where trends of demand in electricity (for every 30 minutes) in New South Wales and Victoria states are obtained by comparing the actual value of the demand (present in the original dataset) for the current sample with the previous sample. Similarly, another feature is derived which monitors the change in demand over 24 hours. These features are then used in the labeling functions since they give us a better class separability.

For high-dimensional datasets (like Sensor-Drift and Malware), most important features are selected using a scoring function which provides a statistic for each of the features in the dataset. In the experiments, two scoring functions (for more ro-
bust results and avoiding dependence on just one function) have been used for the high-dimensional datasets which are chi-squared statistic and F-value of ANOVA test (both of which are measured between each feature and the class). Based on these values, the k-best features are chosen which are then studied for writing the labeling_functions. The process of selecting the k-best features is implemented using Scikit-learn’s feature selection module (Pedregosa et al., 2011). For Malware and Sensordrift, 30 features have been selected. Once these features are obtained, the dataset is then visualised using JMP Pro 16 as a basis for the labeling_functions. A screenshot of such visualisation is in Figure 1.

- **label_model**: an unsupervised naive Bayes model that is trained on votes obtained from the labeling_functions. Given the votes for an example, it can predict the probability of each output class as well as the log likelihood of the votes.

- **end_model**: a supervised model that is trained on X with unsupervised_labels. It is usually a logistic regression. It predicts the probability of each output class, for comparison against the true labels Y.

### 2.2 Training and Evaluation Methods

The first step is to select training data, a subset of the data prior to the current timestep. Our experimental approach fits a naive Bayes model on the target timestep’s votes. We then use this trained model to evaluate the likelihood of votes from previous timesteps. We select the timesteps with the highest log likelihood to use as our training subset. We have experimented with different thresholds for inclusion. We primarily use all the timesteps that scores above the median log likelihood. The control approach simply uses all timesteps prior to the target timestep. This is shown in Figure 2. Both the experiment and control approaches use the same training and evaluation technique as shown in Figure 3. A label_model is trained using the training subset of votes. This produces probabilistic unsupervised_labels to use in training of end_model. This is preferable to using the majority vote or the weighted average of votes because label_model learns statistical dependencies between the labeling_functions that were used to generate the votes.

End_model is trained using the training subset of X as inputs, and unsupervised_labels as outputs. Before training, X is standardized to a mean of 0 and standard deviation of 1 to improve regularization. End_model is usually a logistic regression model that uses an L-BFGS solver with L2 regularization. Finally, we evaluate the trained end_model on the target timestep, and compare the predictions to the true labels Y to calculate our accuracy score.

### 3 Datasets

- **Electricity**: This data is about the electricity pricing of two Australian states which are not fixed and are affected by demand and supply of the market (Harries and Wales, 1999). It has been collected from the Australian New South Wales Electricity Market. The dataset contains 45,312 instances from 7 May 1996 to 5 December 1998, with each sample collected after a period of 30 minutes. Therefore, there are 48 instances for one day. The dataset has 8 features and 2 classes (price increased, price decreased).

- **Weather**: This dataset is a subset of NOAA data which collects data from over 9000 weather stations worldwide (Ueno et al., 2001). The subset chosen is that of Offutt Air Force Base in Bellevue, Nebraska since this location had 50 years worth of data (1949-1999), which is a good benchmark for cyclical seasonal changes as well as long-term climate change (Elwell and Polikar, 2011). The dataset has 18,519 samples with 8 features and 2 classes to predict (rain, no rain).

- **Malware**: This dataset is a collection of over 100,000 executable files which are selected from a corpus of 1 million files ex-
Figure 1: SensorDrift Feature Visualization to create Label Function 5

Figure 2: Training data selection for Experimental and Control model

Figure 3: Training and evaluation pipeline
tracted from cuckoo sandbox (Guarnieri et al., 2013). The data is collected over a period of 44 months and has 482 hand-crafted features. This is a regression problem to predict the percentage of 52 antivirus programs that identified the executable as a virus (Huynh et al., 2017).

- **SensorDrift**: This dataset is collected over a period of 36 months (2007-2011) which consists of collection of readings from 16 sensors in a gas delivery system (Vergara et al., 2012). The sensors are activated when a particular gas is present in the chamber and hence, the classification task is to identify which gas is being measured by the sensors. The dataset has 2,926 samples with 128 features and 6 output classes.

### 4 Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>control</th>
<th>experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricity</td>
<td>61.1% acc</td>
<td>60.1% acc</td>
</tr>
<tr>
<td>Weather</td>
<td>72.7% acc</td>
<td>64.1% acc</td>
</tr>
<tr>
<td>Malware</td>
<td>37.4% mae</td>
<td>37.0% mae</td>
</tr>
<tr>
<td>SensorDrift</td>
<td>59.3% acc</td>
<td>57.2% acc</td>
</tr>
</tbody>
</table>

Table 1: Accuracy (classification tasks) or mean absolute error (regression task) for control and experimental settings for each dataset

The Weather control model uses all historical training data, which consists of about 37% rainy data. The control model proves to be flexible, providing good performance on both rainy and dry inputs. Meanwhile, the experimental model selects heavily imbalanced subsets of the data, often <10% rainy or >80% rainy. If the training distribution matches the target timestep’s distribution, the model performs slightly better than the control. However, the cutoff model occasionally selects an incorrect training distribution, such as 90% rain for a target timestep of 13% rain. This leads to a decreased average accuracy.

Electricity, on the other hand, did not suffer from this issue. The experimental and control models had very similar performance. However, it appears that the control model had slightly better accuracy due to a favorable tradeoff between more training data and more relevant training data.

The Malware dataset was evaluated as a regression problem, using mean absolute error. The Malware experimental model had a 0.4% lower error than the control model. The models had an equal number of timesteps with lowest error, but when the experimental model did better, it outperformed the control model by 0.7%. Meanwhile, when the control model did better, it only improved by 0.1%.

The SensorDrift dataset had similar performance between the experimental and control models. However, when the performance is visualized over different timesteps, there is a significant drop in the accuracy at timestep 7 for both the models. A deeper analysis into the data that is present at timestep 7 showed that the data is homogeneous in terms of class separability. This is also reflected in the way labeling functions respond. Labeling functions abstain from deciding which class a particular sample belongs to for most of the samples at this timestep which is shown in Table 2 while Table 3 shows the actual class distribution for data in timestep 7. It can be seen that the class Acetone has the maximum number of samples at this timestep, however, all the labeling functions vote very few samples as Acetone. Similar to the Electricity dataset, the control model had slightly better performance due to having more training data.
As seen in all of our experiments, the experimental models did not provide a significant boost in performance. It proved to be difficult for the experimental model to detect the ideal training data timesteps without the use of ground truth labels. The experimental model used similarities in label function votes to select training data timesteps. However, the labeling functions had unreliable performance, averaging 37% accuracy on the Electricity dataset.

5 Project Evolution

This project underwent substantial evolution before we landed on our final method. Our first attempt at changepoint detection used the PELT algorithm (Killick et al., 2012). This dynamic programming algorithm detects optimal changepoints in a time series with linear time complexity. Given a time series, and a cost function over a contiguous segment of the series, the algorithm computes the optimal segmentation of the time series so as to minimize the total sum cost of all the segments. We implemented this algorithm in Python, using a naive Bayes cost function. Calculating the naive Bayes cost involves fitting a naive Bayes model to the segment’s label functions’ votes and then computing the negative log likelihood of that data using the trained model.

We use the optimal segmentation to determine the ideal training data for a desired timestep. For example, if the optimal changepoints are at 201, 258, and 394; and we want to predict at t=375, we will use training data from t=258 to t=374. This can boost performance because it excludes outdated training data that is from an older distribution.

As seen in all of our experiments, the experimental models did not provide a significant boost in performance. It proved to be difficult for the experimental model to detect the ideal training data timesteps without the use of ground truth labels. The experimental model used similarities in label function votes to select training data timesteps. However, the labeling functions had unreliable performance, averaging 37% accuracy on the Electricity dataset.

5 Project Evolution

This project underwent substantial evolution before we landed on our final method. Our first attempt at changepoint detection used the PELT algorithm (Killick et al., 2012). This dynamic programming algorithm detects optimal changepoints in a time series with linear time complexity. Given a time series, and a cost function over a contiguous segment of the series, the algorithm computes the optimal segmentation of the time series so as to minimize the total sum cost of all the segments. We implemented this algorithm in Python, using a naive Bayes cost function. Calculating the naive Bayes cost involves fitting a naive Bayes model to the segment’s label functions’ votes and then computing the negative log likelihood of that data using the trained model.

We use the optimal segmentation to determine the ideal training data for a desired timestep. For example, if the optimal changepoints are at 201, 258, and 394; and we want to predict at t=375, we will use training data from t=258 to t=374. This can boost performance because it excludes outdated training data that is from an older distribution.

We tested this on a toy dataset based on Animals with Attributes (Xian et al., 2018). This dataset consists of 15 timesteps with: distribution 1 (t=1-4), distribution 2 (t=5-8), return to distribution 1 (t=9-12), distribution 2 (t=13-15).

Using our implementation of the PELT model, we successfully detected changepoints at t=5,9,13. We have two experiments, the first is a conventional weak supervision model that uses all previous timesteps as training data (use 1-6 when predicting t=7), and the second only trains on the current segment (use 5-6 when predicting t=7). The accuracy of these two models is shown below:

The next step was to adapt this to an online setting, where the changepoints must be detected using only the data prior to the current timestep. This means we must run changepoint detection at every timestep, and we still only train on the data from the most recent segment.
Here, we see that the online changepoint detection slightly improves performance, however it struggles with finding the correct segmentation when the target timestep is at the start of a new segment (such as timesteps 7 and 11 on the graph).

We realized that we only need to minimize the cost of the current segment rather than to sum cost of all segments since we're only making predictions for the current timestep.

This technique for changepoint detection was unreliable. When predicting at the start of a new segment, sometimes the algorithm will choose to include all of the previous segment because cost(full old segment + anomalous current timestep) < cost(anomalous current timestep). This is because the old segment is internally similar even though it differs from the current timestep.

Instead of quantifying the similarity of the entire segment, the cost function should actually be quantifying the similarity of past timesteps with the current timestep, since we are trying to predict on the current timestep.

To do this, we fit the naive Bayes model on the current timestep, and then evaluate log likelihood on the rest of the segment, to find the segment's similarity to the current timestep.

We had another insight to remove the requirement of contiguous segments. Many datasets—including our toy dataset and others such as weather predictions—have cyclical patterns where the data returns to an earlier distribution.

These two insights brought us to our current algorithm. We fit the naive Bayes model on the current timestep, evaluate log likelihood on all past timesteps, and then choose the best past timesteps to use as our “segment” of training data.

6 Conclusion

We proposed a new method for online weak supervision that selects ideal training data to mitigate the effects of concept drift. Our experiments yielded mixed results as it was hard to select the best timesteps without labels. Future work should consider new ways to develop labeling functions that have higher accuracy and coverage, as we have seen with the SensorDrift experiment. In addition, a more complex end_model could increase performance on high-dimensional datasets by learning better feature representations. We are excited to see future progress in this field.

7 Acknowledgements

We would like to thank Dr. Stephen Bach for advising us on this project.

References


Jing Gao, Wei Fan, Jiawei Han, and Philip S Yu. 2007. A general framework for mining concept-drifting data streams with skewed distributions. In Proceedings of the 2007 siam international conference on data mining, pages 3–14. SIAM.


