Learning from Data Streams with Concept Drift

Summary

Increasing access to incredibly large, nonstationary datasets and corresponding demands to analyse these data has led to the development of new online algorithms for performing machine learning on data streams. An important feature of real-world data streams is “concept drift,” whereby the distributions underlying the data can change arbitrarily over time. The presence of concept drift in a data stream causes many classical data mining techniques to become unsuitable, and therefore new approaches must be developed in their place.

In pursuit of this goal, we introduce the dynamic logistic regressor (DLR), a sequential Bayesian approach for performing binary classification on nonstationary data streams. We proceed to show how the DLR framework can be extended to cope with missing observations and missing and corrupted labels.

We proceed to describe a new meta-algorithm for performing classification and regression on data streams with concept drift. The convex hull of receiver operating characteristic (ROC) curves has long been used for identifying potentially optimal classifiers. Unfortunately, the ROC curve does not perform as expected when learning from data streams exhibiting concept drift. We introduce a modification to the ROC curve that provides an easily maintainable online summary of a classifier’s performance, even in the presence of concept drift. We similarly modify the recently introduced regression error characteristic (REC) curve, giving analogous dynamic summaries of online regressors.

We then introduce a system for online ensemble learning utilizing these dynamic performance curves. Using the convex hulls of these curves, we develop a simple framework for supervised learning with drifting data streams. We present empirical evidence with real and simulated data that demonstrates that the proposed method performs better than selected previous solutions.
Contents

1 Introduction 2

2 Introduction to Probability Theory 4
   2.1 What is Probability? 4
   2.2 Definitions and Basic Results of Probability Theory 5
      2.2.1 Probability density functions 6
      2.2.2 Random variables and expectation 8
      2.2.3 Conditional probability and Bayes’ theorem 10

3 Problem Definition and Previous Work 13
   3.1 The Online Semi-Supervised Learning Task 13
      3.1.1 Solving multi-class classification problems with binary classifiers 14
   3.2 Proposed Solutions for the Online Classification Task with Concept Drift 16
      3.2.1 Maintaining a set of binary rules 16
      3.2.2 Maintaining a decision tree 16
      3.2.3 Maintaining an ensemble of classifiers 17

4 The Dynamic Logistic Regressor for Semi-Supervised Classification 20
   4.1 Review of the dynamic logistic regressor 20
      4.1.1 The DLR equations 21
   4.2 Notes Regarding the DLR 23
      4.2.1 Estimating the process noise parameter $Q_t$ 23
      4.2.2 Handling missing observations 24
      4.2.3 Handling missing labels 25
   4.3 Modifying the DLR Handle Corrupted Labels 25
4.3.1 A simple noise model and its consequences ........................................ 26
4.3.2 Estimating $\rho$ ................................................................. 27

5 Dynamic Characteristic Curves for Online Ensemble Learning ........................................ 28
5.1 Review of ROC and REC curves ..................................................... 29
5.2 Dynamic Characteristic Curves ....................................................... 32
  5.2.1 Characteristic curves cannot cope with concept drift ......................... 32
  5.2.2 Dynamically weighting performance ........................................... 33
5.3 Using Dynamic Characteristic Curves for Online Ensemble Learning .................. 35
  5.3.1 Learning with dynamic characteristic curve convex hulls .................... 35
  5.3.2 Observations ................................................................. 37

6 Applications .................................................................................. 40
6.1 Results Regarding the Dynamic Logistic Regressor ...................................... 40
  6.1.1 Initialization .................................................................... 40
  6.1.2 An illustrative example ....................................................... 40
6.2 Results Regarding the DCCCH Algorithm ............................................... 42
  6.2.1 Experiment 1: streaming classification with concept drift .................... 44
  6.2.2 Experiment 2: streaming regression with concept drift ....................... 47
  6.2.3 Experiment 3: dynamic feature evaluation ...................................... 49

7 Conclusions and Future Work .................................................................. 51
7.1 Proposals for Future Research into the DLR algorithm ............................. 52
7.2 Proposals for Future Research into the DCCCH algorithm ....................... 53
  7.2.1 Automatic and adaptive selection of the parameter $\alpha$ ...................... 53
  7.2.2 Decentralizing the DCCCH algorithm .......................................... 54
  7.2.3 Theoretical results .................................................................. 55
7.3 Proposals for Other Future Research ................................................... 55
  7.3.1 Gaussian processes for Bayesian optimization .................................. 55
  7.3.2 Reinforcement learning in non-stationary environments ..................... 57
  7.3.3 Online density estimation ....................................................... 58
Chapter 1

Introduction

The amount of data transferred over communication networks is growing exponentially [Roberts, 2000]. With corresponding increases in data storage capacity and processing power, new methods are needed for extracting knowledge from fast-moving, quickly changing, and extremely large data sources. The growing field of data stream mining addresses these problems.

Data streams pose several unique problems that inhibit the application of standard data mining techniques. The dataset is continuously online and growing to include new measurements; therefore, effective algorithms for analyzing these data must be able to work within a constant memory footprint. In particular, the entire dataset cannot be stored in memory and historical data must eventually be forgotten. An additional problem is that the probability distributions associated with the data might change over time, a condition known as concept drift [Tsymbal, 2004]. Any reasonable stream learning algorithm must be able to recognize and cope with these situations.

Data stream mining systems must also cope with missing and corrupted data: noisy communication lines, human error, experimental design, and failing sensors can all alter and interrupt data streams. In online classification systems, both observations and labels can be missing or corrupted at any time. Noisy and missing observations have been the subject of extensive research. Observation noise is often explicitly modeled by learning procedures, and various imputation techniques have been proposed for handling missing values [Schafer, 1997], [Little and Rubin, 2002].

Several solutions have been proposed for performing online classification in the presence of concept drift. These solutions are generally based on the adaptive maintenance of some discriminating structure; those considered include sets of binary rules [Widmer and Kubat, 1996], decision trees [Hulten et al., 2001], and ensembles of classifiers [Street and Kim, 2001], [Kolter and Maloof, 2003], [Kolter and Maloof, 2005]. The streaming regression problem has been the subject of considerably
less investigation, but proposed solutions do exist [Pankratz, 1991], [Kolter and Maloof, 2005].

Considerably less research has focused on classification with missing and corrupted labels, especially in the online setting. The classification task with missing labels is called *semi-supervised learning*; see [Zhu, 2005] for an excellent survey. Nearly all proposed solutions for the semi-supervised learning problem assume stationarity and require multiple passes over the dataset, and are therefore not applicable for data streams that exhibit concept drift. Very little research has investigated classification with corrupted labels, although some theoretical work has been published [Jackson et al., 1999].

In this report, we address these shortcomings by systematically applying techniques from Bayesian probability theory to these problems. We begin with a short introduction to probability theory. After defining the problem and discussing previously proposed methods for learning from drifting data streams, we introduce the *dynamic logistic regressor* (DLR) [Penny and Roberts, 1999], [Lowne et al., 2007], a method for performing semi-supervised binary classification on non-stationary data streams. We then show how to modify the DLR to cope with possibly corrupted labels. The result is a powerful online learning algorithm that can continue to operate even with noisy or missing observations and labels.

Following the introduction of the DLR, we introduce the *dynamic characteristic curve convex hull* (DCCCH) algorithm, which provides a novel and powerful method for combining the outputs of arbitrary learning algorithms trained on a common drifting data stream [Garnett and Roberts, 2007]. The algorithm allows for a simple method of identifying the optimal learners to use at a particular instant in time, and can adapt quickly to concept drifts. The algorithm has a number of useful properties that will be discussed.

The effectiveness of these algorithms will be demonstrated through a series of experiments on real and simulated data. Finally, avenues for future research on these and other problems will be proposed and discussed.
Chapter 2

Introduction to Probability Theory

Our approach to questions regarding data streams will be chiefly motivated by probabilistic reasoning. In this chapter we give a brief introduction to the basic definitions and results of probability theory. It is by no means complete; for a more in-depth treatment see [Jaynes, 2003], [Berger, 1985], and [Halpern, 2003]. Before we begin the discussion, we will take a moment to consider what exactly we mean by “probability.”

2.1 What is Probability?

The proper meaning of “probability” is the subject of some debate. The two chief interpretations have come to be known as frequency probability and Bayesian probability.

The frequency interpretation of probability defines probability in terms of an idealized repeated experiment. The possible outcomes of an experiment are considered, and the probability of an event (which is simply a subset of the experiment’s possible outcomes) is defined as the relative frequency that event would occur during a theoretical infinite series of experimental trials.

The chief inference procedure in frequency probability is known as a hypothesis test. Given the results of an experiment, a null hypothesis is chosen, which represents the conjecture that the findings of the experiment were not unusual. The purpose of the null hypothesis is to provide a convenient method for providing hypothetical alternative experimental results. A statistical test is constructed to calculate the probability that the results of a typical experiment would be “at least as unusual” as the data set under consideration; when this probability is low, the null hypothesis is rejected with a confidence specified by the calculated probability.

The Bayesian interpretation of probability eschews the formalism of repeated experiments and instead regards a probability as measuring the “degree of belief” in a particular proposition. By
2.2 Definitions and Basic Results of Probability Theory

convention, the value 1 is chosen to represent absolute confidence in an proposition, 0 is chosen
to represent absolute confidence in its negation, and the interval (0, 1) represents a continuum of
degrees of confidence between these extremes. Probability is established as a formal mechanism
for working with these degrees of belief that obeys certain rationality requirements. Because of this
looser meaning of “probability,” the Bayesian interpretation of probability can consider propositions
that cannot be repeatedly tested by an experiment; for example, the proposition “a Democrat will win
the next presidential election in the United States” is perfectly valid in Bayesian probability, but is
ill-posed and intractable in frequency probability.

A result central to Bayesian probability is Bayes’ theorem, which provides a method for up-
dating one’s belief in a proposition after new evidence is discovered. This will be particularly useful
when dealing with concept-drifting data streams, as each newly observed datum can potentially pro-
vide valuable information for the proposition at hand; we will discuss this later.

Throughout this text, we adopt the Bayesian interpretation of probability. We provide the theo-
retical basis of this interpretation below.

2.2 Definitions and Basic Results of Probability Theory

We begin by defining a probability space. It is useful to first define a measure space.

Definition 2.2.1 (Measure space, probability space). A measure space is a tuple \((\Omega, \Sigma, \mu)\), where \(\Omega\)
is a set, \(\Sigma \subseteq \mathcal{P}(\Omega)\) is a \(\sigma\)-algebra of subsets of \(\Omega\), and \(\mu : \Sigma \rightarrow [0, \infty]\) is positive, extended real-valued
function on \(\Sigma\), called a measure, that satisfies the following conditions:

- The measure of the empty set is zero:
  \[ \mu(\emptyset) = 0. \] (2.2.1)

- The measure is countably additive: for a countable collection of pairwise disjoint sets \(\{G_i\}_{i=1}^{\infty} \subseteq \Sigma\), we have
  \[ \mu \left( \bigcup_{i=1}^{\infty} G_i \right) = \sum_{i=1}^{\infty} \mu(G_i). \] (2.2.2)

If, in addition, \(\mu(\Omega) = 1\), the measure \(\mu\) is called a probability measure and the measure space
\((\Omega, \Sigma, \mu)\) is called a probability space.
When \((\Omega, \Sigma, \mu)\) is a probability space, the set \(\Omega\) is called the *sample space* or the *set of possible worlds*; each member of \(\Omega\) is considered the potential outcome of some process. Typically, \(\Omega\) will be a discrete or countable set, a real vector space \(\mathbb{R}^n\), or a mixture of the two. The members of \(\Sigma\) are called *events*, and represent the sets of possible outcomes we can assign a probability to. When \(\Omega\) is countable, it is customary, but not required, to set \(\Sigma\) equal to the power set of \(\Omega\), \(\mathcal{P}(\Omega)\). When \(\Omega = \mathbb{R}^n\), the power set is usually too large to be of use and a restricted set, typically the Borel-measurable or Lebesgue-measurable sets, is assigned to \(\Sigma\) instead.

We will often denote probability measures with \(\Pr(\cdot)\).

From Definition 2.2.1, we may immediately deduce several useful “sum rules.”

**Theorem 2.2.2** (Sum rules of probability). If \((\Omega, \Sigma, \Pr)\) is a probability space, and \(A, B \in \Sigma\) are two events, then the following statements are true:

(i) *(the sum rule)*

\[
\Pr(A) + \Pr(\Omega \setminus A) = 1
\]

(ii) *(the extended sum rule)*

\[
\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B).
\]

### 2.2.1 Probability density functions

Given a probability space \((\Omega, \Sigma, \Pr)\), we may write the probability of an event \(A \in \Sigma\) as an integral with respect to the probability measure \(\Pr\):

\[
\mu(A) = \int_A \mathrm{d}\Pr.
\]

The measure \(\Pr\) will almost certainly not provide a simple method for integration with respect to it. For this reason, it is often useful to be able to represent probabilities in terms of integrating a function with respect to a simpler, more easily managed with measure.

If \(\Omega\) is a countable set, and \(\Sigma\) is the power set of \(\Omega\), the probability measure is completely determined by its value on singletons. For convenience, we define a function \(P : \Omega \to [0, 1]\) trivially by \(P(\omega) \triangleq \Pr(\{\omega\})\). For \(A \in \Sigma\), we may write

\[
\mu(A) = \sum_{\omega \in A} P(\omega).
\]
The function \( P \) is called the **probability mass function** of the probability space \((\Omega, \Sigma, \Pr)\).

If \( \Omega \) is the real numbers \( \mathbb{R} \), we may derive a similar construction. We begin by defining a function \( f : \mathbb{R} \to [0, 1] \) by \( f(x) \triangleq \Pr((0,x]) \). The function \( f \) is called the **cumulative distribution function** (CDF) of the probability space. When \( f \) is absolutely continuous (equivalently, when \( f \) is differentiable almost everywhere), we define a function \( P : \mathbb{R} \to \mathbb{R} \) by

\[
P \triangleq \frac{d}{dx} f(x).
\]

The function \( P \) is called the **probability density function** of the probability space and has the following useful property: given an event \( A \in \Sigma \), the probability of \( A \) is given by

\[
\Pr(A) = \int_A P(x) \, dx.
\]

It should be noted that the probability density function evaluated at \( x \) does not give the probability of the event \( \{x\} \) occurring; rather, for an infinitesimal \( dx \), the interval \([x, x + dx]\) has probability \( P(x) \, dx \). Nonetheless, we may often manipulate probability density functions as if they were *bona fide* probabilities.

There is an intimate relationship between the cumulative distribution functions on \( \mathbb{R} \) and probability measures on the Lebesgue-measurable sets in \( \mathbb{R} \); specifically, there is a one-to-one correspondence between them. The probability measure corresponding to a given CDF is said to be *induced* by the CDF.

If \( \Omega \) is a real vector space \( \mathbb{R}^n \), a probability density function can be found using a similar process as above, assuming the probability measure is absolutely continuous with respect to the Lebesgue measure.

Finally, if \( \Omega \) is a more complicated set, all is not lost. If the probability measure \( \Pr \) is absolutely continuous with respect to a more useful measure \( \mu \) on the space, the Radon-Nikodym theorem guarantees the existence of a function \( P : \Omega \to \mathbb{R} \) such that the probability of a function \( A \in \Sigma \) is given by

\[
\Pr(A) = \int_A P \, d\mu.
\]

This \( P \), when it exists, can also be called a probability density function. Of course, the probability mass and probability density functions defined above are special cases of this theorem, where the more useful measures are the counting measure for countable sets and the Lebesgue measure for \( \mathbb{R}^n \). In fact, the requirement that a cumulative distribution function be absolutely continuous is equivalent to
requiring that the probability measure it induces be absolutely continuous with respect to the Lebesgue measure.

Probability measures on \( \mathbb{R}^n \) that do not admit a probability density function are somewhat pathological, and for the remainder of the current work we will assume that probability density functions will always exist.

### 2.2.2 Random variables and expectation

A fundamental concept in probability theory is that of a \textit{random variable}.

**Definition 2.2.3 (Random variable).** Suppose \( (\Omega, \Sigma, \Pr) \) is a probability space. A \textit{random variable} is a measurable function \( X : \Omega \rightarrow S \) from \( \Omega \) into some set \( S \). The set \( S \) is called the \textit{state space}.

An important class of random variables is the class of real-valued random variables, corresponding to \( S = \mathbb{R} \). Given a probability space \( (\Omega, \Sigma, \Pr) \), a random variable \( X : \Omega \rightarrow \mathbb{R} \) induces a probability measure on the real line in a straightforward way.

It will be useful to introduce a notational device. If \( x \in \mathbb{R} \) is a real number, we will write \( \Pr( X \leq x ) \) as a shorthand for \( \Pr( \{ \omega \in \Omega \mid X(\omega) \leq x \} ) \). Similarly, we will write \( \Pr( X = x ) \), \( \Pr( X \in [0,1] ) \), etc. Now if we define a function \( f : \mathbb{R} \rightarrow \mathbb{R} \) by \( f(x) \triangleq \Pr(X \leq x) \), we will have created a valid cumulative distribution function, which corresponds to a probability measure on \( \mathbb{R} \). Assuming this CDF is absolutely continuous, there will be a probability density function corresponding to this probability measure, which we will noteate with \( P(X) \). We will often refer to \( P(X) \) simply as the \textit{distribution} of \( X \). We will write \( P(X = x) \) for the value of this function evaluated at \( x \).

In a similar fashion, if we have \( n \) real-valued random variables \( \{X_1, X_2, \ldots, X_n\} \) over a common probability space \( (\Omega, \Sigma, \Pr) \), they induce a probability measure on \( \mathbb{R}^n \). This may be accomplished by defining the function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) by

\[
f(x_1, x_2, \ldots, x_n) \triangleq \Pr(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n),
\]

creating an \( n \)-dimensional analogue of the cumulative distribution function. When this function is absolutely continuous, the function

\[
P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \triangleq \frac{\partial^n f(x_1, x_2, \ldots, x_n)}{\partial x_1 \partial x_2 \cdots \partial x_n}
\]

will be a valid probability density function on \( \mathbb{R}^n \). We will call the function \( P(X_1, X_2, \ldots, X_n) \) the \textit{joint distribution} of \( \{X_1, X_2, \ldots, X_n\} \).
Given a real-valued random variable $X$ over a discrete space $\Omega$, we may consider the sum of the possible values of $X$ weighted by their probability of occurring:

$$
\sum_{\omega \in \Omega} X(\omega) \Pr(X = \omega).
$$

This expression represents the value that can be “expected” to occur, on average, given a large number of observations of the random variable. Note that this value itself may not be likely or even attainable by the random variable.

This notion of “expected” value generalizes trivially for any real-valued random variable.

**Definition 2.2.4 (Expected value).** Suppose $(\Omega, \Sigma, \Pr)$ is a probability space and $X : \Omega \rightarrow \mathbb{R}$ is a real-valued random variable on $\Omega$. The expected value of $X$, written $\mathcal{E}(X)$, is defined by

$$
\mathcal{E}(X) \triangleq \int_{\Omega} X \, d\Pr.
$$

If $X$ has a corresponding probability density function, $P(X)$, the expected value may be calculated as

$$
\mathcal{E}(X) = \int_{-\infty}^{\infty} x P(X = x) \, dx.
$$

If $f : \mathbb{R} \rightarrow \mathbb{R}$ is measurable, the expected value of $f(X)$ is given by

$$
\mathcal{E}(f(X)) \triangleq \int_{\Omega} f(X) \, d\Pr
= \int_{-\infty}^{\infty} f(x) P(X = x) \, dx.
$$

The definition of expected value as an integral allows us to conclude some basic linearity properties.

**Theorem 2.2.5.** If $X$ and $Y$ are random variables on a common probability space, and $a \in \mathbb{R}$ is a constant, then the expected value function has the following properties:

(i) *(homogeneity)*

$$
\mathcal{E}(aX) = a \mathcal{E}(X)
$$

(ii) *(additivity)*

$$
\mathcal{E}(X + Y) = \mathcal{E}(X) + \mathcal{E}(Y).
$$
2.2 Definitions and Basic Results of Probability Theory

2.2.3 Conditional probability and Bayes’ theorem

It will often be useful to consider the probability of an event after discovering that the true state of the world lies in a restricted subset of the sample space. This type of probability is called *conditional probability*.

**Definition 2.2.6 (Conditional probability).** Given a probability space \((\Omega, \Sigma, \Pr)\) and two events \(A, B \in \Sigma\) with \(\Pr(B) > 0\), the conditional probability of \(A\) given that \(B\) is true, written \(\Pr(A \mid B)\), is defined to be

\[
\Pr(A \mid B) \triangleq \frac{\Pr(A \cap B)}{\Pr(B)}.
\]

(2.2.14)

If \(\Pr(B) = 0\), then \(\Pr(A \mid B)\) is undefined.

Conveniently, probabilities conditional on a given event provide a valid probability measure.

**Theorem 2.2.7.** If \((\Omega, \Sigma, \Pr)\) is a probability space, and \(B \in \Sigma\) with \(\Pr(B) > 0\), then the function \(\Pr(\cdot \mid B) : \Sigma \to [0, 1]\) is a probability measure on \(\Omega\).

There is a construction analogous to conditional probability for real-valued random variables over a common probability space.

**Definition 2.2.8 (Conditional probability density function).** Given a probability space \((\Omega, \Sigma, \Pr)\) and real-valued random variables \(X, Y : \Omega \to \mathbb{R}\) with probability density functions \(P(X)\) and \(P(Y)\) and with joint probability function \(P(X, Y)\), the function

\[
P(X \mid Y) \triangleq \begin{cases} 
P(X, Y) / P(Y) & P(Y) > 0; \\
0 & \text{otherwise,}
\end{cases}
\]

(2.2.15)

is called *conditional probability distribution of \(X\) given \(Y\).*

Again, the conditional probability distribution provides a valid probability density function on \(\mathbb{R}\).

**Theorem 2.2.9.** Suppose \(X\) and \(Y\) are real-valued random variables on a probability space \((\Omega, \Sigma, \Pr)\), and \(P(X \mid Y)\) is the conditional probability distribution of \(X\) given \(Y\). Let \(y \in \mathbb{R}\) be fixed with \(P(Y = y) > 0\). The function \(P(X = x \mid Y = y) : \mathbb{R} \to \mathbb{R}\) as a function of \(x\) is a valid probability density function.

We now derive a useful result from the definition of conditional probability distributions.
The significance of this result can be realized by considering a typical application. Suppose we are interested in the probability of some event $A$ and learn during the course of our investigation that the true state of the world lies in some other event $B$. Before we discovered this new evidence, we considered the probability of $A$ to be $\Pr(A)$. In this context, $\Pr(A)$ is called the prior probability of $A$, as it represents our outlook before learning the information contained in $B$. Similarly, the probability $\Pr(A \mid B)$ is called the posterior probability of $A$ after taking $B$ into account. Bayes’ theorem provides us a mechanism for updating our beliefs about a proposition after discovering new evidence.

In the context of data stream mining, we will often be interested in the probability of a certain proposition $A$ given the data observed up to a specified time. Suppose we have observed data $D = \{x_1, x_2, \ldots, x_n\}$. The prior probability $\Pr(A \mid D)$ will represent our beliefs about $A$ after observing the
data $D$. If we now observe a new datum $x_{n+1}$, we may use Bayes’ theorem to derive the posterior probability $\Pr(A \mid D \cup \{x_{n+1}\})$. We will now use this posterior probability to reason about $A$. If we observe a new datum $x_{n+2}$, the posterior probability just derived will become the prior probability in the next application of Bayes’ theorem. In this manner, we progress through the data stream, at each time step substituting the last calculated posterior probability for the prior probability. This process of continually updating our probabilities in light of new observations is called \textit{recursive Bayesian estimation}. 
Chapter 3

Problem Definition and Previous Work

Let us begin by defining our problem and establishing some notation.

3.1 The Online Semi-Supervised Learning Task

The problem we consider for the majority of this paper is the semi-supervised learning task with missing values. We observe two random variables: a *predictor* (or *independent*) variable $X$, and a *response* (or *dependent*) variable $Y$, and presume a functional relationship between the two. Let $\mathcal{X}$ and $\mathcal{Y}$ represent the ranges of $X$ and $Y$ respectively. We suppose that there is an unknown function $f : \mathcal{X} \rightarrow \mathcal{Y}$ such that $Y = f(X)$. We seek to find a good approximation to $f$ that will allow us to predict the value of $Y$ that will accompany a particular value of $X$.

When $\mathcal{Y}$ is a discrete set $\{c_1, c_2, \ldots, c_N\}$ of $N$ class labels, this task is called *classification*; when $\mathcal{Y}$ is the set of real numbers $\mathbb{R}$, the task is called *regression*. In general, the random variable $X$ takes the form of an $n$-dimensional vector $\mathbf{x} = (x_k)_{k=1}^n$. Each entry in this vector is called a *feature* and may be of any type: real, integral, ordinal, discrete, etc. For convenience, we assume $\mathcal{X} = \mathbb{R}^n$ in the remainder of this paper, but this is not strictly necessary.

The set $\mathcal{Y}$ can in general be any discrete set, but in this paper we restrict the discussion to the important *binary classification* problem, corresponding to $\mathcal{Y} = \{0, 1\}$. Classification problems with more than two classes can be effectively solved with multiple binary classifiers; for this reason, this impact of this restriction is not great. We discuss multiple methods for reducing multi-class problems to the binary case in Section 3.1.1.

In the absence of other information, we approximate the function $f$ from a set of known values $\{(x_i, y_i)\}$ drawn from $(X, Y)$; this set is called the *training set*. In the data stream case, the training set is presented as a sequence indexed by time. For convenience this sequence is usually assumed to be a
simple time series, in which case the series is indexed by the natural numbers \( \mathbb{N} \).

Our goal may be hindered by the presence of missing predictors or labels; in particular, we allow any predictive feature and any label to be missing at any time.

In many interesting cases, some of the probability distributions associated with the learning problem will change over time. We allow each of the distributions \( P(X) \), \( P(Y) \), and \( P(Y \mid X) \) to change in any way at any moment. We call such changes concept drifts [Tsymbal, 2004]. In this case, we must reformulate the semi-supervised learning task slightly. We assume that there is a sequence of functions \( \{f_t : \mathcal{X} \rightarrow \mathcal{Y}\} \) such that at a given time \( t \), \( Y = f_t(X) \). We seek to approximate this sequence of functions.

### 3.1.1 Solving multi-class classification problems with binary classifiers

The simplicity afforded by restricting the classification problem to the binary case has resulted in a great deal of research devoted to solving this particular problem. Some of these methods (such as decision trees) can be easily extended to solve problems with more than two classes; however, in many cases no natural extension is apparent [Allwein et al., 2001]. As a result, several methods have been proposed for solving multi-class classification problems with a collection of arbitrary binary classifiers. We give a brief review below; see [Allwein et al., 2001] for an excellent overview.

A naïve solution for a \( k \)-class problem is to construct \( k \) binary classification problems, one for each class. For class \( c_i \), a classifier is constructed that attempts to classify observations into the groups “class \( c_i \)” or “everything else.” The outputs of these classifiers need to be combined in some fashion; assuming that each classifier provides an estimate of the posterior class probability, \( \Pr(y = c_i) \), a natural approach would be to output

\[
\text{argmax}_i \Pr(y = c_i),
\]

that is, the class with the highest posterior probability estimated by its corresponding classifier. This method is very simple, but is far from ideal. A great deal of important information is ignored, for example, the correlation between the posterior probability estimated by each classifier and the true class.

A number of more complicated schemes have been proposed. One proposed method builds a classifier for every pair of classes [Hastie and Tibshirani, 1998]; although this often works well in practice, it requires the construction of \( \binom{k}{2} \) classifiers and therefore significantly increases the amount
of computation required.

An interesting compromise between these two extremes was proposed in [Dietterich and Bakiri, 1995]. In the proposed framework, a collection of binary classifiers are trained for different binary partitions of the available classes. By choosing these partitions carefully, an *error-correcting output code* can be constructed, whereby the outputs of the constructed classifiers can be viewed as a code-word that may be decoded according to the selected class partitions. In [Allwein et al., 2001], the authors introduce an extension to this scheme whereby the confidence of each classifier’s output is considered in the decoding step (rather than the predicted class alone), increasing performance.

In the special case where the output classes have a natural ordering, still another approach is useful [Frank and Hall, 2001]. Suppose that the classes may be ordered in such a way that

$$c_1 < c_2 < \cdots < c_{k-1} < c_k.$$  \hspace{1cm} (3.1.2)

An example application with such an ordering is a problem requiring classification into one of a variety of arbitrarily defined size classes, for example,

$$Y = \{”small,” ”medium,” ”large”\}.$$  \hspace{1cm} (3.1.3)

In this case, we may efficiently solve the problem by constructing $k - 1$ two-class models that successively evaluate the probability that the true label is less than label index $k$. We therefore evaluate for $2 \leq i \leq k$

$$Y_i \triangleq \Pr(y < c_i),$$  \hspace{1cm} (3.1.4)

which infers the cumulative density over the class posteriors. We may then form the class posteriors using

$$\begin{align*}
\Pr(y = c_1) &= Y_2, \\
\Pr(y = c_i) &= Y_{i+1} - Y_i, \quad \text{for } 2 \leq i \leq k - 1, \\
\Pr(y = c_k) &= 1 - Y_k.
\end{align*}$$  \hspace{1cm} (3.1.5)

There are still other approaches to this problem; for example, [Lee and Oh, 2003] suggests constructing a tournament of binary classifiers to solve the $k$-class problem. In any case, the restriction of our attention to the binary classification problem should not be regarded with apprehension.
3.2 Proposed Solutions for the Online Classification Task with Concept Drift

A number of previous solutions have been proposed for the task of performing online classification in the presence of concept drift. We provide a brief survey of existing methods. In general, each of these proposed methods define a particular discriminating structure for performing the classification task, then specify a method whereby this structure may be updated to adapt to changes in the system.

3.2.1 Maintaining a set of binary rules

An early approach to this problem, the FLORA family of algorithms, adopted a simple framework where collections of binary rules, or “descriptors” (of the form “feature = value,” and conjunctions of these atoms) are maintained using a straightforward algorithm [Widmer and Kubat, 1996]. Three sets of rules are constructed: ADES, for “accepted descriptors,” NDES, for “negative descriptors,” and PDES for “potential descriptors.” Every time a new observation is made, rules are added, removed, and moved between these sets as appropriate to maintain consistency with current conditions. The authors of [Widmer and Kubat, 1996] introduced several extensions of this algorithm that allow for adaptively forgetting rules, storing learned sets of rules for future reuse, etc.

The FLORA set of algorithms works well on very simple constructed problems; however, it is applicable only in the case where all input variables are discrete with a small range, a rather severe limitation. The work nonetheless remains an important early approach for the online classification problem with concept drift.

3.2.2 Maintaining a decision tree

Decision trees [Breiman et al., 1984], [Quinlan, 1993] have enjoyed enormous popularity in the classical classification setting, because they are simple to implement and evaluate, easy to interpret, and generalize fairly well. Nearly all methods for constructing decision trees require multiple passes over the entire dataset and are therefore inappropriate for the data stream setting. Additionally, most decision tree methods assume that the training data represent a random sample from a stationary distribution, and therefore cannot efficiently handle concept drifts.

In [Domingos and Hulten, 2000], the authors introduce the very fast decision tree (VFDT) algorithm for efficiently building a decision tree from a data stream. The general technique employed is
to use the data streaming in to determine the next needed split point in the decision tree. For example, the first data points seen are used to determine the split point for the root node; after determining this split, the node remains fixed and the next data points are used to grow the decision tree from this node. The algorithm employs a technique known as Hoeffding bounds [Hoeffding, 1963] to ensure probabilistic bounds on the quality of the chosen split point. These bounds can guarantee that the chosen split is the best possible (with a specified confidence), as long as enough data have been observed.

The VFDT algorithm is attractive for a number of theoretical reasons; most importantly, it can be shown that the decision tree grown by the VFDT algorithm will asymptotically approach the decision tree that would have been grown from a traditional multi-pass algorithm [Domingos and Hulten, 2000]. Unfortunately, the VFDT algorithm assumes that the data stream is free from concept drift. For this reason, the concept-adapting very fast decision tree (CVFDT) algorithm was introduced [Hulten et al., 2001]. The CVFDT algorithm represents a simple extension to the VFDT framework, wherein the purity of internal nodes are monitored for quality. If a particular node in the decision tree moves outside specified required quality bounds, the subtree rooted at that node is removed by the CVFDT algorithm, and learning commences anew from that point. This simple mechanism allows for quickly constructing a good decision tree from a data stream, even when the data undergo concept drift.

As originally introduced, the CVFDT algorithm could only handle discrete inputs; this restriction was later removed by the creators [Hulten and Domingos, 2003].

3.2.3 Maintaining an ensemble of classifiers

Ensemble learning\footnote{Not to be confused with the similarly named “ensemble learning” of variational Bayesian inference.} is an increasingly popular approach to supervised learning problems. Instead of forming a single model to explain the data, multiple models are constructed and their outputs are combined to form a single prediction. Empirical studies have shown that ensembles of learners often perform better than single learners [Dietterich, 2000]. Under certain conditions, ensembles perform provably better than a single learner, with the improvement dependent on the diversity of the ensemble [Tumer and Ghosh, 1996]. Numerous methods for combining learners have been proposed for the classical setting, including bagging [Breiman, 1996], boosting [Freund and Schapire, 1995], and stacking [Wolpert, 1990]. Unfortunately, many of these solutions are impractical in a streaming environment with concept drift. Several alternative methods for constructing ensembles in the data
stream setting have been proposed [Street and Kim, 2001], [Kolter and Maloof, 2003], [Kolter and Maloof, 2005]; however, many classical techniques remain unexplored in the streaming case.

The streaming ensemble algorithm

Street and Kim provide a simple meta-algorithm (the streaming ensemble algorithm, or SEA) for constructing and maintaining a set of arbitrary classifiers for performing classification on drifting data streams. The authors do not assume the existence of adaptive instance-based learners and instead provide a general method for constructing an ensemble using any available base learning algorithm, including classical techniques for offline learning.

To accomplish this task, the authors suggest segmenting the data stream into a series of successive fixed-size blocks. Each time a new block of data is observed, that block is used as training data for a new candidate classifier. If the ensemble has not reached a pre-defined maximum capacity, the candidate is added to the ensemble. Otherwise, the next block of data is then used to test the previously constructed candidate as well as the classifiers present in the ensemble at that time. The authors suggest an ad hoc scoring algorithm to measure the quality of each of these learners; this score is designed to encourage diversity among the ensemble members as well as reward accuracy. The classifiers with the highest scores are retained until the next next block of data is observed.

When the SEA algorithm must make a classification decision, the authors suggest simply determining the outputs of each of the learners in the ensemble, and reporting the output class picked by the majority.

Although the SEA algorithm is simple and efficient, it suffers from a number of problems. The classifiers comprising the ensemble are generated from a single block of data and never updated. If a drastic concept drift occurs, at least half the members of the ensemble will need to be replaced before the ensemble can be expected to report reasonable results. Given a large block size (which must be chosen to avoid overfitting the member classifiers of the ensemble), this might take a very long time. Additionally, if concept drifts occur more frequently than the chosen block size, the candidate classifiers will be built from disparate data and have trouble latching on to a single concept. For these reasons, most block-based approaches to the online learning task fail except under favorable conditions.
The dynamic weighted majority algorithm

The authors of [Kolter and Maloof, 2003] introduce a mechanism for maintaining an ensemble of classifiers called the dynamic weighted majority (DWM) algorithm. The method is similar to methods proposed earlier, including the weighted majority (WM) [Littlestone and Warmuth, 1994] and Winnow algorithms [Littlestone, 1988]. The DWM algorithm is quite simple. Each member of the ensemble is assigned an associated weight. When a classification must be made, the outputs of each of the ensemble members are combined using a simple weighted majority vote determined by these weights. When an ensemble member makes the wrong prediction, its weight is decreased by a multiplicative constant. Finally, when the ensemble at large makes an incorrect prediction, a new ensemble member is added with an initial weight of 1. The creators of DWM also provide simple mechanism for removing poorly performing ensemble members, although the suggested method cannot guarantee an upper bound on the number of member classifiers and is therefore not suitable for the general data stream case.

Through a series of experiments, the DWM algorithm was show to work well in practice, although its ad hoc weighting scheme could not be well motivated.

The additive expert family of algorithms

Although DWM was shown to work well in empirical tests, no theoretical results could guarantee its efficiency. Indeed, it was later discovered that DWM can perform very poorly in the worst case [Kolter and Maloof, 2005]. Nonetheless, the algorithm inspired the interesting additive expert (AddExp) series of algorithms. By making small adjustments to the DWM algorithm, the authors of [Kolter and Maloof, 2005] introduced the AddExp.D ("D" for discrete classes) algorithm for classification, and were able to prove theoretical bounds on its performance. In addition, The authors introduce the very similar AddExp.C ("C" for continuous classes) algorithm for performing online regression in the presence of concept drift. Finally, the authors suggest methods for pruning the ensemble when it grows too large, allowing for the algorithm to maintain a constant memory footprint.

Although the theoretical results regarding the AddExp family of algorithms are interesting, the performance of the algorithm is not always satisfactory, which will be demonstrated in later experiments.
Chapter 4

The Dynamic Logistic Regressor for Semi-Supervised Classification

In [Penny and Roberts, 1999] Penny and Roberts introduced the dynamic logistic regressor (DLR) for online binary classification in the presence of concept drift. Recently, Lowne, et al. have extended this framework for tackling the online semi-supervised learning problem [Lowne et al., 2007]. In this paper, we demonstrate how to extend the DLR algorithm to cope with a corrupted label stream. The result is an online method for performing nonstationary semi-supervised learning with missing data and corrupted labels.

We begin with an introduction of the DLR algorithm.

4.1 Review of the dynamic logistic regressor

We will now briefly review the dynamic logistic regressor. Suppose at time $t$ we observe data $(x_t, y_t)$ drawn from $(X, Y)$. Before observing the true label $y_t$, we wish to estimate the conditional class probability

$$z_t \triangleq \mathbb{P}(y_t = 1 | x_t).$$

(4.1.1)

Regarding $y_t$ as being drawn from a Bernoulli distribution with mean $z_t$, we can use the variance of this distribution, $z_t(1 - z_t)$, as a measure of our uncertainty about the true value of $y_t$.

The logistic regression approach to this problem takes the form of a generalized linear model; in this framework we estimate $z_t$ with

$$z_t(x_t) \triangleq g(w^T x_t'),$$

(4.1.2)
4.1 Review of the dynamic logistic regressor

where $\mathbf{w}$ is a vector of weights, $\mathbf{x}_t'$ is a vector that augments $\mathbf{x}_t$ with a constant term

$$\mathbf{x}_t' = \left\{ \mathbf{x}_t, 1 \right\},$$

(4.1.3)

and $g(\cdot)$ is the logistic function

$$g(a) = \frac{\exp(a)}{1 + \exp(a)}.$$  

(4.1.4)

The argument $\mathbf{w}^T \mathbf{x}_t'$ to $g$ is called the *activation* at time $t$. For notational convenience, we will assume that the vectors $\mathbf{x}_t$ have already been augmented with a constant feature and will drop the prime symbol for the remainder of the discussion.

The standard logistic regression model uses a fixed weight vector $\mathbf{w}$ at every time $t$, and is therefore unable to cope with concept drift. To address this problem, the DLR adjusts the model by allowing the weights to vary with time using a recursive Bayesian estimation approach. The DLR assumes that the weights are distributed according to a multivariate normal distribution, and the evolution of their distribution is tracked using the extended Kalman filter (EKF) framework [Penny and Roberts, 1999], [Lowne et al., 2007].

4.1.1 The DLR equations

In the following discussion we will adopt the standard Kalman filter notation and use the subscript $t_1 | t_2$ to indicate estimates for time $t_1$ after observing data up to and including time $t_2$.

In the DLR model, before the data point $(\mathbf{x}_t, y_t)$ arrives the weights are distributed according to the prior distribution

$$P_{t-1 | t-1}(\mathbf{w}) = N(\mathbf{w}; \bar{\mathbf{w}}_{t-1 | t-1}, \mathbf{P}_{t-1 | t-1})$$

(4.1.5)

with mean $\bar{\mathbf{w}}_{t-1 | t-1}$ and covariance $\mathbf{P}_{t-1 | t-1}$. After observing $\mathbf{x}_t$, this distribution is used to estimate the conditional class probability

$$z_t = \Pr(y_t = 1 \mid \mathbf{x}_t).$$

(4.1.6)

Finally, after observing $y_t$, the distribution is updated to obtain the posterior distribution

$$P_{t|t}(\mathbf{w}) = N(\mathbf{w}; \bar{\mathbf{w}}_{t|t}, \mathbf{P}_{t|t}),$$

(4.1.7)

which is used as the prior distribution for the next data point. In this way, the DLR progresses though the data stream recursively.
4.1 Review of the dynamic logistic regressor

The weights are assumed to progress under a dynamical system given by

\[ w_t = f(w_{t-1}, v_t), \quad (4.1.8) \]

where \( f \) is a function that encapsulates the dynamics of \( w \), and \( v_t \) is a process noise term that is assumed to be multivariate normal with zero mean and covariance \( Q_t \):

\[ v_t \sim N(0, Q_t). \quad (4.1.9) \]

For simplicity, in [Penny and Roberts, 1999] and [Lowne et al., 2007], no systematic dynamics are assumed for the weights, which reduces (4.1.8) to a diffusion given by

\[ w_t = w_{t-1} + v_t; \quad (4.1.10) \]

for the present discussion we will also make this assumption.

By assuming a multivariate normal distribution for the weights, we observe that the activation at time \( t \) is also normally distributed according to

\[ a_{t|t-1} \sim N(\bar{a}_{t|t-1}, s_{t|t-1}^2) \quad (4.1.11) \]

with

\[ \bar{a}_{t|t-1} = \bar{w}_{t|t-1}^T x_t \quad (4.1.12) \]

and

\[ s_{t|t-1}^2 = x_t^T P_{t|t-1} x_t. \quad (4.1.13) \]

This observation allows us to define the “moderated” output

\[ z_{t|t-1} = P(y_t = 1 \mid x_t) \]

\[ = \int P(y_t = 1 \mid a_{t|t-1}) P(a_{t|t-1}) da_{t|t-1} \]

\[ = \int g(a_{t|t-1}) P(a_{t|t-1}) da_{t|t-1}. \quad (4.1.14) \]

The integral in (4.1.14) cannot be evaluated analytically, but can be approximated by [MacKay, 1992]

\[ z_{t|t-1} \approx g(\kappa(s_{t|t-1})\bar{a}_{t|t-1}) \quad (4.1.15) \]

where

\[ \kappa(s_{t|t-1}) = \left(1 + \frac{\pi s_{t|t-1}^2}{8}\right)^{-\frac{1}{2}}. \quad (4.1.16) \]
The effect of this transformation is to make our prediction of the conditional class probability $P(y_t = 1 \mid x_t)$ more conservative (that is, nearer to 0.5) by an amount that is dictated by our uncertainty in the activation.

We may now write down the EKF equations for the DLR. After observing $x_t$, we predict new estimates of the weight parameters with

$$\bar{\mathbf{w}}_{t-1} = f(\bar{\mathbf{w}}_{t-1}, 0) = \bar{\mathbf{w}}_{t-1} \mid t-1$$

(4.1.17)

and

$$P_{t-1} = P_{t-1} \mid t-1 + Q_t.$$  

(4.1.18)

We use these estimates to derive the activation parameters $\bar{a}_{t \mid t-1}$ and $s^2_{t \mid t-1}$ defined in (4.1.12–4.1.13), which allow us to calculate the conditional class probability $z_{t \mid t-1}$ with (4.1.15). Finally, after observing $y_t$, we update our estimates $\bar{\mathbf{w}}_{t \mid t-1}$ and $P_{t \mid t-1}$ to derive their posteriors. This is facilitated by the definition of the Kalman gain, given by

$$K_t = \frac{P_{t \mid t-1}}{1 + z_{t \mid t-1}(1 - z_{t \mid t-1})s^2_{t \mid t-1}} x_t.$$  

(4.1.19)

The posterior estimates are now given by

$$\bar{\mathbf{w}}_{t \mid t} = \bar{\mathbf{w}}_{t \mid t-1} + K_t(y_t - z_{t \mid t-1})$$

(4.1.20)

and

$$P_{t \mid t} = P_{t \mid t-1} - z_{t \mid t-1}(1 - z_{t \mid t-1})K_t x_t^T P_{t \mid t-1}^T.$$  

(4.1.21)

This completes the specification of the DLR.

### 4.2 Notes Regarding the DLR

Let us briefly discuss various issues related to the DLR, including how to estimate the process noise parameter $Q_t$, and how to handle missing observations and labels.

#### 4.2.1 Estimating the process noise parameter $Q_t$

The discussion in the previous section does not specify how the noise parameter $Q_t$ is derived. In practice, this parameter will not be known exactly and instead must be inferred from the data. A
method for deriving a suitable $Q_t$ from the data stream is discussed at length in [Lowne et al., 2007].

The suggested approach is to utilize a time-variant, isotropic diagonal covariance matrix

$$Q_t = q_t I.$$  \hspace{1cm} (4.2.1)

At time $t = 0$, $q_t$ is initially set to 0. For $t > 0$, define

$$a_{t|t} = \bar{w}_{t|t}^T x_t,$$  \hspace{1cm} (4.2.2)

$$s_{t|t}^2 = x_t^T P_{t|t} x_t,$$  \hspace{1cm} (4.2.3)

and

$$z_{t|t} = g \left( \kappa(s_{t|t}^2) \bar{a}_{t|t} \right).$$  \hspace{1cm} (4.2.4)

That is, $z_{t|t}$ represents the conditional class probability of $y_t$ estimated by the DLR after observing $y_t$ and updating its parameters. We measure the difference in our uncertainty about $y_t$ before and after observing the true value:

$$\mathcal{J}_t = z_{t|t} (1 - z_{t|t}) - z_{t|t-1} (1 - z_{t|t-1}).$$  \hspace{1cm} (4.2.5)

We now define

$$q_{t+1} = \max(\mathcal{J}_t, 0).$$  \hspace{1cm} (4.2.6)

That is, we use a nonzero value for $q_{t+1}$ only if we suffered an increase in uncertainty about $y_t$ after observing the true value.

This approach is computationally fast and works well in practice, but is not the only solution. In [Sykacek et al., 2004], Sykacek, et al., suggest an alternative solution using a variational Bayes approach to the Kalman filter.

### 4.2.2 Handling missing observations

In [Shumway and Stoffer, 2005], Shumway and Stoffer provide a very simple method for addressing missing observations in the Kalman filter framework. The solution is to simply replace the missing features of $x_t$ with zeros. This has the effect of replacing the corresponding components of the Kalman gain in (4.1.19) with zeros, and therefore the corresponding entries of $\bar{w}_{t|t}$ and $P_{t|t}$ are not modified by the update equations (4.1.20–4.1.21).

If the fact that a particular feature is missing might be indicative of the class label (that is, the features are not missing at random (MAR); see [Little and Rubin, 2002], [Schafer, 1997] for details),
it might be helpful to augment the data $x_t$ with a vector of binary variables indicating whether each possibly missing variable was observed at time $t$.

### 4.2.3 Handling missing labels

Lowne, et al. extend the DLR algorithm to address the semi-supervised learning problem in [Lowne et al., 2007]. The suggested solution is simple. If the true class label $y_t$ is missing, we replace the missing label with the so-called “quasi-target”

$$y'_t = I(z_{t|t-1} > 0.5), \quad (4.2.7)$$

where $I(\cdot)$ is the indicator function. The DLR then proceeds as before with $y'_t$ in place of the missing true value. To offset potential errors in the imputed quasi-targets, the authors of [Lowne et al., 2007] suggest increasing the process noise parameter $q_{t+1}$ by our uncertainty about the missing true value of $y_t$:

$$q_{t+1} = \max(\mathcal{S}_t, 0) + z_{t|t-1}(1 - z_{t|t-1}). \quad (4.2.8)$$

### 4.3 Modifying the DLR Handle Corrupted Labels

In many applications, the labels cannot be expected to always represent the ground truth. The labeling process could be corrupted by a number of factors, including errors during communication and data entry. In a situation with many errors in the label stream, the DLR method described above can suffer from overcompensation and overly drastic changes to the weight parameter $w$.

To illustrate, assume the DLR has developed the ability to make reasonably confident decisions, that is, decisions with $z$ near 0 or 1. In this case, given an incorrect label $y_t$, the innovation residual in Equation 4.1.20, $(y_t - z_{t|t-1})$, would be near ±1. Correspondingly, the weights $w$ experience a very large shift, and as a result the decision boundary moves drastically. This behavior should be expected when the labels can be guaranteed to be correct; after all, if we are very confident about a decision that turns out to be incorrect, something must be wrong with our classifier. When the truth of the labels cannot be assured, however, we should tread more carefully and temper our decisions by our uncertainty in the labels. We may modify the DLR model accordingly.
4.3 Modifying the DLR Handle Corrupted Labels

4.3.1 A simple noise model and its consequences

Let us adopt a simple model for the noise present in the labeling process. For the present discussion, let $\tilde{y}_t$ represent the (possibly incorrect) observed label, and let $y_t$ represent the true, uncorrupted label.

We assume the probability that an observed label is incorrect is independent of the data observed and fixed throughout time:

$$\Pr(\tilde{y}_t \neq y_t | x_t, t) = \Pr(\tilde{y}_t \neq y_t) = \rho.$$  \hspace{1cm} (4.3.1)

This assumption may not always be true, but in many important examples (such as errors introduced by a noisy communication channel), it will be valid. In many other cases, it can serve as a useful substitute for a more complicated model. To simplify the present discussion, we assume that the probability $\rho$ is known a priori; of course, in almost any situation this will not be true. We will present an online method for estimating $\rho$ from the data stream presently.

In the presentation of the DLR, we assumed that the label $y_t$ was determined from a Bernoulli distribution with parameter $z_t$:

$$P(y_t | z_t) = z_t^{y_t}(1-z_t)^{(1-y_t)}.$$  \hspace{1cm} (4.3.2)

Under our label noise model, this assumption is violated. Instead, we may derive the distribution of $\tilde{y}_t$ given $z_t$ and $\rho$:

$$P(\tilde{y}_t | z_t, \rho) = (1 - 2\rho) \left( z_t^{\tilde{y}_t}(1-z_t)^{(1-\tilde{y}_t)} \right) + \rho.$$  \hspace{1cm} (4.3.3)

Recalling that we defined $z_t = P(y_t = 1 | x)$, the above discrepant distribution may reconciled by replacing $z_t$ with an appropriately modified output $\tilde{z}_t$, given by

$$\tilde{z}_t = (1 - 2\rho)z_t + \rho.$$  \hspace{1cm} (4.3.4)

With this definition, we now observe that

$$P(\tilde{y}_t | \tilde{z}_t) = \tilde{z}_t^{\tilde{y}_t}(1-\tilde{z}_t)^{(1-\tilde{y}_t)},$$  \hspace{1cm} (4.3.5)

that is, the original relation between the output of the DLR and the observed label is now preserved. Under the assumed noise model, the only required modification to the DLR framework is to replace the original output of the model $z_t$ (which assumes no noise in the labels) with the value $\tilde{z}_t$ given above. The definition of $\tilde{z}_t$ serves to moderate the certainty of our classifications according to our uncertainty in the labels.
4.3 Modifying the DLR Handle Corrupted Labels

4.3.2 Estimating $\rho$

As mentioned above, the true value of $\rho$ will almost certainly not be known. Fortunately, we may estimate the true value of $\rho$ online from the data. Given our assumptions, a simple calculation shows

$$\Pr(|\tilde{y} - z| > (1 - \varepsilon) | z, \rho) = [\rho + \min(z, 1 - z)] [I(\min(z, 1 - z) < \varepsilon)],$$

(4.3.6)

where $I(\cdot)$ is the indicator function. We marginalize out the nuisance parameter $z$:

$$\Pr(|\tilde{y} - z| > (1 - \varepsilon) | \rho) = \int \Pr(|\tilde{y} - z| > (1 - \varepsilon) | z, \rho) P(z) dz.$$  (4.3.7)

Applying Equation 4.3.6, we may separate this integral in a convenient manner:

$$\Pr(|\tilde{y} - z| > (1 - \varepsilon) | \rho) = \rho \int_0^\varepsilon P(z) dz + \int_0^\varepsilon \min(z, 1 - z) P(z) dz.$$  (4.3.8)

This result may be understood as separating our misclassifications

$$\Pr(|\tilde{y} - z| > (1 - \varepsilon) | \rho)$$

(4.3.9)

into the errors expected given the uncertainty in our predictions

$$\int_0^\varepsilon \min(z, 1 - z) P(z) dz$$

(4.3.10)

and the errors expected given the uncertainty present in the labeling process

$$\rho \int_0^\varepsilon P(z) dz.$$  (4.3.11)

Let us consider the significance of Equation 4.3.7. Under our assumptions, the left-hand side, $\Pr(|\tilde{y} - z| > (1 - \varepsilon) | \rho)$, can be estimated from the data without knowledge of $\rho$ by simply keeping a tally of the portion of previously observed labels $\tilde{y}$ for which $|\tilde{y} - z| > (1 - \varepsilon)$. Additionally, everything in the right-hand side (except the unknown parameter $\rho$) can be determined by keeping an online estimate of the distribution of DLR outputs $z$ assigned; a simple histogram suffices. This knowledge allows us to easily solve for $\rho$ algebraically. Thus, with very little additional storage requirements (on the order of a few hundred bytes), we may effectively solve for the unknown parameter $\rho$.

The above discussion provides a simple modification to the DLR framework that effectively handles label noise with no additional parameters and very little additional computational overhead. Even when the labels are known to be correct, the above formulation may still be effectively used; the estimated parameter $\rho$ will quickly assume a small value accordingly.
Chapter 5

Dynamic Characteristic Curves for Online Ensemble Learning

One method for combining the outputs of classifiers appeals to their receiver operating characteristic (ROC) curves [Fawcett, 2003], which provide useful summaries of a classifier’s performance. Given a set of classifiers, a useful result shows that a classifier is potentially optimal if and only if it lies on the convex hull of the collection’s ROC curves [Provost and Fawcett, 2001]. Recently an analogue of the ROC curve, called the regression-error characteristic (REC) curve, has been proposed for the regression problem, with the REC convex hull providing the same utility as the ROC convex hull. These results would be useful in the data stream setting; however, ROC and REC curves are not immediately applicable because they assign an equal weight to each training point. If the data undergo a drift in concept, the ROC and REC curves can become quickly outdated.

We propose a method for online ensemble learning using dynamic characteristic curves. We modify the ROC and REC curves to place more emphasis on a learner’s recent performance than on its historical performance. Using the convex hull of these dynamic performance curves, we can choose the appropriate learners to apply at any given point in time, even in the presence of concept drift. The result is a unified algorithm for performing both classification and regression in a streaming environment. Empirical evidence demonstrates that the proposed method performs better than several existing solutions, both on actual and simulated data. In addition, the suggested method has several useful properties. For example, if the ensemble members are built using different sets of features, we can gain insight into the relative importance of the dataset’s various features throughout time as a natural byproduct of the algorithm.
5.1 Review of ROC and REC curves

We will now briefly review ROC and REC curves and discuss their relevant properties.

Suppose we have a binary classification problem; that is, a classification problem with \( |Y| = 2 \). For convenience, let us call the two classes “positive” (or \(+\) ) and “negative” (or \(-\) ). One very general way to construct a classifier in this context is to generate a function \( s : \mathcal{X} \rightarrow \mathbb{R} \) that assigns a score to each data point \( x \) commensurate with the degree to which \( x \) is believed to belong to the positive class. A natural choice for \( s \), for example, would be \( s(x) = \Pr(Y = + | X = x) \), but other functions might be useful in the context of a particular problem. Equipped with a score function \( s \), we can choose to classify an unknown input \( x \) according to the function \( I(s(x) > \theta) \), where \( I \) is the indicator function and \( \theta \in \mathbb{R} \) is some chosen threshold. In this manner we can construct a whole family of related classifiers by simply varying the threshold \( \theta \).

For a particular choice of \( \theta \), we can measure the performance of the resulting classifier as follows. We use some method to approximate the probability distributions \( P(s(X) | Y = +) \) and \( P(s(X) | Y = -) \), the distributions of scores given to positive and negative examples, respectively. Then the value of the integral

\[
\int_{\theta}^{\infty} P(s(X) = x | Y = +) dx
\]

represents the portion of positive examples classified correctly by the classifier. Call this number \( TP_\theta \) (for “true positives”). Similarly, the value of the integral

\[
\int_{\theta}^{\infty} P(s(X) = x | Y = -) dx
\]

represents the portion of negative examples classified incorrectly. Call this number \( FP_\theta \) (for “false positives”). We use the pair \( (FP_\theta, TP_\theta) \) to summarize the performance of the classifier with the threshold \( \theta \).

If we now let the value of \( \theta \) range from \(-\infty \) to \(+\infty \) we can generate a collection of points \( \{(FP_\theta, TP_\theta)\} \). Plotting these values reveals a curve in \([0, 1]^2\) parameterized by \( \theta \). Figures 1–2 illustrate this procedure. This curve is called the receiver operating characteristic (ROC) curve. By constructing the ROC curve in this manner, we see that it will range from \((0, 0)\) (corresponding to \( \theta = -\infty \), or a classifier that always outputs \(-\)), to \((1, 1)\) (corresponding to \( \theta = +\infty \), a classifier that always outputs \(+\)). The point \((0, 1)\) is realized by a classifier that has perfect performance. The space \([0, 1]^2\) considered in this context is called \( ROC \) space.
5.1 Review of ROC and REC curves

Figure 5.1: The creation of an ROC curve. Shown are example distributions $P(s(X) \mid Y = -)$ and $P(s(X) \mid Y = +)$. The shaded areas determine FP and TP for $\theta = 2$.

Figure 5.2: The ROC curve corresponding to Figure 1. The point determined by $\theta = 2$ is circled.
The ROC curve has several useful properties. It provides a concise visual summary of a classifier’s performance and can facilitate the comparison of multiple classifiers. The area under the curve (AUC) statistic, which is equivalent to the Wilcoxon–Mann–Whitney statistic [Hanley and McNeil, 1982], has a special interpretation: it represents the probability that a randomly chosen positive example will be assigned a higher score by \( s \) than a randomly chosen negative example. The AUC statistic is often used to further summarize the performance of a classifier.

For a set of classifiers \( \{C_i\} \), an important concept is the ROC convex hull (ROCCH). Suppose that we have generated ROC curves for each of the classifiers \( C_i \). The upper convex hull of these curves in ROC space is the ROC convex hull. The classifiers in \( \{C_i\} \) may be interpolated to achieve any performance point on the ROCCH [Fawcett, 2003]. It is clear that the AUC of the ROCCH is at least as large as the largest AUC among the \( \{C_i\} \), making the ROCCH an intuitively beneficial concept. Further, it can be shown that the classifiers that lie on the ROCCH are optimal in the following sense. Suppose that associated with our problem we have a function \( c : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+ \) that specifies the cost of misclassifying a particular data instance. Further, let \( P(+) \) represent the probability that a random instance is from the positive set. Then, given any choice for the cost function \( c \) and any value of \( P(+) \), there is at least one classifier among the \( C_i \) that minimizes the expected cost under these conditions, and every such classifier lies on the ROC convex hull [Provost and Fawcett, 2001]. This result facilitates the identification and selection of the potentially optimal classifiers among a set of candidates.

The ROC curve and the ROC convex hull can be generalized to classification problems with more than two classes [Srinivasan, 1999]. We focus on the binary classification problem for the remainder of the text because it simplifies the presentation and because problems with more than two classes can be solved using multiple binary classifiers. See [Lee and Oh, 2003] for a recently proposed scheme for doing so.

Recently, the regression-error characteristic (REC) curve has been introduced as an analogue of the ROC curve for the regression case [Bi and Bennett, 2003]. Suppose we have a regression problem, and that we have generated a regression function \( g : \mathcal{X} \rightarrow \mathbb{R} \) as a potential solution. Suppose further that associated with our problem we have a function \( c : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+ \) that measures the cost of predicting an incorrect value for \( Y \). If \( (x,y) \) is a particular value drawn from \( (X,Y) \), common choices for \( c \) include the squared error

\[
c(g(x), y) = (g(x) - y)^2
\]  
(5.1.3)
and absolute deviation
\[ c(g(x), y) = |g(x) - y|. \]  
(5.1.4)

To build the REC curve, we generate an estimate of the probability distribution \( P(c(g(X), Y)) \). The regression-error characteristic curve is then the cumulative distribution function of this distribution.

REC curves have many properties analogous to ROC curves. They provide concise visual summaries of a regressor’s performance and can facilitate the comparison of different regressors. Just as the AUC statistic for ROC curves has a special interpretation, the area over the curve (AOC) of an REC curve is a biased estimator for the regressor’s expected cost [Bi and Bennett, 2003].

Suppose that we have a set of regressors \( \{R_i\} \) and have generated the REC curves for each of the collection’s members. Bi and Bennet suggest using the upper convex hull of the REC curves to generate ensembles of regressors [Bi and Bennett, 2003]. We will call this upper convex hull the REC convex hull (RECCH). Intuitively, the AOC of the RECCH will be at least as small as the smallest AOC among the \( \{R_i\} \).

We will refer to ROC and REC curves collectively as characteristic curves.

### 5.2 Dynamic Characteristic Curves

It is clear that the ROCCH and RECCH are very useful for identifying potentially optimal learners among a collection of candidates. For this reason, it is reasonable to believe that they might be useful for guiding ensemble learning on data streams. Unfortunately, when the data stream is subject to concept drift, the characteristic curve loses some of its utility.

#### 5.2.1 Characteristic curves cannot cope with concept drift

ROC and REC curves implicitly assume that the samples \( \{x_i, y_i\} \) used to build them are drawn from a stationary joint distribution \( P(X, Y) \). This assumption becomes invalidated if concept drift occurs. To illustrate how this could interfere with online learning, we present a hypothetical example.

Suppose that we have two learners built from a common large data stream. Imagine that the first predicted the first 99% of the data points very well, but predicted the most recent 1% very poorly. Conversely, imagine that the second predicted the first 99% very poorly, but predicted the most recent 1% very well.

Suppose that we were given the task of choosing a learner to predict the response of the next
data point. If we built characteristic curves for these two learners, the first learner would appear to be overwhelmingly more preferable than the second. Considering the situation, however, the second learner is probably the one to be preferred. It seems reasonable to believe that a concept drift might have occurred at the 99% mark, incapacitating the first learner. The recent performance of the second learner, however, has been exemplary. Because the characteristic curves assign an equal weight to each point in the data stream, this nuance is masked.

This example illustrates that when comparing learners in the presence of concept drift, a learner’s most recent performance is more important than its historical performance. Standard characteristic curves cannot make this distinction. We describe a simple solution to this problem in the next subsection.

5.2.2 Dynamically weighting performance

We seek to modify standard characteristic curves to operate on data streams so that they place more emphasis on recent performance. Any solution to this problem must further be compatible with the operating requirements of an online environment. The potentially huge size of the data stream, for example, prohibits the storage of the data seen thus far.

In the data stream setting, we observe data in a time-indexed sequence \{(x_t, y_t)\}, which we use to train a learner \(L\). Our goal is to construct a sequence of characteristic curves for \(L\) indexed identically by time.

To clarify the following discussion and to make the introduced method more general, we make a slight modification to the data stream learning problem. In some online systems, it might not be desired (or even possible) to update the system after observing each data point. Instead, the system might update after observing a certain number of data points, or after a fixed period of time. For this reason, at a particular time \(t\), we allow the system to observe a sample (of any size) from \((X_t, Y_t)\). When the observed samples are single points \((x_t, y_t)\), the problem is identical to the one presented above.

From Section 5.1, we observe that all the information associated with a characteristic curve is contained within estimated probability density functions: an ROC curve is completely described by the distributions \(P(s(X) \mid Y = +)\) and \(P(s(X) \mid Y = -)\), and an REC curve is completely described by the distribution \(P(c(g(X), Y))\). When building a sequence of characteristic curves, these are the only structures that need to be maintained. We construct our dynamic characteristic curves by updating
these distributions appropriately.

Suppose that at time \( t - 1 \) we have generated an estimate of some univariate probability density function \( P : \mathbb{R} \rightarrow \mathbb{R} \); call this estimate \( P_{t-1}(\cdot) \). Suppose that at time \( t \) we observe a new sample \( S_t = \{x_i\} \) and wish to update our density estimate in light of the data in \( S_t \). To proceed we build an estimate of the probability density function of the data in \( S_t \); let \( \bar{P}_t(\cdot) \) represent this estimate. If \( S_t \) is a single data point \( x_t \), for example, a reasonable choice might be

\[
\bar{P}_t(x) = \delta(x - x_t),
\]

where \( \delta(\cdot) \) is the Dirac delta distribution. If \( S_t \) is a larger sample, any number of techniques could be used, including histograms, mixture models, kernel density estimation, etc.; see [Silverman, 1986] for an overview. Once \( \bar{P}_t(\cdot) \) has been constructed, we generate our estimate of \( P(\cdot) \) at time \( t \) according to

\[
P_t(\cdot) = \alpha P_{t-1}(\cdot) + (1 - \alpha)\bar{P}_t(\cdot),
\]

where \( \alpha \in (0, 1) \) is a parameter that determines how much emphasis to place on the historical estimate \( P_{t-1}(\cdot) \).

The method of updating probability density estimates described in the preceding paragraph has been suggested in other contexts [Zhou et al., 2003], [Cortes and Pregibon, 2001]. The update scheme can be understood as an analogue of the exponentially weighted moving average (EWMA) for probability density estimates. The estimate \( P_t(\cdot) \) can be rewritten as a weighted sum of all of the estimates \( \{\bar{P}_i\}_{i=1}^t \):

\[
P_t(\cdot) = \sum_{i=1}^t w_i \bar{P}_i(\cdot),
\]

where

\[
w_i = (1 - \alpha)\alpha^{t-i}.
\]

The weights therefore drop off exponentially with time, placing more weight on the data observed in the most recent samples. The parameter \( \alpha \) controls the rate of this drop off. Because of the special nature of the weights \( w_i \), we only need to know the previous estimate \( P_{t-1}(\cdot) \) and the estimate built from the current sample \( \bar{P}_t(\cdot) \) to generate the estimate at time \( t \). We can therefore generate the sequence of estimates \( \{P_t\} \) within a fixed amount of memory; the previous estimates and data points do not need to be stored.

Time series typically have a characteristic time scale associated with their progression; when our prior knowledge of the dataset includes this information, we can use it to guide our choice of \( \alpha \).
5.3 Using Dynamic Characteristic Curves for Online Ensemble Learning

Natural choices for $\alpha$ will proscribe a weighting scheme that is on the order of a time series’s natural time scale.

To create dynamic characteristic curves with the desired behavior, we simply create dynamic estimates of the required probability distributions, and build characteristic curves from those distributions as needed. For example, suppose we have a regression problem and at time $t$ we observe a new data point $x_t$. At time $t-1$, we will have built the estimate

$$P_{t-1}(c(g(X), Y)),$$  \hspace{1cm} (5.2.5)

and if desired, we can use this estimate to generate a REC curve. The generated curve will represent the behavior of the regressor up to time $t-1$, emphasizing its most recent performance. As usual, we predict the response variable with $g_t(x_t)$. If is then revealed that the actual response was $y_t$, we generate the estimate

$$\bar{P}_t(x) = \delta(x - c(g_t(x_t), y_t))$$  \hspace{1cm} (5.2.6)

and update our estimate of $P(c(g(X), Y))$:

$$P_t(c(g(X), Y)) = \alpha P_{t-1}(c(g(X), Y)) + (1 - \alpha) \bar{P}_t(\cdot).$$  \hspace{1cm} (5.2.7)

5.3 Using Dynamic Characteristic Curves for Online Ensemble Learning

The dynamic characteristic curves introduced in the preceding section may be used as a part of any online learning algorithm. In this section we describe a method for performing online ensemble learning using convex hulls of dynamic characteristic curves. After the development in the previous sections, this task will be relatively straightforward.

5.3.1 Learning with dynamic characteristic curve convex hulls

As described in Section 5.1, convex hulls of characteristic curves are useful for identifying potentially optimal learners among a collection of candidates. We exploit this fact. Suppose we have an online supervised learning task (either classification or regression). We generate a collection of candidate learners $\{L_i\}$ and allow them to observe and learn from the data stream. The learners $\{L_i\}$ may be of any type; we only require that each learner $L_i$ provide a sequence of functions $g_t : \mathcal{X} \to \mathcal{Y}$ that allow us to predict the response $Y$ given an observation of the predictor variable $X$ at time $t$. The
5.3 Using Dynamic Characteristic Curves for Online Ensemble Learning

learners may update themselves after observing each instance or after observing a batch of data; the
definitions given in the preceding section allow either case.

As we progress through the data stream, we generate dynamic characteristic curves for each of
the learners \( \{L_i\} \). Suppose now that at time \( t+1 \) we are presented with an observation \( x_{t+1} \) of the
independent variable \( X \). We wish to make a global prediction of the response from the predictions of
the candidate learners.

The solution is simple. For each learner in \( \{L_i\} \), we have an associated dynamic characteristic
curve that summarizes the learner’s performance up to time \( t \). We find the convex hull of these
characteristic curves, identify the learners that lie on the convex hull, and combine the predictions of
those learners to form our global prediction.

This combination can be done in any number of ways. Let \( CH_t \subseteq \{L_i\} \) represent the set of
learners on the dynamic characteristic curve convex hull at time \( t \). We can combine their outputs
using a simple unweighted combination (such as majority voting for classification or the mean of
their outputs for regression), or using a weighted combination (such as weighted majority voting
or a weighted mean of their outputs). These weights could be anything; possible choices from this
framework include the AUC statistics for the associated characteristic curves (which is appropriate
for both classification and regression) or the portion of the arc length along the characteristic curve
convex hull contributed by the associated learner.

In the classification case, we propose another possibility. Suppose that in addition to the distrib-
utions required to calculate the dynamic ROC curves, we have also kept an online estimate of the
distribution \( P(+) \), giving the probability of observing a positive example. This estimate can be made
in the manner described in the previous section. Recall that each classifier \( C \in CH_t \) has an associ-
ated time-dependent function \( C.s_t : \mathcal{X} \rightarrow \mathbb{R} \) that assigns a score to an unknown value \( x \) according
to the belief that \( x \) belongs to the positive class. Let \( s_{t+1} \) represent the scores assigned to \( x_{t+1} \) by
the classifiers in \( CH_t \). We may estimate \( \Pr[Y = + | X = x_{t+1}] \) based on these scores by appealing to
Bayes’ theorem. Specifically, each learner \( C \) on the convex hull contains a summary of the distribu-
tion \( P(C.s(X) | Y = +) \). Given a the particular scores assigned by the learners at time \( t \), we may use
these distributions directly to synthesize a global estimate of the desired probability. We first apply
Bayes’ theorem directly:

\[
\Pr[Y = + | \tilde{s}_{t+1}, t] = \frac{P_t(\tilde{s}_{t+1} | Y = +)P_t(+)}{P_t(\tilde{s}_{t+1})}. \tag{5.3.1}
\]
If we assume that the score functions are conditionally independent, we may decompose the density functions on the right-hand side:

\[
P_t(\tilde{s}_{t+1} \mid Y = +) = \prod_{C \in CH_t} P_t(C.s_t(x_{t+1}) \mid Y = +) \tag{5.3.2}
\]

and

\[
P_t(\tilde{s}_{t+1}) = \prod_{C \in CH_t} P_t(C.s_t(x_{t+1})) \tag{5.3.3}
\]

We will have generated online estimates of each of the probability distributions involved in (5.3.1–5.3.3) in the course of generating the dynamic characteristic curves.

We will call the algorithm described above the dynamic characteristic curve convex hull (DCCCH) method. Pseudocode for the DCCCH algorithm appears in Algorithm 1. As written, the pseudocode assumes that the learners’ convex hulls are updated after every time step. To work with batch learners, UPDATE-LEARNER would simply have to be called less often. The possibilities for COMBINE-PREDICTIONS are detailed above.

### 5.3.2 Observations

We make a number of observations about the framework suggested above. It is interesting to consider that the set of learners that lie on the dynamic characteristic curve convex hull at a particular time might give us additional insight into the learning task. For example, a common method for encouraging diversity between ensemble members is to build each component learner from a different subset of the available features. Random forests, for example, is one common method that employs this approach [Breiman, 2001]. Suppose that we have adapted this strategy and that each learner in \( \{L_i\} \) has been trained using a different subset of the available features. We can now gain insight into the relative importance of each feature at a particular time \( t \) by noting how many times each feature appears among the learners in \( CH_t \). If the learners in \( \{L_i\} \) include an internal mechanism for measuring feature importance, this information can be used to enhance our estimate. By generating such estimates as the data stream progresses, we can gain an understanding of how the features important for prediction change with time. This feature evaluation in an integral part of the algorithm and the required excess computation is trivial. We demonstrate this method in the next section.

Finally, we observe that the DCCCH algorithm is not only independent of the learning algorithms used, but is also independent of which learners are present in the ensemble at any given time. The algorithm could be combined with a mechanism for introducing new ensemble members
5.3 Using Dynamic Characteristic Curves for Online Ensemble Learning

DYNAMIC CHARACTERISTIC CURVE CONVEX HULL(α)

1 for t ← 1 to end
2 do $CH_{t-1} ← CONVEX-HULL(\{L_i\}_{t-1})$
3 $\hat{y}_t ← COMBINE-PREDICTIONS(CH_{t-1}, t)$
4 output $\hat{y}_t$ as prediction of $y_t$
5 $y_t ←$ actual value
6 for $L ∈ \{L_i\}$
7 do $L ←$ UPDATE-LEARNER($L, t, \alpha, y_t$)

COMBINE-PREDICTIONS($CH, t$)

1 for $i ← 1$ to $|CH|$
2 do predictions[$i$] ← $L_t(x_t)$
3 return unspecified combination of predictions

UPDATE-LEARNER($L, t, \alpha, y_t$)

1 if classification task
2 then if $y_t = +$
3 then $L.P_t(s(X) \mid Y = +) ←$
4 $\alpha L.P_{t-1}(s(X) \mid Y = +) + (1 - \alpha)(\delta(x - L.s(x_t)))$
5 else $L.P_t(s(X) \mid Y = -) ←$
6 $\alpha L.P_{t-1}(s(X) \mid Y = -) + (1 - \alpha)(\delta(x - L.s(x_t)))$
7 else $L.P_t(c(g(X), Y)) ← \alpha L.P_{t-1}(c(g(X), I)) + (1 - \alpha)(\delta(x - c(L.g(x_t), y_t)))$
8 return $L$

Algorithm 1: Pseudocode for the DCCCH algorithm.
or removing old ensemble members. In the next section, we test an implementation of the DCCCH algorithm with a simple ensemble maintenance mechanism.
Chapter 6

Applications

In this chapter we provide empirical results to establish the usefulness of the methods introduced in previous chapters. We will use a variety of datasets including real and simulated data.

6.1 Results Regarding the Dynamic Logistic Regressor

We begin by demonstrating the effectiveness of the DLR for non-stationary dynamic classification and the method described above for handling corrupted label streams.

6.1.1 Initialization

In all the experiments reported here, we initialized all components of \( \mathbf{w} \) to zero and both \( P \) and \( Q \) to the identity matrix. All components of the observed data were normalized to have mean 0 and variance 1.

6.1.2 An illustrative example

This example is extracted from [Lowne et al., 2007].

As a simple example of a drifting non-stationary system, we consider two overlapping Gaussian distributions rotating in a circular fashion around a central point at (0,0), with the two distributions out of phase by \( \pi \) radians. The Bayes error was 4%. We wish to adapt a discriminant boundary that captures the non-stationarity in the system without relying on constant feedback. Figure 6.1 shows snapshots of the decision boundary and the last 50 data points for \( t = 50 \) and 300. When presented with a full set of labels \( y \), the DLR model achieved a performance of near 96%, that is, near the Bayes error. A static classifier trained on the same data achieved only 50% accuracy due to the rotating nature of the problem.
6.1 Results Regarding the Dynamic Logistic Regressor

Figure 6.1: The decision boundary and last 50 data points observed at $t = 50$ (left) and $t = 300$ (right) samples in the rotating Gaussian dataset. 50% of labels were observed. Circles and crosses denote the true class labels.

Figure 6.2: The mean accuracy as a fraction of correct classifications with one standard deviation error bars. The top curve is on a two-class problem with 4% Bayes error and the lower curve on a noisy version with 22% Bayes error.

We then successively removed label information. The performance with 50% labeling was hardly degraded, achieving 95% accuracy; with only 20% class labels, the classifier still achieved a 91% accuracy.

The performance of the classifier was further investigated on this data with variation in fractions of observed labels from 0% to 100%. Figure 6.2 shows the average performance of the algorithm over 10 runs; the variation in performance is very robust to large numbers of unobserved labels.

The effect of label error inference

The effect of label errors becomes increasingly important in decision problems with high Bayes error (that is, intrinsic overlap between classes). This error may be due to incorrect labeling or to noise in the observation space, meaning that classes are highly overlapping. In either case we do not want to
adapt the dynamic model in cases where it is believed that a datum has crossed a decision boundary due to noise. Figure 6.3 shows the evolution of the inferred target label noise for the simple rotating system previously considered. The Bayes error is 22%. The parameter $\rho$ was set to 0.1 for the first 100 data samples, and then was updated adaptively as described in Section 4.3.2. Figure 6.4 shows the effect of taking this into account in our algorithm. Plots (c) and (d) respectively show the maximum posterior probability with no inferred label noise (c) and taking it into account (d). We see that the posterior probability trace is reduced in the latter plot indicating the algorithm has taken this extra uncertainty into account. Plots (a) and (b) show the parameters $w_t$ for the two cases. We note that the plot in (b) follows the rotating boundary considerably better.

We note that in this case the label noise independence condition given in Equation 4.3.1 is violated; nonetheless, the adaptive procedure given for predicting the label noise parameter $\rho$ works very well.

### 6.2 Results Regarding the DCCCH Algorithm

We conducted several experiments to evaluate the methods presented in Chapter 5, using a synthetic classification problem an actual regression problem. Both data streams exhibited concept drift. In each case we compared the dynamic characteristic curve convex hull algorithm with other online learning methods. The results were universally favorable for the new method. We give details about
Figure 6.4: Effect of label errors: (a) parameters without label noise inference and (b) parameters after inference of label noise. (c) maximum posterior without label error inference. (d) maximum posterior taking label noise into account.
the various experiments and their results below.

6.2.1 Experiment 1: streaming classification with concept drift

To test the algorithm’s ability to perform classification on non-stationary data streams, we subjected it to a simulated dataset based on one previously presented for this purpose, but modified to be more difficult. The dataset, which was first presented in [Street and Kim, 2001] to test the streaming ensemble algorithm (SEA), is a binary classification problem with 60,000 data points. A similar dataset was used in [Kolter and Maloof, 2005] to test the AddExp family of algorithms. Each feature is independent and identically distributed with distribution $U[0, 10]$. In the original dataset there were three features, but we use up to 200. In each case we select a set of features $\{x_i\}_{i \in I}$ which we use to derive the response. The class label is completely determined by the mean value of the selected features according to the function $I(\text{mean}_{i \in I}(x_i) < \xi)$ for some value $\xi$. In the original dataset four concept drifts were introduced by periodically modifying the value of $\xi$. To increase the problem’s difficulty, we introduce many more. Every 500 data points we randomly select a new value for $\xi$ from the distribution $U[3.5, 6.5]$. We also increase the problem’s difficulty by choosing a new feature subset $\{x_i\}_{i \in I}$ every 6,000 data points. Finally, label noise is introduced by replacing 10% of the responses with values selected from a Bernoulli distribution with parameter $p = 1/2$.

We implemented the dynamic characteristic curve convex hull algorithm using an ensemble of size 25. Each member of the ensemble was a dynamic logistic regressor. The DLR algorithm is already quite good at handling concept drift, making it an ideal candidate for inclusion in the DCCCH algorithm. Each DLR was built using a random subset of the features available, where each feature was included with probability 0.25. The DLR classifier directly estimates $\Pr(Y = + | X = x)$; this is the score function $s$ used to generate the dynamic ROC curves. For each classifier, we generated online estimates of the density functions $P(s(X) | Y = +)$ and $P(s(X) | Y = -)$ using (5.2.2) with $\alpha = 0.99$ \footnote{With $\alpha = 0.99$, the previous 230 data points account for 90% of the total available weight.}. The functions $\bar{P}_t(\cdot)$ were chosen to make these density estimates histograms over $[0, 1]$ with 50 equal-width bins. To classify a data point $x_t$, we found the classifiers on the convex hull at the last time step $CH_{t-1}$, and combined their outputs into a global response using (5.3.1–5.3.3).

Our implementation of the DCCCH algorithm used a simple ensemble maintenance mechanism. If the ensemble produced an incorrect response at time $t$, the learner with the worst recent performance (as measured by the area under the ROC curve) was replaced with a new DLR at time $t + 1$. This is
Table 6.1: The mean portion of points in the synthetic classification dataset classified incorrectly using different methods (standard deviation times $10^3$ in parentheses).

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of relevant features</th>
<th>Number of features</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDLR</td>
<td>2</td>
<td>50</td>
<td>0.221 (3.23)</td>
<td>0.253 (3.25)</td>
<td>0.273 (3.43)</td>
<td>0.296 (4.54)</td>
</tr>
<tr>
<td>CVFDT</td>
<td>2</td>
<td>100</td>
<td>0.443 (10.8)</td>
<td>0.451 (10.3)</td>
<td>0.456 (7.60)</td>
<td>0.456 (8.70)</td>
</tr>
<tr>
<td>SEA</td>
<td>2</td>
<td>150</td>
<td>0.453 (11.7)</td>
<td>0.455 (8.23)</td>
<td>0.458 (7.76)</td>
<td>0.461 (5.28)</td>
</tr>
<tr>
<td>MV</td>
<td>2</td>
<td>200</td>
<td>0.320 (8.56)</td>
<td>0.326 (10.2)</td>
<td>0.330 (8.20)</td>
<td>0.331 (10.0)</td>
</tr>
<tr>
<td>EWMV</td>
<td>2</td>
<td>50</td>
<td>0.291 (9.33)</td>
<td>0.301 (10.8)</td>
<td>0.307 (8.88)</td>
<td>0.308 (11.6)</td>
</tr>
<tr>
<td>AddExp.D</td>
<td>2</td>
<td>100</td>
<td>0.156 (3.70)</td>
<td>0.195 (2.95)</td>
<td>0.214 (3.77)</td>
<td>0.229 (3.04)</td>
</tr>
<tr>
<td>DCCCH</td>
<td>2</td>
<td>150</td>
<td><strong>0.142</strong> (4.50)</td>
<td><strong>0.161</strong> (3.36)</td>
<td><strong>0.171</strong> (4.32)</td>
<td><strong>0.180</strong> (5.92)</td>
</tr>
<tr>
<td>SDLR</td>
<td>10</td>
<td>50</td>
<td>0.225 (1.95)</td>
<td>0.264 (3.47)</td>
<td>0.293 (3.71)</td>
<td>0.329 (4.39)</td>
</tr>
<tr>
<td>CVFDT</td>
<td>10</td>
<td>100</td>
<td>0.447 (9.40)</td>
<td>0.453 (9.93)</td>
<td>0.456 (12.7)</td>
<td>0.458 (7.56)</td>
</tr>
<tr>
<td>SEA</td>
<td>10</td>
<td>150</td>
<td>0.483 (14.2)</td>
<td>0.488 (11.7)</td>
<td>0.491 (11.9)$^a$</td>
<td>0.493 (11.5)$^a$</td>
</tr>
<tr>
<td>MV</td>
<td>10</td>
<td>200</td>
<td>0.242 (6.49)</td>
<td>0.250 (7.31)</td>
<td>0.258 (9.00)</td>
<td>0.263 (7.68)</td>
</tr>
<tr>
<td>EWMV</td>
<td>10</td>
<td>50</td>
<td>0.238 (6.00)</td>
<td>0.246 (6.80)</td>
<td>0.254 (8.61)</td>
<td>0.259 (7.39)</td>
</tr>
<tr>
<td>AddExp.D</td>
<td>10</td>
<td>100</td>
<td>0.266 (4.87)</td>
<td>0.292 (7.25)</td>
<td>0.313 (7.55)</td>
<td>0.327 (6.51)</td>
</tr>
<tr>
<td>DCCCH</td>
<td>10</td>
<td>150</td>
<td><strong>0.219</strong> (4.66)</td>
<td><strong>0.226</strong> (4.81)</td>
<td><strong>0.232</strong> (6.49)</td>
<td><strong>0.237</strong> (5.58)</td>
</tr>
<tr>
<td>SDLR</td>
<td>25</td>
<td>50</td>
<td>0.208 (3.78)</td>
<td>0.249 (5.40)</td>
<td>0.282 (6.29)</td>
<td>0.325 (8.01)</td>
</tr>
<tr>
<td>CVFDT</td>
<td>25</td>
<td>100</td>
<td>0.433 (11.7)</td>
<td>0.441 (13.4)$^a$</td>
<td>0.443 (4.20)$^a$</td>
<td>did not run$^a$</td>
</tr>
<tr>
<td>SEA</td>
<td>25</td>
<td>150</td>
<td>0.492 (23.5)</td>
<td>0.495 (20.1)</td>
<td>0.502 (16.6)</td>
<td>0.495 (19.9)</td>
</tr>
<tr>
<td>MV</td>
<td>25</td>
<td>200</td>
<td>0.187 (7.23)</td>
<td>0.191 (8.35)</td>
<td>0.199 (8.66)</td>
<td>0.203 (9.28)</td>
</tr>
<tr>
<td>EWMV</td>
<td>25</td>
<td>50</td>
<td>0.185 (7.09)</td>
<td>0.189 (8.05)</td>
<td>0.196 (8.43)</td>
<td>0.200 (9.05)</td>
</tr>
<tr>
<td>AddExp.D</td>
<td>25</td>
<td>100</td>
<td>0.217 (6.33)</td>
<td>0.236 (9.93)</td>
<td>0.252 (9.63)</td>
<td>0.263 (9.87)</td>
</tr>
<tr>
<td>DCCCH</td>
<td>25</td>
<td>150</td>
<td><strong>0.182</strong> (6.04)</td>
<td><strong>0.186</strong> (7.30)</td>
<td><strong>0.191</strong> (7.90)</td>
<td><strong>0.193</strong> (7.55)</td>
</tr>
</tbody>
</table>

$^a$ The reference CVFDT implementation exited prematurely with a segmentation fault for some or all of these datasets.
6.2 Results Regarding the DCCCH Algorithm

the same mechanism used by the AddExp.D algorithm [Kolter and Maloof, 2005].

Table 1 shows the results of testing this implementation of the DCCCH method and several other algorithms on the constructed dataset. The other methods tested were: a single DLR built using the entire dataset (SDLR), the concept-varying very-fast decision tree algorithm (CVFDT) [Hulten et al., 2001], the SEA algorithm [Street and Kim, 2001] using classification and regression trees (CART) [Breiman et al., 1984] as the base learner, and the AddExp.D algorithm [Kolter and Maloof, 2005] using DLRs as the base learner. We also tested combining the outputs of the ensemble members using unweighted and error-weighted majority voting (MV and EWMV, respectively).

The Very Fast Machine Learning (VFML) toolkit [Hulten and Domingos, 2003] supplied the code for the CVFDT algorithm. The CVFDT implementation provided in the toolkit has been extended by the VFML creators to handle continuous data, a feature not provided by the original published specification for the CVFDT algorithm. Because of this ability, the dataset did not need to be discretized for CVFDT. The VFML default parameters were used. The default parameters for CVFDT were used, except the allowed chance for error in each decision was increased from 1% to 20%, which improved its performance slightly.

The SEA algorithm was implemented according to the specification in [Street and Kim, 2001]. The ensemble size was fixed at 25.

The AddExp.D algorithm was implemented according to the specification in [Kolter and Maloof, 2005]. The \( \beta \) and \( \gamma \) parameters were set at 0.5 and 0.1, respectively. The ensemble size was fixed at 25. When the ensemble grew larger than this, the worst-performing learners (according to the weights assigned by the AddExp.D algorithm) were removed from the ensemble. The base learners were DLRs built from random subsets of the available features, where each feature was chosen with probability 0.25; these are the same learners used for the DCCCH algorithm.

To eliminate problems caused by random fluctuations in the generated datasets, twenty-five datasets were created for each (number of features, number of relevant features) pair, and the performance of each algorithm was determined from its aggregate performance on these datasets.

In every case, the dynamic ROC curve algorithm outperformed every other method tested. In most cases the degree of this improvement increased with the number of extraneous features. This can be explained as a benefit of building the ensemble members using random feature subsets. Although the majority voting schemes were built from the same ensembles, they were not able to effectively ignore the poorly performing classifiers. The dynamic convex hull algorithm, on the other hand, was
able to quickly identify those classifiers that were potentially optimal among the candidates, and base its prediction on their input alone.

In a few cases, it is not immediately clear that the DCCCH algorithm provides a significant improvement over other methods. In particular, with 25 relevant features, the DCCCH method appears to be very close to the error-weighted majority voting algorithm. If you compare their performance on each of the 25 individual datasets, however, the DCCCH algorithm almost invariably performs better. Even in the closest case (comparing EWMV and DCCCH with 50 features and 2 relevant features), a Wilcoxon signed-rank test rejects the hypothesis that the distribution of differences in performance has zero median with a \( p \)-value of \( 1.13 \times 10^{-4} \). In almost every other case, this \( p \)-value decreases by an order of magnitude.

It is clear that the DCCCH method can perform classification on highly non-stationary data streams very well.

### 6.2.2 Experiment 2: streaming regression with concept drift

Next we verified that the DCCCH method can perform online regression in the presence of concept drift. We tested the algorithm on an actual dataset gathered from inter-currency trading information throughout twenty-five full trading days. Financial markets naturally exhibit constant concept drift; if it weren’t for this drift, the authors would be much richer!

Each dataset consists of 5,760 points, containing information about five inter-currency rates (EUR–CHF, EUR–USD, USD–JPY, EUR–JPY, and USD–CHF). For each of these rates, each data point includes the ask and bid prices, the mid-price, defined as

\[
mp(t) = \frac{\text{ask}(t) + \text{bid}(t)}{2},
\]  
(6.2.1)

the order flow on the ask and bid sides of the book, and the return, defined as

\[
\frac{mp(t) - mp(t - T)}{mp(t - T)}.
\]  
(6.2.2)

For these data, the time scale for the return \( T \) is equal to 10 time steps. The linearly dependent mid-prices were intentionally included as redundant features.

From these data we constructed five different regression problems. For each trading rate of interest, the problem was to predict its return at a particular time from the remaining features, as well as the value of the return at the last time step. It should be noted that we were not using these features
to predict the return at the next time step, but rather using the available values to predict a missing value at the current time step.

We implemented the DCCCH algorithm using an ensemble of size 25. Each member of the ensemble was an order-1 Kalman filter [Kalman, 1960]. To promote diversity in the ensemble, each member regressor was built from a different random subset of the available features, where each feature was included with probability 0.5. The chosen cost function $c$ was the squared error:

$$c(g(x), y) = (g(x) - y)^2.$$  \hspace{1cm} (6.2.3)

For each regressor, we generated online estimates of the density function $P(c(g(X), Y))$ using (5.2.2) with $\alpha = 0.99$. This density function was estimated by a histogram with 750 bins representing squared errors from 0 to a maximum value $\kappa$ chosen prior to each run of the DCCCH algorithm. To determine $\kappa$, we observed the squared error realized by a single Kalman filter on each point of the dataset, and set $\kappa$ equal to the 95th percentile of these measurements. To predict the response for a data point $x_t$, we found the regressors on the convex hull at the last time step $CH_{t-1}$, and combined their outputs into a global prediction by taking a weighted mean, where the output of each regressor was weighted by the area under its dynamic REC curve.

We compared this implementation with a single Kalman filter (SKF) and simple regressors built from the described ensemble that predicted the unweighted mean of the member regressors’ outputs (ME) and the weighted mean of the member regressors’ outputs (WE), using the areas under the dynamic REC curves for the weights. We also compared our implementation with the AddExp.C algorithm [Kolter and Maloof, 2005]. The accuracy of each regressor was measured by finding its root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{\sum_t (g(x_t) - y_t)^2}{N}}.$$  \hspace{1cm} (6.2.4)

The AddExp.C algorithm was implemented according to the specification in [Kolter and Maloof, 2005]. The $\beta$ and $\gamma$ parameters were set at 0.5 and 1/25, respectively. The $\tau$ parameter was set equal to the $\kappa$ value defined above. The ensemble was fixed at size 25. When the ensemble grew too large, the worst-performing learners (according to the weights assigned by the AddExp.C algorithm) were removed. This scheme was suggested in [Kolter and Maloof, 2005].

Table 2 shows the aggregate results over the twenty-five datasets. For each of the five trading rates of interest, the DCCCH algorithm performs the best, its distribution of RMSEs having both the


### 6.2 Results Regarding the DCCCH Algorithm

Table 6.2: Mean RMSE times $10^3$ for the twenty-five financial datasets achieved by different methods (standard deviation times $10^3$ in parenthesis).

<table>
<thead>
<tr>
<th>Method</th>
<th>EUR–CHF ($10^3$)</th>
<th>EUR–USD ($10^3$)</th>
<th>USD–JPY ($10^3$)</th>
<th>EUR–JPY ($10^3$)</th>
<th>USD–CHF ($10^3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKF</td>
<td>7.52 (2.54)</td>
<td>7.54 (2.86)</td>
<td>7.22 (2.45)</td>
<td>7.35 (2.26)</td>
<td>7.38 (2.40)</td>
</tr>
<tr>
<td>ME</td>
<td>6.32 (2.10)</td>
<td>6.61 (2.61)</td>
<td>6.57 (2.38)</td>
<td>6.48 (2.03)</td>
<td>6.50 (2.37)</td>
</tr>
<tr>
<td>WE</td>
<td>4.11 (2.02)</td>
<td>4.54 (2.06)</td>
<td>4.48 (2.29)</td>
<td>4.35 (1.93)</td>
<td>4.35 (2.03)</td>
</tr>
<tr>
<td>AddExp.C</td>
<td>30.7 (3.09)</td>
<td>30.8 (3.06)</td>
<td>30.4 (3.06)</td>
<td>30.4 (3.04)</td>
<td>30.2 (3.18)</td>
</tr>
<tr>
<td>DCCCH</td>
<td><strong>2.87</strong> (1.32)</td>
<td><strong>3.06</strong> (1.87)</td>
<td><strong>2.56</strong> (1.08)</td>
<td><strong>2.66</strong> (1.75)</td>
<td><strong>2.80</strong> (1.90)</td>
</tr>
</tbody>
</table>

lowest mean and the lowest standard deviation. The closet competitor was the weighted ensemble, which used the area under the REC curve as a weight. The DCCCH algorithm was able to improve upon this scheme by only utilizing those regressors on the REC convex hull, allowing the algorithm to quickly identify the potentially optimal learners. The DCCCH algorithm proved again that it can handle large-scale online learning tasks, this time on a regression problem.

#### 6.2.3 Experiment 3: dynamic feature evaluation

Finally, we conducted an experiment to evaluate the DCCCH feature selection method we described in Section 5.3.2. We used a synthetic binary classification problem very similar to the one used in Experiment 1. The dataset consisted of 5,000 data points drawn from $U[0, 10]$. The response was determined by the mean of two features in the dataset, but the relevant features were randomly reassigned every 500 points. If $x_i$ and $x_j$ were the two relevant features at a particular time, the response was determined by $I(\text{mean}\{x_i, x_j\} < 10)$. No label noise was introduced.

We built a DCCCH classifier from an ensemble of 25 DLR classifiers built from random feature subsets, with each feature being selected with probability 0.5. Note that as a generalized linear model, the DLR algorithm performs internal feature evaluation—at every time step, each feature is assigned a coefficient, the magnitude of which can be interpreted as a measure of its importance.

At every time step $t$ the relative importance of each feature was determined in the following manner. Each feature was assigned an initial weight of 0. Each time a feature appeared in a classifier $C \in CH_t$, the weight associated with that feature was increased by the magnitude of its regression coefficient in $C$. After considering each classifier on the convex hull, the feature weights were normalized to sum to 1. This defines a distribution over the features, which we used to build an online
Figure 6.5: The mean portion of the total available mass accounted for by the relevant features in the Experiment 3 dataset over time. The mean plus- and minus-one standard deviation are shown in dotted lines.

estimate of the features’ relative importance using (5.2.2) with $\alpha = 0.99$.

To eliminate problems caused by random fluctuations in the generated dataset, twenty-five datasets were created.

The aggregate results are shown in Figure 3. The plot shows the portion of the total mass accounted for by the relevant features throughout time. The method works very well; the two relevant features often account for 70–90% of the available mass. The method was also able to quickly recover from concept drifts that affected the relative importance of the features.
Chapter 7

Conclusions and Future Work

The rapid expansion of the internet and increasing ease with which extremely large datasets may be collected has inspired a great deal of research into performing data mining on fast-moving data streams. The unique issues posed by these problems often renders the use of classical techniques impossible. Among these issues, one of the most important and drastic is the prevalence of arbitrary and often unexpected concept drifts in typical data streams.

After a brief review of relevant research into the nonstationary online learning problem, we introduced the dynamic logistic regressor (DLR), a Bayesian approach to the non-stationary, semi-supervised binary classification task. We then demonstrated how the DLR framework could be extended to effectively cope with possible uncertainty in the label stream. The result is a very capable classifier able to handle an extraordinary range of online classification situations.

With an excellent online classifier in hand, we proceeded to introduce a new meta-algorithm for performing online ensemble classification and regression problems.

The convex hull of characteristic curves provides a simple mechanism for identifying potentially optimal learners from a set of candidates. Unfortunately, standard ROC and REC curves are not suitable for use in a streaming environment with concept drift because they assign a equal weight to each point.

We introduced an online method for creating dynamic characteristic curves that places more emphasis on recent performance. We used these dynamic characteristic curves to introduce a new framework for online ensemble learning called the dynamic characteristic curve convex hull (DCCCH) method. This framework gives a unified approach for performing classification and regression on large, non-stationary data streams. The algorithm has other useful characteristics, including providing a method for automatically performing online feature evaluation. The DCCCH method out-
perform other available methods on large classification and regression problems with concept drift.

The methods discussed in this paper offer interesting avenues for future research. We will discuss possible future work related to the DLR and DCCCH in the next few sections, then proceed to discuss several proposals for interesting research opportunities in problems related to learning from drifting data streams.

7.1 Proposals for Future Research into the DLR algorithm

The DLR algorithm has been the subject of extensive research and is quite mature [Penny and Roberts, 1999], [Lowne et al., 2007]. Nonetheless, the algorithm could be refined in a number of ways. Perhaps the most significant improvement would be the addition of a predictive model for the weight parameters \( w \); that is, by choosing a more complicated form for the function \( f(w_{t-1}, v_t) \) in Equation 4.1.8. In Section 4.1, we made the simplifying assumption that the weights don’t undergo any sort of symptomatic drift, prescribing instead the very simple function

\[
f(w_{t-1}, v_t) = w_{t-1} + v_t. \quad (7.1.1)
\]

Although this assumption works well in practice, it is not the best we could do. In many cases, the weights will undergo predictable drift at least over short time scales; in this case, we should take advantage of that information. Suppose then that we have developed a function \( f_t(w_{t-1}, v_t) \) whose purpose is to predict the location of the weight parameter \( w \) at time \( t \). This is essentially an arbitrary time-series prediction problem that can be solved in any number of ways. Gaussian processes, for example, provide a simple and natural Bayesian method for performing this prediction [Rasmussen and Williams, 2005], [Osborne and Roberts, 2007]. Given this function \( f_t \), we can use it to derive the weights \( w_{t|t-1} \), allowing us to better predict the estimated posterior probability \( z_{t|t-1} \).

Perhaps more importantly, however, this adaptation allows for a more natural method for handling missing labels. Instead of defining the “quasi-target” (4.2.7)

\[
y'_t = I(z_{t|t-1} > 0.5), \quad (7.1.2)
\]

we may instead calculate \( z_{t|t-1} \) with Equations (4.1.12)–(4.1.14) using the now better estimate of \( w_{t|t-1} \). Then, if the label \( y_t \) is missing, when we derive the updated weight parameters \( w_{t|t} \), we set

\[
y_t = z_{t|t-1}. \quad (7.1.3)
\]
This has the effect of setting
\[ w_t = w_{t|t-1}; \]  
(7.1.4)
that is, we do not presume to learn anything from the observation \((x_t, y_t)\) beyond what our predictive function \(f_t\) had, unlike the illusionary knowledge provided by quasi-targets. As a result, this procedure is less likely to bring the system to an unlikely state in the presence of many missing labels. On the other hand, the EKF framework may be made more complicated by the more complex predictive function \(f_t\), requiring a possibly difficult derivative to be calculated.

## 7.2 Proposals for Future Research into the DCCCH algorithm

The DCCCH method introduced in this paper offers interesting possibilities for future improvement.

### 7.2.1 Automatic and adaptive selection of the parameter \(\alpha\)

It is interesting to consider that the weighting parameter \(\alpha\) used to construct the probability density estimates in (5.2.2) does not need to be a constant. Instead, the weight could be made a function \(\alpha(\cdot)\) of some other variable, which could be useful in many circumstances. For example, in practice many data streams do not present themselves as simple time series. Instead, the data often appear intermittently, with variable-length gaps between successive points. If this were the case, it might be desirable to assign a weight to the historical knowledge \(P_{t-1}(\cdot)\) based on the delay between the observations at time \(t-1\) and time \(t\). Intuitively, we might have less confidence in our old information the longer we have to wait for new data. This could be accomplished by modeling the gaps with an exponential distribution
\[ P(\Delta t = x) = \lambda e^{-\lambda x}, \]  
(7.2.1)
whose parameter \(\lambda\) could be estimated online. This distribution could then help guide the choice of \(\alpha\). As another example, some learners (such as Kalman models) have built-in measures of the non-stationarity of the observed data. It would be reasonable to consider this information when selecting \(\alpha\)—if we believe that the data have experienced a drift in concept, we should have less faith in our previous estimates.

Another option for setting the \(\alpha\) parameter can be developed from a recent paper on online Bayesian change-point detection [Adams and MacKay, 2007]. In the paper, the authors present a
novel approach to the online change-point detection algorithm; that is, the problem of detecting when an observed data stream undergoes a drift in concept. Their approach defines a posterior distribution over the likely run-length of the current concept, or equivalently, the number of observations made since a likely concept drift. This posterior distribution could be used in a straightforward matter to select the DCCCH $\alpha$ parameter; specifically, by setting $\alpha$ to represent a past time-scale commensurate with the expected run-length. Again, this procedure will chose a smaller $\alpha$ when we suspect a concept drift has recently occurred, effectively ignoring the information contained in the now out-of-date distributions used to build the characteristic curves.

The automatic selection of $\alpha$ is an interesting and open-ended problem that requires further investigation.

### 7.2.2 Decentralizing the DCCCH algorithm

The DCCCH algorithm as described requires a central server to calculate the convex hulls and generate the ensemble’s global prediction. It would be interesting to explore the use of dynamic characteristic curve convex hull methods in distributed agent-based systems with limited communication.

This should be fairly easy to accomplish. Assume we have a network of sensors attempting to solve a common learning problem, and that each sensor has a limited neighborhood of other sensors with which it may communicate. One way to proceed is to have each sensor individually build a local copy of its neighbors’ characteristic curves, calculate the convex hull of these curves, and generate predictions based on the DCCCH as before. This can be realized using a very simple protocol whereby each sensor simply communicates its prediction at time $t$ to each of its neighbors; this should be sufficient for constructing the required characteristic curves.

This algorithm may offer interesting side effects. For example, if a sensor discovers that its own characteristic curve does not lie on the DCCCH of it and its neighbors. In this case, it may be advantageous for the sensor to take less frequent measurements (for example, to save power) until it begins to perform better. Additionally, under this scheme, there is no particular reason why the sensors must communicate with each other synchronously or at fixed time intervals, especially if an adaptive weighting scheme like the one discussed in the previous section is used.
7.3 Proposals for Other Future Research

7.2.3 Theoretical results

The DCCCH algorithm as presented in Chapter 5 works very well in practice, but as of yet no theoretical results can underscore this performance. The algorithm is grounded in several well-known theoretical results, for example the optimality of classifiers that lie on the ROC convex hull. With the proper assumptions, it may be possible to prove similar theorems about the DCCCH. By fixing the parameter $\alpha$, it should be possible to prove useful theorems using an inductive proof similar to those used in [Kolter and Maloof, 2005].

7.3 Proposals for Other Future Research

In addition to the proposed work on the DLR and DCCCH above, we make further suggestions for future research related to learning from data streams with concept drift.

7.3.1 Gaussian processes for Bayesian optimization

The problem we consider in this section is the global optimization task. In this problem, we have a parameter space $\Omega$, which we will consider to be $\mathbb{R}^n$, and an unknown real-valued function $f: \Omega \rightarrow \mathbb{R}$ that might be very expensive to calculate. The goal is to find a maximal value of the function $f$ after sampling the function at as few points in the parameter space as possible.

Many solutions have been proposed for this problem, including simulated annealing [Kirkpatrick et al., 1983], stochastic tunneling [Wenzel and Hamacher, 1999], genetic algorithms [Fraser and Burnell, 1970], and particle swarm optimization [Kennedy and Eberhart, 1995]. Many of these solutions are heuristic approaches to the problem that may work well in practice but do not have much, if any, theoretical support.

We propose a different approach to this problem using Gaussian processes, which provide for simple and powerful Bayesian inference over function spaces. The theory of Gaussian processes is deep and well-established [Rasmussen and Williams, 2005]; we provide only the scantest of overviews here.

A Gaussian process is an infinite collection of random variables, any finite number of which are jointly Gaussian distributed. A common application of Gaussian processes is to perform regression; that is, fit a function to a set of observations $\{(x,f(x))\}$. In the global optimization function, this is exactly what is required.
A Gaussian process is completely defined over a space $\Omega$ by a mean function $\mu : \Omega \to \mathbb{R}$ and a covariance function $K : \Omega \times \Omega \to \mathbb{R}$ that, given two points $x, x' \in \Omega$, determines the strength of the correlation between the function values at $x$ and $x'$.

To perform the regression task, the process is a straightforward application of Bayesian formalism. We begin by defining a prior distribution over the function space of interest, specifying a prior mean function $\mu$ (usually taken to be zero or another constant), and a convenient prior covariance function $K$. Then, after observing data of the form $(x, f(x))$, we update these distributions using Bayes’ rule to find the posterior mean and covariance functions, which then become the prior distributions before observing the next sample.

In the case of the global optimization function, we are only interested in finding a point $x \in \Omega$ that has the largest associated value $f(x)$ possible. We are not concerned with the value of any other sample obtained. This allows us to define a simple utility function $U : \Omega \to \mathbb{R}$ appropriate for the problem. Suppose that before observing the value of the function at a point $x^*$, the maximum function value we have observed thus far is some value $M$. The utility of observing the value $f(x^*)$ can be given by

$$U(x^*) = \begin{cases} f(x^*) & \text{if } f(x^*) > M, \\ M & \text{otherwise}; \end{cases} \quad (7.3.1)$$

that is, we only experience an increase in utility when we observe a point with a greater function value than we’ve previously observed. This allows us to calculate the expected utility of observing the function value at a sample point $x^*$ given our current knowledge of the function. Suppose that we have observed the function at a sample $f_D = \{(x_i, f(x_i)) \}$. The expected utility of observing $f(x^*)$ is then

$$\mathbb{E}[U(x^*)] = \int_{M}^{\infty} y P(f(x^*) = y \mid f_D) \, dy + M \int_{-\infty}^{M} P(f(x^*) = y \mid f_D) \, dy. \quad (7.3.2)$$

The use of a Gaussian process distribution over our function space ensures a convenient form for the density function

$$P(f(x^*) = y \mid f_D); \quad (7.3.3)$$

it is, in fact, guaranteed to be a Gaussian distribution. This allows us to evaluate the required integrals in (7.3.2) easily; the first has a closed-form solution and the second is simply the error function.
7.3 Proposals for Other Future Research

If we were allowed to sample the function at just one more point \( x^* \), the correct Bayesian decision would be to sample at the point that has the highest expected utility as defined in (7.3.2). This becomes more interesting, however, if we are allowed more than one sample. In this case, we can calculate the expected utility of sampling at a particular set of points. In this case, it may be difficult to derive a closed-form solution for the expected utility; however, Bayesian Monte Carlo techniques should be able to provide adequate estimates. The power of this technique is that, given enough samples, we may discover that the best decision is to first sample in areas of high uncertainty (to gain more information about the function) before sampling in areas of low uncertainty (where we might be relatively certain the function attains high values). In any case, the decision-theoretical approach we have chosen will naturally guide us toward the correct choices.

An interesting extension to this problem can be realized by allowing the function \( f \) to undergo concept drift; that is, allowing the unknown function \( f \) to change arbitrarily over time. In this case, \( f \) becomes a function of location and time. The Gaussian process framework prescribed above handles this modification to the problem in a natural way: we simply model the function over \( \Omega \) and one additional dimension signifying time. In this way, our uncertainty about the function will increase with the time lapsed since our last observation. In fact, this property of Gaussian processes makes them attractive for other learning situations with concept drift; this is an exciting avenue for future research.

This technique for approaching the global optimization problem is quite promising, because it offers a formalized Bayesian framework for solving the problem that is more motivated by theory than many of the existing, ad hoc methods. A great deal of research and experimental analysis remains to be done.

7.3.2 Reinforcement learning in non-stationary environments

The global optimization problem discussed above is an example of a much more general class of problems; specifically, it is an example of a prototypical reinforcement learning problem. In the general reinforcement learning task, an agent operates within an environment characterized by a state space \( S \). At each time step \( t \), the agent is informed of its location in the state space, \( s_t \in S \). The agent is also presented with a set of possible actions from that state \( A(s_t) \). After selecting an action \( a \in A(s_t) \), the agent is taken to a new state \( s_{t+1} \in S \) and presented with a real-valued reward or reinforcement \( R \in \mathbb{R} \). The transition rule determining the next state given a state and action pair is unknown, but usually assumed to be fixed throughout time. The goal of the agent is to establish a policy \( \pi : S \to A \)
in the space to maximize its long-term cumulative reward in some sense; its only feedback guiding
the creation of this policy is the reinforcement it receives after every action.

A common problem faced in reinforcement learning systems the decision between exploration
and exploitation. Should the agent explore unknown regions of the state space, or should it instead
exploit regions known to give relatively large rewards? In the global optimization problem described
above, this conflict was realized by having to choose between sampling the function $f$ at regions of
high uncertainty (to explore the space) or sampling $f$ at regions of low uncertainty and high expected
value (to exploit our knowledge of the space).

A number of solutions have been proposed to solve the classical reinforcement learning problem
with a static transition rule; see [Kaelbling et al., 1996] for a survey. Less research has investigated the
reinforcement problem in nonstationary environments, although changing environments and reward
functions typify real-world reinforcement learning problems. In the previous section, we discuss how
to use Gaussian processes to approach a rather specific reinforcement learning problem; it is possible
that such approaches could be adapted or extended to solve a more general class of problems. Re-
gardless of whether similar methods can be used or a completely different approach is needed, the
nonstationary reinforcement learning is very important and provides a very interesting opportunity
for open-ended research.

### 7.3.3 Online density estimation

A common problem that must be solved as part of machine learning systems is the estimation of
the probability density function for a particular random variable. When this distribution is station-
ary, well-known approaches such as histograms, frequency polygons, and kernel density estimation
suffice [Silverman, 1986], [Scott, 1992].

Little research has investigated approaches to the problem when the distribution undergoes con-
cept drift. The procedure outlined in Chapter 5 provides a simple mechanism for maintaining density
estimates in nonstationary environments; however, the simple method is motivated more by comput-
tational simplicity than by theoretical considerations. Similar ad hoc approaches was described in
[Zhou et al., 2003] and [Cortes and Pregibon, 2001].

It is possible that Gaussian processes may provide a more theoretically grounded approach for
performing online density estimation for nonstationary distributions, but this idea needs much more
investigation before its merit may be adequately judged.


