COMS 4771 Spring 2015

Online-to-batch

Daniel Hsu *Warning: notes are incomplete and unchecked for correctness*

1 Batch learning via online learning

In the usual model of statistical learning for binary classification, there is a sequence of labeled examples

$$((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)) \in (\mathcal{X} \times \{\pm 1\})^n$$

called the "training data", and the goal is to find a classifier with small classification error. The training data are assumed to be drawn i.i.d. from a fixed probability distribution $\mathbb{P}$. The classification error of a classifier $f \colon \mathcal{X} \to \{\pm 1\}$ is the probability that the classifier mis-classifies a random example (x, y) drawn from $\mathbb{P}$:

$$\mathrm{err}(f) := \Pr(f(x) \neq y).$$

A "batch" learning algorithm takes as input the training data and returns a classifier. We'll also talk about *randomized* classifiers f; the probability defining $\mathrm{err}(f)$ is then taken with respect to both the internal randomization in f as well as the random draw (x, y) from $\mathbb{P}$.

We abstractly think of an "online" learning algorithm as taking as input a sequence of n labeled examples, and producing a sequence of classifiers $f_1, f_2, \ldots, f_n, f_{n+1} \colon \mathcal{X} \to \{\pm 1\}$. The t-th classifier f_t can only depend on the first $t-1$ labeled examples. Although not all online learning algorithms fit this template, it is sufficient for most cases.

1.1 Relating expected classification error and mistake bounds

In expectation, the classification error of the classifier returned by the batch learning algorithm $\mathcal{B}$ can be written as

$$\mathbb{E}\Big(\mathbb{1}\{\hat{f}(x_{n+1}) \neq y_{n+1}\}\Big)$$

where $\hat{f} := \mathcal{B}(S_{1:n})$ and

$$S_{1:n+1} := ((x_1, y_1), (x_2, y_2), \ldots, (x_{n+1}, y_{n+1}))$$

is a sequence of $n+1$ labeled examples drawn i.i.d. from $\mathbb{P}$, and $S_{1:n}$ denotes the sequence of the first n such examples. If $\mathcal{B}$ is a randomized algorithm, then the expectation above is also taken with respect to the random bits used by $\mathcal{B}$.

The following is a simple way to convert an online learning algorithm $\mathcal{A}$ such as Online Perceptron into a (randomized) batch learning algorithm $\mathrm{OTB}_{\mathcal{A}}$ in a way that bounds the expected classification error of $\mathrm{OTB}_{\mathcal{A}}$ in terms of a mistake bound for $\mathcal{A}$.

Algorithm 1 Online-to-batch learning algorithm $\mathrm{OTB}_{\mathcal{A}}$

input Training data $S_{1:n} := ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$.

1: Run $\mathcal{A}$ with the sequence $S_{1:n}$ to produce classifiers $f_1, f_2, \ldots, f_n, f_{n+1}$.
2: Pick $T \in \{1, 2, \ldots, n+1\}$ uniformly at random.
3: **return** f_T.

Another way to interpret Algorithm 1 is as follows.

1. Run $\mathcal{A}$ on the training data sequence, and keep all $n+1$ classifiers $f_1, f_2, \ldots, f_n, f_{n+1}$. generated along the way.
2. The final classifier used for prediction is a randomized classifier: to predict on a new instance x, first randomly choose $f_T \in \{f_1, f_2, \ldots, f_{n+1}\}$, and then predict $f_T(x)$.

Theorem 1. *Let $M_{\mathcal{A},n+1}$ be the number of mistakes made by online learning algorithm $\mathcal{A}$ on a sequence of $n+1$ i.i.d. examples $S_{1:n+1}$ drawn from $\mathbb{P}$. The expected classification error of $\mathrm{OTB}_{\mathcal{A}}(S_{1:n})$ is $\mathbb{E}(M_{\mathcal{A},n+1})/(n+1)$.*

Proof. By direct computation: letting $f_T := \mathrm{OTB}_{\mathcal{A}}(S_{1:n})$,

$$\begin{aligned}
&\mathbb{E}(\mathbb{1}\{f_T(x_{n+1}) \neq y_{n+1}\}) \\
&= \sum_{t=1}^{n+1} \Pr(T = t) \cdot \mathbb{E}(\mathbb{1}\{f_T(x_{n+1}) \neq y_{n+1}\} \,|\, T = t) \\
&= \sum_{t=1}^{n+1} \frac{1}{n+1} \mathbb{E}(\mathbb{1}\{f_t(x_{n+1}) \neq y_{n+1}\}) \\
&= \sum_{t=1}^{n+1} \frac{1}{n+1} \mathbb{E}(\mathbb{1}\{f_t(x_t) \neq y_t\}) \\
&= \frac{1}{n+1} \mathbb{E}\left(\sum_{t=1}^{n+1} \mathbb{1}\{f_t(x_t) \neq y_t\} \right).
\end{aligned}$$

The penultimate step is true because f_t is only a function of $S_{1:t-1}$, and because $(S_{1:t-1}, (x_{n+1}, y_{n+1}))$ has the same distribution as $(S_{1:t-1}, (x_t, y_t))$. The sum in the final expectation is $M_{\mathcal{A},n+1}$. □

Theorem 1 shows that an online learning algorithm with a small mistake bound can be effectively used for batch learning. If the mistake bound of $\mathcal{A}$ for a sequence of $n+1$ examples is sublinear in n, then the expected classification error of $\mathrm{OTB}_{\mathcal{A}}$ is $o(1)$. Indeed, the mistake bound for Online Perceptron is constant in the "linearly separable by a margin" setting, in which case the expected classification error of $\mathrm{OTB}_{\mathrm{OnlinePerceptron}}$ is $O(1/n)$.

Computationally, Algorithm 1 can be quite attractive if the online learning algorithm has constant time updates and has a constant memory requirements (with respect to n). Online Perceptron certainly fits this bill. In these cases, the running time scales linearly with n. In fact, the algorithm only needs streaming access to the data.

One drawback of Algorithm 1 is that its final classifier is a randomized classifier: it randomly picks among the $n+1$ classifiers generated by the online learning algorithm, and uses the chosen classifier to predict. However, the randomized classifier can be *derandomized* by using a majority vote over the $n+1$ classifiers. The classification error of the majority-vote classifier is at most twice the classification error of the randomized classifier (and is possibly much smaller).

Some online learning algorithms such as Perceptron don't update their weight vectors unless a mistake is made, and thus there may actually be fewer than $n+1$ different classifiers. In such cases, it can be more efficient to represent the final randomized (or majority-vote) classifier as a weighted combination of these different classifiers. The weight is proportional to the number of rounds in which it is used by the online learning algorithm—specifically, one plus the number of examples on which it predicts correctly.

In the case of linear classifiers, it is reasonable to use a (weighted) average of the weight vectors corresponding to the different classifiers. This can be thought of as an approximation to the majority-vote classifier. In some cases, averaging can be rigorously justified and in fact shown to at least as good as the randomized classifier.