An Introduction to Incremental Learning

By Qiang Wu and Dave Snell

Machine learning provides useful tools for predictive analytics. The typical machine learning problem can be described as follows: A system produces a specific output for each given input. The mechanism underlying the system can be described by a function that maps the input to the output. Human beings do not know the mechanism but can observe the inputs and outputs. The goal of a machine learning algorithm is to infer the mechanism by a set of observations collected for the input and output. Mathematically, we use (x_i,y_i) to denote the i-th pair of observation of input and output. If the real mechanism of the system to produce data is described by a function f^*, then the true output is supposed to be $f^*(x_i)$. However, due to systematic noise or measurement error, the observed output y_i satisfies $y_i = f^*(x_i)+\epsilon_i$ where ϵ_i is an unavoidable but hopefully small error term. The goal then, is to learn the function f^* from the n pairs of observations $\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$.

A machine learning algorithm must first specify a loss function $L(y,f(x))$ to measure the error that will occur when we use $f(x)$ to predict the output y for an unobserved x. We use the term unobserved x to describe new observations outside our training sets. We wish to find a function such that the total loss on all unobserved data is as small as possible. Ideally, for an appropriately designed loss function, f^* is the target function. In this case, if we can compute the total loss on all unobserved data, we can exactly find f^*. Unfortunately, computing the total loss on unobserved data is impossible. A machine learning algorithm usually searches for an approximation of f^* by minimizing the loss on the observed data. This is called the empirical loss. The term **generalization error** measures how well a function having small empirical loss can predict unobserved data.

There are two machine learning paradigms. **Batch learning** refers to machine learning methods that use all the observed data at once. **Incremental learning** (also called online learning) refers to the machine learning methods that apply to streaming data collected over time. These methods are used to update the learned function accordingly when new data come in. Incremental learning mimics the human learning process from experiences. In this article, we will introduce three classical incremental learning algorithms: the stochastic gradient descent for linear regression, perceptron for classification and incremental principal component analysis.

STOCHASTIC GRADIENT DESCENT

In linear regression, $f^*(x) = w^Tx$ is a linear function of the input vector. The usual choice of the loss function is the squared loss $L(y,w^Tx) = (y-w^Tx)^2$. The gradient of L with respect to the weight vector w is given by

$$\nabla_w L = -2(y - w^T x)x.$$

Note the gradient is the direction for the function to increase, so if we want the squared loss to decrease, we need to let the weight vector move opposite to the gradient. This motivates the stochastic gradient descent algorithm for linear regression as follows: the algorithm starts with the initial guess of w as w_0. At time t, we receive the t-th observation x_t and we can predict the output as

$$\hat{y}_t = w_{t-1}^T x_t.$$

After we observe the true output y_t, we can update the estimate for w by

$$w_t = w_{t-1} + \eta_t(y_t - \hat{y}_t)x_t$$

The number η_t>0 is called the step size. Theoretical study shows that w_t becomes closer and closer to the true coefficient vector w provided the step size is properly chosen. Typical choice of the step size is

$$\eta_t = \frac{\eta_0}{\sqrt{t}}$$

for some predetermined constant η_0. Another quantity to measure the effectiveness is the accumulated regret after T steps defined by

$$Regret = \sum_{t=1}^{T}(y_t - \hat{y}_t)^2 - \sum_{t=1}^{T}(y_t - w^T x_t)^2$$

If this algorithm is used in a financial decision-making process and $w^T x_t$ is the optimal decision at step t, the regret measures the total **additional**[1] losses because the decisions are not optimal. In theory, the regret is bounded, implying that the average additional loss resulting from one decision is minimal when T is large.

We use a simulation to illustrate the use and the effect of this algorithm. Assume that in a certain business, there are five risk factors. They may either drive up or down the financial losses. The loss is the weighted sum of these factors plus some fluctuation due to noise: $y = x_1 - x_2 + 0.5x_3 - 0.5x_4 + x_5 + \epsilon$. So the true weight coefficients are given by w=[1, -1, 0.5, -0.5, 2]. We assume each risk factor can take values between 0 and 1 and the noise follows a mean zero normal distribution with variance 0.01. The small variance choice is empirically selected to achieve a smaller signal to noise ratio. We generate 1,000 data points sequentially to mimic the data-generating process and perform the learning with an initial estimate w_0=[0,0,0,0,0]. In Figure 1, we plot the distance between w_t and w, showing estimation error decays fast (which is desirable). In Figure 2, we plot the regret for each step. We see most additional losses occur at the beginning because we have used a stupid initial guess. They increase very slowly after 50 steps, indicating the decisions become near optimal. In other words, even a poor guess can lead to excellent results after a sufficient number of steps.

Figure 1: Estimation Error vs. Iterations

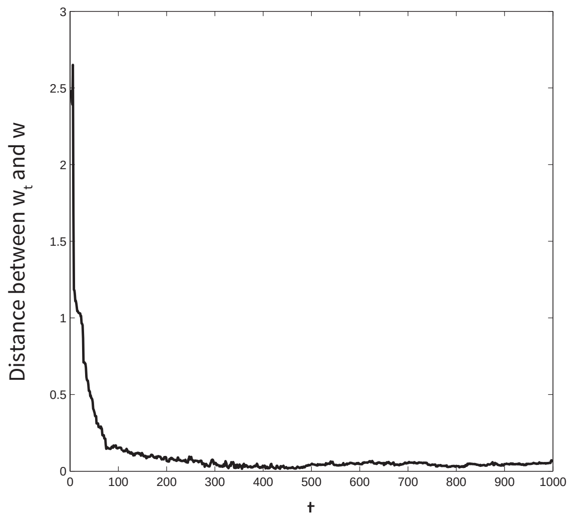
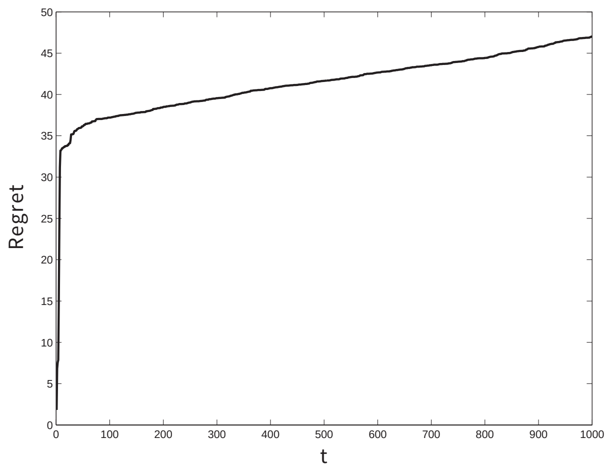


Figure 2: Regret vs. Iterations



PERCEPTRON

In a classification problem, the target is to develop a rule to assign a label to each instance. For example, in auto insurance, a driver could be labeled as a high risk or low risk driver. In financial decision-making, one can determine whether an action should be taken or not. In a binary classification problem where there are two classes, the labels for the two classes are usually taken as 0 and 1 or –1 and +1. When –1 and +1 are used as the two labels, the classifier could be determined by the sign of a real valued function. A linear classifier is the sign of a linear function of predictors $f(x) = sign(w^T x)$. Mathematically $w^T x = 0$ forms a separating hyperplane in the space of predictors. The perceptron for binary classification is an algorithm to incrementally update the weight vectors of the hyperplane after receiving each new instance. It starts with an initial vector w_0 and when each new instance (x_t, y_t) is received, the coefficient vector is updated by

$$w_t = \begin{cases} w_{t-1} + \eta_t y_t w_t, \text{if } y_t(\beta_{t-1} x_t) < \gamma, \\ w_{t-1}, \end{cases}$$

otherwise, where y is a user specified parameter called the **margin**. The original perceptron introduced by Rosenblatt in the 1950s has a margin 0, i.e., $y = 0$. The perceptron can be explained as follows. If $y_t(\beta_{t-1} x_t) < 0$, the t-th observation is classified incorrectly and thus the rule is updated to decrease the chance for it being classified incorrectly. If $y_t(\beta_{t-1} x_t) > 0$, the t-th observation is classified correctly, and no update is necessary. The idea of using a positive margin is from the well-known support vector machine classification algorithm. The motivation is that the classification is considered unstable if the observation is too close to the decision boundary even when it is classified correctly. Updating is still required in this case as a penalty. The classification rule is not updated only when an instance is classified correctly

Principal component analysis (PCA) is probably the most famous feature extraction tool for analytics professionals.

and has a margin from the decision boundary. For perceptron, the cumulative classification accuracy, which is defined as the percentage of the classified instances, can be used to measure the effectiveness of the algorithm.

In Figure 3, we simulated 1,000 data points for two classes: the positive class contains 500 data points centered at (1, 1) and the negative class contains 500 data points centered at (–1, –1). Both classes are normally distributed. The optimal separating line is $x_1 - x_2 = 0$, which can achieve a classification accuracy of 92.14 percent. That is, there is a systematic error of 7.86 percent. We assume the data points come in sequentially and apply the perceptron algorithm. The cumulative classification accuracy is shown in Figure 4. As desired, the classification ability of the perceptron is near optimal after some number of updates.

Figure 3: Data for a Binary Classification Problem

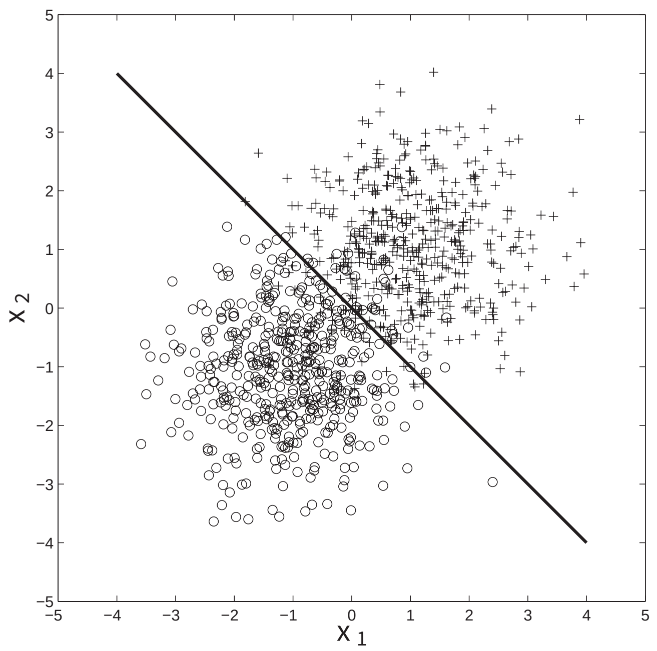
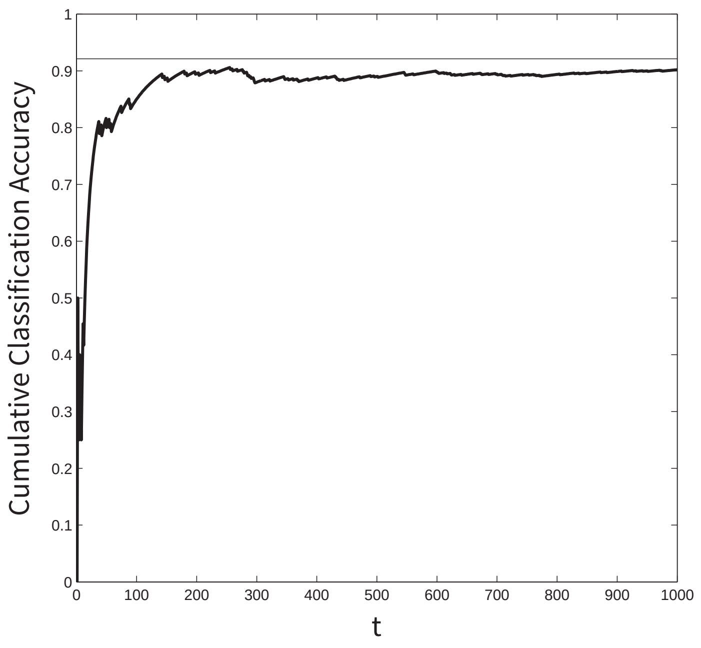


Figure 4: Cumulative Classification Accuracy of Perceptron Technique



INCREMENTAL PCA

Principal component analysis (PCA) is probably the most famous feature extraction tool for analytics professionals. The principal components are linear combinations of predictors that preserve the most variability in the data. Mathematically they are defined as the directions on which the projection of the data has largest variance and can be calculated as the eigenvectors associated with the largest eigenvalues of the covariance matrix. It can also be implemented by an incremental manner. For the first principal component v_1, the algorithm can be described as follows. It starts with an initial estimation $v_{1,0}$ and when a new instance x_t comes in, the estimation is updated by

$$u_{1,t} = (t-1)v_{1,t-1} + \left(v_{1,t-1}^T x_t\right)x_t,$$
$$v_{1,t} = \frac{u_{1,t}}{\|u_{1,t}\|}.$$

The accuracy can be measured by the distance between the estimated principal component and the true one.

Again, we use a simulation to illustrate its use and effectiveness. We generated 1,000 data points from a multivariate normal distribution with mean $\mu = [1,1,1,1,1]$ and covariance matrix

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}.$$

The first principal component is [0.9517, –0.2898, 0, 0, 0]. In Figure 5, we used the scatter plot to show the first two variables of the data with the red line indicating the direction of the first principal component. After applying the incremental PCA algorithm, the distance between the estimated principal component and the true principal component is plotted for each step in Figure 6. As expected, the distance shrinks to 0 as more and more data points get in.

Figure 5: Feature Abstraction via Principal Component Analysis

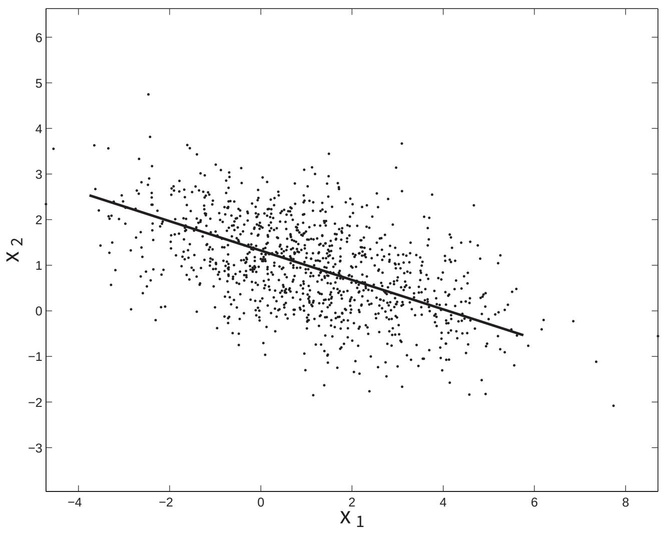
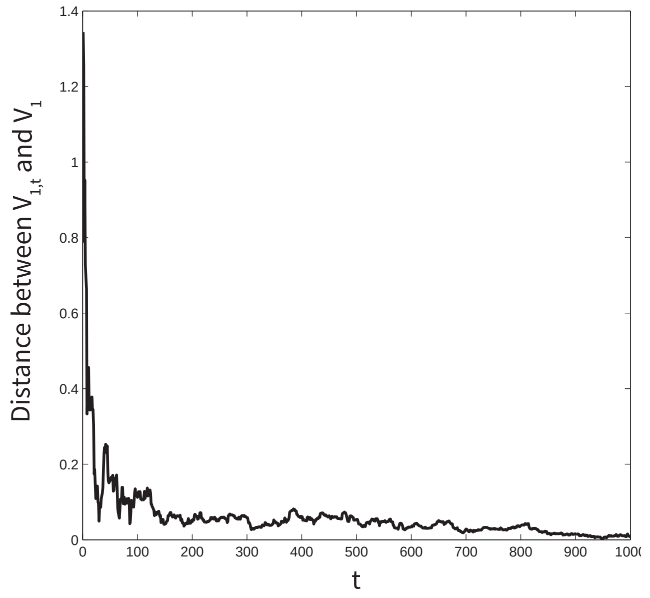


Figure 6: Estimation Error from Principal Component Analysis



REMARKS

We close with a few remarks. First, incremental learning has very important application domains, for example, personalized handwriting recognition for smartphones and sequential decision-making for financial systems. In the real applications, batch learning methods are usually used with a number of experiences to set up the initial estimator. This helps avoid large losses at the beginning. Incremental learning can then be used to refine or "personalize" the estimation. Second, we have introduced the algorithm for linear models. All these algorithms can be extended to nonlinear models by using the so-called kernel trick in machine learning. Finally, we would mention that it seems the term "online learning" is more popular in machine learning literature; however, we prefer the term "incremental learning" because "online learning" is widely used to refer to the learning system via the Internet and can easily confuse people. Actually, in *Google*, you probably cannot get what you want by searching "online learning." Instead, "online machine learning" should be used. ■

Qiang Wu, PhD, ASA, is asociate professor at Middle Tennessee State University in Murfreesboro, Tenn. He can be reached at *qwu@mtsu.edu.*

Dave Snell, ASA, MAAA, is technology evangelist at RGA Reinsurance Company in Chesterfield, MO. He can be reached at *dave@ActuariesAndTechnology.com.*

ENDNOTES

[1] Vladimir N. Vapnik, Statistical Learning Theory, John Wiley & Sons,1998.

[2] Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang, Candid Covariance-Free Incremental Principal Component Analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(8), 2003, 1034-1039.

[3] Wikipedia, Online Machine Learning. *https://en.wikipedia.org/wiki/Online_machine_learning*

[4] Wikipedia, Perceptron. *https://en.wikipedia.org/wiki/Perceptron*