Machine Learning, Spring 2019
EN 2105
MWF 9-9:50am

Chao Lan
EN 4084A
clan@uwyo.edu

Introduction

Machine learning studies "how can we build computer programs that automatically improve their performance through experience?" It is often applied to make predictions, such as

- predict students' GPA based on their profiles and current performance
- predict loan applicants' dependability based on their profiles and credit records
- predict users' product ratings based on their profiles and existing ratings
- classify documents into different topics based on their content (e.g., words)
- classify images into different scenes based on their content (e.g., pixel values)
- classify patients' health into different conditions based on their profiles and medical records

Note all these tasks can be performed by human experts based on their domain knowledge and experience. But machine learning *automate* this process, by learning experience from historical data, encoding it into a prediction model, and applying the model to predict future data.

Let us be more formal. Consider the student GPA prediction task. A student x is an example or instance, and his GPA y is his label (the variable to predict). His profile is often represented by a p-dimensional feature vector $x = [x_{\cdot 1}, x_{\cdot 2}, ..., x_{\cdot p}]^T$, where each variable $x_{\cdot j}$ is a feature or attribute of the student, e.g., $x_{\cdot 1}$ is his current GPA and $x_{\cdot 2}$ is his number of missing assignments.

If we denote the set of all students as $\mathcal{X} = \{x\}$ and the set of all GPA's as $\mathcal{Y} = \{y\}$, then a prediction model is a function $f : \mathcal{X} \to \mathcal{Y}$, i.e., model predicts the GPA of student x as $f(x)$.

The machine learning task is to estimate f from a training sample S of n students. Let x_i be the i_{th} student and y_i be his/her true label. If S has both student features and labels, i.e. $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, we call it a supervised learning task and each (x_i, y_i) a labeled example. If S has student features but no label, i.e., $S = \{x_1, \ldots, x_n\}$, we call it an unsupervised learning task and each x_i an unlabeled example. If S contains some labeled examples and some unlabeled examples, we call it a semi-supervised learning task. Another way to categorize learning tasks is based on Y. If Y is continuous, e.g., GPA = 3.72 or 2.89, we call it a regression task. If Y is discrete, e.g., GPA = A or B+, we call it a classification task.

The process of estimating f from S is called training. Typically, we find an f that can minimize the empirical loss on S, i.e.

$$f_* = \arg\min_f \sum_{(x,y)\in S} loss(f(x), y), \tag{1}$$

where $loss$ is the loss function; it has many choices and a common one is squared loss

$$loss(f(x), y) = (f(x) - y)^2. \tag{2}$$

Essentially, f is a function with unknown parameters, and training is the process of estimating

these parameters from observed samples. For example, f can be a linear function

$$f(x) = \beta_1 x_{\cdot 1} + \beta_2 x_{\cdot 2}, \tag{3}$$

where β's are unknown and can be estimated from S by minimizing the squared loss and give

$$[\beta_1, \beta_2]^T = (X^T X)^{-1} XY, \tag{4}$$

where $X = [x_1, \ldots, x_n]^T$ and $Y = [y_1, \ldots, y_n]^T$ with $(x_i, y_i) \in S$.

After training f, we can test it on a testing sample T of students. For regression task, we often use the evaluation metric of mean-squared-error (MSE)

$$\hat{er}_T(f) = \frac{1}{|T|} \sum\nolimits_{(x,y)\in T} (f(x) - y)^2. \tag{5}$$

For classification task, we often use the evaluation metric of classification error[1]

$$\hat{er}_T(f) = \frac{1}{|T|} \sum\nolimits_{(x,y)\in T} \mathbf{1}_{f(x)\neq y}, \tag{6}$$

where $\mathbf{1}_E$ is an indicator function and $\mathbf{1}_E = 1$ if event E is true and $\mathbf{1}_E = 0$ otherwise.

For clarity, the error of f on training sample will be called training error, denoted by $\hat{er}_S(f)$, and the error of f on testing sample will be called testing error, denoted by $\hat{er}_T(f)$. Ideally, we hope f can minimize $\hat{er}_T(f)$ over any any future T, and to that end we often find an f that can minimize $\hat{er}_S(f)$ over an observed S, presuming examples of S and T are taken from the same distribution (the stationary setting).[2] In other words, we hope f learned from S can generalize well on T. In practice, however, f may not generalize well, especially when S does not contain sufficient amount of examples. This typically results in a small training error $\hat{er}_S(f)$ but very large testing error $\hat{er}_T(f)$. We call this problem overfitting and say f overfits S.

More ideally, we hope f can minimize generalization error, which is the prediction error on a random example taken from a distribution D (from which S and T are taken), i.e., [3]

$$er_D(f) = E_{(x,y)\sim D}[(f(x) - y)^2]. \tag{7}$$

[*Exercise*] Prove $er_D(f) = E_{T\sim D^{|T|}}[\hat{er}_T(f)]$ if examples in T are taken from D.

[*Discussion*] What would happen to $er_D(f)$ if overfitting occurs?

One goal of machine learning research and practice is to avoid overfitting. In general, a model is less likely to overfit if it is simple. Here are four rule-of-thumbs:

1. a model designed with fewer unknown parameters is less likely to overfit

2. a model designed with smaller ranges of unknown parameters is less likely to overfit

3. a model learned from a larger sample is less likely to overfit

4. a model learned with more constraints is less likely to overfit

[1] For convenience, we will use the same notation for both MSE and classification error.
[2] We will assume stationary setting throughout the semester, unless specified otherwise.
[3] Here we consider regression task. Discussion on classification task is similar.

Rule 1 is easy to understand: if one can build an accurate model using 5 features, there is no need to include more features. The latter is harmful because it not only increases the chance of overfitting and computational cost but may also cause numerical problems.

Rule 2 can be implemented by regularization techniques, which restrict model parameters within particular ranges. For example, to learn f in (3), we can solve the optimization problem

$$\min_f \sum_{(x,y)\in S} (f(x) - y)^2 + \lambda \sum_{j=1}^{2} \alpha_j^2, \tag{8}$$

where $\sum_{j=1}^{2} \alpha_j^2$ is the regularization term and $\sum_{(x,y)\in S}(f(x) - y)^2$ is often called loss.

In (8), minimizing $\sum(f(x)-y)^2$ requires the model to make accurate prediction on S, minimizing $\sum \alpha_j^2$ requires the model to have small parameters, and λ is a hyper-parameter which controls how small α's should be. Hyper-parameter is often introduced to control model complexity; it is often set manually by experience or using K-fold cross-validation, which works as follows:

- prepare a pool of J candidate hyper-parameters $\lambda_1, \ldots, \lambda_J$
- partition training sample S into K validation sets $S_1, \ldots, S_K$
- for j = 1 : J
  - choose hyper-parameter λ_j for f
  - for k = 1 : K
    - train f on $\{S/\ S_k\}$, test it on S_k and obtain validation error er_{kj}
  - get average validation error for λ_j as $er_j = \frac{1}{K} \sum_{k=1}^{K} er_{kj}$
- select λ_j with the smallest average validation error

Cross-validation selects an optimal hyper-parameter to reduce overfitting, because smaller validation error implies f generalizes better (on unseen validation sets). In general, the process of selecting optimal hyper-parameters for a model is called model selection.

Rule 3 encourages us to collect more training data. This is perhaps the simplest and most ideal solution, but is often difficult to realize. For instance, in crime prediction we can only collect 10 examples if only 10 crimes occurred in Laramie, and in medical image diagnosis it is expensive and time-consuming to have experts diagnose which breast images indicate cancer.

Rule 4 is related to Rule 2, because a model learned with more constraints probably has smaller ranges of parameters. Take (8) for example. Standard learning approach will only minimize the first term but not the second, and thus f is more likely to be overfitting.

Reducing overfitting is not a worry-free task, however. One may over-simplify the model so it cannot accurately fit the data, resulting in underfitting. Consider (3). If the optimal parameters are $\beta_1 = 0.5$ and $\beta_2 = 0.6$, but the regularization technique (8) restricts $\beta_1, \beta_2 \in [0, 0.3]$, then we will never learn the optimal model and there will always be some prediction error.

[*Discussion*] What would you do in the exercise task?

Suppose we are given three features to predict student GPA: (1) current GPA, (2) number of registered courses and (3) number of missed assignments. If we use all features to construct a model, we obtain training error 0.12 and testing error 0.35. If we use only two features, in the best case we obtain training error 0.33 and testing error 0.36. What's wrong when we use three features? What's wrong when we use two features? How to avoid the problem?

The relation between overfitting and underfitting can be quantified as a bias-variance trade-off. To see the trade-off, we can first decompose the prediction error on a *fixed* example (x, y) by a *random* model f_S (learned from a random sample S) as

$$\begin{aligned} er(f_S; (x,y)) &= E[(f_S(x) - y)^2] \\ &= E[f_S^2(x) + y^2 - 2 \cdot f_S(x) \cdot y] \\ &= E[f_S^2(x)] + E[y^2] - E[2 \cdot f_S(x) \cdot y] \\ &= Var[f_S(x)] + E[f_S(x)]^2 + Var[y] + E[y]^2 - 2 \cdot E[f_S(x)] \cdot E[y] \\ &= Var[f_S(x)] + Var[y] + (\, E[f_S(x)]^2 + E[y]^2 - 2 \cdot E[f_S(x)] \cdot E[y] \,) \\ &= Var[f_S(x)] + Var[y] + (E[f_S(x)] - E[y])^2. \end{aligned} \tag{9}$$

where all expectations are taken over the randomness of S and we can further introduce randomness of (x, y) to keep the expectation and variance of y. The fourth equation is based on the property that any random variable a has

$$Var[a] = E[a^2] - E[a]^2. \tag{10}$$

[*Exercise*] Prove (10).

Look at (9). The leftmost prediction error is a constant since we average out all randomness. In the rightmost formula, we can interpret $Var[f_S(x)]$ as the variance of model prediction and $E[f_S(x)] - E[y]$ as the bias of model prediction; $Var[y]$ is data noise and has nothing to do with the model. Then we see model variance and (squared) model bias adds up to a constant, suggesting a trade-off between them, i.e., reducing model bias will increase model variance and vice versa. This further implies a 'trade-off' between overfitting and underfitting, because large model variance means overfitting and large model bias means underfitting. In general, we have

(a) overfitting = large model variance = complex model

(b) underfitting = large model bias = simple model

Besides prediction, machine learning is also applied in tasks of clustering, dimensionality reduction and reinforcement learning. Clustering aims to group similar examples while separating dissimilar examples in the feature space (for data categorization, scientific or anomaly discovery, etc). Dimensionality reduction aims to reduce feature dimension (for data visualization, computation speedup, etc) while preserving or enhancing certain data structures. Reinforcement learning aims to identify a sequence of optimal actions for an agent so that it can collect maximum rewards over these actions taken in a rewarding environment (for sequential decision making).