Machine Learning Theory (CSC 431/531) - Lecture 3

Nishant Mehta

1 Towards Statistical Learning: the Consistency Model

We now begin our study of the first major component of this course: *statistical learning theory.* Our starting point will be classification in the statistical learning setting (recall the protocol introduced in Lecture 1) under the realizability assumption.

Let's review the statistical learning protocol in the realizable case. Let $\mathcal{C} \subseteq 2^{\mathcal{X}}$ be a fixed and known concept class.

1. Nature selects a distribution P over the input space $\mathcal{X}$ and selects a concept $c \in \mathcal{C}$.
2. Learner is given a training set of n labeled examples $(x_1, y_1), \dots, (x_n, y_n)$, where each x_j is drawn i.i.d. from P and each $y_j = c(x_j)$.
3. Learner wishes to select a hypothesis f in some hypothesis space $\mathcal{F}$ which obtains low risk with respect to 0-1 loss, i.e., for which the misclassification probability $\Pr_{X \sim P}(f(X) \neq c(X))$ is small.

A natural approach for Learner is to take $\mathcal{F} = \mathcal{C}$ and select any hypothesis $f \in \mathcal{F}$ which is consistent with the training sample.

Definition 1. A hypothesis $f : \mathcal{X} \to \mathcal{Y}$ is *consistent* with training sample $(x_1, y_1), \dots, (x_n, y_n)$ if $f(x_j) = y_j$ for all $j \in [n]$.

If we are lucky and have a training sample that is representative of the actual distribution, then it will be the case that the *empirical risk*

$$\frac{1}{n} \sum_{j=1}^{n} \mathbf{1}[f(x_j) \neq c(x_j)]$$

is close to the true risk

$$R(f) = \mathsf{E}_{X \sim P}[\mathbf{1}[f(X) \neq c(X)]] = \Pr_{X \sim P}(f(X) \neq c(X)).$$

In this lucky event, minimizing the empirical risk is a good proxy for minimizing the actual risk.

For now, we will consider the algorithmic question of how to find a hypothesis consistent with the training sample. Once that is settled, at least in a few scenarios, we will directly address the question of how much data we need before the empirical risk is "close enough" to the true risk; answering this question is critical if we wish to justify algorithms that attempt to minimize the empirical risk.

Definition 2. An algorithm $\mathcal{A}$ learns a class $\mathcal{C}$ in the *consistency model* if, for any training set $(x_1, y_1), \dots, (x_n, y_n)$, the algorithm outputs a concept $f \in \mathcal{C}$ consistent with the training set if one exists and otherwise outputs `False` (indicating that no such concept exists in $\mathcal{C}$).

As before, our primary interest is in efficient algorithms, whose runtime is polynomial in the number of examples n and the size of an example (typically the dimension of the data, d). Note that unlike the mistake bound model, our algorithm must also handle the situation where the examples were not labeled according to any concept $c \in \mathcal{C}$.

2 Examples of learning in the consistency model

2.1 Monotone conjunctions

When $\mathcal{C}$ is the class of monotone conjunctions over $\{0,1\}^d$, we can reuse the key idea for learning monotone conjunctions in the mistake bound model:

> Start with the conjunction of all variables $x_1 \wedge x_2 \wedge \ldots \wedge x_d$, and kick out any feature which takes the value of 0 in some positive example. If this conjunction is consistent with the negative examples, output the conjunction; otherwise, output `False`.

Analysis. The resulting hypothesis, call it $\hat{f}$, only kicks out features when necessary to be consistent with the positive examples, and thus subject to this constraint, it tries to label as many examples as negative as possible. Therefore, if there is a monotone conjunction consistent with the data, then $\hat{f}$ can never make a mistake on a negative example and hence is consistent with the data. If $\hat{f}$ does make a mistake on a negative example, then no monotone conjunction is consistent with the data, and so the algorithm should indeed output `False`.

Efficiency. The above algorithm is efficient; its runtime is $O(dn)$, since it only needs to inspect each feature of each example at most once.

Extensions. The class of monotone disjunctions, concepts of the form $x_{j_1} \vee x_{j_2} \vee \ldots \vee x_{j_k}$ for $k \in \{0,1,\ldots,d\}$, can be learned in an identical way if we first transform the data as follows:

- Flip the sign of each label.
- For each example x, replace each feature x_j by its negation $\bar{x}_j = 1 - x_j$.

Learning monotone disjunctions over the original data is now equivalent to learning monotone conjunctions on the transformed data.

We leave as a simple exercise the task of efficiently learning (not necessarily monotone) conjunctions over $\{0,1\}^d$ in the consistency model: a conjunction is of the form (e.g.) $\bar{x}_2 \wedge x_4 \wedge x_7$. Another good exercise is to design an algorithm for efficiently learning k-CNF, the class of formulas in *conjunctive normal form* for which each clause has at most k terms. For instance, $x_1 \wedge (\bar{x_1} \vee x_2) \wedge (x_3 \vee \bar{x}_4)$ is an example of a concept in 2-CNF. A related exercise is the task of learning k-DNF, the class of formulas in *disjunctive normal form* for which each clause has at most k terms. For instance, $\bar{x}_1 \vee (x_1 \wedge \bar{x}_2) \vee (\bar{x}_3 \wedge x_4)$ is an example of a concept in 2-DNF.

2.2 Linear separators

Suppose that we are learning homogeneous linear separators in the realizable case, where there exists a hypothesis w^* obtaining margin $\gamma > 0$ over the training sample $(x_1, y_1), \ldots, (x_n, y_n)$. We later will see a simple way to find a w consistent with the data using the Perceptron algorithm. However, rather than using Perceptron, it is possible to try and find such a w directly.

Without loss of generality, assume that w^* has unit norm. Since w^* obtains margin γ over the training sample, it holds that

$$y_j \langle w^*, x_j \rangle \geq \gamma \text{ for all } j \in [n].$$

Therefore, taking $\tilde{w} = \frac{w^*}{\gamma}$, we have

$$y_j \langle \tilde{w}, x_j \rangle \geq 1 \text{ for all } j \in [n].$$

Clearly, this vector $\tilde{w}$ induces a linear separator consistent with the data, and so we need only find some $w \in \mathbb{R}^d$ which satisfies the linear constraints

$$y_j \langle w, x_j \rangle \geq 1 \text{ for all } j \in [n].$$

We can find such a vector using linear programming. A linear program is an optimization problem which has a linear objective function and linear constraints. In our case, we do not have an objective function, and so we simply set the parameter for the objective function to equal the zero vector $\mathbf{0}$. Framed as a linear program, our task becomes

$$\begin{aligned} \underset{w \in \mathbb{R}^d}{\text{minimize}} \quad & \langle \mathbf{0}, w \rangle \\ \text{subject to} \quad & \langle w, y_j\, x_j \rangle \geq 1, \; j = 1, \ldots, n. \end{aligned}$$

There is a rich theory for solving linear programs, which includes efficient (polynomial time) algorithms. Thus, it is possible to learn linear separators efficiently in the consistency model.

References

Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory.* MIT press, 1994.