Machine Learning
Lecture 5

Lecture by Dr. Shai Fine
Typeset by Steven Karas

2017-11-19
Last edited 21:00:04 2017-11-19

Disclaimer These lecture notes are based on the lecture for the course Machine Learning, taught by Dr. Shai Fine at IDC Herzliyah in the fall semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Shai Fine.

Agenda

- Instance based learning
- Generative modeling
- Discriminative classification

1 Classification

Classification problems are decision problems.

Formally, we want to learn a classification model $f : X \to Y$ with a training set $X = \{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$ where the feature vector $x_i \in X$ is a d-dimensional vector where each feature is either numeric or nominal, and the label y_i is either a binary classification, or a multi-class classification.

Global assumption is that the training examples are i.i.d.

Once we have modeled the function, we can predict the classification y for a new instance's feature vector x.

2 Instance based learning

Instance based learning simply stores all the training examples and calculates a prediction based on the stored data.

2.1 k-Nearest-Neighbors

kNN stores the training instances in an efficient data structure for querying based on distance from a provided instance.

kNN is parameterized by k, which is typically odd, to avoid ties.

Formally, kNN is calculating the conditional probability that x is a particular class given that there are $n \leq k$ neighbors of a given class:

$$\Pr(y \mid x) = \frac{|\{(x_i, y_i) \mid y_i = y,\, x_i \in kNN(x)\}|}{|kNN(x)|}$$

Training algorithm For each training vector in the training set: store the training vector.

Prediction algorithm Given a feature vector x, calculate the k neighbors closest to x. Predict the class of x as the most common class of the neighbors.

There are some variants of this, such as outputting the margin as the confidence, or weighting closer neighbors.

Efficient Indexing KD-Trees implement finding the nearest neighbor in $O(\log n)$

Selection of k k is typically chosen to be very small. As k grows, the neighborhood of x grows, which gives us a larger bias (inaccurate estimation). As k shrinks, the neighborhood of x shrinks, which gives us larger variance (unreliable estimation).

Usage considerations kNN trains very fast, doesn't lose information, and can learn complex target functions. However, the query time can be very slow, and can get fooled by irrelevant attributes.

It also doesn't scale well to high dimensional data (20 features at most), and requires lots of training data.

3 Generative modeling

Estimates $\Pr(x \mid C_k)$.

3.1 Naive Bayes

Assume feature independence, and calculate:

$$\Pr(x \mid C_k) \approx \prod_{i=1}^{d} \Pr(x_i \mid C_k)$$

$$\begin{aligned}\Pr(C_k \mid x) &= \frac{\Pr(x \mid C_k)\Pr(C_k)}{\sum_j \Pr(x \mid C_j)\Pr(C_j)} \\ &= \frac{1}{1 + \sum_{j \neq k} \frac{\Pr(x|C_j)\Pr(C_j)}{\Pr(x|C_k)\Pr(C_k)}} \\ &= \frac{1}{1 + \sum_{j \neq k} \frac{\Pr(C_j)}{\Pr(C_k)} \prod_{i=1}^{d} \frac{\Pr(x_i|C_j)}{\Pr(x_i|C_k)}}\end{aligned}$$

This is a bad approximation, but works well enough. The ratio is the only part that matters, and because of the feature independence, the ratio factors nicely.

We then train our decision boundaries by $C = \arg\max_k \Pr(C_k \mid x)$.

Take log, and assume $k = 2$ for simplicity:

$$\ln \frac{\Pr(C_1 \mid x)}{\Pr(C_2 \mid x)} = \ln \frac{\Pr(x \mid C_1)\Pr(C_1)}{\Pr(x \mid C_2)\Pr(C_2)} = \ln \frac{\Pr(C_1)}{\Pr(C_2)} + \sum_{i=1}^{d} \ln \frac{\Pr(x_i \mid C_1)}{\Pr(x_i \mid C_2)}$$

Multinomial distribution

$$\ln \frac{\Pr(C_1 \mid x)}{\Pr(C_2 \mid x)} = \ln \frac{\Pr(C_1)}{\Pr(C_2)} + \sum_{\omega \in \Omega} n_\omega \ln \frac{\Pr(\omega \mid C_1)}{\Pr(\omega \mid C_2)}$$

Where n_ω is the appearances of the symbol ω in x and $\sum_{\omega \in \Omega} n_\omega = |x| = n$.

Gaussians with equal covariance If $\Pr(x \mid C_k) = \mathcal{N}(x \mid \mu_k, \Sigma)$:

We will show this derivation later in the course

$$\ln \frac{\Pr(C_1 \mid x)}{\Pr(C_2 \mid x)} \propto \ln \frac{\Pr(C_1)}{\Pr(C_2)} - x^\intercal \Sigma^{-1}(\mu_1 - \mu_2)$$

All three of these Naive Bayes approximation results in linear decision boundaries in the probability space:

$$f_{\mathbf{w}}(\mathbf{x}) = b + \mathbf{x}^\intercal \mathbf{w}$$

Note that because we independently evaluate the distribution of the classes, it is possible to combine dissimilar distributions, however this leads to some uglier math.

4 Discriminative classification

4.1 Decision Trees

Tree where each internal node represents a decision. Decisions may be univariate, or multivariate. Each decision partitions the feature space into separate branches of the tree. Leaves represent classification labels, or regressions either as an average or a local fit.

Training process Greedy algorithm, find the best split recursively. Tree depth is bound by training parameters.

More details on the training process are available on slides 15-18

5 Linear Classification

5.1 Perceptron

Introduced by Rosenblatt in 1958.

Uses a generic update rule (called the perceptron learning rule):

$$\mathbf{w}^t = \mathbf{w}^{t-1} + \Delta\mathbf{w}^t$$

The algorithm only updates the weight on classification error, with the update proportional to the magnitude of the error multiplied by the learning rate η, typically with a decaying learning rate $\eta \propto \frac{1}{t}$:

$$\Delta\mathbf{w}_j^t = \eta(\hat{y}^t - y^t)x_j^t$$

Effectively, this does gradient descent, and converges on linear subspaces. If the feature space is nonlinear, then it will not converge.

5.2 Support Vector Machines

Linear SVMs find the maximal margin linear classifiers which have a unique solution.