CS 194-129 Deep Neural Networks
Spring 2018 Canny Lecture 3

1 Reading

6.10. Training Logistic Regression

To minimize the logistic regression loss function, we can use batch or stochastic gradient descent. It doesn't matter which initial values we pick for our weights, because the loss function is convex and doesn't have any local minima.

6.11. Support Vector Machines

If we only care about assigning each data point a class, we don't really need to care about modeling probability distributions. Instead, we can just find a "good" decision boundary. In this spirit, SVMs are an attempt to model decision boundaries directly.

Setup: we are given a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$. We would like to find a $(d-1)$-dimensional hyperplane to serve as our decision boundary H. It should separate the $+1$s from the -1s.

Motivation for SVMs

We already have the (much simpler) perceptron classifier for finding these kinds of decision boundaries. Why do we need SVMs? Well, perceptrons fail to find stable solutions if the data is not linearly separable. (Soft-margin SVMs fix this issue by allowing best-fit decision boundaries even when the data is not linearly separable.) Furthermore, even if the data is linearly separable, a perceptron can find infinitely many decision boundaries – some are better than others, of course, but the perceptron cannot distinguish between them. This leads to generalization issues. (Basically, perceptrons have no notion of a margin.)

Hard-Margin SVMs

A **hard-margin SVM** finds a hyperplane H that maximizes the margin, i.e. the minimum distance from H to any of the training points. Intuitively, this allows us to generalize better to unseen data.

Maximizing the margin m means that there exists at least one point on each side of the hyperplane whose distance to the hyperplane is exactly equal to m. These two points are the **support vectors**.

Soft-Margin SVMs

A hard-margin SVM has a unique solution only if the data is linearly separable, and no solution otherwise. (Its constraints depend on being able to draw a hyperplane that separates the $+1$s from the -1s.) Also, hard-margin SVMs are very sensitive to outliers. Soft-margin SVMS have neither of these problems.

A soft-margin SVM modifies the constraints from the hard-margin SVM by allowing some points to violate the margin. Formally, it introduces slack variables ξ_i (one for each training point), enforcing that each point x_i need only be a distance of $1 - \xi_i$ from the separating hyperplane (replacing the previously-hard distance of 1).

1.6. Bias-Variance Tradeoff

For a fixed input x, our measurement Y is a noisy measurement of the true underlying response $f(x)$, i.e. $Y = f(x) + N$ where N is a zero-mean random variable typically represented as a Gaussian distribution. Our goal in regression is to recover the underlying model $f(x)$ as closely as possible. In this section, we would like to form a metric to measure the effectiveness of a hypothesis function h.

If the metric is represented as the expected squared error between the hypothesis and the (noisy) observation $Y = f(x) + N$, we can decompose it into a *squared bias* (how well the average hypothesis over all training sets can approximate the true underlying value $f(x)$), a *variance* (of the hypothesis over all possible training sets), and an *irreducible error*.

Models that are very complex have little bias because on *average* they can fit the true underlying model value $f(x)$ very well. However, on an individual basis they tend to have high bias and be far off from $f(x)$.

Takeaways

- Generally, underfitting is equivalent to high bias, and overfitting correlates to high variance.
- Training error reflects bias but not variance, while test error reflects both.
- As n goes to infinity, variance goes to 0.
- Adding good features will decrease the bias, while adding a bad feature will rarely increase the bias (its coefficient will simply be set to 0).
- Adding a feature will usually increase the variance. Add one only if it decreases bias more than it increases variance!
- Noise in the test set only affects $var(N)$, while noise in the training set affects both bias and variance.
- f is rarely known for real-world data (and the noise model might be incorrect), so typically we can't calculate bias and variance. However, we can test algorithms over synthetic data.

2 Lecture

Recap

Last time, we covered the fundamental different types of ML models: generative and discriminative models. As a refresher, deep networks (the focus of this course) are discriminative models – which may learn more slowly at first but have better asymptotic accuracy than generative models (because the assumptions made by generative models are often violated).

Generative models try to solve a harder problem, and try to represent the data distribution of x as well as the conditional distribution of labels given the data. *But* we don't need to model the data distribution if we just want to predict things!

We also covered loss as a point-to-point measurement of the difference between predictions $\hat{y}$ and actual targets y. A good model should minimize loss on its predictions. *Risk* is expected loss, so it applies to new data as well as existing data. Empirical risk is just an average of the loss measurements across some dataset. ML models attempt to solve the problem of minimizing risk by minimizing empirical risk.

What else? We looked at a few prediction functions $\hat{y} = f(x)$, studying linear regression and logistic regression. In linear regression, we predict a real value with a squared loss. (Squared loss is natural for probabilistic arguments, as it models a Gaussian disturbance in y.) In logistic regression, we predict a binary target (which could be on continuous inputs), and define a cross-entropy loss (basically the negative-log probability of a correct answer).

SVMs

Say we have a 2D classification task: we have points each belonging to one of two classes, and we'd like to draw a linear decision boundary between points belonging to different classes. Each decision boundary can be represented as a weight vector

$$\begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T$$

Each decision function, which encodes a decision boundary, can be described as $f(x) = w_0 + w_1 x_1 + w_2 x_2 > 0$ (read: we will decide that a point belongs to the positive class if $f(x) > 0$). Formally, this will be written as $f(x) = w^T x + b$, where the weight vector is

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix}^T$$

Note that in this formulation, w_0 has become b.

Even if there are multiple decision boundaries that separate the sets, some might be preferable over others. We'd like to **maximize the margin** (the distance between the closest point and the line). Big margins are better because any new, unseen data point has to be at least as far away as the margin [from the rest of its brethren] in order to be misclassified. A large margin implies that new points will have to be far from current data to be incorrectly classified, which is just less likely. In short, a large margin gives us more protection for future data.

How do we maximize the classifier's margin? We need to turn everything into a loss that can formally be minimized. If $\|w\|_2 = 1$, it turns out that $f(x) = w^T x + b$ is a signed measure of the Euclidean distance of a point x from the decision boundary $f(x) = 0$.

Let's try to create a margin of 1 (again, a margin is a region where there are no points). In order for this to happen, it has to be true that $f(x) \geq 1$ for any point that's in the class, and $f(x) \leq -1$ for any point that isn't in the class. We will use a common labeling trick: instead of letting y be either 0 or 1, we will let it be either -1 or 1:

$$y = \begin{cases} 1 & \text{if } x \in C \\ -1 & \text{if } x \notin C \end{cases}$$

Note that C is the positive class.

Now our two inequalities ($f(x) \geq 1$ for $x \in C$, and $f(x) \leq -1$ for $x \notin C$) simplify to one:

$$yf(x) \geq 1$$

We want to measure how much this is violated. (If it's satisfied, there is no loss, because this corresponds to there being no points inside the margin space.) If the constraint is violated, $yf(x)$ is less than 1. We can use the *degree* to which $yf(x)$ is less than 1 as our loss:

$$\max(0, 1 - yf(x))$$

This is called a **hinge loss**.

If we have a whole collection of points, we can use the sum of per-point losses as our total hinge loss:

$$l = \sum_{i=1}^{n} \max(0, 1 - y_i f(x_i))$$

One problem – we've assumed so far that $\|w\| = 1$. We should now make sure that it *is* true. If we scale w with some factor, $y_i f(x_i)$ will be scaled by the same factor. This means that as $\|w\|$ increases, the hinge loss tends to decrease (because a larger $y_i f(x_i)$ means a larger number is subtracted from 1 in the loss expression).

Therefore, minimizing l will generally force $\|w\|$ to be bigger (because a bigger $\|w\|$ will make $y_i f(x_i)$ bigger and drive down the loss). Instead of forcing $\|w\| = 1$ (which is tricky), we can then add a quadratic loss term $\lambda\|w\|^2$ (which is easy to optimize, because we can easily take the derivative). The quadratic term creates a quadratic pressure as a function of the length of w, thereby tending to decrease $\|w\|$.

We can also tune λ to get a more or less aggressive margin. In this way the classifier learns the margin as well as the position of the decision boundary!

Note that correctly-classified points outside of the margin have zero loss, correctly-classified points inside the margin have positive loss $(0, 1]$, and incorrectly-classified points have an even more positive loss $(1, \infty)$! Meanwhile, loss grows with distance from the margin.

Incidentally, logistic regression loss also fits this scheme (except with a smooth hinge instead of a sharp one). So logistic regression can also be used as a soft-margin classifier!

Multiclass Classification

To which of k possible classes does a data point belong? There are two approaches we can take, both of which build multiclass classifiers on top of binary classifiers:

- **One-versus-rest:** Construct k functions $f_i(x)$, each designed to estimate the probability that x is in class i. Train each $f_i(x)$ with labeled data: $y = 1$ if $x \in C$ and $y = 0$ if $x \notin C$. The predicted class label for x will be $\arg\max_i f_i(x)$.
- **One-versus-one:** Construct $\binom{k}{2}$ functions $f_{ij}(x)$ with $i < j \in \{1, ..., k\}$ – one for every pair of classes. Each $f_{ij}(x)$ will be a binary classifier between classes i and j; $f_{ij} > 0$ is a vote for class i, while $f_{ij}(x) < 0$ is a vote for class j. For a new data point x, we can tally the votes from all $\binom{k}{2}$ classifiers and output the class with the highest vote.

We will mainly consider the first paradigm, because it is more scalable (we only need k classifiers). Again we want a notion of a margin; in this case we will take the difference between the score for the correct class $f_y(x)$ and the score for any other class $f_j(x)$. We would like it to be the case that

$$f_y(x) - f_j(x) \geq 1$$

In other words, we would like the winning class score to be very different from all of the other classes' scores. The loss for class j will be

$$\max\left(0, 1 - (f_y(x) - f_j(x))\right)$$

To arrive at the overall loss for a particular input, we will sum over all of the classes j not equal to y:

$$l = \sum_{j \neq y} \max\left(0, 1 - (f_y(x) - f_j(x))\right)$$

To simplify notation, let $f_j(x, w^j)$ be the linear function $f_j(x)$ with weight vector w^j. We can write the full predictor (over all classes) as $f(x, W) = Wx$ where

$$W = \begin{bmatrix} (w^1)^T \\ (w^2)^T \\ \vdots \\ (w^k)^T \end{bmatrix}$$

Now $f(x, W) = Wx$ will give us all of the respective scores for each class.

The loss over all N samples (x_i, y_i) in a dataset is

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max\left(0, 1 - (f(x_i, W)_{y_i} - f(x_i, W)_j)\right)$$

Again we have assumed $\|w^j\|_2 = 1$, and again (since minimizing risk tends to make $\|w^j\|_2$ grow) we can fix this by minimizing an additional regularizing term $\lambda\|W\|_2$:

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max\left(0, 1 - (f(x_i, W)_{y_i} - f(x_i, W)_j)\right) + \lambda\|W\|_2$$

Multiclass Logistic Regression and Softmax

Again we will consider the one-versus-rest paradigm. Assume x is an m-dimensional vector (it can be binary or real-valued). We would like $f_j(x)$ to estimate the probability that x is in class j. Let $s = Wx$, where W is a $k \times m$ weight matrix. Then let

$$f_j(x) = \frac{\exp(s_j)}{\exp(s_1) + \exp(s_2) + ... + \exp(s_k)}$$

Note that $f_j(x)$ – this function from scores to probabilities – is called a **softmax** of the vector $(s_1, ..., s_k)$. Just to be clear, the scores s_i are scalars. Clearly,

$$\sum_{j=1}^{k} f_j(x) = 1$$

meaning we can treat it as a probability.

It's called a "softmax" because the winning s_j, if somewhat larger than the others, will have an exponential that dominates the sum. Then the output $f_j(x)$ over different j's will be close to $(0, ..., 1, ..., 0)$ where the 1 is in the jth position. So it's kind of a maximum-identifying function.

Multiclass logistic regression (taking the softmax of Wx) is the discriminative version of multiclass naive Bayes! If x is a binary vector, then there is a weight matrix W such that the softmax classifier implements the multiclass naive Bayes classifier (making it the optimal classifier under the assumptions of naive Bayes, i.e. that features are independent given the class).

Cross-Validation

We want to use as much data as possible for training (to get the most accurate model), and we also want to use as much data as possible for testing (to get the most accurate measurement). However, there has to be a train/test split. Therefore, we can simply use different test data subsets in different measurements (and the measurements will mostly be independent)!

References

[1] S. Nasiriany, G. Thomas, W. Wang, and A. Yang.
A Comprehensive Guide to Machine Learning.
`http://snasiriany.me/files/ml-book.pdf`