Introduction to Machine Learning **Fall 2017**

Lecture 2: Linear Classification and Logistic Regression

Lecturer: Ethan Fetaya *Scribe: Lily Li*

2.1 From Last Week

Note $p(\mathbf{w})$ is the prior. Then $p(w|data)$ is the most probable model given the data. Convenient prior (conjugate): $p(\mathbf{w}) = \mathcal{N}(0, \sigma_w^2)$. Linear models works when the features are great they are also quite fast (fast to deploy and run on your weak sauce machine). Otherwise the model will fail.

Definition 2.1 *Functions which are linear in the unknown parameter* $\mathbf{w}$ *are called* ***linear models****.*

2.2 Linear Classification

This week we are going to map $\mathbf{x} \in \mathbb{R}^d$ into categorical y which is a finite set S. Usually S is taken to be $\{1, ..., k\}$, $\{0, 1\}$ or $\{-1, 1\}$. We will focus on the last one for now.

The linear model $\hat{y} = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$ outputs a real score. To turn this into a binary decision we will take f to be the threshold function:

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = sign(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

The decision boundary is then the hyperspace defined by $\mathbf{w}$.

$\mathbf{w}^T\mathbf{x} = 0$ is a hyperplane passing through the origin orthogonal to w. Thus

$$\mathbf{w}^T\mathbf{x} + w_0 = 0$$

is the the same hyperplane shifted by w_0. Notice that the decision boundary, $\mathbf{w}$, is invariant under scaling. If the classes can be separated by a hyperplane, then the problem is **linearly separable**.

Reasons that separation might not be possible include:

1. Model is too simple
2. Noise
3. Errors in the data target (missing labels for data)
4. Simple features that do not account for all variation (underlying problem is not linear); you could use a transformation to make data linearly separable
5. Poorly chosen feature parametrization

Learning in the context of this class amounts to find a good decision boundary. In particular we need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary. The quality of our boundary will be defined using the natural loss function *zero-one loss*

$$l_{0-1}(\hat{y}, y) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y} \end{cases}$$

Unfortunately this loss is hard to optimize because it is not continuous.

Another loss function to consider is the *asymmetric binary loss*

$$l_{ABL}(\hat{y}, y) = \begin{cases} \alpha & \text{if } y = 0 \wedge \hat{y} = 1 \\ \beta & \text{if } y = 1 \wedge \hat{y} = 0 \\ 0 & \text{if } y = \hat{y} \end{cases}$$

Our goal is to optimize either loss function, but the problem is NPH and piecewise constant. Thus we will make use of a **surrogate loss** $\hat{l}$ in our optimization instead. Good surrogate losses have the property that: (1) they are easy to optimize — smooth and convex — and (2) they are representative — upper bound $\forall y \forall \hat{y}(y, \hat{y}) leq \hat{l}(y, \hat{y})$ — (since surrogate loss is proportional to original loss). If we just use our continuous l_2 loss it is not representative, because the value of the prediction — not just the sign — is taken into account.

We evaluate the quality of a classifier with **metrics**. Typically we cannot directly optimize for the metrics. The following are possible metrics

1. *Accuracy:* percent of correct predictions, $1 - l_{0-1}(w)$. Problems when the data is very unbalanced (small number of TP).

2. *Recall:* fraction of relevant instances that are retrieved:

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground-truth instances}}.$$

3. *Precision:* fraction of retrieved instances that are correct

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all positive predictions}}.$$

4. *F1 score:* harmonic mean of precision and recall

$$F1 = 2\frac{P \cdot R}{P + R}$$

(Note here TP are the true positives, FN are the false negatives, and FP are the false positives.) There is a trade-off between precision and recall. The **Precision-Recall curve** records this given a particular decision threshold. The **Average Precision (AP)** is the area under the Precision-Recall curve. Increasing both precision and recall is a good idea.

Definition 2.2 ***Receiver Operator Characteristic (ROC)*** *is the trade-off between false-positive-rate (FPR) and true-positive-rate (TPR) using the decision threshold. Note that a better ROC implies a better PR but the opposite is not always the case. The difference can be quite large with unbalanced data.*

2.3 Logistic Regression

Continuing with binary classification with $\{0, 1\}$ labels: we have turned a real score $\mathbf{w}^T\mathbf{x} = w_0 \cdot 1 + \sum_{i=1}^{d} w_i \cdot x_i$ to a binary decision by thresholding.

An alternative first models the probability $P(y = 1|\mathbf{x})$. Then squishes $\mathbf{w}^T\mathbf{x}$ into $[0, 1]$, $p(y = 1|\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$. Then we know that $P(y = -1|\mathbf{x}) = 1 - P(y = 1|\mathbf{w}) = 1 - f(\mathbf{w}^T\mathbf{x})$.

A useful squashing function is the **sigmoid (a.k.a. logistic function** $\sigma(z) = \frac{1}{1+\exp(-z)}$. Nice properties include: differentiable, monotonic increasing, $\sigma(0) = 0.5$, and $lim_{z\to-\infty}\sigma(z) = 0$ and $lim_{z\to\infty}\sigma(z) = 1$. By modifying $\mathbf{w}$ we can change the shape of the sigmoid function. In the 1D case $y = \sigma(w_1 x + w_0)$. w_1 determines the "sharpness" of the increase from zero to one on the y-axis. w_0 shifts the curve along the x-axis.

The decision boundary for logistic regression is $\mathbf{w}^T\mathbf{x} = w_0 + \sum_{j=1}^{d} w_j x_j = 0$. When the output $p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) \geq 0.5$ then this implies that $\mathbf{w}^T\mathbf{x} \geq 0$.

We will use **maximum likelihood** to learn the weights $\mathbf{w} = (w_0, ..., w_d)$ given the probabilistic model. Assume $y \in \{0, 1\}$ and write the probability distribution of each training data point (likelihood) as

$$p(y^{(1)}, ..., y^{(N)}|\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}; w).$$

Assuming the training examples are sampled i.i.d, the *likelihood function* is:

$$L(\mathbf{w}) = p(y^{(1)}, ..., y^{(N)}|\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}; w) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

Never use test data for tuning the hyper-parameters. You are suppose to divide the data into training and validation sets. Use the training data to estimate the weights $\mathbf{w}$ for different values of α. Use the validation data to estimate the best α. Ultimately there is suppose to be three set.

2.4 Gradient Decent

The issue with gradient decent is that calculating the gradient might be quite slow if your data set is huge. The way to deal with this is to randomize your data and take a certain subset of it to calculate your gradient. This method is called **Stochastic Gradient Decent**.

Our goal it to find $\theta^* = \arg\min_\theta f(\theta)$ where $\theta \in \mathbb{R}^n$ is our optimization variable and $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function. Remember, maximizing f is the same as minimizing $-f$. We need to ask: is θ discrete or continuous, what form do constraints on θ take, and is f "well-behaved". θ is the parameters of the model we want to learn. Our goal is to minimize the loss function e.g. if the data are pairs (x, y) then we might want to maximize the likelihood, $P(y|x, \theta)$. This is equivalent to minimize $-\log P(y|x, \theta)$.

We use gradients to find the minimum of f. The gradient is given by $\nabla_\theta f$. Let η be the learning rate and T be the number of iterations. Initialize θ_0 randomly. For $t = 1$ to T:

$$\delta \leftarrow -\eta \nabla_{\theta_{t-1}} f + \alpha \delta_{t-1}$$
$$\theta_t \leftarrow \theta_{t-1} + \delta_t$$

You can do Gradient Descent with Line-Search which first finds a step size value η_t such that $f(\theta_t - \eta_t \nabla_{\theta_{t-1}} < f(\theta_t)$ at each iteration. Note that $\alpha \in [0, 1)$ is the momentum coefficient so that the update takes into account the previous step. Adding this momentum can help speed up convergence.

2.5 Thoughts about the Course

The course material is *not* that difficult — I would even say that it is the same as what Andrew teaches — but there is just *so much* jargon. All of this *maximum likelihood, prior, model, etc* stuff and all of the mathematic notation on the slides makes it both intimidating and difficult to follow. Also *why* is he doing what he is doing? He is taking the logarithm of the likelihood function $L(\mathbf{w})$. But he really doesn't give a good reason for why we do this. There are lots of notational inconsistencies and there is *a lot* of nation. When the first assignment comes out we will really be able to tell how difficult the course is actually going to be. You do have a couple of blind spots: covariance, loss functions, and some linear algebra, but if you can fill in those blanks and get the jargon, you will be fine.

2.6 The Basics

As we have noted, there are some holes in your understanding. Thus here are some tid-bits which might be useful for machine learning.

Definition 2.3 *The **variance** $\mathbf{Var}(X)$ of a random variable X is the square of its standard deviation. Formally it is defined to be*

$$\mathbf{Var}(X) = \mathbf{E}[(X - \mu)^2]$$

Further the variance can be seen as a special case of the covariance $\mathbf{Cov}(X, X)$ where you take the covariance of a random variable with itself. If you take the definition of the variance and play with the math a little you will find that $\mathbf{Var}(X) = \mathbf{E}[X^2] - \mathbf{E}[X]^2$.

Definition 2.4 *The **covariance** measures how closely two values are linearly related as well as the scales of these variables. It is defined as*

$$\mathbf{Cov}(f(x), g(y)) = \mathbf{E}[(f(x) - \mathbf{E}[f(x)])(g(y) - \mathbf{E}[g(y)])]$$