Note 2

02 Perceptrons

by Alvin Wan

With machine learning in general, we learn a prediction function f, which gives us a prediction $\hat{y}$ when given a sample point x, $\hat{y} = f(x)$. We will narrow our focus to a specific problem: consider a question to which we answer "yes" or "no". We can choose to assign all x where $f(x) > 0$ to "yes" and otherwise to "no". We call this problem a **binary classification** problem, and in such a problem, we learn a **decision boundary**, which separates sample points that belong to a class C and those that do not. Formally, the decision boundary is the set of all points x such that $f(x) = 0$, $\{x \in \mathbb{R}^d : f(x) = 0\}$. In the two-dimensional case, this decision boundary is a line separating $x \in \mathbb{R}^d$. In higher dimensions, this decision boundary is a **hyperplane**.

What about other classification problems with $k > 2$ classes? As it turns out, one such option is to run binary classification k times, with the added feature (or bug) that one sample x_i could be tagged with multiple classes.

1 Intuition

Let us begin by gathering some linear algebra intuition. This intuition is critical to understanding the loss functions we introduce later on. Above, we considered a prediction function f. We now consider a specific structure for this prediction function $f(x) = w^T x + \beta$, for $w, x \in \mathbb{R}^d, \beta \in \mathbb{R}$, where w and β are the parameters of the model that we learn during training. They are additionally called **regression coefficients**. As stated before, the decision boundary is then a hyperplane defined by

$$H = \{x \in \mathbb{R}^d : f(x) = w^T x + \beta = 0\}$$

Theorem 1 For some point x not on the decision boundary, $\frac{f(x)}{\|w\|_2}$ is the signed distance d from x to H, where $f(x) = w^T x + \beta$.

Proof: We wish to show $d = w^T x + \beta$. Consider d to be the signed distance from x to H. Let x_H be the vector in H that is "closest" to x. First, since $x_H \in H$, we know

$$w^T x_H + \beta = 0$$

Recall that for any hyperplane H, w is orthogonal to all points in H. Consider the direction of w, $\frac{w}{\|w\|_2}$ multiplied by the distance between x and x_H. This is precisely $x - x_H$: first, they share the same distance, by definition. Second, they share the same direction, since we know the shortest vector between x and H is orthogonal to H. Rearrange to isolate x_H.

$$x - x_H = \frac{w}{\|w\|_2} d$$
$$x_H = x - \frac{w}{\|w\|_2} d$$

Plug this into our first equation to get our final formulation. We then distribute and rearrange. Note $w^T w = \|w\|_2^2$.

$$\begin{aligned} w^T(x - \frac{w}{\|w\|_2} d) + \beta &= 0 \\ w^T x - \frac{\|w\|_2^2}{\|w\|_2} d + \beta &= 0 \\ w^T x + \beta &= d\|w\|_2 \\ d &= \frac{f(x)}{\|w\|_2} \end{aligned}$$

Theorem 2 The distance from the origin of $\mathbb{R}^d$ to the decision boundary is $\frac{\beta}{\|w\|_2}$.

Proof: Given Theorem 1, we can plug in $d = f(\vec{0}) = \frac{1}{\|w\|_2}(w^T \vec{0} + \beta) = \frac{\beta}{\|w\|_2}$

2 Centroid Method

We begin with a simple classifier, called the **centroid method**. We take an average μ_C of all sample points in class C and μ_X of all points not in C. Our prediction function $f(x)$ outputs a negative value if x is closer to μ_X and otherwise outputs a positive value.

$$f(x) = (\mu_C - \mu_X)^T x - \frac{1}{2}\|\mu_C - \mu_X\|_2^2$$

With the above, we take a scaled projection of x along the difference between means $\mu_X - \mu_C$. This is effectively the amount of x that leans towards either mean. Subtracting half the distance between the two means, we get a positive value if x leans closer to C and a negative value otherwise. We can consider another interpretation of the same prediction function.

$$f(x) = (\mu_C - \mu_X)^T (x - \frac{(\mu_C - \mu_X)}{2})$$

We interpret $\frac{(\mu_C - \mu_X)}{2}$ to be the midpoint between the two means. The above function then takes the difference between the sample point x and the midpoint. This is then projected onto the difference $\mu_C - \mu_X$ between means.

3 Risk Function

The **perceptron** is one linear classification method that will always converge to the correct decision boundary for linearly separable data. If the data is not linearly separable, the perceptron algorithm will not converge.

First, we consider a dataset with n sample points $\{x_i\}_{i=1}^n$, where some x_i are in class C, $x_i \in C$. Define the labels y_i to be the following.

$$y_i = \begin{cases} 1 & \text{if } x_i \text{ in class } C \\ -1 & \text{if } x_i \text{ not in class } C \end{cases}$$

Recall from Theorem 1 that the distance from a point x_i to the decision boundary $H = \{x : w^T x + \beta\}$ is $d = w^T x_i + \beta$. For simplicity, we will consider decision boundaries that cross the origin, or where $\beta = 0$. Thus, the distance from x_i to H is $w^T x_i$. Our goal is to achieve the following predictions $\hat{y}_i$ for each x_i, using our model w:

$$\hat{y}_i = \begin{cases} w^T x_i > 0 & \text{in class } C \\ w^T x_i < 0 & \text{if } x_i \text{ not in class } C \end{cases}$$

Note that $\hat{y}_i$ is not necessarily positive and negative in the correct scenarios, for a given model w. The above statement is simply our *goal*. Using these two definitions, we develop the notion of a **loss function**, which evaluates to 0 if the prediction is correct and is otherwise a negative quantity. We can construct a simple loss function like below:

$$L(\hat{y}_i, y_i) = \begin{cases} 0 & \text{if } y_i = \hat{y}_i \\ 1 & \text{if } y_i \neq \hat{y}_i \end{cases}$$

However, we can construct a more clever, continuous loss function. Continuity will be important later, when we take its gradient. We note that if the prediction is correct, y_i and $\hat{y}_i$ are both negative or are both positive. In either case, $y_i \hat{y}_i > 0$. If the prediction is incorrect, y_i and $\hat{y}_i$ have opposite signs, meaning $y_i \hat{y}_i < 0$.

$$L(\hat{y}_i, y_i) = \begin{cases} 0 & \text{if } y_i \hat{y}_i > 0 \\ -y_i \hat{y}_i & \text{if } y_i \hat{y}_i < 0 \end{cases}$$

Moving forward, we will need to consider the **objective function** (a.k.a. **risk function**), which gives us the quantity we are interested in minimizing. For now, our objective function is simply the **loss function** evaluated for all sample points x_i. Below is the objective function, where V is the set of all indices i where $y_i \neq \hat{y}_i$ or equivalently, $y_i \hat{y}_i = w^T x_i < 0$, $V = \{i : y_i w^T x_i < 0\}$.

$$R(w) = \sum_{i=1}^{n} L(\hat{y}_i, y_i) = \sum_{i \in V} -y_i \hat{y}_i = \sum_{i \in V} -y_i w^T x_i \approx \sum_{i \in V} 1 = |V|$$

In other words, the above objective function R approximately counts the number of misclassified points. If $R(w) = 0$, we have achieved 100% accuracy. Otherwise, we take what is called a gradient step.

4 Gradient Descent

In your very first single-variable calculus class, you learned how to solve $\min_x f(x)$: take $\frac{\partial}{\partial x} f(x) = 0$ and solve for x. In your multi-variable calculus class, you learned to repeat for $x \in \mathbb{R}^d$, setting $\nabla f(x) = 0$ and solving for x. In this course, we add another general technique for this **optimization problem** - by employing gradient descent. We will discuss descent methods in more detail later on.

Gradient descent is particularly applicable in scenarios where there is no closed-form solution for x given $\nabla f(x) = 0$. We will apply gradient to the loss function above R to obtain the perceptron algorithm. Consider the gradient of our objective function from above $R(w) = -\sum_{i \in V} y_i w^T x_i$ below:

$$\nabla R(w) = \frac{\partial}{\partial w}(-\sum_{i \in V} y_i w^T x_i) = -\sum_{i \in V} \frac{\partial}{\partial w}(y_i w^T x_i) = -\sum_{i \in V} y_i x_i$$

This gives rise to the following algorithm, where we update w at each iteration with the gradient $\nabla R(w)$ and **step size** ϵ.

Algorithm 1 Running perceptron algorithm with gradient descent

```
function PERCEPTRON(x, y)
    w ← arbitrary non-zero value
    while R(w) > 0 do
        V ← set of indices i s.t. y_i w^T x_i < 0
        w ← w + ε Σ_{i∈V} y_i x_i
    return w
```

However, this method takes $O(d)$ computing $w^T x_i$ for every x_i, where there are up to n x_i. This makes $O(nd)$ per iteration. We can alternatively use **stochastic gradient descent** which samples x_i at random and for perceptrons, also converges to the optimal w^* if gradient descent does. Note that stochastic gradient descent is not always guaranteed to converge.

We briefly return to our original problem, where the decision function has the form $f(x) = w^T x + \beta$ for some non-zero β. As it turns out, we can simply increase the dimension of x so that the decision boundary again passes through the origin. With a decision boundary that passes the origin, we can then apply the perceptron algorithm above. Say our decision boundary satisfied the following condition for $x, w \in \mathbb{R}^d$.

Algorithm 2 Running perceptron algorithm with stochastic gradient descent

```
1: function PERCEPTRON(x, y)
2:     w ← arbitrary non-zero value
3:     while some y_i w^T x_i < 0 do
4:         w ← w + ε y_i x_i
       return w
```

$$f(x) = w^T x + \beta = \begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} + \beta = 0$$

We can then take $x', w' \in \mathbb{R}^{d+1}$ for the following decision boundary:

$$f(x) = w'^T x' = \begin{bmatrix} w_0 & w_1 & \cdots & w_d & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \\ \beta \end{bmatrix} = 0$$

Note that both functions give us the same value of $f(x) = w_0 x_0 + w_1 x_1 + \cdots + w_d x_d + \beta$. As it turns out, the perceptron algorithm fairly slow. Fortunately, we can find solutions much faster using quadratic programming, the topic of our next note.