Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Spring 2023

Probabilities in multi-class

So, we now know our **problem**: we're taking in a data point $x \in \mathbb{R}^d$, and **outputting** one of the classes as a **one-hot vector**.

So, now that we know what sorts of data we're **expecting**, we need to decide on the formats of our **answer**.

We'll be returning a vector of length-k: **one** for each **class**. When we were doing **binary** classification, we estimated the **probability** of the positive class.

So, it should make sense to do the same **here**: for each class, we'll return the estimated **probability** of our data point being in that class.

$$g = \begin{bmatrix} \mathbf{P}\{x \text{ in } C_1\} \\ \mathbf{P}\{x \text{ in } C_2\} \\ \vdots \\ \mathbf{P}\{x \text{ in } C_k\} \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} \tag{1}$$

We need one **additional** rule: the probabilities need to add up to **one**: we should assume our point ends up in some class or **another**.

$$g_1 + g_2 + \ldots + g_k = 1 \tag{2}$$

Concept 1

The different terms of our **multi-class** guess g_i represent the **probability** of our data point being in class C_i.

Because we should assume our data point is in **some** class, all of these probabilities have to **add** to 1.

Let's be careful, though: this is only true for probabilities within a single data point.

Example: Suppose you have two animals (data points).

- It's impossible for the first animal to be **both** 90% cat and 90% dog.
- *But*, there's no issue with the first animal being 90% cat and the second animal being 90% dog.

Clarification 2

It's only true that all of the probabilities for the **same data point** need to add to 1.

If you have $\mathbf{P}\{\text{class 1}\}$ for one data point and $\mathbf{P}\{\text{class 2}\}$ for another data point, those **aren't related**.

So, we want to scale our values so they add to 1: this is called **normalization**. How do we do that?

Well, let's say each class gets a **value** of r_i, before being **normalized**. For now, let's ignore how we got r_i, just know that we have it.

To make the total 1, we'll **scale** our terms by a factor C:

$$C(r_1 + r_2 + ...r_k) = C\left(\sum_{i=1}^{k} r_i\right) = 1 \tag{3}$$

We can get our factor C just by dividing:

$$C = \frac{1}{\sum r_i} \tag{4}$$

We've got our desired g_i now!

$$g = \begin{bmatrix} r_1/\sum r_i \\ r_2/\sum r_i \\ \vdots \\ r_k/\sum r_i \end{bmatrix} \tag{5}$$

Turning sigmoid multi-class

Now, we just need to compute r_i terms to plug in. To do that, we'll see how we did it using sigmoid:

$$g = \sigma(u) = \frac{1}{1 + e^{-u}} \tag{6}$$

This function is 0 to 1, which is good for being a probability.

Just for our convenience, we'll switch to positive exponents: all we have to do is multiply by e^u/e^u.

Negative numbers are easy to mess up in algebra.

$$g = \frac{e^u}{e^u + 1} \tag{7}$$

We'll think of **binary** classification as a special case of **multi-class** classification. The above probability could be thought of as g_1: the chance of our first class.

Concept 3

Binary classification is a **special** case of **multi-class** classification with only **two** classes.

So, we can use it to figure out the **general** case.

So, what was our **second** probability, $1 - g$? This will be our second class, g_2.

$$g_2 = 1 - g = \frac{1}{1 + e^u} \tag{8}$$

This follows an $r_i / (\sum r_i)$ format: the numerators (1 and e^u) add to **equal** the denominator $(1 + e^u)$.

$$g = \begin{bmatrix} 1/(1+e^u) \\ e^u/(1+e^u) \end{bmatrix} \tag{9}$$

How do we **extend** this to **more** classes? Well, 1 and e^u are **different** functions: this a problem. We want to be able to **generalize** to many r_i.

How do they make them **equivalent**? We could say $1 = e^0$. So, we could treat both terms as **exponentials**!

$$g_1 = \frac{e^u}{e^0 + e^u} \tag{10}$$

We can do this for an **arbitrary** number of terms. We'll treat them as **exponentials**, just like for e^u and e^0

$$g_i = \frac{r_i}{\sum r_j} = \frac{e^{u_i}}{\sum e^{u_j}} \tag{11}$$

Now, we have a template for expanding into higher dimensions!

Our Linear Classifiers

What are each of those u_i terms? When we were doing **binary classification**, we used a **linear regression** function to help generate the probability:

Remember that $u(x)$ is not a probability yet: we used a sigmoid to turn it *into* a probability.

$$u(x) = \theta^T x + \theta_0 \tag{12}$$

Now, we want multiple probabilities. So, we create multiple different functions u_i: k different linear regression models (θ, θ_0). We'll represent each vector as $\theta_{(i)}$.

$$\theta_{(1)} = \begin{bmatrix} \theta_{1(1)} \\ \theta_{2(1)} \\ \vdots \\ \theta_{d(1)} \end{bmatrix} \qquad \theta_{(2)} = \begin{bmatrix} \theta_{1(2)} \\ \theta_{2(2)} \\ \vdots \\ \theta_{d(2)} \end{bmatrix} \qquad \theta_{(k)} = \begin{bmatrix} \theta_{1(k)} \\ \theta_{2(k)} \\ \vdots \\ \theta_{d(k)} \end{bmatrix} \tag{13}$$

Each of these models could be seen as a "different perspective" of our data point: what about that data point is prioritized (large θ_i magnitudes), how do we bias the result (θ_0)?

This "perspective" we call $\theta_{(i)}$ will tell us if our data point is "closer" to the class it represents. And we compute the result with:

$$u_1(x) = \theta_{(1)}^T x + \theta_{0(1)} \qquad u_2(x) = \theta_{(2)}^T x + \theta_{0(2)} \qquad u_k(x) = \theta_{(k)}^T x + \theta_{0(k)} \tag{14}$$

In the last section, we emphasized that we can only use $\sum p_i = 1$ for the probabilities of a **single** data point. Based on this, we'll focus on only one data point.

Clarification 4

In this section, x represents only **one data point** $x^{(i)}$.

Softmax treats each data point **individually**, so it's easier to not group them together.

Having all these separate equations for θ_i is tedious. Instead, we can combine them all into a $(d \times k)$ **matrix**.

k classes, so we need k classifiers. We'll stack them side-by-side like how we stacked multiple data points to create X.

$$\theta = \begin{bmatrix} \theta_{(1)} & \theta_{(2)} & \cdots & \theta_{(k)} \end{bmatrix} = \begin{bmatrix} \theta_{1(1)} & \theta_{1(2)} & \cdots & \theta_{1(k)} \\ \theta_{2(1)} & \theta_{2(2)} & \cdots & \theta_{2(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d(1)} & \theta_{d(2)} & \cdots & \theta_{d(k)} \end{bmatrix} \tag{15}$$

And our constants, θ_0, in a $(k \times 1)$ matrix:

$$\theta_0 = \begin{bmatrix} \theta_{0(1)} \\ \theta_{0(2)} \\ \vdots \\ \theta_{0(k)} \end{bmatrix} \tag{16}$$

Concept 5

We can combine **multiple classifiers** $\Theta_{(i)} = \left(\theta_{(i)}, \theta_{0(i)}\right)$ into large **matrices** θ and θ_0 to compute **multiple** outputs u_i at the **same** time.

This will put all of our terms into a $(1 \times k)$ vector u.

$$u(x) = \theta^T x + \theta_0 = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} \tag{17}$$

Softmax

We now have all the pieces we need. Our **linear regression** for each class:

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} = \theta^T x + \theta_0 \tag{18}$$

The **exponential** terms, to get **logistic** behavior:

$$r_i = e^{u_i} \tag{19}$$

The **averaging** to get probability = 1:

$$g = \begin{bmatrix} r_1 / \sum r_i \\ r_2 / \sum r_i \\ \vdots \\ r_k / \sum r_i \end{bmatrix} \tag{20}$$

And so, our multiclass function is...

Definition 6

The **softmax function** allows us to calculate the probability of a point being in each class:

$$g = \begin{bmatrix} e^{u_1} / \sum e^{u_i} \\ e^{u_2} / \sum e^{u_i} \\ \vdots \\ e^{u_k} / \sum e^{u_i} \end{bmatrix}$$

Where

$$u_i(x) = \theta_{(i)}^T x + \theta_{0(i)} \tag{21}$$

If we are forced to make a **choice**, we choose the class with the **highest probability**: we return a **one-hot encoding**.

A side comment: Sigmoid vs. Softmax

Let's pause real quick and clarify something.

Usually, we expect to use **softmax** if we have more than 2 classes, because that's what we built it for.

However, this isn't always the case.

There's another aspect we haven't focused on: **softmax** represents k different classes/events. These classes are assumed to be **mutually exclusive**: you can't be in multiple at the same time.

In other words, they are **disjoint**.

Definition 7

If two events are **disjoint**, they **can't** happen at the **same time**.

If n events are **disjoint**, only **one** of them can happen at a time.

Example: We usually wouldn't classify an animal as both a cat and a dog: it's either one or the other.

When events are disjoint, their probabilities are separate:

Concept 8

If two events are **disjoint**, then they have **separate** probabilities: there's no overlap. Since $\mathbf{P}\{A \cap B\} = 0$, we can say:

$$\mathbf{P}\{A \cup B\} = \mathbf{P}\{A\} + \mathbf{P}\{B\}$$

If we have **every** event and they're all **disjoint**, then their probabilities sum to 1.

$$\sum_i p_i = 1 \tag{22}$$

Example: If the weather options are rain, cloudy, and sunny, and you have to only choose one, you should expect that:

$$\mathbf{P}\{\text{Rain}\} + \mathbf{P}\{\text{Cloudy}\} + \mathbf{P}\{\text{Sunny}\} = 1 \tag{23}$$

But this only makes sense **if** events can't happen at the same time.

But, what if they can? For example: there might be k different people we could find in an **image**. But, there can be **multiple** people in the same image!

So, it doesn't make sense to assume that each event is **mutually exclusive**: multiple events can all happen, which just isn't an option with softmax!

The solution: we still have **probabilities**, so we just use **one sigmoid per class**.

Clarification 9

Softmax is used when each of our k classes is **disjoint** (mutually exclusive).

However, if they aren't, then we **can't** use softmax.

Instead, we use k **sigmoid** functions: one for each of our k classes. We're using **binary classification** on each class separately.

The i^{th} sigmoid tells us how likely the **data point** is to be in the i^{th} class.

Example: We might have an algorithm figuring out which **products** a customer might want. They might want **multiple**, so we can't treat them as disjoint.

In this case, each product is a class, and we determine the result based on the matching sigmoid.