Multiclass Classification

Running example Suppose we want to train a multiclass classifier to classify sentences as being headlines of one of several types. We have the possible labels $\mathcal{Y} =$ HEALTH, SPORTS, SCIENCE.

Furthermore, take as an example the sentence:

too many drug trials, too few patients

Finally, suppose our feature space is a set of three indicators:
[I[sentence contains *drug*], I[sentence contains *patients*], I[sentence contains *baseball*]]
which take the values [1, 1, 0] on the example above.

Different weights In the "different weights" version of multiclass perceptron, we define our features as a function $\mathbf{f}(\mathbf{x})$ which returns a "base set" of features (three features in the above examples). Each class $y \in \mathcal{Y}$ has a distinct weight vector $\mathbf{w}_y$ that scores how likely an example is to be in that class. Prediction consists of taking the dot product of each weight vector with the features and returning the highest scoring class:

$$\arg\max_y \mathbf{w}_y^\top \mathbf{f}(\mathbf{x})$$

In total, we end up with a number of parameters equal to the number of features in $\mathbf{f}$ times the number of classes (3 x 3 = 9 total parameters for our running example).

For neural networks, we can think of the last layer of the neural network as a matrix consisting of stacked $\mathbf{w}_y$ vectors that essentially implement a "different weights" logistic regression computation.

Different features The Eisenstein book uses another view of classification where we think of each possible label as inducing a different set of features. Our feature vector in our running example now consists of 9 features obtained by *conjoining* each base feature with the label:

$\mathbf{f}(\mathbf{x}, y)$ = [I[sentence contains *drug* $\wedge$ y = HEALTH], I[sentence contains *patients* $\wedge$ y = HEALTH], I[sentence contains *baseball* $\wedge$ y = HEALTH], I[sentence contains *drug* $\wedge$ y = SPORTS], I[sentence contains *patients* $\wedge$ y = SPORTS], I[sentence contains *baseball* $\wedge$ y = SPORTS], I[sentence contains *drug* $\wedge$ y = SCIENCE], I[sentence contains *patients* $\wedge$ y = SCIENCE], I[sentence contains *baseball* $\wedge$ y = SCIENCE]]

Now, on this example, we have

$\mathbf{f}(\mathbf{x} =$ *too few drug trials, too few patients*, $y =$ HEALTH) = [1, 1, 0, 0, 0, 0, 0, 0, 0]

$\mathbf{f}(\mathbf{x} =$ *too few drug trials, too few patients*, $y =$ SPORTS) = [0, 0, 0, 1, 1, 0, 0, 0, 0]

$\mathbf{f}(\mathbf{x} =$ *too few drug trials, too few patients*, $y =$ SCIENCE) = [0, 0, 0, 0, 0, 0, 1, 1, 0]

Under this framework, we now have a single weight vector $\mathbf{w}$. This vector can be thought of as containing blocks of features corresponding to scores associating each feature with each class label. This is equivalent to simply concatenating the $\mathbf{w}_y$ vectors from the "different weights" view.

We find the highest scoring class by extracting features for each class in turn and taking the dot product with the feature vector:

$$\arg\max_y \mathbf{w}^\top \mathbf{f}(\mathbf{x}, y)$$

Multiclass Perceptron See Algorithm 3 in Section 2.3.1 the textbook. This is the "different features" form of perceptron exactly as we discussed in lecture (with $\boldsymbol{\theta}$ instead of $\mathbf{w}$).

Multiclass Logistic Regression Definition (under different features):

$$P(y|\mathbf{x}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{x}, y'))}$$

Section 2.5 in the book presents this algorithm, but is different in two major ways from the in-class version. First, we do not include regularization, since it usually makes a fairly minor difference and traditional notions of regularization don't apply to our deep learning models. Second, the notation is a bit different than what we've used.

Using our notation, the gradient of the logistic regression loss (negative log likelihood) on an example $(\mathbf{x}^{(i)}, y^{(i)})$ is:

$$-\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \sum_{y' \in \mathcal{Y}} P(y'|\mathbf{x}^{(i)}) \mathbf{f}(\mathbf{x}^{(i)}, y')$$

where $P(y|\mathbf{x}^{(i)})$ is the probability the *model* assigns to class y. (This term also shows up in the binary logistic regression case.) As always, note that the weights are updated by subtracting off the gradient. You can verify that if the model is nearly correct, the update ends up being very close to zero, whereas if almost all of the mass is on the wrong label, you have an update very close to the multiclass perceptron update.