TTIC 31190: Natural Language Processing
Hints for Assignment 2: Text Classification

Instructor: Kevin Gimpel

1 Worked-out Example for Training a Linear Text Classifier

The assignment asks you to implement and experiment with simple ways of building text classifiers based on linear models. Consider a classifier defined by the function classify for label set $\mathcal{L} = \{0, 1, 2\}$:

$$\text{classify}(\boldsymbol{x}, \mathbf{w}) = \underset{y \in \{0,1,2\}}{\text{argmax}} \ \text{score}(\boldsymbol{x}, y, \mathbf{w}) \tag{1}$$

where $\boldsymbol{x}$ is a textual input, y is an output class label, and the parameters are contained in the parameter (weight) vector $\mathbf{w}$. The score function is defined:

$$\text{score}(\boldsymbol{x}, y, \mathbf{w}) = \sum_i w_i f_i(\boldsymbol{x}, y) \tag{2}$$

where each f_i is a feature function and w_i is the corresponding weight of the feature function.

We'll work through an example of minimizing the perceptron loss by applying the stochastic subgradient descent (SSD) update for a single training example $\langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle$ where $\boldsymbol{x}^{(1)}$ = "great film" and $y^{(1)} = 2$. Let's assume we have two features in our model:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = 2] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = 0] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

(Note: if we were using a feature count cut-off of 1, then this would not be the feature set that would result from a training set that included $\langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle$, but for purposes of this worked-out example of the SSD update rule for the perceptron loss, we won't worry about where the feature set came from. We'll just proceed assuming that the features above are the only two features in the model.)

Then we also have two weights w_1 and w_2, both of which we will initialize to be 0.

Learning proceeds by iterating through all training examples. When processing an example $\langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle$, we will compute the subgradient of the loss with respect to the weight vector $\mathbf{w}$ and update each entry in $\mathbf{w}$ based on its subgradient component. The assignment pdf derives the following update rule for a single weight w_j (written below for our single training example):

$$w_j \leftarrow w_j + 0.01 f_j(\boldsymbol{x}^{(1)}, y^{(1)}) - 0.01 f_j(\boldsymbol{x}^{(1)}, \text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}))$$

where we have plugged in our fixed step size $\eta = 0.01$. Let's start with w_1. Its subgradient component consists of the difference in feature values between two y's. First, compute $f_1(\boldsymbol{x}^{(1)}, y^{(1)})$. Since $\boldsymbol{x}^{(1)}$ contains "great" and $y^{(1)} = 2$, $f_1(\boldsymbol{x}^{(1)}, y^{(1)}) = 1$.

Next, we have to compute $f_1(\boldsymbol{x}^{(1)}, \text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}))$. To compute this, we first need to compute $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w})$, which is shown in Eq. 1. This requires computing the score function (Eq. 2) for each

possible label. Here things get slightly odd, because we initialized both weights w_1 and w_2 to be 0. So, the scores for all labels will be 0. But since the classify function has to return a y, we need to break ties somehow. Let's break ties arbitrarily by simply choosing the y with the lowest integer value among those with equal scores. This means that $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}) = 0$. Plugging this into the update rule, we then have to compute the feature value $f_1(\boldsymbol{x}^{(1)}, \text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w})) = f_1(\boldsymbol{x}^{(1)}, 0)$. What's the value of this feature function? Even though $\boldsymbol{x}^{(1)}$ contains "great", the label being passed to the feature function is 0, which doesn't match the label that the feature is looking for, so $f_1(\boldsymbol{x}^{(1)}, 0) = 0$.

So, the update for weight w_1 is:

$$w_1 \leftarrow 0 + 0.01 \times 1 - 0.01 \times 0$$

The new value of w_1 will be 0.01. This feature "fired" (i.e., had nonzero value) with the gold standard label, but did not fire for the predicted label (the output of $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w})$), so the SSD update made the feature weight slightly more positive than it had been.

Now let's do the update for w_2. Its subgradient component again consists of the feature function value difference between two y's. First, compute $f_2(\boldsymbol{x}^{(1)}, y^{(1)})$. While the input $\boldsymbol{x}^{(1)}$ contains "great", the given label $y^{(1)} = 2$ does not match what the feature f_2 is looking for, so $f_2(\boldsymbol{x}^{(1)}, y^{(1)})$ returns 0.

For the next term in the update, we have to compute $f_2(\boldsymbol{x}^{(1)}, \text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}))$. We already computed $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w})$ above and obtained the predicted label 0.[1] So we then compute the feature value $f_2(\boldsymbol{x}^{(1)}, 0)$, which is 1.

So the update for weight w_2 is:

$$w_2 \leftarrow 0 + 0.01 \times 0 - 0.01 \times 1$$

and the new value of w_2 will be -0.01. This feature fired on the predicted label but not the gold standard label, so the SSD update made the feature weight slightly more negative than it had been.

(Note: We will only update weights for features that fire for the current $\boldsymbol{x}$ when paired with either the gold standard label or the predicted label. For all other features, the update will be zero. You can speed up computation by avoiding the loop over all features for a given $\boldsymbol{x}$ and only considering the features that could possibly be nonzero for it (e.g., if the textual input does not contain "great", you can skip over f_1 and f_2).)

Let's denote our new updated weight vector by $\mathbf{w}^{(new)}$. Consider what happens if we process this same example $\langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle$ again, i.e., if we do another SSD update on this training example. Let's first compute $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}^{(new)})$. Below are the scores for the three labels:

$$\begin{aligned}
\text{score}(\boldsymbol{x}^{(1)}, 0, \mathbf{w}^{(new)}) &= w_2 = -0.01 \\
\text{score}(\boldsymbol{x}^{(1)}, 1, \mathbf{w}^{(new)}) &= 0 \\
\text{score}(\boldsymbol{x}^{(1)}, 2, \mathbf{w}^{(new)}) &= w_1 = 0.01
\end{aligned}$$

Therefore, $\text{classify}(\boldsymbol{x}^{(1)}, \mathbf{w}^{(new)}) = 2$. Since this predicted label is in fact the correct label, the weight updates for both weights will be 0!

[1] Note: we could recompute the classify function using our updated value for w_1, but this means that we would have different parameter updates depending on the order in which we updated the weights. As far as I know, this is not commonly done. I'd suggest computing classify as the first thing you do when processing a training example and then using the saved output of classify for updating all parameters for that training example (as described in this document).