Homework 2: Sentiment

Brandon McKinzie

Problem 1: Building Intuition

(a). *Suppose we run stochastic gradient descent, updating the weights according to*

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w})$$

once for each of the four examples in the order given above. After the classifier is trained on the given four data points, what are the weights of the six words ("pretty", "good", "bad", "plot", "not", "scenery") that appear in the above reviews? Use $\eta = .5$ as the step size and initialize $\mathbf{w} = [0, ..., 0]$. Assume that $\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) = 0$ when the margin is exactly 1.

Let $w_i :=$ words$[i]$, where words =["pretty", "good", "bad", "plot", "not", "scenery"] (zero-indexed).

The sparse feature vectors for each of the four examples are:

$$\phi(x_1) = \{\text{pretty} : 1, \text{bad} : 1\} \tag{1}$$
$$\phi(x_2) = \{\text{good} : 1, \text{plot} : 1\} \tag{2}$$
$$\phi(x_3) = \{\text{not} : 1, \text{good} : 1\} \tag{3}$$
$$\phi(x_4) = \{\text{pretty} : 1, \text{scenery} : 1\} \tag{4}$$

As seen in lecture, we can use the chain rule to obtain

$$\nabla_{\boldsymbol{w}} L(x_i, y_i) = -\mathbb{1}\left[\boldsymbol{w} \cdot \phi(x_i) y_i < 1\right] \phi(x_i) y_i \tag{5}$$

We can then evaluate the gradients in order as follows:

1.

$$\nabla_{\boldsymbol{w}} L(x_1, y_1) = -\mathbb{1}\left[(w_0 + w2)(-1) < 1\right] \phi(x_1)(-1) \tag{6}$$
$$= \phi(x_1) \tag{7}$$
$$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_0 - (0.5)\phi(x_1) \tag{8}$$
$$= -0.5 \cdot \{\text{pretty} : 1, \text{bad} : 1\} \tag{9}$$

2.

$$\nabla_{\boldsymbol{w}} L(x_2, y_2) = -\mathbb{1}\left[(w_1 + w_3)(1) < 1\right] \phi(x_2)(1) \tag{10}$$
$$= -\phi(x_2) \tag{11}$$
$$\boldsymbol{w}_2 \leftarrow \boldsymbol{w}_1 + (0.5)\phi(x_2) \tag{12}$$
$$= 0.5 \cdot \{\text{pretty} : -1, \text{good} : 1, \text{bad} : -1, \text{plot} : 1\} \tag{13}$$

3.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(x_3, y_3) &= -\mathbb{1}\left[(0+0.5)(-1) < 1\right]\phi(x_3)(-1) && (14)\\
&= \phi(x_3) && (15)\\
\boldsymbol{w}_3 &\leftarrow \boldsymbol{w}_2 - (0.5)\phi(x_3) && (16)\\
&= 0.5 \cdot \{\text{pretty}: -1, \text{good}: 0, \text{bad}: -1, \text{plot}: 1, \text{not}: -1\} && (17)
\end{aligned}$$

4.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(x_4, y_4) &= -\mathbb{1}\left[(-0.5+0)(1) < 1\right]\phi(x_4)(1) && (18)\\
&= -\phi(x_4) && (19)\\
\boldsymbol{w}_4 &\leftarrow \boldsymbol{w}_3 + (0.5)\phi(x_4) && (20)\\
&= 0.5 \cdot \{\text{pretty}: 1, \text{good}: 0, \text{bad}: -1, \text{plot}: 1, \text{not}: -1, \text{scenery}: 1\} && (21)
\end{aligned}$$

Which gives us the answer:

$$\boldsymbol{w} = 0.5 \cdot \{\text{pretty}: 1, \text{good}: 0, \text{bad}: -1, \text{plot}: 1, \text{not}: -1, \text{scenery}: 1\} \qquad (22)$$

(b). *Create a small labeled dataset of four mini-reviews using the words "not", "good", and "bad", where the labels make intuitive sense. Each review should contain one or two words, and no repeated words. Prove that no linear classifier using word features can get zero error on your dataset.*

Remember that this is a question about classifiers, not optimization algorithms; your proof should be true for any linear classifier, regardless of how the weights are learned. After providing such a dataset, propose a single additional feature that we could augment the feature vector with that would fix this problem. (Hint: think about the linear effect that each feature has on the classification score.)

Define our map from feature to zero-based index as { bad: 0, good: 1, not: 2 }. My four mini-reviews, along with their associated $\boldsymbol{w} \cdot \phi(x_i)$, are as follows:

1. (-1) not good; $\boldsymbol{w} \cdot \phi(x_0) = w_1 + w_2$
2. (+1) good; $\boldsymbol{w} \cdot \phi(x_1) = w_1$
3. (+1) not bad; $\boldsymbol{w} \cdot \phi(x_2) = w_0 + w_2$
4. (-1) bad; $\boldsymbol{w} \cdot \phi(x_3) = w_0$

Proceed with proof by contradiction: assume that there exists a linear word classifier that can correctly classify all the 4 examples above. In other words, assume my classifier gets a loss value of zero for all examples. This would require that it satisfies the following system of inequalities (in order of my aforementioned data points):

$$1 + w_1 + w_2 \leq 0 \Rightarrow w_2 \leq -1 - w_1 \tag{23}$$
$$1 - w_1 \leq 0 \Rightarrow w_1 \geq 1 \tag{24}$$
$$1 - w_0 - w_2 \leq 0 \Rightarrow w_2 \geq 1 - w_0 \tag{25}$$
$$1 + w_0 \leq 0 \Rightarrow w_0 \leq -1 \tag{26}$$

Plugging in the inequalities for w_0 and w_1 leads to a contradiction:

$$w_2 \leq -2 \tag{27}$$
$$w_2 \geq 2 \tag{28}$$

Therefore, there does not exist any linear classifier using word features that correctly classifies all four of my data points.

Problem 2: Predicting Movie Ratings

(a). *Write out the expression for Loss*$(x, y, \mathbf{w})$.

$$L(x, y, \boldsymbol{w}) = ||y - \sigma(\boldsymbol{w} \cdot \phi(x))||^2 \tag{29}$$

(b). *Compute the gradient of the loss with respect to* $\mathbf{w}$. *Hint: you can write the answer in terms of the predicted value* $p = \sigma(\mathbf{w} \cdot \phi(x))$

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(x, y, \boldsymbol{w}) &= 2(y - \sigma(\boldsymbol{w} \bullet \phi(x)))\nabla_{\boldsymbol{w}}(y - \sigma(\boldsymbol{w} \bullet \phi(x))) && (30)\\
&= 2(y - \sigma(\boldsymbol{w} \bullet \phi(x)))(-1)\sigma(\boldsymbol{w} \bullet \phi(x))(1 - \sigma(\boldsymbol{w} \bullet \phi(x)))\nabla_{\boldsymbol{w}}(\boldsymbol{w} \bullet \phi(x)) && (31)\\
&= -2(y - \sigma(\boldsymbol{w} \bullet \phi(x)))\sigma(\boldsymbol{w} \bullet \phi(x))(1 - \sigma(\boldsymbol{w} \bullet \phi(x)))\phi(x) && (32)\\
&= -2(y - p)p(1 - p)\phi(x) && (33)
\end{aligned}$$

(c). *Suppose there is one datapoint (x, y) with some given $\phi(x) and y = 1$. Can you choose a* $\mathbf{w}$ *to make the magnitude of the gradient of the loss with respect to* $\mathbf{w}$ *arbitrarily small (i.e., minimize the magnitude of the gradient and make it asymptotically approach some value)? If so, how small? Can the magnitude of the gradient ever be exactly zero? You are allowed to make the magnitude of* $\mathbf{w}$ *arbitrarily large.*

Hint*: try to understand intuitively what is going on and the contribution of each part of the expression. If you find yourself doing too much algebra, you're probably doing something suboptimal.*

Motivation*: the reason why we're interested in the magnitude of the gradients is because it governs how far gradient descent will step. For example, if the gradient is close to zero when* $\mathbf{w}$ *is very far from the optimum, then it could take a long time for gradient descent to reach the optimum (if at all). This is known as the vanishing gradient problem when training neural networks.*

The gradient for the specified data point is

$$\begin{aligned} \nabla_{\boldsymbol{w}} L(x, 1, \boldsymbol{w}) &= -(1-p)p(1-p)\phi(x) \quad (34) \\ &= -p(1-p)^2\phi(x) \quad (35) \end{aligned}$$

If we can make $p{=}\sigma(\boldsymbol{w} \cdot \phi(x))$ either arbitrarily close to 0 or to 1, then the gradient will approach zero. We can do this by setting

$$w_i = \lim_{c \to \infty} c \cdot \operatorname{sign}(\phi_i(x)) \quad (36)$$

with the convention that $\operatorname{sign}(0) := +1$. This would make $p \to 1$ and thus $\nabla_{\boldsymbol{w}} L \to 0$.

(d). *Assuming the same data point as above, what is the largest magnitude that the gradient can take? Leave your answer in terms of $\|\phi(x)\|$.*

$$\begin{aligned}
\frac{\partial \nabla_{\boldsymbol{w}} L(x,1,\boldsymbol{w})}{\partial p} &= \frac{\partial}{\partial p}\left[-p(1-p)^2\phi(x)\right] && (37)\\
&= -\left[(1-p)^2 - 2p(1-p)\right]\phi(x) && (38)\\
&= (1-p)(3p-1)\phi(x) && (39)
\end{aligned}$$

which has zeros at $p = 1$ and $p = 1/3$. We already know that $p = 1$ corresponds to the gradient being zero. Taking another gradient confirms that $p = 1/3$ is concave up (while $p = 1$ is concave down), meaning that at $p = 1/3$ the gradient is furthest from the origin, with magnitude:

$$\begin{aligned}
||\nabla_{\boldsymbol{w}} L(x,1,\boldsymbol{w})\Big|_{p=1/3}|| &= ||-\frac{1}{3}(1-\frac{1}{3})^2\phi(x)|| && (40)\\
&= \frac{1}{3}\frac{4}{9}||\phi(x)|| && (41)\\
&= \frac{4}{27}||\phi(x)|| && (42)
\end{aligned}$$

(e). *The problem with the loss function we have defined so far is that is it is non-convex, which means that gradient descent is not guaranteed to find the global minimum, and in general these types of problems can be difficult to solve. So let us try to reformulate the problem as plain old linear regression. Suppose you have a dataset* **D** *consisting of (x,y) pairs, and that there exists a weight vector* **w** *that yields zero loss on this dataset. Show that there is an easy transformation to a modified dataset* $\mathbf{D}' of (x, y')$ *pairs such that performing least squares regression (using a linear predictor and the squared loss) on* $\mathbf{D}'$ *converges to a vector* $\mathbf{w}^*$ *that yields zero loss on* $\mathbf{D}'$. *Concretely, write an expression for* y' *in terms of* y *and justify this choice. This expression should not be a function of* **w**.

If there exists a linear predictor of the transformed dataset that can obtain the same (zero) loss, then we can assert

$$\begin{aligned} y &= \frac{1}{1+e^{-\boldsymbol{w}\cdot\phi(x)}} && (43)\\ &= \frac{1}{1+e^{-y'}} && (44)\\ 1+e^{-y'} &= \frac{1}{y} && (45)\\ e^{-y'} &= \frac{1-y}{y} && (46)\\ -y' &= \ln\left(\frac{1-y}{1-y}\right) && (47)\\ y' &= \ln\left(\frac{y}{1-y}\right) && (48) \end{aligned}$$

This is known as the *logit transformation.*

Problem 3: Sentiment Classification

(a). *When you run the grader.py on test case 3b-2, it should output a weights file and a error-analysis file. Look through some example incorrect predictions and for five of them, give a one-sentence explanation of why the classification was incorrect. What information would the classifier need to get these correct? In some sense, there's not one correct answer, so don't overthink this problem. The main point is to convey intuition about the problem.*

1. **Sentence**: *home alone goes hollywood , a funny premise until the kids start pulling off stunts not even steven spielberg would know how to do . besides , real movie producers aren't this nice .* **Explanation**: the model overly emphasized the positiveness of words like "funny" (0.39) compared to words like "arent́" (0.11), while also giving a lot of positive weight to words that should be neutral (if not given context) such as “start” (0.27).
2. **Sentence**: *a heady , biting , be-bop ride through nighttime manhattan , a loquacious videologue of the modern male and the lengths to which he'll go to weave a protective cocoon around his own ego .* **Explanation**: many neutral words like "a", "his", ".", etc. were labeled as negative. This is a consequence of it being a simple bag of unigrams model.
3. **Sentence**: *it's painful to watch witherspoon's talents wasting away inside unnecessary films like legally blonde and sweet home abomination , i mean , alabama .* **Explanation**: It didn't realize that "sweet home abomination" was a mock-reference to a name, and the weight of "sweet" (0.57) dwarfed the rest of the sentence, resulting in an incorrectly "positive" label. The model could have benefited from being able to recognize references to titles here, or entities in general.
4. **Sentence**: *wickedly funny , visually engrossing , never boring , this movie challenges us to think about the ways we consume pop culture .* **Explanation**: the model labeled both "never" and "boring" as individually very negative words, and this ultimately caused it to incorrectly label the whole sentence as negative. The model would have greatly benefited from bigrams here, i.e. understanding the "never boring" together is a positive statement.
5. **Sentence**: *patchy combination of soap opera , low-tech magic realism and , at times , ploddingly sociological commentary .* **Explanation**: the interesting thing here is that every single one of the negative cues ("patchy", "low-tech", "ploddingly") were all assigned a weight of 0 and thus not contributing to the prediction at all. The words "magic" and "realism" were both heavily positive and ended up determining the prediction. The model would've benefited from understanding the negative connotation of the three aforementioned words.

In general, the classifier may have had a better chance of getting these correct if it took into consideration context (e.g. by using n-grams instead of bag of words).

(f). *Run your linear predictor with feature extractor extractCharacterFeatures. Experiment with different values of n to see which one produces the smallest test error. You should observe that this error is nearly as small as that produced by word features. How do you explain this?*

For me, $n = 5$ produced the smallest test error of about 0.2718. I tried $n \in [1..10]$. The most likely reason that error increases beyond this point is that the features become more specific to the training set, and we are more likely to see n-grams that we've never seen before in the test set for large n. It's possible that $n = 5$ is a "sweet spot" because (a) it is large enough to capture negations like "not", while (b) small enough that it's still basically just grabbing individual words (thus able to share features with the test set). Since it is extremely common to say "not X" in order to negate the word "X", having $n \approx 5$ means the model is at least capable of recognizing negations.

Problem 4: K-Means Clustering

Suppose we have a feature extractor ϕ that produces 2-dimensional feature vectors, and a toy dataset $\mathcal{D}_{train} = \{x_1, x_2, x_3, x_4\}$ with

$$\phi(x_1) = [1, 0] \tag{49}$$
$$\phi(x_2) = [1, 2] \tag{50}$$
$$\phi(x_3) = [3, 0] \tag{51}$$
$$\phi(x_4) = [2, 2] \tag{52}$$

(a) *Run 2-means on this dataset until convergence. Please show your work. What are the final cluster assignments z and cluster centers μ? Run this algorithm twice with the following initial centers:*

$$\mu_1 = [2, 3] \text{ and } \mu_2 = [2, -1]$$

$$\mu_1 = [0, 1] \text{ and } \mu_2 = [3, 2]$$

Initial Centroids: (2, 3) and (2, -1)

First, compute the closest centroid to each $\phi(x_i)$, and assign the index of the closest centroid to z_i. I'll break ties by picking the winner uniformly at random. The distances shown below are the squared euclidean distances. The argmin's are 1-indexed (so they align with the μ_i).

$$z_1 = \arg\min[10, 2] = 2 \tag{53}$$
$$z_2 = \arg\min[2, 10] = 1 \tag{54}$$
$$z_3 = \arg\min[10, 2] = 2 \tag{55}$$
$$z_4 = \arg\min[1, 9] = 1 \tag{56}$$

Now, update the centroids.

$$\mu_1 = \frac{1}{2}\left[(1, 2) + (2, 2)\right] = \left(\frac{3}{2}, 2\right) \tag{57}$$
$$\mu_2 = \frac{1}{2}\left[(1, 0) + (3, 0)\right] = (2, 0) \tag{58}$$

Update assignments.

$$z_1 = \arg\min[4.25, 1] = 2 \tag{59}$$
$$z_2 = \arg\min[0.25, 5] = 1 \tag{60}$$
$$z_3 = \arg\min[6.25, 1] = 2 \tag{61}$$
$$z_4 = \arg\min[0.25, 4] = 1 \tag{62}$$

The assignments have converged to the above values, and with $\mu_1 = \left(\frac{3}{2}, 2\right)$ and $\mu_2 = (2, 0)$.

Initial Centroids: (0, 1) and (3, 2)

First, compute the closest centroid to each $\phi(x_i)$, and assign the index of the closest centroid to z_i.

$$z_1 = \arg\min[2, 8] = 1 \tag{63}$$
$$z_2 = \arg\min[2, 4 = 1 \tag{64}$$
$$z_3 = \arg\min[10, 4] = 2 \tag{65}$$
$$z_4 = \arg\min[5, 1] = 2 \tag{66}$$

Now, update the centroids.

$$\mu_1 = \frac{1}{2}\left[(1, 0) + (1, 2)\right] = (1, 1) \tag{67}$$
$$\mu_2 = \frac{1}{2}\left[(3, 0) + (2, 2)\right] = \left(\frac{5}{2}, 1\right) \tag{68}$$

Update assignments.

$$z_1 = \arg\min[1, 3.25] = 1 \tag{69}$$
$$z_2 = \arg\min[1, 3.25] = 1 \tag{70}$$
$$z_3 = \arg\min[5, 1.25] = 2 \tag{71}$$
$$z_4 = \arg\min[2, 1.25] = 2 \tag{72}$$

The assignments have converged to the above values, and with $\mu_1 = (1, 1)$ and $\mu_2 = \left(\frac{5}{2}, 1\right)$.

(c). *Sometimes, we have prior knowledge about which points should belong in the same cluster. Suppose we are given a set S of example pairs (i,j) which must be assigned to the same cluster. For example, suppose we have 5 examples; then* $S = \{(1,5),(2,3),(3,4)\}$ *says that examples 2, 3, and 4 must be in the same cluster and that examples 1 and 5 must be in the same cluster. Provide the modified k-means algorithm that performs alternating minimization on the reconstruction loss:*

$$\sum_{i=1}^{n} \|\mu_{z_i} - \phi(x_i)\|^2$$

where μ_{z_i} *is the assigned centroid for the feature vector* $\phi(x_i)$.

Recall that alternating minimization is when we are optimizing two variables jointly by alternating which variable we keep constant.

Ultimately, each point will "belong" to a group of points that must be in the same cluster. Since we should allow for S only including a subset of the data, also let any point not mentioned in S be in its own group. In the example above, the two groups are $\{(1,5),(2,3,4)\}$. To compute the assignments z_i, we must now include all points x_s that are in the same group as x_i. Let G_i denote the group of point indices that must be in the same cluster as x_i (note that this includes x_i). The modified assignment step is then

$$z_i = \arg\min_{k\in[1..K]} \sum_{s\in G_i} ||\mu_k - \phi(x_s)||^2 \tag{73}$$

The next step, where we re-compute the centroids μ, is unchanged. It is still the average over the points that were assigned to it in the previous assignment step.

(d). *What is the advantage of running k-means multiple times on the same dataset with the same K, but different random initializations?*

K-means is not guaranteed to converge to a global optimum, but rather a local optimum. By running it multiple times with different random initializations, we are likely to obtain different results. This allows us to choose the result that had the lowest error.

(e). *If we scale all dimensions in our initial centroids and data points by some factor, are we guaranteed to retrieve the same clusters after running k-means (i.e. will the same data points belong to the same cluster before and after scaling)? What if we scale only certain dimensions? If your answer is yes, provide a short explanation. If it is no, provide a counterexample.*

First, the question seems to implicitly assume that by "data points" we mean the individual $\phi(x_i)$ (not the x_i themselves), otherwise we wouldn't be able to say anything here (ϕ is an arbitrary black box as far as we're concerned).

If we scale everything by a constant factor c, then we get the following reconstruction loss, assignment step, and centroid update:

$$\begin{aligned}
L &= \sum_{i=1}^{n} ||c\phi(x_i) - c\mu_{z_i}||^2 && (74)\\
&= c^2 \sum_{i=1}^{n} ||\phi(x_i) - \mu_{z_i}||^2 && (75)\\
z_i &= \arg\min_{k\in[1..K]} ||c\phi(x_i) - c\mu_k||^2 && (76)\\
&= \arg\min_{k\in[1..K]} ||\phi(x_i) - \mu_k||^2 && (77)\\
\mu_k &= \frac{1}{|\{i : z_i = k\}|} \sum_{i:z_i=k} c\phi(x_i) && (78)\\
&= \frac{c}{|\{i : z_i = k\}|} \sum_{i:z_i=k} \phi(x_i) && (79)
\end{aligned}$$

and we see that the resulting cluster assignments z_i will be the same. The argmin is not affected by a constant factor, and the other equations are just scaled by c or c^2.

If we only scale certain dimensions, then the assignments are not guaranteed to stay the same. If each dimension d has a different factor c_d, then that factor essentially controls how "important" it is for us to minimize the distance along that dimension. Dimensions with larger c_d will contribute more on average to the argmin, and therefore the optimization will care more about minimizing those dimensions' distances.