CS221 Fall 2015 Homework 1

SUNet ID: jaigupta
Name: Jai Gupta

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) Given the four examples, we have the following word list for our vocabulary

$$words = \{pretty, bad, good, plot, not, scenery\}$$

Initially $\mathbf{w} = [0, 0, 0, 0, 0, 0]$.

- Example 1: "pretty bad", y = -1

$$\begin{aligned}
\phi(x) &= \{pretty : 1, bad : 1\} \\
Loss_{hinge}(x, y, \mathbf{w}) &= max\{0, 1 - \mathbf{w}\phi(x)y\} && = 1 \\
\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) &= -\phi(x)y \\
&= [1, 1, 0, 0, 0, 0] \\
\mathbf{w} &= \mathbf{w} - \eta\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) \\
&= [0, 0, 0, 0, 0, 0] - 0.5[1, 1, 0, 0, 0, 0] \\
&= [-0.5, -0.5, 0, 0, 0, 0]
\end{aligned}$$

- Example 2: "good plot", y = +1

$$\begin{aligned}
\phi(x) &= \{good : 1, plot : 1\} \\
Loss_{hinge}(x, y, \mathbf{w}) &= max\{0, 1 - \mathbf{w}\phi(x)y\} && = 1 \\
\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) &= -\phi(x)y \\
&= [0, 0, -1, -1, 0, 0] \\
\mathbf{w} &= \mathbf{w} - \eta\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) \\
&= [-0.5, -0.5, 0, 0, 0, 0] - 0.5[0, 0, -1, -1, 0, 0] \\
&= [-0.5, -0.5, 0.5, 0.5, 0, 0]
\end{aligned}$$

- Example 3: "not good", y = -1

$$
\begin{aligned}
\phi(x) &= \{not: 1, good: 1\} \\
Loss_{hinge}(x, y, \mathbf{w}) &= max\{0, 1 - \mathbf{w}\phi(x)y\} \qquad = max\{0, 1 + [-0.5, -0.5, \\
&= 1.5 \\
\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) &= -\phi(x)y \\
&= [0, 0, 1, 0, 1, 0] \\
\mathbf{w} &= \mathbf{w} - \eta \nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) \\
&= [-0.5, -0.5, 0.5, 0.5, 0, 0] - 0.5[0, 0, 1, 0, 1, 0] \\
&= [-0.5, -0.5, 0, 0.5, -0.5, 0]
\end{aligned}
$$

- Example 4: "pretty scenery", $y = +1$

$$
\begin{aligned}
\phi(x) &= \{pretty: 1, scenery: 1\} \\
Loss_{hinge}(x, y, \mathbf{w}) &= max\{0, 1 - \mathbf{w}\phi(x)y\} \qquad = max\{0, 1 - [-0.5, - \\
&= 1.5 \\
\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) &= -\phi(x)y \\
&= [-1, 0, 0, 0, 0, -1] \\
\mathbf{w} &= \mathbf{w} - \eta \nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) \\
&= [-0.5, -0.5, 0.5, 0.5, 0, 0] - 0.5[-1, 0, 0, 0, 0, -1] \\
&= [0, -0.5, 0, 0.5, -0.5, 0.5]
\end{aligned}
$$

(b) Let us take the following example set:
examples = {"good", "bad", "not good", "not bad"}
labels = {1, -1, -1, 1}
A linear classifier on the unigrams will have the features for count of "good", "bad", "not" and can also include a bias term. Let the weights corresponding to these four features be w_1, w_2, w_3, w_4. Since we want to get zero error, the four equations from the four examples above would be:

$$w_1 + w_4 = 1 \tag{1}$$

$$w_2 + w_4 = -1 \tag{2}$$

$$w_1 + w_3 + w_4 = -1 \tag{3}$$

$$w_2 + w_3 + w_4 = 1 \tag{4}$$

If there exists any linear classifier, then it should satisfy all the above four equations. But from equation 1 - equation 2, we get $w_1 - w_2 = 2$. Additionally, from equation 3 - equation 4, we get $w_1 - w_2 = -2$. Since the two are conflicting, we cannot find any linear classifier that can satisfy all the four equations.

In order for a linear classifier to be able to correctly classify all the examples, we will need to provide more than just unigram count features since they are proving to be insufficient in this case. This will increase the dimensionality of the problem space thereby allowing us to find a linear classifier that satisfies our needs. One such example would be to add bigram counts alongwith the existing unigram counts as features. For this problem, instead of adding all possible bigrams, just adding the feature for count of bigram "not good" would suffice.

Problem 2

(a) $Loss(x, y, \mathbf{w}) = (\sigma(\mathbf{w} \cdot \phi(x)) - y)^2$

(b) Let us calculate $\frac{\partial \sigma(z)}{\partial z}$. Once we have that, we can use chain rule to calculate the full derativative of $Loss(x, y, \mathbf{w})$.

$$\begin{aligned}
\frac{\partial \sigma(z)}{\partial z} &= \frac{\partial (1 + e^{-z})^{-1}}{\partial z} \\
&= -\frac{1}{(1 + e^{-z})^2} \frac{\partial e^{-z}}{\partial z} \\
&= \frac{e^{-z}}{(1 + e^{-z})^2} \\
&= \frac{e^{-z} + 1 - 1}{(1 + e^{-z})^2} \\
&= \frac{1}{1 + e^{-z}} - \frac{1}{(1 + e^{-z})^2} \\
&= \sigma(z) - \sigma^2(z) \\
&= \sigma(z)(1 - \sigma(z))
\end{aligned}$$

$$\begin{aligned}
\nabla_{\mathbf{w}} Loss(x, y, \mathbf{w}) &= 2(\sigma(\mathbf{w} \cdot \phi(x)) - y) \frac{\partial(\sigma(\mathbf{w} \cdot \phi(x)) - y)}{\partial \mathbf{w}} \\
&= 2(\sigma(\mathbf{w} \cdot \phi(x)) - y)\sigma(\mathbf{w} \cdot \phi(x))(1 - \sigma(\mathbf{w} \cdot \phi(x))\phi(x)
\end{aligned}$$

(c) Since $y = 1$, we can rewrite the above expression as:

$$\begin{aligned}
\nabla_{\mathbf{w}} Loss(x, y, \mathbf{w}) &= 2(\sigma(\mathbf{w} \cdot \phi(x)) - 1)\sigma(\mathbf{w} \cdot \phi(x))(1 - \sigma(\mathbf{w} \cdot \phi(x))\phi(x) \\
&= -2\phi(x)(1 - \sigma(\mathbf{w} \cdot \phi(x))^2 \sigma(\mathbf{w} \cdot \phi(x))
\end{aligned}$$

If the magnitude of $\mathbf{w}$ is very large, then we can have $\mathbf{w}$ such that $\mathbf{w} \cdot \phi(x)$ will tend towards $\pm\infty$.

- In case of $-\infty$, $\sigma(\mathbf{w} \cdot \phi(x))$ will tend towards 0^+. Thereby, the overall product will tend towards 0^+ (assuming $\phi(x)$ non-zero is finite).

- In case of $+\infty$, $\sigma(\mathbf{w}\cdot\phi(x)$ will tend towards 1^- . So, $1-\sigma(\mathbf{w}\cdot\phi(x)$ will tend towards 0^+. Hence, in this case too, the overall product tends towards 0^+ (assuming $\phi(x)$ is non-zero inite).

Hence the limit tends to 0^+, but it will never be zero except for the case when $\phi(x)$ itself is zero.

Note that in the above solution, superscript +/- on the top right is used to denote whether the limit is approaching from the left or the right side.

(d)

$$||\nabla_{\mathbf{w}} Loss(x,y,\mathbf{w})|| = 2||\phi(x)|| * ||(1-\sigma(\mathbf{w}\cdot\phi(x))^2\sigma(\mathbf{w}\cdot\phi(x))||$$

Lets start with finding the max of function $f(z) = \sigma(z)(1-\sigma(z))^2$

$$\begin{aligned}
f(z) &= \sigma(z)(1-\sigma(z))^2 \\
f'(z) &= (1-\sigma(z))^2 * \sigma(z)(1-\sigma(z)) + \sigma(z) * 2(1-\sigma(z)) * -\sigma(z)(1-\sigma(z)) \\
&= \sigma(z)(1-\sigma(z))^2(1-\sigma(z)-2\sigma(z)) \\
&= \sigma(z)(1-\sigma(z))^2(1-3\sigma(z))
\end{aligned}$$

Putting $f'(z)=0$, we get $\sigma(z)=\frac{1}{3}$. At this value, $f(z)=\frac{4}{27}$. We know this is a maxima since this is the only point with $f'(z)=0$ and $f(z)$ has smaller values at other values of z (e.g., 0). Hence, the max value of $||\nabla_{\mathbf{w}} Loss(x,y,\mathbf{w})||$ is $\frac{8||\phi(x)||}{27}$.

(e) From $\mathbf{D}$, we know that the following expression is valid for all examples $(x_i,y_i)\epsilon\mathbf{D}$

$$y_i = \sigma(\mathbf{w}\cdot\phi(x_i)) \tag{5}$$

Additionally, from $\mathbf{D}'$, we know that:

$$\begin{aligned}
y_i' &= \mathbf{w}^*\cdot\phi(x_i) \\
\sigma(y_i') &= \sigma(\mathbf{w}^*\cdot\phi(x_i))
\end{aligned} \tag{6}$$

Comparing equations 5 and 6, if we take $\mathbf{w}^*$ same as $\mathbf{w}$, then we can find an easy transformation from each y_i to y_i' by setting:

$$\sigma(y_i') = y_i$$

Solving for y' from y:

$$\begin{aligned}
\sigma(y') &= y \\
(1+e^{-y'})^{-1} &= y \\
e^{-y'} &= \frac{1}{y} - 1 \\
-y' &= log(\frac{1}{y} - 1) \\
y' &= log(\frac{y}{1-y})
\end{aligned}$$

Problem 3

(a) in submission.py

(b) in submission.py

(c) in submission.py

(d) (1) rain is a small treasure , enveloping the viewer in a literal and spiritual torpor that is anything but cathartic .
Truth: 1, Prediction: -1 [WRONG]
Explanation: The word *torpor* has a positive meaning in this context which has been scored heavily negative. This primarily brought down the score for the prediction.

Words should be scored differently based on the context they are part of. Instead of one hot vectors, word embedding have become pretty popular in NLP tasks. An extension to them are contextual word embedding such as BERT. These can be used as input features along with the one-hot vector to provide some contextual information about the words.

(2) patchy combination of soap opera , low-tech magic realism and , at times , ploddingly sociological commentary .
Truth: -1, Prediction: 1 [WRONG]
Explanation: Words *magic* and **realism** do not have positive meaning in this context, but are scored extremely positive.

We have already taken a look at contextual word embeddings as a possible solution. That will work for this case too. Another possible solution to this problem is to use bigrams as features. But bigrams might be too infrequent, so another way is to supply weights of the individual tokens from the bigram and combine them in some way (e.g., multiply) to create additional features.

(3) . . . standard guns versus martial arts cliche with little new added .
Truth: -1, Prediction: 1 [WRONG]
Explanation: *new* is scored very positively, but in this context it actually is qualified with *little* which negates its meaning.

Yet again, providing contextual information as a feature would help. In this case, using the bigram features as mentioned above should work great since here too we have an adjective exists that changes the meaning of the word *new.*

(4) even during the climactic hourlong cricket match , boredom never takes hold.
Truth: 1, Prediction: -1 [WRONG]
Explanation: Words *boredom*, *never* and *hold* have a positive meaning in this context, but are scored pretty negatively.

Additional features that will help provide the contextual meaning of the words will help here too. Other than that, another way that has been found useful is to parse sentences into a dependency tree structure (See Stanford sentiment treebak – https://nlp.stanford.edu/sentiment/treebank.html) and perform inferences in a bottom to top manner on the tree.

(5) this may be the dumbest, sketchiest movie on record about an aspiring writer's coming-of-age.
Truth: -1, Prediction: 1 [WRONG]
Explanation: It seems like *sketchiest* and *dumbest* are words that did not occur in the tranining set. These two terms had a pretty strong indication of the review being negative, but the predictor failed to capture it and assigned them weight 0 as these were not present in the training dataset.

Out of vocabulary words have always been a problem for NLP tasks and one methodology that has proved very successful is the use of character n-gram based features. This is same as the *extractCharacterFeatures* feature extractor used in the assignment. Since these feature extractors can match portions of words, they deal better with out of vocabulary words. For examples, the model can learn the score for *dumb* and *est* from other words and use that to approximate the score for the word *dumbest.*

(e) in submission.py

(f) After running the features extractor for different values of n, the best results we found were at $n = 5$. This had 0 test error, while the dev error was 0.27068. This is a bit better than the word based feature extractor which produced a test error of 0.02842 while a test error of 0.27687.
The reason we think that character based feature extractor work better is because it is able to work better with out of vocabulary words. For example, if *dumber* was present in the training dataset, but *dumbest* is present in the test dataset (which was not present in train dataset), character based feature extractor can still match partial words and give good approximations for the sentiment.

An example is: "Though slow , a pretty-sincerely-awesomest movie.".
I ran it through the predictor with word feature extractor and it assigned negative score to *slow* and 0 to *pretty-sincerely-awesomest* thereby leading to a negative overall score.
When using the char feature extractor ($n = 5$), I found the predictor correctly mark it as a positive review with some of the key scoring feature being *"aweso"*, *"esome"* and *"wesom"*.

Problem 4

(a) • μ_1 = (2, 3), μ_2 = (2, -1)

Evaluating example to cluster assignments:

	$dist^2(x_i, \mu_1)$	$dist^2(x_i, \mu_2)$	cluster assignment
x_1	10	2	2
x_2	2	10	1
x_3	10	2	2
x_4	1	9	1

Updating centroids:
$\mu_1 = ((1, 2) + (2, 1))/2 = (1.5, 2)$
$\mu_2 = ((1, 0) + (3, 0))/2 = (2, 0)$

Updating example to cluster assignments:

	$dist^2(x_i, \mu_1)$	$dist^2(x_i, \mu_2)$	cluster assignment
x_1	4.25	1	2
x_2	0.25	5	1
x_3	6.25	1	2
x_4	0.25	4	1

The assignments do not change, which means that convergence has been reached.

- $\mu_1 = (0, 1)$ $\mu_2 = (3, 2)$

  Evaluating example to cluster assignments:

	$dist^2(x_i, \mu_1)$	$dist^2(x_i, \mu_2)$	cluster assignment
x_1	2	8	1
x_2	2	4	1
x_3	10	4	2
x_4	5	1	2

Updating centroids:
$\mu_1 = ((1, 0) + (1, 2))/2 = (1, 1)$
$\mu_2 = ((3, 0) + (2, 2))/2 = (2.5, 1)$

Updating example to cluster assignments:

	$dist^2(x_i, \mu_1)$	$dist^2(x_i, \mu_2)$	cluster assignment
x_1	1	3.25	1
x_2	1	3.25	1
x_3	5	1.25	2
x_4	2	1.25	2

The assignments do not change, which means that convergence has been reached.

(b) in submission.py

(c) Our objective function still remains the same when calculated over individual points which is:

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

Let us split S into smaller subset of points $S = \{S_1, S_2, ...S_m\}$ where all points in any subset must belong to the same cluster. Hence we can rewrite the objective as:

$$\sum_{i=1}^{m} \sum_{j=1}^{||S_i||} ||\mu_{z_i} - \phi(s_j)||^2$$

Here μ_{z_i} denotes the centroid of the cluster that subset S_i belongs to. Also s_j denotes the j_{th} point in S_i.

For alternating minimization, we initialize the K centroids using random selection of K points from the original set of points. After that we alternate between the following two steps until convergence.

- For each subset S_i, find the centroid μ_k that minimizes the sum of distance from all points in S_i.

$$\arg\min_k \sum_{j=1}^{||S_i||} ||\mu_k - \phi(s_j)||^2$$

This gives us the assignment of centroid for every subset. Since the sum of the cost over all subsets is same as the original cost function over individual points, this objective function works.

- Once we have the assignments, we can update the centroids using the average of all points assigned to it. This step is same as the original k-means algorithm on individual points since the overall cost function is still the same.

(d) k-means algorithm as shows in Problem 3.a uses a greedy approach and can easily converge to a local minima. This local minima is dependent on the initialization of the centroids. Hence, if we run the same algorithm multiple times, we might be able to extract different local minima and discover better ones. It is also possible to find the global minima during these trials and the probability of discovering may increase with the increase in the number of trials.

(e) Yes, we are guaranteed to retrieve the same cluster if all data points and the inital centroids are scaled by the same factor.

Lets say that factor is λ. While performing cluster assignments, we assign the points to the centroid with least distance. Since all distances will scale by same factor λ, argmin will still return the same result.
When calculating the mean of the assigned points to update the centroid, the mean will also scale by the same factor λ.
Hence overall, the final centroids we get will also be λ times the scenario without scaling.

Case when scaling only certain dimenstions:
Let us start with the following dataset:

$$x_1 = (2, 2)$$
$$x_2 = (2, -2)$$
$$x_3 = (-2, 2)$$
$$x_4 = (-2, -2)$$
$$\mu_1 = (1, 1)$$
$$\mu_2 = (-1, -1)$$

Now consider these two cases:

- *Case 1:* Scale dimension 0 by a factor of 100.
The four points will be (200, 2), (200, -2), (-200, 2) and (-200, -2). Also the centroids will be (100, 1) and (-100, -1). It is easy to say that (200, 2) and (200, -2) will group with (100, -1) and rest with the other centroid due to dimension 0 dominating the distance metric due to scaling by 100. The algorithm will converge with clusters: $\{(x_1, x_2), (x_3, x_4)\}$.
- *Case 2:* Scale dimension 1 by a factor of 100.
The four points will be (2, 200), (2, -200), (-2, 200) and (-2, -200). Also the centroids will be (1, 100) and (-1, -100). It is easy to say that (2, 200) and (-2, 200) will group with (1, 100) and rest with the other centroid due to dimension 1 dominating the distance metric due to scaling by 100. The algorithm will converge with clusters: $\{(x_1, x_3), (x_2, x_4)\}$.

Hence, scaling can lead to different convergence clusters. Note that even though we performed two different scaling factors, dimension 1 of case #1 can basically just be multiplied by 10,000 to produce same clustering as case #2.