Sentiment - Stanford CS 221 Fall 2017-2018

Rohit Apte

2017-10-06

1 Problem 1: Warmup

We'll warm up with the following set of four mini-reviews, each labeled positive (+1) or negative (1):
1. (-1) pretty bad
2. (+1) good plot
3. (-1) not good
4. (+1) pretty scenery

Each review x is mapped onto a feature vector $\phi(x)$, which maps each word to the number of occurrences of that word in the review. For example, the first review maps to the (sparse) feature vector $\phi(x) = \{pretty : 1, bad : 1\}$. Recall the definition of the hinge loss:
$Loss_{hinge}(x, y, w) = max\,\{0, 1 - w \cdot \phi(x)y\}$
where y is the correct label

a Suppose we run stochastic gradient descent, updating the weights according to $w \leftarrow w - \eta \nabla_w Loss_{hinge}(x, y, w)$ once for each of the four examples in order. 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 $\eta = 1$ as the step size and initialize $w = [0, ..., 0]$. Assume that $\nabla_w Loss_{hinge}(x, y, w) = 0$ when the margin is exactly 1.

Lets label each review as $x^{(1)}$ to $x^{(4)}$ and each label as $y^{(1)}$ to $y^{(4)}$.
We will vectorize the words by alphabetic order (bad,good,not,plot,pretty,scenery).
Then
$\phi(x)^{(1)} = [1, 0, 0, 0, 1, 0], y^{(1)} = -1$
$\phi(x)^{(2)} = [0, 1, 0, 1, 0, 0], y^{(2)} = +1$
$\phi(x)^{(3)} = [0, 1, 1, 0, 0, 0], y^{(3)} = -1$
$\phi(x)^{(4)} = [0, 0, 0, 0, 1, 1], y^{(4)} = +1$
$w = [0, 0, 0, 0, 0, 0]$
$\eta = 1$

$$\nabla_w = \begin{cases} [0, \ldots, 0] \text{ if } w \cdot \phi(x)y <= 1 \\ -\phi(x)y \text{ otherwise} \end{cases}$$

Then we have
$\nabla_w = -\phi(x)^{(1)}y^{(1)} = [1, 0, 0, 0, 1, 0]$
$w = w - \eta\nabla_w = [-1, 0, 0, 0, -1, 0]$

for case 2
$\nabla_w = -\phi(x)^{(2)}y^{(2)} = [0, -1, 0, -1, 0, 0]$

$w = w - \eta \nabla_w = [-1, 1, 0, 1, -1, 0]$
for case 3
$\nabla_w = -\phi(x)^{(3)} y^{(3)} = [0, 1, 1, 0, 0, 0]$
$w = w - \eta \nabla_w = [-1, 0, -1, 1, -1, 0]$

for case 4
$\nabla_w = -\phi(x)^{(4)} y^{(4)} = [0, 0, 0, 0, -1, -1]$
$w = w - \eta \nabla_w = [-1, 0, -1, 1, 0, 1]$

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.

If we use the following mini reviews
(-1) bad
$(+1)$ not bad
$(+1)$ good
(-1) not good
If we plot these vectors with axes representing good, bad and not, then the vectors for "good" fall on the good axis, the vectors for "bad" fall on the bad axis. The vector "not good" falls in between the not and good axis. The vector "not bad" falls between the not and bad axis. There is no single hyperplace that can separate these 4 points (see image below).

To fix this we need to do the kernel trick - transform the data so it is separable by a hyperplane. A single additional feature that would fix this is an interaction between features (e.g. (good-bad)*not).

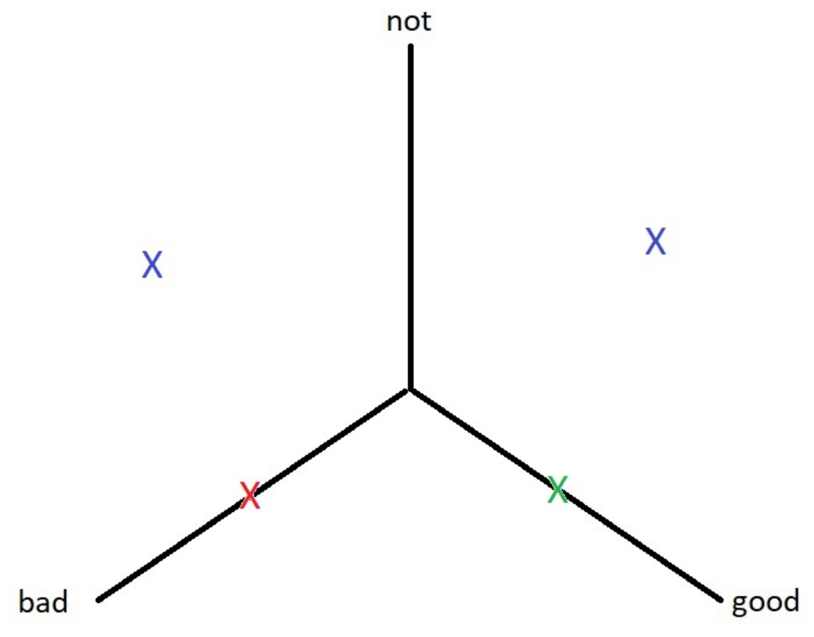

Figure 1: Plotting review vectors

2 Problem 2: Suppose that we are now interested in predicting a numeric rating for movie reviews. We will use a non-linear predictor that takes a movie review x and returns $\sigma(w\dot{\phi}(x))$, where $\sigma(z) = (1 + e^z)^1$ is the logistic function that squashes a real number to the range $[0, 1]$. Suppose that we wish to use the squared loss. For this problem, assume that the movie rating y is a real-valued variable in the range $(0, 1)$.

a Write out the expression for $Loss(x, y, w)$.

$Loss(x, y, w) = (\sigma(w \cdot \phi(x)) - y)^2 = (\frac{1}{1+e^{-w\cdot\phi(x)}} - y)^2$

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

We know that $\frac{d(e^{ax})}{dx} = ae^{ax}$. We have
$Loss(x, y, w) = (\sigma(w \cdot \phi(x)) - y)^2 = (\frac{1}{1+e^{-w\cdot\phi(x)}} - y)^2$
Let $p = \sigma(w \cdot \phi(x)) = \frac{1}{1-e^{w\cdot\phi(x)}}$. Then $Loss(x, y, w) = (p - y)^2$

Therefore we have
$\frac{d(Loss(x,y,w))}{dw} = \frac{d(p-y)^2}{dw} = 2(p - y)\frac{d(p)}{dw}$
Solving for $\frac{d(p)}{dw}$ we have
$\frac{d(p)}{dw} = \frac{d(\left[1+e^{-w\cdot\phi(x)}\right]^{-1})}{dw} = -1\left(\frac{1}{1+e^{-w\cdot\phi(x)}}\right)^{-2}\left(-\phi(x)e^{-w\cdot\phi(x)}\right)$
$= \phi(x)\left(\frac{1}{1+e^{-w\cdot\phi(x)}}\right)^{-2}\left(e^{-w\cdot\phi(x)}\right) = \phi(x)\left[\frac{1}{1+e^{-w\cdot\phi(x)}}\frac{e^{-w\cdot\phi(x)}}{1+e^{-w\cdot\phi(x)}}\right]$
$= \phi(x)\left[\frac{1}{1+e^{-w\cdot\phi(x)}}\frac{1+e^{-w\cdot\phi(x)}-1}{1+e^{-w\cdot\phi(x)}}\right] = \phi(x)\left[\frac{1}{1+e^{-w\cdot\phi(x)}}\left(1 - \frac{1}{1+e^{-w\cdot\phi(x)}}\right)\right]$
$= \phi(x)p(1 - p)$
Combining the two we have
$\frac{d(Loss(x,y,w))}{dw} = 2\phi(x)(p - y)p(1 - p)$

c Assuming $y = 1$, what is the smallest magnitude that the gradient can take? That is, find a way to set w to make $\nabla Loss(x, y, w)$ as small as possible. You are allowed to let the magnitude of w go to infinity. Hint: try to understand intuitively what is going on and the contribution of each part of the expression. If you find 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 w is very far from the origin, 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 smallest magnitude the gradient can hypothetically take is 0 (that is the limit as $-w \cdot \phi(x) \to -\infty$ or $-w \cdot \phi(x) \to \infty$).

$Loss(x, y, w) = (p - y)^2$
$\frac{d(Loss(x,y,w))}{dw} = 2\phi(x)(p - y)p(1 - p)$
If $y = 1$, and we are solving for $\|\nabla Loss(x, y, w)\| = 0$. This can happen in 3 ways
$p - y = 0 \implies y = p = 1 \implies w \to \infty$. i.e. w is very large
$1 - p = 0 \implies p = 1 \implies w \to \infty$ i.e. w is very large
$p = 0 \implies w \to -\infty$ i.e. w is very small

d Assuming $y = 1$, what is the largest magnitude that the gradient can take? Leave your answer in terms of $\phi(x)$

Our loss function is $2\phi(x)(p-y)p(1-p)$. If $y = 1$ we have $2\phi(x)(p-1)p(1-p) = -2\phi(x)(1 - p)^2 p = -2\phi(x)(p^2 - 2p + 1)p = -2\phi(x)(p^3 - 2p^2 + p)$
To find the largest magnitude, we need the critical points of the cubic equation. We take the derivative and solve for 0. i.e.
$\frac{d(-2\phi(x)(p^3-2p^2+p))}{dp} = -2\phi(x)(3p^2 - 4p + 1)$.
Solving for the derivative = 0
$(3p^2 - 4p + 1) = 0 \implies p = 1/3, 1$
Plugging this back in we get the maximum value when $p = 1/3$. Therefore the largest magnitude the gradient can take is
$-2\phi(x)\left[\frac{1}{3} - 1\right]^2 \frac{1}{3} = -2\phi(x)\frac{4}{27} = -\phi(x)\frac{8}{27}$

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 D' of (x, y') pairs such that performing least squares regression (using a linear predictor and the squared loss) on D' converges to a vector w that yields zero loss on D'. Concretely, write an expression for y' in terms of y and justify this choice. This expression should not be a function of w**

For zero loss, the error must be zero, i.e. the term $(\sigma(w \cdot \phi(x)) - y)^2$ must be zero. Therefore we have
$(\sigma(w \cdot \phi(x)) - y)^2 = 0$
$\implies \sigma(w \cdot \phi(x)) - y = 0$
$\implies \sigma(w \cdot \phi(x)) = y$
$\implies \frac{1}{1+e^{-w\cdot\phi(x)}} = y$
$\implies \frac{1}{y} = 1 + e^{-w\cdot\phi(x)}$
$\implies \frac{1}{y} - 1 = e^{-w\cdot\phi(x)}$
$\implies ln\left(\frac{1}{y} - 1\right) = -w \cdot \phi(x)$
Therefore the transformation is $y' = ln\left(\frac{1}{y} - 1\right)$ and $w^* = -w$. This will yield zero loss on D'.

3 Problem 3: Sentiment Classification

a See submission.py

b See submission.py

c See submission.py

d 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.

Lets take a look at one of the reviews that we had an error on
=== 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 .
Truth: -1, Prediction: 1 [WRONG]
The review begins with a positive note but that view is negated with until... The classifier naively just adds the weighted sentiment for each word so it does not understand the negation.
One solution could be to split the sentence into parts, look at the sentiment for each part and negate or add to it if we encounter words like "until", "plus", "also", "besides", etc. This requires much more NLP.

=== 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 .
Truth: 1, Prediction: -1 [WRONG]
Too many uncommon words (which got weight 0), and stop words have weights (. has weight -0.03, , has weight-0.01, a has weight -0.02 ,which has weight -0.13, etc). Need to remove stop words since they don't usually convey sentiment.

=== the lousy lead performances . . . keep the movie from ever reaching the comic heights it obviously desired .
Truth: -1, Prediction: 1 [WRONG]
The words "performance" and "keep" have strong positive weights outweighing words like "lousy". Again, we need to look at composite words to get the overall sentiment. So "lead performance" is say 0.68 but "lousy" should negate that weight. Instead its adding a small negative weight.

=== mazel tov to a film about a family's joyous life acting on the yiddish stage .
Truth: 1, Prediction: -1 [WRONG]
Stop words have negative weights causing the prediction to be negative.

=== suffers from a decided lack of creative storytelling .
Truth: -1, Prediction: 1 [WRONG]
Both "creative" and "storytelling" are strong positive words (0.74 and 0.52). "Lack" should negate that part but we are just taking the sum of positive and negative weights.

e See submission.py

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?

We get the lowest error when n=5. This is similar to the error we get with whole words as features. The reason they are similar is the average size of words is 5. A review in which n grams probably outperform word features is "Greatest movie ever - videography was superlative". Reason is some of the words like videography are superlative are rare and may have low weights in the word features, but 5 grams like super and video would be common.

4 Problem 3: K-means clustering Suppose we have a feature extractor ϕ that produces 2-dimensional feature vectors, and a toy dataset $D_{train} = \{x_1, x_2, x_3, x_4\}$ with $\phi(x_1) = [0, 0]$ $\phi(x_2) = [0, 2]$ $\phi(x_3) = [2, 0]$ $\phi(x_4) = [1, 2]$

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: $\mu_1 = [1, 3]$ and $\mu_2 = [1, -1]$ $\mu_1 = [-1, 1]$ and $\mu_2 = [2, 2]$

I am using the square of euclidean distance here. It will give the same result as euclidean distance. Running for $\mu_1 = [1, 3]$ and $\mu_2 = [-1, 1]$
Euclidean distance squared for $\phi(x_1)$ to $\mu_1 = 10$ and to $\mu_2 = 2$.
$z_1 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_1) - \mu_k\|^2 = 2$.
Euclidean distance squared for $\phi(x_2)$ to $\mu_1 = 2$ and to $\mu_2 = 10$.
$z_2 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_2) - \mu_k\|^2 = 1$.
Euclidean distance squared for $\phi(x_3)$ to $\mu_1 = 10$ and to $\mu_2 = 2$.
$z_3 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_3) - \mu_k\|^2 = 2$.
Euclidean distance squared for $\phi(x_4)$ to $\mu_1 = 1$ and to $\mu_2 = 9$.
$z_4 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_4) - \mu_k\|^2 = 1$.
$\mu_1 = average(\phi(x_2), \phi(x_4)) = (0.5, 2)$
$\mu_2 = average(\phi(x_1), \phi(x_3)) = (1, 0)$
Euclidean distance squared for $\phi(x_1)$ to $\mu_1 = 4.25$ and to $\mu_2 = 1$.
$z_1 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_1) - \mu_k\|^2 = 2$.
Euclidean distance squared for $\phi(x_2)$ to $\mu_1 = 0.25$ and to $\mu_2 = 5$.
$z_2 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_2) - \mu_k\|^2 = 1$.
Euclidean distance squared for $\phi(x_3)$ to $\mu_1 = 6.25$ and to $\mu_2 = 1$.
$z_3 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_3) - \mu_k\|^2 = 2$.
Euclidean distance squared for $\phi(x_4)$ to $\mu_1 = 0.25$ and to $\mu_2 = 4$.

$z_4 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_4) - \mu_k\|^2 = 1.$

Therefore we have convergence. $\phi(x_2), \phi(x_4) \in$ cluster 1. $\phi(x_1), \phi(x_2) \in$ cluster 2.

Running for $\mu_1 = [-1, 1]$ and $\mu_2 = [2, 2]$
Euclidean distance squared for $\phi(x_1)$ to $\mu_1 = 2$ and to $\mu_2 = 8$.
$z_1 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_1) - \mu_k\|^2 = 1.$
Euclidean distance squared for $\phi(x_2)$ to $\mu_1 = 2$ and to $\mu_2 = 4$.
$z_2 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_2) - \mu_k\|^2 = 1.$
Euclidean distance squared for $\phi(x_3)$ to $\mu_1 = 10$ and to $\mu_2 = 4$.
$z_3 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_3) - \mu_k\|^2 = 2.$
Euclidean distance squared for $\phi(x_4)$ to $\mu_1 = 5$ and to $\mu_2 = 1$.
$z_4 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_4) - \mu_k\|^2 = 2.$
$\mu_1 = average(\phi(x_1), \phi(x_2)) = (0, 1)$
$\mu_2 = average(\phi(x_3), \phi(x_4)) = (1.5, 1)$
Euclidean distance squared for $\phi(x_1)$ to $\mu_1 = 1$ and to $\mu_2 = 3.25$.
$z_1 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_1) - \mu_k\| = 1.$
Euclidean distance squared for $\phi(x_2)$ to $\mu_1 = 1$ and to $\mu_2 = 3.25$.
$z_2 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_2) - \mu_k\| = 1.$
Euclidean distance squared for $\phi(x_3)$ to $\mu_1 = 5$ and to $\mu_2 = 1.25$.
$z_3 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_3) - \mu_k\| = 2.$
Euclidean distance squared for $\phi(x_4)$ to $\mu_1 = 2$ and to $\mu_2 = 1.25$.
$z_4 \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_4) - \mu_k\| = 2.$
Therefore we have convergence. $\phi(x_1), \phi(x_2) \in$ cluster 1. $\phi(x_3), \phi(x_4) \in$ cluster 2.

b See submission.py

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.

This is basically COP-KMeans (constrained Kmeans). The original algorithm has Must-link (certain pairs must be in a specific cluster) and Cannot-link (certain pairs cannot be in a particular cluster). In our case its only Must-link.
Algorithm is as follows:
Initialize $\mu_1, \cdots, \mu_K$ randomly
Step 1
for each point $i = 1, \ldots n$:

Assign i to cluster with closest centroid without violating constraints.

$z_i \leftarrow arg \min_{k=1,\cdots,K} \|\phi(x_i) - \mu_k\|^2$ without violating constraints.

If $(i, j) \in S$ and i and j not in the same cluster then constraint is violated. Look for next closest cluster. If there is no next closest cluster, then fail.

Step 2
for each cluster $k = 1, \cdots, K$:

set μ_k to the average of points assigned to cluster k

The pseudocode for the algorithm is as follows:

```
initialize $\mu_1 \cdots \mu_k$ to be randomly selected pairs
for each point $i$:
    assign $i$ to closest cluster $\mu_j$ such that violateConstraints($i$,$\mu_j$,$S$)=False
    if no cluster is found, fail and exit
for each cluster $\mu_j$ update its center as the average of all the points
in the cluster.
repeat the assign $i$ and cluster average steps until convergence
return $\mu_1 \cdots \mu_k$

violateConstraints(point,cluster,setOfMustHave)
    for each pair of points $point, j$ in setOfMustHave
        if j not in cluster return True (constraint violated)
    return False (constraint not violated)
```