CS221 Fall 2015 Homework [sentiment]

Problem 1: Warmup

1a. x_1 = pretty good y_1 = 1 x_2 = bad plot y_2 = -1
x_3 = not bad y_3 = 1 x_4 = pretty scenery y_4 = 1

And using Dictionary = ('pretty', 'good', 'bad', 'plot', 'not', 'scenery')
We extract feature vectors as follows:

$\phi(x_1)$ = [1, 1, 0, 0, 0, 0] $\phi(x_2)$ = [0, 0, 1, 1, 0, 0]
$\phi(x_3)$ = [0, 0, 1, 0, 1, 0] $\phi(x_4)$ = [1, 0, 0, 0, 0, 1]

Next, run stochastic gradient descent once for each of the four examples in order:
We start with weight vector w = [0,0,0,0,0,0]

for $\phi(x_1)$ and y_1 the Loss function is:

max(0, 1 – [0,0,0,0,0,0].[1,1,0,0,0,0] * 1) = 1

Therefore $\nabla_w Loss_{hinge}(x_1, y_1, w)$ is the gradient of $1 - w \cdot \phi(x_1) y_1$ is $-\phi(x_1) y_1$ so then we update the weights in using $w \leftarrow w - \eta \nabla_w Loss_{hinge}(x, y, w)$ using $\eta = 1$ as follows:

w = [0,0,0,0,0,0] – 1 * (-[1, 1, 0, 0, 0, 0] * 1) = [1, 1, 0, 0, 0, 0]

for $\phi(x_2)$ and y_2 the Loss function is:

max(0, 1 - [1, 1, 0, 0, 0, 0] . [0, 0, 1, 1, 0, 0] *-1) = 1

Therefore $\nabla_w Loss_{hinge}(x_2, y_2, w)$ is $-\phi(x_2) y_2$ and

w = [1, 1, 0, 0, 0, 0] - 1 (- [0, 0, 1, 1, 0, 0] * -1) = [1, 1,-1,-1, 0, 0]

for $\phi(x_3)$ and y_3 the Loss function is:

max(0, 1 - [1, 1,-1,-1, 0, 0] . [0,0,1,0,1,0] * 1) = 2

Therefore $\nabla_w Loss_{hinge}(x_3, y_3, w)$ is $-\phi(x_3) y_3$ and

w = [1, 1,-1,-1, 0, 0] – 1 (-[0, 0, 1, 0, 1, 0] * 1) = [1, 1, 0, -1, 1, 0]

for $\phi(x_4)$ and y_4 the Loss function is:

max(0, 1 - [1, 1, 0, -1, 1, 0] . [1, 0, 0, 0, 0, 1] * 1) = max(0, 0)

For this case we are given that $\nabla_w Loss_{hinge}(x_4, y_4, w) = 0$

Therefore the weight vector does not change from the previous step.
Which results in a final weight vector *w* of [1, 1, 0, -1, 1, 0] for the words ('pretty', 'good', 'bad', 'plot', 'not', 'scenery').

1b.
Let the four mini-reviews be as follows:
bad (-1)
good (+1)
not good (-1)
not bad (+1)
Dictionary = (bad, good, not)
Let there be a linear classifier that uses the weight vector $[w_1 w_2 w_3]$ for $[bad\ ,\ good\ ,\ not]$
This means that for the each mini-review that the following should be true:
bad $w \cdot [1,0,0] + w_0 < 0$ equivalently is $w_1 + w_0 < 0$ which we'll call *inequality1*
good $w \cdot [0,1,0] + w_0 \geq 0$ equivalently is $w_2 + w_0 \geq 0$ which we'll call *inequality2*
not good $w \cdot [0,1,1] + w_0 < 0$ equivalently is $w_2 + w_3 + w_0 < 0$ which we'll call *inequality3*
not bad $w \cdot [1,0,1] + w_0 \geq 0$ equivalently is $w_1 + w_3 + w_0 \geq 0$ which we'll call *inequality4*
Now *inequality1* and *inequality2* yields $w_2 > w_1$
However *inequality3* and *inequality4* yields $w_1 > w_2$
Since these results are inconsistent, this proves that no linear classifier can yield zero error.

The single addition feature that will fix this problem is adding the presence of two words, such as one of the following: “not bad” or “not good” such that:
Dictionary = (bad, good, not, not bad)
Now a linear classifier will use the weight vector $[w_1 w_2 w_3 w_4]$
bad $w \cdot [1,0,0,0] + w_0 = -1$ equivalently is $w_1 + w_0 = -1$
good $w \cdot [0,1,0,0] + w_0 = 1$ equivalently is $w_2 + w_0 = 1$
not good $w \cdot [0,1,1,0] + w_0 = -1$ equivalently is $w_2 + w_3 + w_0 = -1$
not bad $w \cdot [1,0,1,1] + w_0 = 1$ equivalently is $w_1 + w_3 + w_4 + w_0 = 1$
One solution for the above equations is the following w = [-1, 1, -2, 4] with $w_0 = 0$
This shows that the linear classifier with this one additional feature and the above weights can successfully classify the four mini-reviews.

Problem 2: Predicting Movie Ratings

2a.

First calculate the logistic function: $\sigma(w \cdot \phi(x)) = \frac{1}{(1 + e^{-(w \cdot \phi(x))})}$

Then the squared error is the Loss: $Loss(x, y, w) = \left(y - \frac{1}{1 + e^{-(w \cdot \phi(x))}} \right)^2$

2b.

The gradient of Loss is equal to the derivative with respect to w, such that:

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

The derivative is determined using the chain rule , differentiating and simplifying the terms as follows:

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

2c.

When y=0 then the gradient of the Loss function, $\nabla\ Loss$ simplifies to the following:

$$\frac{2\phi(x)}{e^{w\cdot\phi(x)}\left(1+e^{-(w\cdot\phi(x))}\right)^3}$$

When the magnitude of w goes to infinity $\| w \| \to \infty$ then both $w\cdot\phi(x) \to \infty$ and $e^{w\cdot\phi(x)} \to \infty$ go to infinity. However $e^{-(w\cdot\phi(x))} \to 0$ goes to zero. Therefore, when $\| w \| \to \infty$ then the denominator of $\frac{2\phi(x)}{e^{w\cdot\phi(x)}\left(1+e^{-(w\cdot\phi(x))}\right)^3}$ goes to infinity. Therefore when $\| w \| \to \infty$ then the magnitude of the Loss function, $\| \nabla\ Loss \|$ goes to zero $\| \nabla\ Loss \| \to 0$ which is it's minimum possible value.

2d.

When y=0 then the gradient of the Loss function, $\nabla\ Loss$ simplifies to the following:

$$\frac{2\phi(x)}{e^{w\cdot\phi(x)}\left(1+e^{-(w\cdot\phi(x))}\right)^3}$$

Substituting $e^{w\cdot\phi(x)}$ with z equals the following:

$$\frac{2\phi(x)}{z\ \left(1+\frac{1}{z}\right)^3}$$

Maximizing the magnitude of the gradient is equivalent to minimizing the denominator. To minimize the denominator we take the derivative with respect to z and set it to zero as follows:

$\frac{d}{dz}\ z\left(1+\frac{1}{z}\right)^3 = 3z\left(1+\frac{1}{z}\right)^2\left(\frac{-1}{z^2}\right)+\left(1+\frac{1}{z}\right)^3 = 0$ therefore, $\frac{3}{z} = 1+\frac{1}{z}$ therefore $z=2$ therefore $e^{w\cdot\phi(x)}=2$

So, we substitute the value of $e^{w\cdot\phi(x)}=2$ back into the $\nabla\ Loss$ function to get the maximum $\| \nabla\ Loss \|$ equal to $\frac{8}{27}\| \phi(x) \|$ which is the answer.

As a side note, we know that $z=2$ minimizes the denominator because the second order derivative is positive.

Problem 3: Sentiment Classification

3d.

After much experimenting and tweaking hyper parameter values using the supplied polarity data, the following are the results that produced the smallest test error:

numIters = 20

stepSize = 0.15

Read 3554 examples from polarity.train

Read 3554 examples from polarity.dev

10935 weights

Official: train error = 0.0402363534046, dev error = 0.270399549803

3e.

For each of the below examples that was "WRONG" in the error-analysis, the feature extractor could be updated to more carefully pre-process the learning data to create more intelligently crafted feature vectors such as to eliminate errors such as these.

For example, the following words are relatively meaningless relative to our goal of sentiment analysis. These words are grammatically necessary in order to create a sentence but could effectively be removed from the feature vectors due to the fact that they are structural grammar words, such as conjunction words, rather than words that denote information in terms of accurately representing sentiment:

and 3 * 0.2449 = 0.7347

an 1 * 0.19764942991 = 0.19764942991

of 1 * 0.0648933787222 = 0.0648933787222

that 1 * 0.00980529298448 = 0.00980529298448

than 1 * -0.081577893862 = -0.081577893862

for 1 * -0.0008 = -0.0008

has 1 * 0.136469088758 = 0.136469088758

with 1 * 0.122732771126 = 0.122732771126

Furthermore, on a similar note, a quick analysis of the remaining words reveal that they are people's names, and typically people's names are common words that may not work well as a generalization to denote sentiment since many people can have the same name, and also some names may be very rare, causing both noise and outliers in the data set, here are two examples:

shawn 1 * 0 = 0

dan 1 * 0 = 0

3g.
The following are results of running the n-gram character feature extractor I wrote multiple times until achieving an average minimum error using n=5:

featureExtractor = extractCharacterFeatures(5)

Read 3554 examples from polarity.train
Read 3554 examples from polarity.dev
112547 weights
Official: train error = 0.0, dev error = 0.267023072594

The error observed on the test data is actually zero and the dev is nearly just as low as in the word features tests. This was surprising, but believe it can be explained due to the n-gram character features still accurately representing the mapping contained in the dataset. When predicting, the part-of-words are mashed together with the the same featureExtractor function as they are when testing. Since the feature vectors are nearly representative of each other, containing almost all the same data, and the function mapping one to the other is the same, it makes sense that the error would be near equivalent.

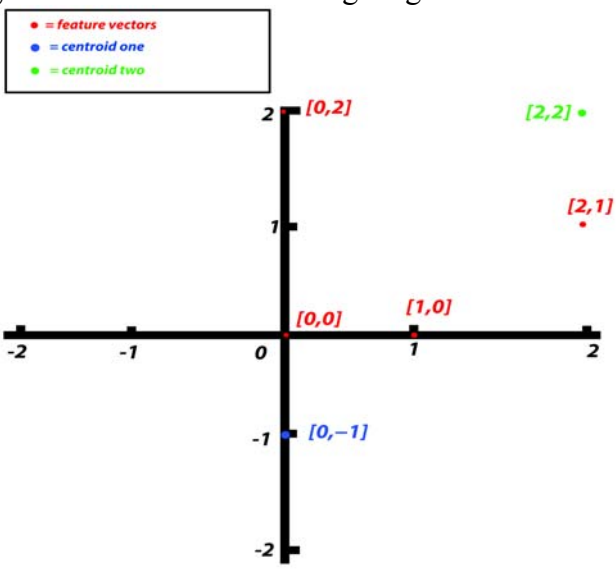
An example of a sentence that the per-character ngram model would do better on would be a sentence that does not contain standard whitespace delimiters, such as the common #hashtag example:

“ThisIsAReallyCoolHashtagSentence”

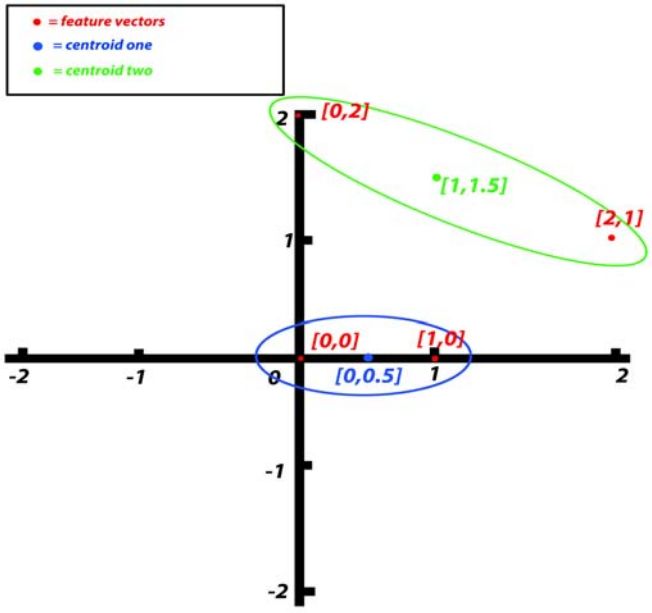
Since the extractCharacterFeatures(n) function is not concerned with whitespace delimiters, it would better seperate this into it's n-component part-of-words which could then be more effectively classified using machine learning; much more effectively than if it were to remain a single word yet contain multiple features within it that denote it's natural language meaning.

Problem 4: K-means clustering

4a. 1. Given initial centers: μ1=[0,−1] and μ2=[2,2] and feature vectors: $\phi(x_n)=[1,0][2,1][0,0][0,2]$
Starting with the above feature vectors: [1,0] [2,1] [0,0] [0,2] and given centroids: [0,−1] [2,2] this simple toy problem is easily illustrated in the following diagram:

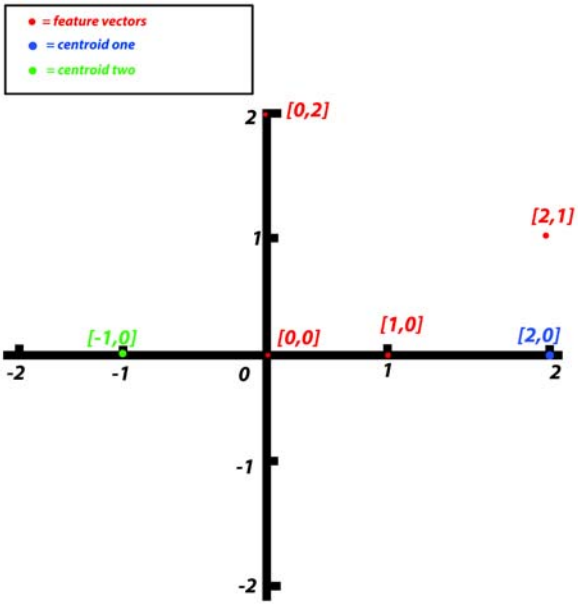


To run k-means algorithm, first start, for each feature vector, find the nearest centroid: for [1,0] the nearest centroid is: [0,−1] then for [2,1] the nearest centroid is: [2,2] then for [0,0] the nearest centroid is: [0,−1] then for [0,2] the nearest centroid is: [2,2] And now, in the next iteration, cluster assignment is determined and the new average cluster centers and are [0,0.5] and [1,1.5] as shown:

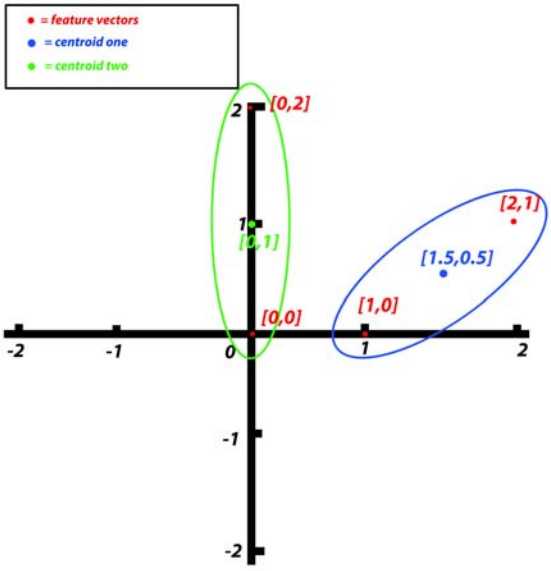


Since the cluster assignments remain the same after the first iteration, the k-means algorithm terminates with μ1=[0,0.5] and μ2=[1,1.5] and the cluster assignments z1 = [0,0] [1,0] and z2 = [0,2] [2,1]

4a.2. Given initial centers: μ1=[2,0] and μ2=[-1,0] and feature vectors: $\phi(x_n)=[1,0][2,1][0,0][0,2]$
Starting with the feature vectors: [1,0] [2,1] [0,0] [0,2] and the centroids: [2,0] [-1,0] again as shown:



Once again for each feature, find nearest center: for [1,0] the nearest centroid is: [2,0] then for [2,1] the nearest centroid is: [2,0] then for [0,0] the nearest centroid is: [-1,0] then for [0,2] the nearest centroid is: [-1,0] and at next iteration, clusters and new centroids are [1.5,0.5] and [0,1] as shown:



Since the cluster assignments remain the same after the first iteration, the k-means algorithm terminates with μ1=[1.5,0.5] and μ2=[0,1] and the cluster assignments z1 = [1,0] [2,1] and z2 = [0,0] [0,2]

The outcome is interesting because the answer for problem 4a.1. and 4a.2. result in different cluster assignments *z*. This shows that the result of the clusters from the k-means algorithm depends on the initial choice of the centers μ ...which could be seen as a pitfall of this algorithm.

4c.

Given set S of example pairs (i,j) we can first construct an undirected graph where the nodes represent the data points and each (i,j) pair is represented as an edge. We can use the connected components of this graph to group the data points into equivalence classes containing points that must be in the same cluster. The modified k-means algorithm should assign each equivalence class to a center μ based on a distance measure. The distance measure between a center μ and an equivalence class of points $[p_1, p_2, p_3 \ldots p_n]$ is calculated using the sum of the individual distances: $\sum_{i=1}^{n} \| p_i - \mu \|_2$

Recomputing the new centers for the next iteration is done as in the regular k-means algorithm.