**Homework #2: Sentiment**

# Problem 1

## Problem 1a

We have a weight vector of dimension 6 for the 6 different unique words appearing in the review. For the first pass we have

From the above, we see that with our initialized weights being zero, the margin for the first pass is 1, but since the gradient , we see that the weights are unable to update via gradient descent, which in turn reduces the weight vector to all zeros, the same as the value to which they were initialized:

## Problem 1b

Suppose that we have the following reviews as our dataset:

* (-1) not good
* (-1) bad
* (+1) good
* (+1) not bad

The fundamental property of the linear classifier is that its learning score is a linear combination of weights and features:

This means that in order for the classifier to have zero error, the features must be linearly separable, i.e. for two categories of points there must be some hyperplane with component coefficients such that every point in satisfies and each point in satisfies (assuming the hyperplane passes through the origin).

However, since the feature-vector is a mapping of word frequencies, **linear separation becomes geometrically impossible, because there is no linear / constant separator which can divide the word frequencies on their own.** Plotting the sample feature vectors from our dataset, which exist in , illustrates this point:

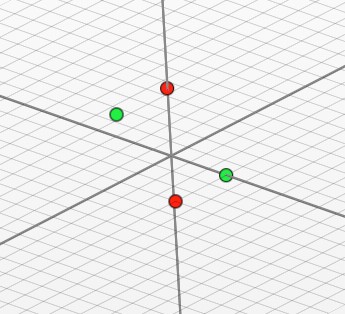


Figure 1: Plot of the sample feature points in . Red points denote bad reviews (y=-1) while green denote good reviews (y=1). Note that no 3D plane can be drawn which perfectly separates bad from good reviews, indicating linear inseparability.

In order to fix this problem we have to augment the feature vector somehow. Since part of the problem is that the existing features are used in both classification outcomes, one such augmentation that would fix it would be **to include bigrams of the words as features (e.g. counting the number of instances of “not good” in succession as opposed to just the words individually).** This will augment the feature vector into a higher dimension in which linear separability is possible.

# Problem 2

## Problem 2a

Since we are using the squared loss, its general expression is given by

In this case, we use a nonlinear predictor in the form of a sigmoid which is of the form

By substitution we then have

## Problem 2b

Computing the gradient of the loss, we apply the chain rule, letting such that:

Applying the derivative identity for the sigmoid function, , we have

## Problem 2c

Substituting y=1 into the above for the SGD run on the data point provided in the problem statement, we have

Finding the points which make the gradient small involve finding the roots of the expression on the left-hand side, which has a non-trivial root at p=1. Therefore, it is **theoretically** possible to obtain a zero-gradient result given the above analytical solution. However, recall that

Therefore the underlying weights which obtain this zero gradient are given by

Which simplifies to

Therefore, for some given value of , it follows that

Based on the above, **it is not actually possible to get the magnitude of the gradient to zero, since it only approaches zero as the magnitude of the weights approaches an infinitely large magnitude.** This is easily verified by visual inspection of the sigmoid curve, i.e. the curve “flattens” for very large values of the independent variables, which mathematically translates to a smaller gradient magnitude.

## Problem 2d

The largest magnitude the gradient can take can be found by differentiating and finding the roots of the derivative, which gives

Which has four roots at the values . Substitution into the original gradient expression shows that the magnitude of the gradient is largest when , which yields the **maximum gradient value of approximately 17.365**.

## Problem 2e

<FINISH THIS> For some dataset , there is a value of the weights such that . For , the loss function is potentially non-convex. We seek to transform into a modified dataset where

Fundamentally, we want to arrive at a choice of which converts to this linear regression from our original loss function which uses the sigmoid:

# Problem 3

# Problem 4

## Problem 4a

To cluster using the K-means algorithm, we first assign each data point to its closest centroid using

Then, we will update the location of the centroids based on the points that fall into their clusters as

With data

First, we have . This leads to assignments of

Which then results in updated centroids of

Which then gives new assignments of

Since these assignments are the same as before, the update step is identical to before, which means the above is the final converged result

Second, we have . The first assignment step gives

Which then updates the centroids to

Which in turn gives new assignments of

These assignments are the same as the previous iteration, indicating convergence at **,**

## Problem 4c

In order to modify the K-means algorithm to minimize the reconstruction loss subject to the constraint function , we would proceed according to the same fundamental steps, with modifications made along the way:

1. **Initialize Centroids:** This is done in the same way as vanilla K-means (i.e. randomly). There is no modification to be made here.
2. **Calculate Distances:** Compute the distances of all data points to all of the K initialized clusters as . There is no modification to be made here.
3. **Assignment Subject to Constraint Function:** Here, we modify by splitting assignment into two steps to account for the constraint.
   * **Assignment of unconstrained points:** Unconstrained points are assigned in the typical manner, i.e. using
   * **Assignment of constrained points:** For the constrained points, we have a pre-defined set of “arbitrary” assignment sets which we must specify to our actual clusters . To do so, we calculate the WCSS (within-cluster sum of squares) for each against each real cluster as , and associate the constrained assignment set to the cluster that is closest (i.e. lowest loss) to where the points already are, i.e. . This minimizes the cost of assignment, and the number of iterations required for convergence.
4. **Update Centroids:** Update the centroids according to their constituent data points using . There is no modification made here.
5. **Iteration**: Repeat steps 3-4 until maximum iterations exceeded or convergence.

Another option to account for the constraint is at the initialization stage (i.e. initialize a cluster to immediately lie at the centroid of each constraint set), but this approach has less generality.

## Problem 4d

The advantage of running K-means multiple times on the same dataset with different initializations is robustness – the nature of the centroid calculation and the K-means algorithm itself means that the converged outcome is deterministic from the initial values of the centroids. Therefore by testing many different initial values of the centroids, one can arrive at a final clustering answer which is most robust (e.g. represents the statistical majority of results for different initial values), which provides confidence that the result obtained is not a contrived / edge case stemming from a certain set of initial values.

## Problem 4e

If we scale all dimensions by the same factor, the clustering result will not change. This is because if all dimensions are scaled the same way, their relative position to one another will not change, which means their spatial proximity to centroids (and thus the eventual K-means clustering result) will be unaffected. (NOTE: This does not hold for any non-linear transformation of the original problem space, e.g. squaring all values in the original domain may result in different results)

However, if only some dimensions are scaled, then the relative positions of the data points and centroids change, which means that the final K-means result can change as a result of this type of scaling.