CS221, Spring 2019, PS2 Sentiment

Haiyuan Mei (hmei0411@stanford.edu)

1.a Stochastic gradient descent, updating the weights

- The gradient of hinge loss w.r.t. $\mathbf{w}$ is
$$\nabla_{\mathbf{w}}\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y, \mathbf{w}\cdot\phi(x)y \le 1 \\ 0, \text{otherwise} \end{cases}$$
- Represent the 4 samples with vectors in the order of ("pretty", "good", "bad", "plot", "not", "scenery"), and their scores/margins starting from $\mathbf{w} = \vec{0}$:

-	x	y	old w	score	margin	gradient	new w
1.	[1,0,1,0,0,0]	-1	[0,0,0,0,0,0]	score=0	margin=0	[1,0,1,0,0,0]	[-0.5,0,-0.5,0,0,0]
2.	[0,1,0,1,0,0]	+1	[-0.5,0,-0.5,0,0,0]	score=0	margin=0	[0,-1,0,-1,0,0]	[-0.5,0.5,-0.5,0.5,0,0]
3.	[0,1,0,0,1,0]	-1	[-0.5,0.5,-0.5,0.5,0,0]	score=0.5	margin=-0.5	[0,1,0,0,1,0]	[-0.5,0,-0.5,0.5,-0.5,0]
4.	[1,0,0,0,0,1]	+1	[-0.5,0,-0.5,0.5,-0.5,0]	score=-0.5	margin=-0.5	[-1,0,0,0,0,-1]	[0,0,-0.5,0.5,-0.5,0.5]

- Conclusion: after the 4 samples trained, the weights for each of the six words are: $[0, 0, -0.5, 0.5, -0.5, 0.5]$

1.b Small labeled dataset of four mini-reviews

Prove that no linear classifier can gain 0 error.

1. The 4 new datasets are:

1. (-1) not good: $x^{(1)} = [1, 0, 1]$, $y^{(1)} = -1$
2. $(+1)$ good: $x^{(1)} = [1, 0, 0]$, $y^{(1)} = +1$
3. $(+1)$ not bad: $x^{(1)} = [0, 1, 1]$, $y^{(1)} = +1$
4. (-1) bad: $x^{(1)} = [0, 1, 0]$, $y^{(1)} = -1$

   If there exists a $\mathbf{w}$ such that it makes no error, then the following should be true:
   $$\begin{cases} w_1 + w_3 < 0 \\ w_1 > 0 \\ w_2 + w_3 > 0 \\ w_2 < 0 \end{cases}$$
   This is impossible, because: if $w_1 > 0$, we must have $w_3 < 0$; since $w_2 < 0$, then $w_2 + w_3 > 0$ is impossible.

2. Add an additional feature that could fix the problem

- we could add a feature the number of words in the review. Suppose it is the 4th component in the review feature vector; the 4 datasets can be denoted as:
  1. (-1) not good: $x^{(1)} = [1, 0, 1, 2]$, $y^{(1)} = -1$
  2. $(+1)$ good: $x^{(1)} = [1, 0, 0, 1]$, $y^{(1)} = +1$
  3. $(+1)$ not bad: $x^{(1)} = [0, 1, 1, 2]$, $y^{(1)} = +1$
  4. (-1) bad: $x^{(1)} = [0, 1, 0, 1]$, $y^{(1)} = -1$
- The problem now becomes whether we can find a $\mathbf{w}$ which make no error on the dataset for the following inequalities:
  $$\begin{cases} w_1 + w_3 + 2w_4 < 0 \\ w_1 + w_4 > 0 \\ w_2 + w_3 + 2w_4 > 0 \\ w_2 + w_4 < 0 \end{cases}$$
- Since $\mathbf{w}$ can be anything and we can scale it, if we can solve the equation
  $$\begin{bmatrix} 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$
  we can find a valid $\mathbf{w}$ for the above inqualities.
- Since $\begin{bmatrix} 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ can be linear transformed to I, it can be inversed and the linear equation has solution

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 \end{bmatrix}^{-1}$$

which means the new feature added is enough to fix the above problem.

2.a Expression for Loss

The Loss expression can be written as:

$$\begin{aligned} \text{Loss}(x, y, \mathbf{w}) &= (y - \sigma(z))^2 \\ &= (y - \sigma(\mathbf{w} \cdot \phi(x)))^2 \\ &= \left(y - \frac{1}{1 + e^{-\mathbf{w} \cdot \phi(x)}}\right)^2 \end{aligned}$$

2.b Compute the gradient of the loss with respect to w

By applying the chain rule of gradient, the gradient w.r.t. $\mathbf{w}$:

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = -2(y - \sigma(z))\sigma(z)(1 - \sigma(z))\phi(x)$$

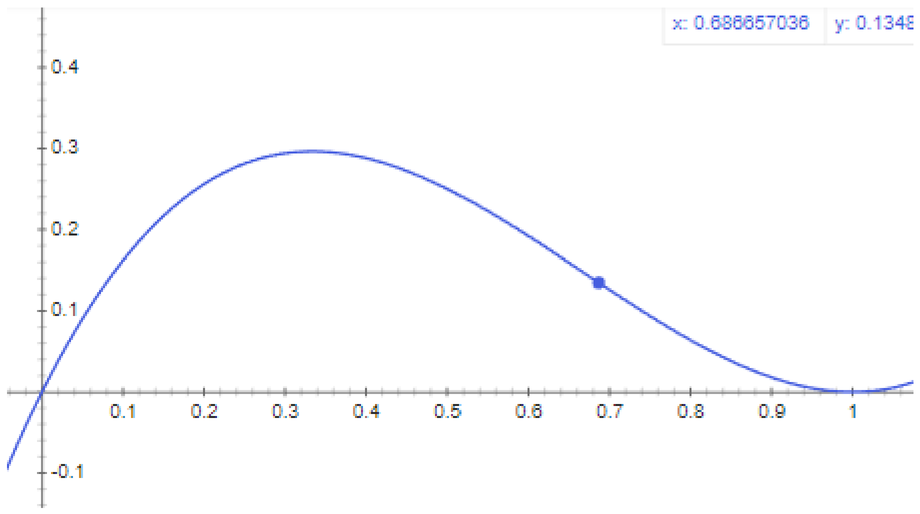
where $z = \mathbf{w} \cdot \phi(x)$

2.C Minimum gradient magnitude

From section b, replace y by 1, think of $||\phi(x)||$ as some constant and denote the magnitude as the following function $f(\sigma)$ (here for the ease of description, the sigmoid function is just denoted by a variable σ):

$$f(\sigma) = \|-2(y-\sigma(z))\sigma(z)(1-\sigma(z))\phi(x)\|$$
$$= 2\|\phi(x)\|(\sigma-1)^2\sigma, \text{where } \sigma \in (0,1)$$

The function looks like following (image generated using google):



This magnitude function touches 0 twice when $\sigma = 0$ or $\sigma = 1$, but σ can never reach 0 or 1, depending on the sign of $\mathbf{w} \cdot \phi(x)$, $\lim_{\|\mathbf{w}\| \longrightarrow \infty} \sigma(\mathbf{w} \cdot \phi(x)) = 0 \text{ or } 1$; So $\lim_{\|\mathbf{w}\| \longrightarrow \infty} f(\sigma(\mathbf{w} \cdot \phi(x))) = 0$ but can never reach a minimum, it is lower bounded by 0, and it can never be 0.

2.d Maximum gradient magnitude

When $\sigma \in (0,1)$, it can reach maximun when it's gradient is 0:

$$\begin{aligned} f'(\sigma) &= -4(\sigma-1)\sigma - 2(\sigma-1)^2 \\ &= -6\sigma^2 + 8\sigma - 2 = 0 \\ &\Rightarrow \sigma = \frac{1}{3}, \text{ when } \sigma \in (0,1) \end{aligned}$$

Which means if we choose a $\mathbf{w}$ which makes $\sigma(\mathbf{w} \cdot \phi(x)) = \frac{1}{3}$, we can reach a maximum gradient magnitude. The max magnitude is $f(1/3) = \frac{8}{27}\|\phi(x)\|$

2.e Conversion to least squares regression

Since $\mathbf{w}$ makes makes no error on dataset $\mathbf{D}$, and it uses non-linear predicator as described above; we have for each data point in $\mathbf{D}$:

$$y = \sigma(\mathbf{w} \cdot \phi(x)) \Rightarrow \mathbf{w} \cdot \phi(x) = \log \frac{y}{1-y}$$

For the transformed y' and least squares regression loss, set the loss to 0 we have:

$$\begin{aligned} L(x, y', \mathbf{w}^*) \ &= \ \frac{1}{2}(\mathbf{w}^* \cdot \phi(x) - y')^2 \\ &\Rightarrow \mathbf{w}^* \cdot \phi(x) = y' \end{aligned}$$

Compare this with the above loss of dataset $\mathbf{D}$, it's obvious that if the new dataset $\mathbf{D}'$ of (x, y') where $y' = \log \frac{y}{1-y}$ when perform least squares regression will also converge to $\mathbf{w}^* = \mathbf{w}$.

3.d One sentence explaination of wrong predictions.

1. === 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 .
    - putting too much weight on neutral words such as {start:0.28, spielberg:0.25, etc.}

1. === 'it's painful to watch witherspoon's talents wasting away inside unnecessary films like legally blonde and sweet home abomination , i mean , alabama . '
    - 'painful' should have large negative weight; 'wasting' should also have a negative weight.

1. === wickedly funny , visually engrossing , never boring , this movie challenges us to think about the ways we consume pop culture .
    - 'never boring', negative of negative is position, but they both have negative weights; context need to be learned.

1. === patchy combination of soap opera , low-tech magic realism and , at times , ploddingly sociological commentary .
    - 'low-tech' should be very negative, and 'magic' and 'realism' seems to be neutral words.

1. === . . . although this idea is " new " the results are tired .
    - Putting too much weight on quotation mark. Should exclude punctuation marks from learning.

3.e n-gram character level learning

So is splitting the words really necessary or can we just naively consider strings of characters that stretch across words?
The answer is no. The sentiment learning can be just done in character level, such as n-gram features.

3.f Compare n-gram feature and word feature

- Find the n-gram length that produces errors nearly as small as word features.The function and result is shown below:

In [ ]:
```
# 3.f code
def test3f():
    trainExamples = readExamples('polarity.train')
    devExamples = readExamples('polarity.dev')
    for i in range(1,10):
        featureExtractor = submission.extractCharacterFeatures(i)
        weights = submission.learnPredictor(trainExamples, devExamples, featur
eExtractor, numIters=20, eta=0.01)
        trainError = evaluatePredictor(trainExamples, lambda(x) : (1 if dotPro
duct(featureExtractor(x), weights) >= 0 else -1))
        devError = evaluatePredictor(devExamples, lambda(x) : (1 if dotProduct
(featureExtractor(x), weights) >= 0 else -1))
        print "%d-gram: train error = %5f, dev error = %5f" % (i,trainError, d
evError)

The result is:
1-gram: train error = 0.458638, dev error = 0.484524
2-gram: train error = 0.314575, dev error = 0.414744
3-gram: train error = 0.002532, dev error = 0.320484
4-gram: train error = 0.000000, dev error = 0.277715
5-gram: train error = 0.000000, dev error = 0.274057
6-gram: train error = 0.000000, dev error = 0.272651
7-gram: train error = 0.000281, dev error = 0.270962
8-gram: train error = 0.000563, dev error = 0.293191
9-gram: train error = 0.000844, dev error = 0.309229
```

- It can be seen that when n is with [4,7] the training error is almost 0, and the dev error is very close to word feature result (train error = 0.027293190771, dev error = 0.270399549803). The reason is probably because on average most of the words have around 4~7 letters.

- Construct a review (one sentence max) in which character n-grams probably outperform word features, and briefly explain why this is so.
  - 'not bad': n-gram can perform better since word feature would learn two negative weight, but 6-gram can just use one feature for 'notbad' and consider it positive.

4.a Run 2-means on this dataset until convergence.

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

From initial to the second iteration, the centroids and point assignments are shown as:

iter	Centroid 1	Point assignment to centroid 1	Centroid 2	Point assignment to centroid 2
0	(2,3)	$\{\phi(x_2) = [1,2], \phi(x_4) = [2,2]\}$	(2,-1)	$\{\phi(x_1) = [1,0], \phi(x_3) = [3,0]\}$
1	(1.5,2)	$\{\phi(x_2) = [1,2], \phi(x_4) = [2,2]\}$	(2,0)	$\{\phi(x_1) = [1,0], \phi(x_3) = [3,0]\}$
2	(1.5,2)	$\{\phi(x_2) = [1,2], \phi(x_4) = [2,2]\}$	(2,0)	$\{\phi(x_1) = [1,0], \phi(x_3) = [3,0]\}$

The assignment is hence $z = [2,1,2,1]$, the two centroids are $(1.5,2),(2,0)$

2. $\mu_1 = [0,1]$ **and** $\mu_2 = [3,2]$

From initial to the second iteration, the centroids and point assignments are shown as:

iter	Centroid 1	Points assignment to centroid 1	Centroid 2	Points assignment to centroid 2
0	(0,1)	$\{\phi(x_1) = [1,0], \phi(x_2) = [1,2]$	(3,2)}	$\{\phi(x_3) = [3,0], \phi(x_4) = [2,2]\}$
1	(1,1)	$\{\phi(x_1) = [1,0], \phi(x_2) = [1,2]$	(2.5,1)}	$\{\phi(x_3) = [3,0], \phi(x_4) = [2,2]\}$
2	(1,1)	$\{\phi(x_1) = [1,0], \phi(x_2) = [1,2]$	(2.5,1)}	$\{\phi(x_3) = [3,0], \phi(x_4) = [2,2]\}$

The assignment is hence $z = [1,1,2,2]$, the two centroids are $(1,1),(2.5,1)$

4.c K-means with prior knowledge

The prior knowledge can help to decide at least partly the initial centroids, it will help making the algorithm more stable;
The algorithm can as such modified to the following steps:

1. Initialize some centroids according to prior knowledge (fixed point);
2. Draw the remaining random centroids from all unfixed points;
3. Preprocess unfixed points (L2 norm for each point which will be used to calculate distance to centroids.);
4. Loop until converge:
    A. Calculate distance to every unfixed points, record the shortest centroids and assign a centroid;
    B. recalculate new centroids according to current assignments (including both fixed and unfixed points);

4.d Benefits running k-means with different initializations

1. Running k-means multiple times on the same dataset with the same K, but different random initializations can help check the correctnessless of the clustering.
2. k-means algorithm is sensitive to initializations, different initialization may have different clustering results.
3. For this reason, works have been done to improve the effectiveless of k-means by carefully choosing the starting centroids.
    - Hierarchical Starting Values, Tseng and Wong (2005)
    - k-means++, Arthur and Vassilvitskii (2007)

4.e Does scaling of features matter?

Applying the same linear transformation on both centroids and data points will not change the final assignment result. For this reason, scaling on all dimensions or only certain dimensions will both result in the same assignment result.