Sentiment Analysis

Stanford CS221 Autumn 2021-2022

Owner CA: Weston Hughes

Version: 03/25/2022

Ed Release Post

General Instructions

This (and every) assignment has a written part and a programming part.

The full assignment with our supporting code and scripts can be downloaded as sentiment.zip.

a. This icon means you should write responses in `sentiment.pdf`.

b. This icon means you should write code in `submission.py`.

All written answers must be **typeset (preferably in LaTeX)**. We strongly recommend using Overleaf. A link to a tex file with prompts can be found on Ed and a link to a starter guide and a generic LaTeX written answer template is provided on the main course page.

Also note that your answers should be **in order** and **clearly and correctly labeled** to receive credit. Be sure to submit your final answers as a PDF and tag all pages correctly when submitting to Gradescope.

You should modify the code in `submission.py` between

```
# BEGIN_YOUR_CODE
```

and

```
# END_YOUR_CODE
```

but you can add other helper functions outside this block if you want. Do not make changes to files other than `submission.py`.

Your code will be evaluated on two types of test cases, **basic** and **hidden**, which you can see in `grader.py`. Basic tests, which are fully provided to you, do not stress your code with large inputs or tricky corner cases. Hidden tests are more complex and do stress your code. The inputs of hidden tests are provided in `grader.py`, but the correct outputs are not. To run the tests, you will need to have `graderUtil.py` in the same directory as your code and `grader.py`. Then, you can run all the tests by typing

```
python grader.py
```

This will tell you only whether you passed the basic tests. On the hidden tests, the script will alert you if your code takes too long or crashes, but does not say whether you got the correct output. You can also run a single test (e.g., `3a-0-basic`) by typing

```
python grader.py 3a-0-basic
```

We strongly encourage you to read and understand the test cases, create your own test cases, and not just blindly run `grader.py`.

Advice for this homework:

a. Words are simply strings separated by whitespace. Note that words which only differ in capitalization are considered separate (e.g. *great* and *Great* are considered different words).

b. You might find some useful functions in `util.py`. Have a look around in there before you start coding.

Problem 1: Building intuition

Here are two reviews of *Perfect Blue*, from Rotten Tomatoes:

Panos Kotzathanasis
Asian Movie Pulse

"Perfect Blue" is an artistic and technical masterpiece; however, what is of utmost importance is the fact that Satoshi Kon never deteriorated from the high standards he set here, in the first project that was entirely his own.

January 26, 2020

Full Review

Derek Smith
Cinematic Reflections

[An] nime thriller [that] often plays as an examination of identity and celebrity, but ultimately gets so lost in its own complex structure that it doesn't end up saying much at all.

August 19, 2006

Full Review | Original Score: 2/4

Rotten Tomatoes has classified these reviews as "positive" and "negative," respectively, as indicated by the intact tomato on the top and the splatter on the bottom. In this assignment, you will create a simple text classification system that can perform this task automatically. 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 $\phi(x) = \{\text{pretty} : 1, \text{bad} : 1\}$. Recall the definition of the hinge loss:

$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{0, 1 - \mathbf{w} \cdot \phi(x) y\},$$

where x is the review text, y is the correct label, $\mathbf{w}$ is the weight vector.

a. Suppose we run stochastic gradient descent once for each of the 4 samples in the order given above, updating the weights according to

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}).$$

After the updates, what are the weights of the six words ("pretty", "good", "bad", "plot", "not", "scenery") that appear in the above reviews?

- Use $\eta = 0.1$ as the step size.
- Initialize $\mathbf{w} = [0, 0, 0, 0, 0, 0]$.
- The gradient $\nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = 0$ when margin is exactly 1.

What we expect: A weight vector that contains a numerical value for each of the tokens in the reviews ("pretty", "good", "bad","plot", "not", "scenery"), **in this order**. For example: $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$.

1. (score=0, loss=1) $[-0.1, 0, -0.1, 0, 0, 0]$
2. (score=0, loss=1) $[-0.1, 0.1, -0.1, 0.1, 0, 0]$
3. (score=+0.1, loss=1.1) $[-0.1, 0, -0.1, 0.1, -0.1, 0]$
4. (score=-0.1, loss=1.1) $[0, 0, -0.1, 0.1, -0.1, 0.1]$

(Only the weights in the last line need to be included in the given answer)

b. Given the following dataset of reviews:

1. (-1) bad
2. $(+1)$ good
3. $(+1)$ not bad
4. (-1) not good

Prove that no linear classifier using word features can get zero error on this dataset. Remember that this is a question about classifiers, not optimization algorithms; your proof should be true for any linear classifier, regardless of how the weights are learned.

Propose a single additional feature for your dataset that we could augment the feature vector with that would fix this problem.

What we expect:

1. a short written proof (~3-5 sentences).
2. a viable feature that would allow a linear classifier to have zero error on the dataset (classify all examples correctly).

Proof: to classify "bad" and "good" correctly, they must have negative and positive weights, respectively. But if "not" has a positive weight then "not good" is classified incorrectly, and if "not" has a negative weight then "not bad" is classified incorrectly.
Or, proof by contradiction: Let w_g, w_b, w_n denote the weights of good, bad, not, respectively. Suppose linear classifier of these weights perfectly classifies the reviews. Then, for each review respectively, to achieve 0 loss we must have:

1. $w_b \leq -1$
2. $w_g \geq 1$
3. $w_n + w_b \geq 1$
4. $w_n + w_g \leq -1$

Note that the +-1 values are interpreting "zero error" as zero on the loss function, which has a margin of 1. If we interpret "zero error" as "all correct prediction" then 0s on the right side of the inequalities is also correct.
From 3,4 we have $w_b \geq w_g$, which contradicts 1,2. Therefore such classfier does not exist.
A single redeeming feature would be (for example) an indicator feature for the phrase "not good".

Problem 2: Predicting Movie Ratings

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 $\sigma(\mathbf{w} \cdot \phi(x))$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function that squashes a real number to the range $(0, 1)$. For this problem, assume that the movie rating y is a real-valued variable in the range $[0, 1]$.
Do not use math software such as Wolfram Alpha to solve this problem.

a. Suppose that we wish to use **squared loss**. Write out the expression for $\text{Loss}(x, y, \mathbf{w})$ for a single datapoint (x, y).

What we expect: A mathematical expression for the loss. Feel free to use σ in the expression.

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

b. Given $\text{Loss}(x, y, \mathbf{w})$ from the previous part, compute the gradient of the loss with respect to $\mathbf{w}$, $\nabla_{\mathbf{w}}\text{Loss}(x, y, \mathbf{w})$. Write the answer in terms of the predicted value $p = \sigma(\mathbf{w} \cdot \phi(x))$.

What we expect: A mathematical expression for the gradient of the loss.

$$\nabla_w \text{Loss}(x, y, \mathbf{w}) = 2(p - y)p(1 - p)\phi(x),$$

where $p = \sigma(\mathbf{w} \cdot \phi(x))$ is the predicted value. Importantly, $p \in (0, 1)$.

c. Suppose there is one datapoint (x, y) with some arbitrary $\phi(x)$ and $y = 1$. Specify conditions for $\mathbf{w}$ to make the magnitude of the gradient of the loss with respect to $\mathbf{w}$ arbitrarily small (i.e. minimize the magnitude of the gradient). Can the magnitude of the gradient with respect to $\mathbf{w}$ ever be exactly zero? You are allowed to make the magnitude of $\mathbf{w}$ arbitrarily large but not infinity.

Hint: try to understand intuitively what is going on and what each part of the expression contributes. If you find yourself 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 $\mathbf{w}$ is very far from the optimum, 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.

What we expect: 1-2 sentences describing the conditions for $\mathbf{w}$ to minimize the magnitude of the gradient, 1-2 sentences explaining whether the gradient can be exactly zero.

To make the magnitude of the gradient arbitrarily small, we need to to make p infinitely close to 1 or 0. For 1, we could select a $\mathbf{w}$ of great magnitude such that $\mathbf{w} \cdot \phi(x)$ approaches positive infinity, causing the sigmoid to approach 1. For 0, select a $\mathbf{w}$ of great magnitude such that $\mathbf{w} \cdot \phi(x)$ approaches negative infinity, causing the sigmoid to approach 0. In either case, the expression $(p-1)p(1-p)$ gets arbitrarily close to 0, but never exactly hits 0 because $p \in (0, 1)$. This causes the magnitude of the gradient to get arbitrarily close to 0 but never exactly reach it.

Problem 3: Sentiment Classification

In this problem, we will build a binary linear classifier that reads movie reviews and guesses whether they are "positive" or "negative."

Do not import any outside libraries (e.g. numpy) for any of the coding parts. Only standard python libraries and/or the libraries imported in the starter code are allowed. In this problem, you must implement the functions without using libraries like Scikit-learn.

a. Implement the function `extractWordFeatures`, which takes a review (string) as input and returns a feature vector $\phi(x)$, which is represented as a `dict` in Python.

b. Implement the function `learnPredictor` using stochastic gradient descent and minimize hinge loss. Print the training error and validation error after each epoch to make sure your code is working. You must get less than 4% error rate on the training set and less than 30% error rate on the validation set to get full credit.

c. Write the `generateExample` function (nested in the `generateDataset` function) to generate artificial data samples.

   Use this to double check that your `learnPredictor` works! You can do this by using `generateDataset()` to generate training and validation examples. You can then pass in these examples as `trainExamples` and `validationExamples` respectively to `learnPredictor` with the identity function `lambda x: x` as a featureExtractor.

d. Some languages are written without spaces between words, so is splitting the words really necessary or can we just naively consider strings of characters that stretch across words? Implement the function `extractCharacterFeatures` (by filling in the `extract` function), which maps each string of n characters to the number of times it occurs, ignoring whitespace (spaces and tabs).

e. Run your linear predictor with feature extractor `extractCharacterFeatures`. Experiment with different values of n to see which one produces the smallest validation error. You should observe that this error is nearly as small as that produced by word features. Why is this the case?

   Construct a review (one sentence max) in which character n-grams probably outperform word features, and briefly explain why this is so.

   Note: There is a function in `submission.py` that will allow you add a test to `grader.py` to test different values of n. Remember to write your final written solution in `sentiment.pdf`.

   What we expect:

   1. a short paragraph (~4-6 sentences). In the paragraph state which value of n produces the smallest validation error, why this is likely the value that produces the smallest error.
   2. a one-sentence review and explanation for when character n-grams probably outperform word features.

The validation error is minimized when n is about 5 to 7. Because words in English are roughly this length, character grams can still mostly pick up on words. In addition, character grams can capture the context of adjacent words. A simple example of a review that character grams perform well on is "not good". Word features do not detect that "not" inverts "good", whereas character grams will generate features such as "notgo", which can be detected as negative.

Problem 4: Toxicity Classification and Maximum Group Loss

Recall that models trained (in the standard way) to minimize the average loss can work well on average but poorly on certain groups, and that we can mitigate this issue by minimizing the maximum group loss instead. In this problem, we will compare the average loss and maximum group loss objectives on a toy setting inspired by a problem with real-world toxicity classification models.

Toxicity classifiers are designed to assist in moderating online forums by predicting whether an online comment is toxic or not, so that comments predicted to be toxic can be flagged for humans to review [1]. Unfortunately, such models have been observed to be biased: non-toxic comments mentioning demographic identities often get misclassified as toxic (e.g., “I am a [demographic identity]”) [2]. These biases arise because toxic comments often mention and attack demographic identities, and as a result, models learn to *spuriously correlate* toxicity with the mention of these identities.

In this problem, we will study a toy setting that illustrates the spurious correlation problem: The input x is a comment (a string) made on an online forum; the label $y \in \{-1, 1\}$ is the toxicity of the comment ($y = 1$ is toxic, $y = -1$ is non-toxic); $d \in \{0, 1\}$ indicates if the text contains a word that refers to a demographic identity; and $t \in \{0, 1\}$ indicates whether the comment includes certain “toxic” words. The comment x is mapped onto the feature vector $\phi(x) = [1, d, t]$ where 1 is the bias term (the bias term is present to prevent the edge case $\mathbf{w} \cdot \phi(x) = 0$ in the questions that follow). To make this concrete, we provide a few simple examples below, where we underline toxic words and words that refer to a demographic identity:

Comment (x)	**Toxicity (y)**	**Presence of demographic mentions (d)**	**Presence of toxic words (t)**
“Stanford <u>sucks</u>!”	1	0	1
“I’m a <u>woman</u> in computer science!”	-1	1	0
“The hummingbird <u>sucks</u> nectar from the flower”	-1	0	1

Suppose we are given the following training data, where we list the number of times each combination (y, d, t) shows up in the training set.

y	d	t	**# data points**
-1	0	0	63
-1	0	1	27
-1	1	0	7
-1	1	1	3
1	0	0	3
1	0	1	7
1	1	0	27
1	1	1	63

Total # examples	**200**

From the above table, we can see that 70 out of the 100 of toxic comments include toxic words, and 70 out of the 100 non-toxic comments do not. In addition, the toxicity of the comment t is highly correlated with mentions of demographic identities d (because toxic comments tend to target them) — 90 out of the 100 toxic comments include mentions of demographic identities, and 90 out of the 100 non-toxic comments do not.

We will consider linear classifiers of the form $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$, where $\phi(x)$ is defined above. Normally, we would train classifiers to minimize either the average loss or the maximum group loss, but for simplicity, we will compare two fixed classifiers (which might not minimize either objective):

- Classifier D: $\mathbf{w} = [-0.1, 1, 0]$
- Classifier T: $\mathbf{w} = [-0.1, 0, 1]$

For our loss function, we will be using the zero-one loss, so that the per-group loss is

$$\text{TrainLoss}_g(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}(g)|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}(g)} \mathbf{1}[f_{\mathbf{w}}(x) \neq y].$$

Recall the definition of the maximum group loss:

$$\text{TrainLoss}_{\max}(\mathbf{w}) = \max_g \text{TrainLoss}_g(\mathbf{w}).$$

To capture the spurious correlation problem, let us define groups based on the value of (y, d). There are thus four groups: $(y = 1, d = 1), (y = 1, d = 0), (y = -1, d = 1)$, and $(y = -1, d = 0)$. For example, the group $(y = -1, d = 1)$ refers to non-toxic comments with demographic mentions.

a. In words, describe the behavior of Classifier D and Classifier T.

What we expect: For each classifier (D and T), an "if-and-only-if" statement describing the output of the classifier in terms of its features when $y = 1$.

1. Classifier D predicts that a comment is toxic ($f_{\mathbf{w}}(x) = 1$) if and only if a demographic identity word is mentioned ($d = 1$).
2. Classifier T predicts that a comment is toxic ($f_{\mathbf{w}}(x) = 1$) if and only if a toxic word is mentioned $(t = 1)$.

b. Compute the following three quantities concerning Classifier D using the dataset above:

1. Classifier D's average loss
2. Classifier D's average loss for each group (fill in the table below)
3. Classifier D's maximum group loss

	$y = 1$	$y = -1$
$d = 1$		
$d = 0$		

What we expect: A value for average loss, a complete table with average loss for each group with the values in the given order, and a value for maximum group loss.

1. Average loss: Out of 200 examples, the classifier would misclassify 20 examples (where y and d do not "agree"). So the zero-one loss is 20/200 = 0.1.

2.

	$y = 1$	$y = -1$
$d = 1$	0	1
$d = 0$	1	0

3. Maximum group loss: 1

c. Now compute the following three quantities concerning Classifier T using the same dataset:

1. Classifier T's average loss
2. Classifier T's average loss for each group (fill in the table below)
3. Classifier T's maximum group loss

Note the groups are still defined by d, the demographic label.

	$y = 1$	$y = -1$
$d = 1$		
$d = 0$		

What we expect: A value for average loss, a complete table with average loss for each group with the values in the given order, and a value for maximum group loss.

1. Average loss: Out of 200 examples, the classifier would misclassify 60 examples (where y and t do not "agree"). So the zero-one loss is 60/200 = 0.3.

2.

	$y = 1$	$y = -1$
$d = 1$	0.3	0.3
$d = 0$	0.3	0.3

3. Maximum group loss: 0.3

d. Now let's compare the two classifiers. Which classifier has lower average loss? Which classifier has lower maximum group loss?

What we expect: First, indicate which classifier has lower average loss, then indicate which classifier has lower maximum group loss.

1. Classifier D has lower average loss (0.1).
2. Classifier T has lower maximum group loss (0.3).

e. As we saw above, different classifiers lead to different numbers of accurate predictions and different people's comments being wrongly rejected. Accurate classification of a non-toxic comment is good for the commenter, but when no classifier has perfect accuracy, how should the correct classifications be distributed across commenters? Here are four well-known principles of fair distribution:

1. According to **utilitarianism**, we should choose the distribution of accurate classifications that results in the *greatest net benefit* or *greatest average well-being*, where the average is a simple average. [3]
2. According to **prioritarianism**, we should choose the distribution of accurate classifications that results in the greatest *weighted* average well-being, where the weights prioritize less-well-off groups. [4]

3. According to John Rawls's **difference principle**, when choosing between distributive systems, we should choose the one that maximizes the well-being of the worst-off people. [5]
4. In order to **avoid compounding prior injustice**, we should ensure that our classifier does not impose a disadvantage on members of a demographic group that has faced historical discrimination. [6]

Note: The above are only a subset of the ethical frameworks out there. If you feel that another principle is more appropriate, you may invoke it, but you must define it in your answer and link to a source in which it is described.

First, choose an objective (average loss or maximum group loss) by appealing to one of the four principles. Which of Classifier D or Classifier T would you deploy on a real online social media platform in order to flag posts labeled as toxic for review? Refer to one of the above principles when justifying your answer.

What we expect: A 2-4 sentence explanation that justifies your choice of objective by referring to one of the four principles. There are many ways to answer this question well; a good answer explains the connection between a classifier and a principle clearly and concisely.

There are many correct answers to this solution. Here is an example: Because of the difference principle, we should choose a classifier that minimizes the maximum group loss. For someone whose comments are getting repeatedly and unfairly removed, a marginal improvement means more to them than someone who almost never gets comments removed. Therefore, the difference principle would suggest that minimizing the maximum group loss is more important than minimizing the average loss. From that perspective, Classifier T should be favored.

f. We've talked about machine learning as the process of turning data into models, but where does the data come from? In the context of collecting data for training a machine learning model for toxicity classification, who determines whether a comment **should** be marked as toxic or not is very important (i.e., whether $y = 1$ or $y = -1$ in the earlier data table). Here are some commonly used choices:

- Recruit people on a crowdsourcing platform (like Amazon Mechanical Turk) to annotate each comment.
- Hire social scientists to establish criteria for toxicity and annotate each comment.
- Ask users of the platform to rate comments.
- Ask users of the platform to deliberate about and decide on community standards and criteria for toxicity, perhaps using a process of participatory design. [7]

Which methods(s) would you use to determine the toxicity of comments for use as data to use for training a toxicity classifier, and why? Explain why you chose your method(s) over the others listed.

What we expect: 1-2 sentences explaining what methods(s) you would use and why you chose those method(s), and 1-2 sentences contrasting your chosen method(s) with alternatives.

Any combination is acceptable as long as there's justification. An example answer might be: "I would use a participatory design method and engage platform users to establish community standards of toxicity. I would then hire social scientists to annotate the comments using these established criteria. I chose this method because the community inclusion element of participatory design and using trained annotators is important. I would not use Amazon Mechnical Turk because Turkers aren't formally trained and are often underpaid.

Problem 5: K-means clustering

Suppose we have a feature extractor ϕ that produces 2-dimensional feature vectors, and a toy dataset $\mathcal{D}_{\text{train}} = \{x_1, x_2, x_3, x_4\}$ with

1. $\phi(x_1) = [10, 0]$
2. $\phi(x_2) = [30, 0]$
3. $\phi(x_3) = [10, 20]$
4. $\phi(x_4) = [20, 20]$

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:

1. $\mu_1 = [20, 30]$ and $\mu_2 = [20, -10]$
2. $\mu_1 = [0, 10]$ and $\mu_2 = [30, 20]$

What we expect: Show the cluster centers and assignments for each step.

1. Initial clusterings: μ_1 has $\{x_3, x_4\}$ and μ_2 has $\{x_1, x_2\}$. Then we set $\mu_1 = [15, 20]$, and $\mu_2 = [20, 0]$. This is stable.
2. Initial clusterings: μ_1 has $\{x_1, x_3\}$ and μ_2 has $\{x_2, x_4\}$. Then we set $\mu_1 = [10, 10]$ and $\mu_2 = [25, 10]$. This is stable.

b. Implement the `kmeans` function. You should initialize your k cluster centers to random elements of `examples`.

After a few iterations of k-means, your centers will be very dense vectors. In order for your code to run efficiently and to obtain full credit, you will need to precompute certain dot products. As a reference, our code runs in under a second on cardinal, on all test cases. You might find `generateClusteringExamples` in `util.py` useful for testing your code.

Do not use libraries such as Scikit-learn.

c. If we scale all dimensions in our initial centroids and data points by some non-zero factor, are we guaranteed to retrieve the same clusters after running k-means (i.e. will the same data points belong to the same cluster before and after scaling)? What if we scale only certain dimensions? If your answer is yes, provide a short explanation; if not, give a counterexample.

What we expect: This response should have two parts. The first should be a yes/no response and explanation or counterexample for the first subquestion (scaling all dimensions). The second should be a yes/no response and explanation or counterexample for the second subquestion (scaling only certain dimensions).

True if we scale all dimensions since the scaling factor can be factored out of the argmin equation and disregarded since it is applied equally to all possible centroids.

$$z_i = \arg \min_{k=1,\ldots,K} \| \alpha\phi(x_i) - \alpha\mu_k \|^2 = \arg \min_{k=1,\ldots,K} \alpha^2 \| \phi(x_i) - \mu_k \|^2 = \arg \min_{k=1,\ldots,K} \| \phi(x_i) - \mu_k \|^2$$

False if we scale only certain dimensions. Here is a counterexample:

1. $\phi(x_1) = [1, 0]$
2. $\phi(x_2) = [0, 1]$
3. $\phi(x_3) = [100, 0]$
4. $\phi(x_4) = [100, 100]$

With initial centroids $\mu_1 = [1, 1]$ and $\mu_2 = [100, 50]$, x_1 and x_2 belong to μ_1 while x_3 and x_4 belong to μ_2. If we divide the first dimension of the data points and centroids by 100, x_1, x_2, and x_3 now belong to μ_1 while x_4 now belongs to μ_4.

[1] https://jigsaw.google.com/the-current/toxicity/

[2] https://medium.com/jigsaw/unintended-bias-and-names-of-frequently-targeted-groups-8e0b81f80a23

[3] There are many varieties of utilitarianism and consequentialism – if you are interested in reading more about these principles, see here and here.

[4] For more on prioritarianism, see here and here.

[5] For more on the difference principle, see here and here.

[6] There are many anti-discrimination principles. This one is drawn from here; for others, see here . Many theories rely on perceived social group membership rather than demographic group membership. This principle also relates to corrective justice, for which see here.

[7] For more on participatory design, see here.