CS221 Summer 2023: Artificial Intelligence: Principles and Techniques

Homework 2: Sentiment Analysis

SUNet ID: alebarro
Name: Alessandro Barro
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

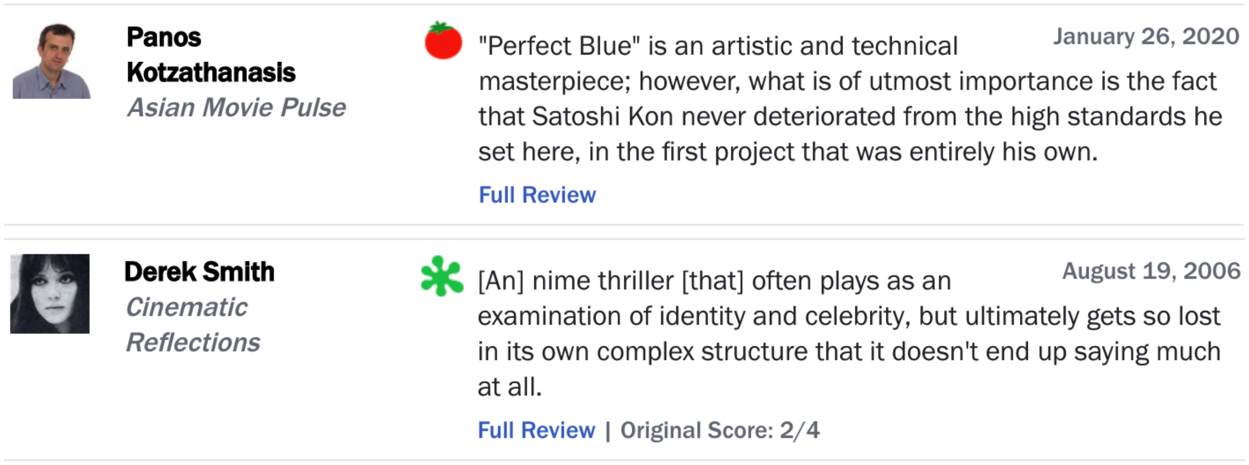
Advice for this homework:

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

Before you get started, please read the Assignments section on the course website thoroughly.

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

1. (-1) not good
2. (-1) pretty bad
3. $(+1)$ good plot
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 second 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. [2 points] 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: $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$.

Your Solution: Given the following initial set
$\eta = 0.1$, $\mathbf{w} = [0, 0, 0, 0, 0, 0]$
We can define the gradient of the hinge loss function as follows

$$\nabla_{\mathbf{w}} Loss_{hinge}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & if \quad 1 - (w\phi(x))y > 0 \\ 0 \quad otherwise \end{cases} \tag{1}$$

Let's now apply stochastic gradient descent (taking in consideration the gradient with respect to $\mathbf{w}$ of a random example instead of the max or average loss) in order to minimize the loss function over the 4 examples

I. $\phi(x_1) = (0,1,0,0,1,0)$, $y = -1$

$$\mathbf{w} = \vec{0} - 0.1((0,1,0,0,1,0) \cdot -1) = (0,-0.1,0,0,-0.1,0) \tag{2}$$

$$= (0,-0.1,0,0,-0.1,0) \tag{3}$$

II. $\phi(x_2) = (1,0,1,0,0,0)$, $y = -1$

$$\mathbf{w} = (0,-0.1,0,0,-0.1,0) - 0.1((1,0,1,0,0,0) \cdot -1) \tag{4}$$

$$= (-0.1,-0.1,-0.1,0,-0.1,0) \tag{5}$$

III. $\phi(x_3) = (0,1,0,1,0,0)$, $y = 1$

$$\mathbf{w} = (-0.1,-0.1,-0.1,0,-0.1,0) - 0.1(-(0,0.1,0,0.1,0,0) \cdot 1) \tag{6}$$

$$= (-0.1,0,-0.1,0.1,-0.1,0) \tag{7}$$

IV. $\phi(x_3) = (1,0,0,0,0,1)$, $y = 1$

$$\mathbf{w} = (-0.1,0,-0.1,0.1,-0.1,0) - 0.1(-(0.1,0,0,0,0,0.1) \cdot 1) \tag{8}$$

$$= (0,0,-0.1,0.1,-0.1,0.1) \tag{9}$$

Our final model's parameter $\mathbf{w} = (0,0,-0.1,0.1,-0.1,0.1)$

b. [2 points] 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).

Your Solution: In order to answer the question and provide a solution we should analyze the role of the last two labels of the dataset. In terms of reviews, "Not good" implies "bad", "Not bad" implies "good": this leads to the conclusion that the set of indipendent variables that compose this dataset are actually not indipendent to each other and a linear classifier won't be able to actually fit in the model. One possible solution could be mapping the indipendent varibles utilizing the following feature map $\hat{\phi}(x) = (bad : k_1, good : k_2, not : k_3)$, $k_i \in [-1, 1]$, allowing, for instance, the model to actually learn the negation "not" in the sentiment of reviews.

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 $[0, 1]$.
NOTE: Do not use math software such as Wolfram Alpha to solve this problem.

a. [2 points] 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.

Your Solution:

$$Loss(x, y, \mathbf{w}) = [\sigma(\mathbf{w}\phi(x)) - y]^2 = \left[\frac{1}{1 + \exp(-\mathbf{w}\phi(x))} - y\right]^2 \tag{10}$$

b. [3 points] 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.

Your Solution: Considering that

$$\frac{\partial}{\partial \mathbf{w}_i} p = \frac{\partial}{\partial \mathbf{w}_i} \sigma(\mathbf{w}\phi(x)) = \frac{\phi(x)\exp(-\mathbf{w}\phi(x))}{\exp(-\mathbf{w}\phi(x)) + 1} = \phi(x)(1 - p) \tag{11}$$

We can summarize the vector form in with the following notation

$$\nabla_{\mathbf{w}} Loss(x, y, \mathbf{w}) = 2\phi(x)(p - y)(1 - p) \tag{12}$$

c. [3 points] 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.

What we expect:

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

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.

Your Solution: In order to answer this question, we should consider

$$\min ||\nabla_{\mathbf{W}} Loss(x, y, \mathbf{w})|| \tag{13}$$

Hence, focus on minimizing each component of $\nabla_{\mathbf{W}} Loss(x, y, \mathbf{w})$. Since $y = 1$ from hypothesis

$$(1 - p)(p - 1) = 0 \tag{14}$$

Let's zoom into the $p = 1$ case, and we obtain

$$\exp(-\mathbf{w}\phi(x)) = 0 \tag{15}$$

Which is impossible, therefore $\nabla_{\mathbf{W}} Loss(x, y, \mathbf{w})$ will never be exactly zero, but still can by asymptotic to it for very large values of $\mathbf{w}$.

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.".

NOTE: 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. [2 points] Implement the function `extractWordFeatures`, which takes a review (string) as input and returns a feature vector $\phi(x)$ (you should represent the vector $\phi(x)$ as a `dict` in Python).

b. [4 points] Implement the function `learnPredictor` using stochastic gradient descent and minimize the 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. [2 points] 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. [2 points] 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. [3 points] 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 here.

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.

Your Solution: 1. The optimal value for n to produce the smallest validation error is $n = 5$. By the time $n > 5$ the error gets smaller by a neglegible amount iteration after iteration. Worth noting is that after this threshold, variation between train error and validation error grows significantly. This is because, for those values of n the model is able to capture enough information to make accurate predictions. If n's were lower values then the model would underfit the data trend (don't capture enough information), and on the other hand very high values of n might overfit it and overall make the model's processing way too complex.
2. n-grams might outperform (and overall get better results in terms of prediction) word feature models in cases where the input text data contains character repetitions (or other "very-human" escamotages) to emphasize the emotions correlated to the text itself, for example in the following text: "The lecture was sooooooo interesting !!!! :D"
.

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	num. examples
-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 num. 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, for example.

a. [2 points] 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 $f_{\mathbf{w}}(x) = 1$.

Your Solution: Let's take a look at the output of the hypothesis function with respect to both of the fixed classifiers

$$f_{\mathbf{w}_d}(x) = sign((-0.1, 1, 0)(1, d, t)) \tag{16}$$

$$f_{\mathbf{w}_t}(x) = sign((-0.1, 0, 1)(1, d, t)) \tag{17}$$

Now, we can clearly observe that the classifiers, in regard to make predictions, behave very similarly since $f_{\mathbf{w}_t}(x) = 1$ if-and-only-if $d, t > 0$, and both of them are independent from each other. The classifier $\mathbf{w}_d$ is sensitive to demographic mentions, ignoring the presence of actually toxic words; while classifier $\mathbf{w}_t$ focuses on the toxic words themselves, without considering demographic mentions at all. In conclusion we can affirm that both of the classifier are "incomplete", since d tends to penalize comments with demographic mentions regardless, while t comments with toxic words, which might not result in a toxic comment at all (also over-focusing on toxic comments containing demographic mentions).

b. [3 points] 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

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

Classifier D	$y = 1$	$y = -1$
$d = 1$	0	1
$d = 0$	1	0

Your Solution: 1. Let's compute the Classifier D's average loss, by considering all the losses in the data set and dividing them by the total number of points $\mathcal{D}_{\text{train}}$. The hypothesis function in the formula will indeed be $f_{\mathbf{w}_d}(x)$. We shall consider as "lossy" the groups where the classifier misfits the data, hence the cases where $(y, d) = (1, 0), (-1, 1)$.

$$\text{TrainLoss}(\mathbf{w}_d) = \frac{1}{200} \cdot (10 + 10) = 0.1 \tag{18}$$

2. The Classifier D's loss for each group, can be calculated similarly, but instead of considering the whole number of points, we will take only the ones correlated to each group. For the points where $(y, d) = (1, 1), (-1, 0)$, the average loss is zero since the model fits the standards. Let's take a look at the other two:

$(1, 0)$

$$\text{TrainLoss}_g(\mathbf{w}_d) = \frac{1}{10} \cdot 10 = 1 \tag{19}$$

$(-1, 1)$

$$\text{TrainLoss}_g(\mathbf{w}_d) = \frac{1}{10} \cdot 10 = 1 \tag{20}$$

3. The maximum group loss is

$$\text{TrainLoss}_{\max}(\mathbf{w}_d) = \max_g \text{TrainLoss}_g(\mathbf{w}_d) = 1 \tag{21}$$

c. [3 points] 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

Which classifier has lower average loss? Which classifier has lower maximum group loss? **Note the groups are still defined by d, the demographic label.**

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. Indicate which classifier has lower average loss, then indicate which classifier has lower maximum group loss.

Classifier T	$y = 1$	$y = -1$
$d = 1$	0	1
$d = 0$	1	0

Your Solution: In this case, for Classifier T, the function $f_{\mathbf{w}_t}(x) = ((-0.1, 0, 1)(1, d, t))$ will be used as the hypothesis function. 1. The "lossy" groups here are still $(y, t) = (1, 0), (-1, 1)$ and the total instances are $20 + 10 = 30$. Then the average loss equals to

$$\text{TrainLoss}(\mathbf{w}_t) = \frac{1}{200} \cdot (30) = 0.15 \tag{22}$$

2. For the groups where $(y, t) = (1, 1), (-1, 0)$, the average loss is zero since the classifier perfectly fits these instances. For the other two we will have
$(1, 0)$

$$\text{TrainLoss}_g(\mathbf{w}_t) = \frac{1}{20} \cdot 20 = 1 \tag{23}$$

$(-1, 1)$

$$\text{TrainLoss}_g(\mathbf{w}_t) = \frac{1}{10} \cdot 10 = 1 \tag{24}$$

3. The maximum group loss is simply the maximum average loss across all the groups. In this case, the maximum group loss is:

$$\text{TrainLoss}_{\max}(\mathbf{w}_t) = \max_g \text{TrainLoss}_g(\mathbf{w}_t) = 1 \tag{25}$$

In terms of average loss, Classifier D has a lower average loss (0.1) compared to Classifier T (0.15). However, both classifiers have the same maximum group loss (1). Thus, while Classifier D performs better on average, it does not outperform Classifier T in terms of "per group" performance.

d. [2 points] 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 three well-known principles of fair distribution:

(a) Your choice of algorithm should result in the greatest net benefit (benefitting the greatest number of people at the lowest cost possible). (Utilitarianism, see here and here)

(b) Your choice of algorithm should prioritize the well-being of the people who are worst-off. (Prioritarianism, see here and here, or the difference principle, see here and here)

(c) Your choice of algorithm should be such that it does not impose a disadvantage on the members of groups that have faced historical discrimination. (Avoiding compounding prior injustice, see here)

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. If you're interested in learning more about these concepts, we've compiled a short video here explaining them in more depth (watching this isn't necessary for answering the question). First, argue for using average loss as your objective by appealing to one of the three above 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? Next, do the same but argue for using maximum group loss as your objective, again appealing to one of the three principles.

What we expect: A 2-3 sentence explanation that justifies using average loss by referring to one of the three principles. Then, a 2-3 sentence explanation that justifies using maximum group loss by referring to one of the three 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.

Your Solution: The use of average loss would optimal inside a social media system, this methods provides the most benefit in comparison to every user for the lowest cost possible (Utilitarianism). The function is applied among all the data set (hence benefits the overall community) and tends to be computationally easier to implement since doesn't need to evaluate every group loss by its own as its "counterpart" max loss. On the other hand, maximum group loss could be more benificial in terms of Prioritarianism, as it evaluates case by case and promotes a well-behaving and correct community. I think that Classifier T would be more beficial in terms of the overall health state of users, and reports a more general gamma of texts, and is only slighly underperforming in comparison with Classifier D.

e. [2 points] 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.[1]

[1]For more on participatory design, see here

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.

Your Solution: I think the best approach overall would be to recruit people on a crowdsearching platform, to actually engage personalities who are estraneous from the environment and can consequentially the most impartial and critic eye to toxicities. The help of social scientists would be great in order to contextualize, parameterize, frame better the levels of toxicities. I think recruiting people from inside the community won't lead the best result possible, since when a toxic environment is established, it is hard to kick away biases, bad manners and preconceptions.

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. $\phi(x_1) = [0, 0]$
2. $\phi(x_2) = [4, 0]$
3. $\phi(x_3) = [6, 0]$
4. $\phi(x_4) = [11, 0]$

a. [2 points] 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:

(a) $\mu_1 = \phi(x_1) = [0, 0]$ and $\mu_2 = \phi(x_4) = [11, 0]$
(b) $\mu_1 = \phi(x_1) = [0, 0]$ and $\mu_2 = \phi(x_2) = [4, 0]$

What we expect: Show the cluster centers and assignments for each step, and the loss for each final clustering.

Your Solution:
(a) $\mu_1 = \phi(x_1) = (0, 0)$, $\mu_2 = \phi(x_4) = (11, 0)$
Let's first assign each data-point to the nearest cluster center

$$z_1 = argmin_K(||(0,0) - (0,0)||^2, ||(0,0) - (11,0)||^2) \rightarrow \mu_1 \tag{26}$$

$$z_2 = argmin_K(||(4,0) - (0,0)||^2, ||(4,0) - (11,0)||^2) \rightarrow \mu_1 \tag{27}$$

$$z_3 = argmin_K(||(6,0) - (0,0)||^2, ||(6,0) - (11,0)||^2) \rightarrow \mu_2 \tag{28}$$

$$z_4 = argmin_K(||(11,0) - (0,0)||^2, ||(11,0) - (11,0)||^2) \rightarrow \mu_2 \tag{29}$$

And then average the obtained z_i's to update the centroids

$$\frac{z_1 + z_2}{2} = \frac{(0,0) + (4,0)}{2} = (2, 0) \tag{30}$$

$$\frac{z_3 + z_4}{2} = \frac{(6,0) + (11,0)}{2} = (8.5, 0) \tag{31}$$

Loss is given from

$$Loss = ||(0,0) - (2,0)||^2 + ||(4,0) - (2,0)||^2 + \tag{32}$$

$$+||(6,0)-(8.5,0)||^2+||(14,0)-(8.5,0)||^2=20.5 \tag{33}$$

(b) $\mu_1=\phi(x_1)=(0,0)$, $\mu_2=\phi(x_4)=(4,0)$
Again, let's first assign each data-point to the nearest cluster center

$$z_1=argmin_K(||(0,0)-(0,0)||^2,||(0,0)-(4,0)||^2)\rightarrow\mu_1 \tag{34}$$

$$z_2=argmin_K(||(4,0)-(0,0)||^2,||(4,0)-(4,0)||^2)\rightarrow\mu_2 \tag{35}$$

$$z_3=argmin_K(||(6,0)-(0,0)||^2,||(6,0)-(4,0)||^2)\rightarrow\mu_2 \tag{36}$$

$$z_4=argmin_K(||(11,0)-(0,0)||^2,||(11,0)-(4,0)||^2)\rightarrow\mu_2 \tag{37}$$

And then average the obtained z_i's to update the centroids

$$\frac{z_1}{1}=(0,0) \tag{38}$$

$$\frac{z_2+z_3+z_4}{3}=\frac{(4,0)(6,0)+(11,0)}{3}=(7,0) \tag{39}$$

Loss is given from

$$Loss=||(0,0)-(0,0)||^2+||(4,0)-(7,0)||^2+ \tag{40}$$

$$+||(6,0)-(7,0)||^2+||(11,0)-(7,0)||^2=26 \tag{41}$$

b. [5 points] 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.

NOTE: Do not use libraries such as Scikit-learn.

c. [2 points] 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).

Your Solution: If we scale all dimensions in out initial centorids and data points by some non-zero factor, we are guaranteed to retrieve the same clusters after running k-means. K-means is indeed an algorithm that is based on euclidean distance, which is a scale-invariant operation. When we are scaling every dimension, we are in fact expanding the whole vector space, resulting in the same relative distance between data points and centroids.
Let $v, w \in \mathbb{R}^n$, $t \in \mathbb{R}$

$$d(tv, tw) = ||tv - tw|| = t||v - w|| = td(v, w) \tag{42}$$

The previous relationship is no more valid if we scale only some of the n dimension, hence we will not retrieve the same clusters after running k-means. To see this, let's expand the terms
Let $v, w \in \mathbb{R}^n$, $t \in \mathbb{R}$ $v = (v_1, v_2, ..., v_i, ...v_n)$
$w = (w_1, w_2, ..., w_i, ...w_n)$
Suppose we scale only the i-th dimension
$\hat{v} = (v_1, ..., tv_i, ..., v_n)$
$\hat{w} = (w_1, ..., tw_i, ..., w_n)$
The relative distance between the data points and centroids is not preserved since
$||\hat{v} - \hat{w}|| \neq t||v - w||$
Hence the vector space isn't expanded omogeneosly among its axes, therefore the euclidean distance is not preserved.

Submission

Submission is done on Gradescope.

Written: When submitting the written parts, make sure to select **all** the pages that contain part of your answer for that problem, or else you will not get credit. To double check after submission, you can click on each problem link on the right side and it should show the pages that are selected for that problem.

Programming: After you submit, the autograder will take a few minutes to run. Check back after it runs to make sure that your submission succeeded. If your autograder crashes, you will receive a 0 on the programming part of the assignment. Note: the only file to be submitted to Gradescope is `submission.py`.

More details can be found in the Submission section on the course website.