Homework 2

(multiple due dates; see below)

CS 485, UMass Amherst, Fall 2024

Overview: Annotation and Modeling Experiment

This assignment consists of designing an annotation task, doing annotations yourself, then analyzing and experimenting with annotations jointly collected among a group. Note there are **multiple due dates**:

- Phase 0: Due Tue Oct 1, as group submission: "Group Formation." Submit your groups to Gradescope, size 2-3. We need this just so we can send you your group's dataset. If you need help finding a group, please post on Piazza or find a group in class on Tuesday!
- Phase 1: Due Monday Oct 7, as group submission: Submit your group's annotation guidelines AND all completed individual annotations to Gradescope. Late submissions are not allowed.
- Phase 2: Due Friday Oct 11, as individual submission: Submit your PDF with answers and analysis/experiment results from the last section below.

In Phase 1, as a group you'll inspect a "reference set" of tweets, and design an annotation task that is interesting to you. You should pilot it working together, and crucially, write annotation guidelines. Next (still Phase 1), a group will receive its "target set" of 250 tweets, where each member INDIVIDUALLY will annotate it for the task the group developed. The annotations must be done without communicating among group members. Once done, the group collects its annotations and submits them plus the guidelines to Gradescope.

In Phase 2, all group members use their collected dataset, but most of your work will be done individually like a traditional individual homework assignment. You'll analyze agreement rates, diagnose issues, aggregate results to create a final dataset for yourself, and perform NLP classification experiments on it. (Note the assignment also includes some math/conceptual questions that are totally separate—you can do these any time!)
Many more details on these steps are spelled out in the next section for Problem 1.

Tasks: Every group will define their *own* unique task. Therefore the researcher will have to draw up guidelines and explanations to allow their annotators to do a good job at reliably identifying whatever categories the researcher has a goal of collecting.

It's OK if this is hard: Designing an annotation task, and collecting annotations, is tricky. This assignment is small-scale as far as annotation projects go, but hopefully it illustrates the challenges that underlie it.

Datasets. We have collected samples of publicly available English sentences for all your experiments. (The data is collected from social media, and may include offensive and inappropriate content.) They are available at:

https://drive.google.com/drive/u/1/folders/1p0HJBVWsBXt2EuD4eMaAtcG1FqCvHqOx
There are two types of datsets:

- Reference sets: You can just look at these any time. They're intended for groups to read through to help define their task. You could also use them as unlabeled data later on if you want.
- Target sets: Each group will receive one target set of 250 tweets, used for individual annotations then later analysis.

The format is JSON; see the Python package "json". We replace hyper links with "[link]". The text is encoded as UTF-8.

1 Phase 1: Task design and annotation collection

1.1 Task design

As a group, conceptualize and design a classification task for the data. Take a look at some reference data. What's an interesting, but also reasonable way to annotate the classification problem for this data? It can be any binary or multiclass classification problem. Here are a few examples:

- Does the sentence contain humor? Sarcasm?
- Does the sentence contain a location?
- Does the sentence mention an entity like person/organization name?
- Does the sentence contain abbreviations?
- Is the sentence about a certain specific topic: politics, movie, campus life, . . .
- Is the sentence written in the passive voice?

A few guidelines:

- You must have a fixed set of categories. Please use no more than 5 categories. Fewer tends to be easier to annotate.
- Do not have any categories that seem to be rare—say, less than 5% incidence. Annotators tend to find rare classes difficult to annotate.
- For many category systems, it's often useful to have a catch-all category like "Not Applicable" or "Other" or "Unknown." Of course, those three category labels can mean different things in different situations, or you might even need more than one of them. (For sentiment, "neutral" is a kind of catch-all.)
- Don't do a really simple problem with an obvious shallow textual indicator, like "Is the sentence a question?" or "Does this sentence contain more than 10 words?" This violates the spirit that we want you to do a real NLP problem. However, seemingly simple tasks might be more subtle than they first appear. For example, "Does this sentence contain an emoticon?" can sometimes be hard, if there are creative or complex ASCII-art emoticons (e.g. horizontal emoticons).

Write **annotation guidelines** for your task. These are instructions you all will use as annotators. When you work as an annotator, you have to read the guide and rely on it when doing annotations. It should be a written document, perhaps a half page long (or more if you think necessary). The guidelines should include:

- A list of the categories under consideration, including the exact string an annotators should use when typing into a spreadsheet.
- Descriptions of the categories and what they mean.
- Example sentences that are illustrative of the categories.
- A discussion of tricky corner cases, and criteria to help the annotator decide them. If you look at the data and think about how annotators could do the task, you will realize a bunch of such issues!

You are in charge of defining your task! This process of boiling it down to something specific, actionable, and thus measurable, called *operationalization*. Every group will be defining a different task, so feel free to make yours specific or unique in some way. But you will want to make it clear and as straightforward as possible for your annotators to do the task.

Deliverable: Include a copy of your annotation guidelines in your Phase 1 submission.

1.2 Annotation collection

It's time to collect annotations! This part is done individually, though remember you'll have to gather everyone's annotations together for submission. Remember, each group member now performs this individually, and on the target set. Please collect your annotations through a Google Sheets document with three columns:

1. Text
2. Label from annotator
3. Notes from annotator

Each annotator should have a separate sheet, each with the exact same texts in the text field. **Annotators should annotate independently of each other—only use the annotation guidelines document at this point.**

(If you like, you could use other software than a spreadsheet for this. It may not be worth the trouble though.)

Deliverable: In both your Phase 1 and Phase 2 submissions, include the separate CSV files for each annotator.

2 Phase 2: Annotations analysis

For this phase of work, you are responsible for writing it up as a report, which you submit individually.

2.1 Annotator Feedback

This is the only part of Phase 2 that has a group component. Do this ASAP after you've finished annotations.

Meet with your group and discuss together what general feedback you all have. What worked well or what didn't? Look at your "annotator notes" column and compare them to other members' notes to help spur discussion. Take notes on your meeting.

Deliverable: In your writeup, summarize the important issues in the annotation task. Did you or the other annotators find it easy or hard? What were the biggest issues? Did the annotators have similar or different experiences? What would you do differently if you were to revise your task?

If this were a real annotation project, what you just did would be considered the first pilot experiment, and you would work more to iteratively refine your guidelines. But for this homework, one round is enough!

2.2 Inter-annotator agreement

Now you'll conduct quantitative analysis of the agreement between annotators.

Deliverable: in your writeup, report:

1. The confusion matrix between the two annotators. Rows = category choices for Annotator1, Columns = category choices by Annotator2, Cells = the number of sentences with that pair of labels from the two annotators. The categories should be in the same order for both rows and columns, so the diagonal cells correspond to cases with agreement.

2. The observed agreement rate, the random chance agreement rate, and chance-adjusted agreement rate (kappa). Explain the calculations you conduct.

If your group is larger than two, calcuate the confusion matrix by summing across all pairs of annotators. (Technically the adjusted agreement rate will then be called "Fleiss' kappa.")

2.3 Aggregate to final dataset

Create the final annotated dataset, by combining the sets of annotations. For cases where annotators agree, you should just use that label. But where they disagree, you will have to adjudicate and decide who's right—you could take a majority vote if there's no tie, or inspect it yourself to make a final decision.

Deliverable: What principles (if any) did you use to make these decisions? (Explain in your writeup). Also, include a copy of your final dataset in CSV format, as 'final.csv' in your code/data submission.

3 Phase 2 continued: Conceptual questions

Not data related, but questions are also part of your Phase 2 submission. There's no need to wait for annotations to do these—try starting early!

3.1 Logistic regression learning

3.1.1

We'll examine the gradient of logistic regression. Assume only two features, x_1 and x_2, thus with parameter weights β_1 and β_2. Let ℓ be the logistic regression log-likelihood for a single example that happens to have ground-truth label $y = 1$:

$$\ell(\beta) = \log p_\beta(y = 1 \mid x)$$

where $p_\beta(y = 1 \mid x) = g(\beta_1 x_1 + \beta_2 x_2)$ when $g(z)$ is the logistic sigmoid function.

Derive and simplify $\partial\ell/\partial\beta_1$, the partial derivative of the log-likelihood with respect to parameter β_1. Simplify it into a form that includes $p(y = 1 \mid x)$, which we'll use to aid our intuition.[1]

3.1.2

One gradient-based learning algorithm is called Stochastic Gradient Descent, where you calculate the parameter gradient for a single example, then immediately update. For the parameter β_1 and this example, the SGD update rule[2] is

$$\beta_1^{\text{new}} = \beta_1^{\text{old}} + \frac{\partial\ell}{\partial\beta_1}$$

Say feature 1 is a document's count of the word "cow," and in the current document, "cow" does not appear at all. How will β_1, the weight for "cow," get updated? Intuitively, why does this make sense?

3.1.3

Say "cow" appears once in some document with ground-truth $y = 1$, and the LR probabilistic prediction is $p(y = 1 \mid x) = 0.6$. How does the "cow" weight get updated?

3.1.4

Say "cow" appears once in some document with ground-truth $y = 1$, and the LR probabilistic prediction is $p(y = 1 \mid x) = 0.999$. How does the "cow" weight get updated? Intuitively, does this make sense compared to the previous update under a prediction of 0.6? Explain any implications for your intuitions about the derivative.

3.2 Softmax and Sigmoid

[INLP ch. 3, #2.] Prove that softmax and sigmoid functions can be made equivalent when the number of possible labels is 2. Specifically, for any weight matrix A and vector z, show how to construct a weight vector θ such that the first element of the softmax output matches the sigmoid output:

$$[S(Az)]_1 = \sigma(\theta \cdot z)$$

[1]Hint: Brendan thinks the sigmoid form $g(z) = e^z/(1 + e^z)$ is easier to work with than $1/(1 + e^{-z})$ only because he's bad at negative signs when doing calculus. YMMV!

[2]Assuming learning rate of 1.

where $\theta \cdot z$ is a dot product, and where $S(x)$ is the softmax function, which inputs and outputs vectors of length n, and the kth element of the output is

$$[S(x)]_k = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}}$$

[*Hint:* if you're stuck getting started, try defining the dimensionality of all the relevant variables to make sure the formulas work; this may help you understand the role each one is playing. It can help to sketch out an example for yourself as well. Then if you give clear and careful definitions in your answer, it will be stronger and easier to follow.]

3.3 Precision and recall: class distribution

Imagine you have a classifier and its outputs on a bunch of data, with some precision and recall. Now imagine you add a bunch of very easy negative examples to the dataset, and your classifier correctly predicts them as negative. How do accuracy, precision, and recall change? What does this tell us about the nature of these three metrics?

4 Phase 2 continued: NLP experiments

OK great, you have some labeled data. What about NLP modeling?

Conduct experiments to build and test classifiers on your dataset. This will be a little difficult, since it is very small; fancier machine learning methods may not work well.

Split your dataset into train and test splits by simply dividing it in half. (For extra credit, try using cross-validation, which is better in this environment, but more annoying to use properly.)

4.1 BOW LogReg

Create a logistic regression model based on features of the presence/count of words. (Word counts aren't much different than word presence indicators, since sentences we provide are so short.)

4.1.1 Feature preproc.

Describe the features and preprocessing you do. You probably want to clean things a bit more beyond simply lowercasing the words. Also, choose a normalization method to normalize the features before using them for training the model. Justify your decisions. And of course, describe any hyperparameter tuning or selection that you conducted.

4.1.2 Results

Report overall accuracy, as well as the precision and recall for each class.

We recommend you use the scikit-learn version of LogisticRegression with L2 regularization, and probably with DictVectorizer to aid construction of the feature vectors.

4.1.3 Model insight: parameters

Look at your features' weights. Show the top-10 highest weighted words for each category. What do they indicate, if anything?

4.1.4 Model insight: worst mistakes

Split the data in half; train on one half and take the model's probabilistic predictions on the other half. Look at examples your model got wrong, and choose the 3 examples with highest model confidence in that wrong answer (try sorting by confidence; e.g. python list ".sort(key=...)" or numpy array ".order()"). For example, where y^* is the gold standard label, and $\hat{y}$ is the model prediction $\hat{y} = \arg\max_k p(y = k \mid x)$, this is asking for the $(x, \hat{y}, y^*)$ test set instances with highest $p(\hat{y} \mid x)$ where $\hat{y} \neq y^*$. For each print out the gold label, the prediction, and prediction confidence.

Try to analyze—why are they classified into the wrong classes? Can you learn any insight from these samples to help you improve your model?

4.2 Improved model

Try a variation of your model to see if you can get an improvement—engineering different features, or change the machine learning model, for example. Report results in a results table. Each row should be one model. The first column is the model name or description, and the second column is the accuracy (or another holistic metric, say, macro-averaged F1) on the test set.

Write up a paragraph or two on your main findings, and why you think you're finding what you found.

[Note that since your dataset is small, evaluation may be noisy. Later in the course we'll discuss statistical testing to account for evaluation variation due to the small size of a test set.]

4.3 Extra Credit: Bag of embeddings LogReg

This is extra credit since it may take more time, and we aren't fully covering this topic until later in the semester. However, it's a good fit for the problem, and integrates well into the logistic regression modeling approach you're using.

Try an alternate model—take word embeddings (vectors) for each word in the data, and average them, to construct the feature vector. You'll use "pretrained" static embeddings, which you can just download. They were created by unsupervised learning on a general text corpus. For each word w in the document, you look up its word embedding (vector), v_w. Say it's a J-dimensional vector, so $v_w \in \mathbb{R}^J$. The feature vector for a document will be $x \in \mathbb{R}^J$, which is simply defined as the vector average of all word embeddings:

$$x = \frac{1}{|\text{doc}|} \sum_{w \in \text{doc}} v_w$$

or to be even more specific, the j'th element of x is just the average of values for that dimension across the words:

$$x_j = \frac{1}{|\text{doc}|} \sum_{w \in \text{doc}} v_{w,j}$$

Use any pretrained GLOVE word embeddings that you like, which can be downloaded from this link: `https://nlp.stanford.edu/projects/glove/` We recommend using the "Twitter" version (glove.twitter.27B) because it is pretrained on similar source of text to the data that you are using for the assignment. Construct a logistic regression classifier based on averaged word embeddings. Report any details in getting it to work, and its performance.