Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission. The course assignment policy is available at [http://nlp.cs.princeton.edu/cos484](http://nlp.cs.princeton.edu/cos484).

- This assignment contains 3 problems.
- We highly recommended that you typeset your submissions in \LaTeX. Use the template provided on the website for your answers. Include your name and NetIDs with your submission. If you wish to submit hand written answers, you can scan and upload the pdf.
- Assignments must be uploaded to Gradescope by **11:59pm** on Oct 7th, 2019.
- Late submissions will incur a penalty of 10% for each day, up to a maximum of 4 days beyond which submissions will not be accepted.

**Problem 1: Naive Bayes**

(5 + 5 = 10 points)

Given the following documents with counts for key sentiment words, with positive or negative class labeled:

<table>
<thead>
<tr>
<th>doc</th>
<th>“good”</th>
<th>“poor”</th>
<th>“great”</th>
<th>“terrible”</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₁</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>D₂</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>−</td>
</tr>
<tr>
<td>D₃</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>−</td>
</tr>
<tr>
<td>D₄</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>−</td>
</tr>
<tr>
<td>D₅</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>D₆</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>−</td>
</tr>
</tbody>
</table>

(a) Compute a naive Bayes model (with add-1 smoothing) on the above documents and assign a class (+/−) to the following sentence:

**great characters and good acting, but terrible plot**

You should just use \( V = \{ \text{good, poor, great, terrible} \} \). Provide details for your answer. **Note that this problem doesn't require programming.**

(b) For sentiment classification and a number of other text classification tasks, whether a word occurs or not matters more than its frequency. A variant of naive Bayes, called **binarized naive Bayes**, is to clip the word counts in each document at 1 (i.e. if a word appears multiple times in a document, it should be only counted once). Compute a binarized naive Bayes model (with add-1 smoothing) on the same documents and predict the label of the above sentence and provide details for your answer. Which of the two models do you prefer and why?

**Problem 2: Understanding word2vec**

(3 + 4 + 4 + 3 + 6 = 20 points)

Given a sequence of words \( w_1, \ldots, w_T \) and context size \( c \), the training objective of Skipgram that we learned in the class is:

\[
\mathcal{L} = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} \mid w_t),
\]

where \( P(w_o \mid w_t) \) is define as:

\[
P(w_o \mid w_t) = \frac{\exp \left( \langle \mathbf{v}_{w_o}^\top, \mathbf{v}_{w_t} \rangle \right)}{\sum_{w \in V} \exp \left( \langle \mathbf{v}_{w}^\top, \mathbf{v}_{w_t} \rangle \right)}.
\]
and $v_w$ and $v'_w$ for every $w \in V$ represent the “target” and “context” vectors.

(a) Derive the following gradient:

$$- \frac{\partial \log P(w_o \mid w_t)}{\partial v_{w_t}}$$

(b) Consider the following sequence of words ($T = 14$):

$$a \text{ dog runs} <s> a \text{ cat runs} <s> a \text{ dog and a cat run}$$

Denote $V = \{a, \text{dog}, \text{cat, and, runs, run, <s>}\}$, compute the gradient of $\mathcal{L}$ with respect to $v_{\text{dog}}$ for this example with context size $c = 1$.

(c) For the above example, compute the gradient of $\mathcal{L}$ with respect to $v'_{\text{runs}}$ with context size $c = 2$.

(d) Imagine that we train the model on a large corpus (e.g. English Wikipedia). Describe the effects of context size $c$ to the resulting word vectors $v_w$, i.e. what if we use context size $c = 1, 5, \text{or 100}$?

(e) In practice, it is very expensive to compute the full cross-entropy loss because it is proportional to the vocabulary size $|V|$. One computationally efficient method called **negative sampling** is to randomly sample $K$ negative words $w_1, \ldots, w_K \in V$ and the loss function is approximated as:

$$- \log P^*(w_o \mid w_t) = - \log \sigma(v_w^T v_{w_t}) - \sum_{k=1}^{K} \log \sigma(-v_w^T v_{w_k}),$$

where $\sigma$ is the sigmoid function. Derive the following gradient:

$$- \frac{\partial \log P^*(w_o \mid w_t)}{\partial v_{w_t}}$$

**Problem 3: Building classifiers for sentiment analysis (programming)**

(20 + 20 = 40 points)

Note that this problem is worth 20 + 20 = 40 points ((a) and (b)). Problem 3(c) is optional but highly encouraged and you’d have a chance to receive extra credit of up to 10 points!

This problem will require programming in Python 3. The goal is to build a Naive Bayes model and a logistic regression model that you learnt from the class on a real-world sentiment classification dataset. Finally, you will explore how to design better features and improve the accuracy of your models for this task.

The dataset you will be using is collected from movie reviews online. All the examples have been already tokenized and lowercased and each example is labeled as 1 (positive) or 0 (negative). The dataset has been split into a training, a development and a test set. We will only release the training and development sets and you are required to upload your code to Gradescope for final grading on the test set.

To get started, you should first download the data and starter code from https://nlp.cs.princeton.edu/cos484/assignments/a2.zip. Try to run:

```
python sentiment_classifier.py --model AlwaysPositive
```

This will load the data and run a default classifier AlwaysPositive which always predicts label 1 (positive) and evaluate it on both the training set and development set. You should be able to see the reported dev accuracy = 50.92%. That says, always predicting positive isn’t that good. Let’s try to build better classifiers for sentiment analysis!

(a) **(Naive Bayes)** In this part, you should implement a Naive Bayes model with add-1 smoothing, as we taught in the class. You are required to implement the `train_nb` function and `NaiveBayesClassifier` class in `models.py`.

- `train_nb`: it takes a list of training examples (class `SentimentExample`) and a feature extractor (class `FeatureExtractor`) as input and is required to output an instance of `NaiveBayesClassifier`. 
• NaiveBayesClassifier: you should at least implement __init__ and predict functions. Feel free to add more arguments to __init__ (actually we expect you to do so!) but don’t change the arguments in predict — it should just take an example ex (class SentimentExample) and predict a label 1 or 0.

• You would probably want to take a look at the UnigramFeatureExtractor class that we have implemented for you already!

(i) (12 points) After you are done, you should run:

```
python sentiment_classifier.py --model NaiveBayes --feature unigram
```

Report your training accuracy and development accuracy on this dataset. How big is the gap between the two? Explain why this might be happening. Note that we will also evaluate your model on the test set.

(ii) (3 points) List the 10 words that, under your model, have the highest ratio of \( \frac{P(w|\text{positive})}{P(w|\text{negative})} \) (the most distinctly positive words). List the 10 words with the lowest ratio. What trends do you see?

(iii) (5 points) Naive Bayes can also use any set of features instead of single words. Implement and experiment with BigramFeatureExtractor in models.py. For a sentence it is not funny, you should extract bigrams it|is, is|not and not|funny. Run

```
python sentiment_classifier.py --model NaiveBayes --feature bigram
```

Report your training accuracy and development accuracy on this dataset. How does it compare to the unigram model in (i)? Explain why. Note that we will also evaluate your model on the test set.

(b) (Logistic Regression) In this part, you should implement a Logistic Regression model. You are required to implement the train_lr function and LogisticRegressionClassifier class in models.py. The formats of train_lr and LogisticRegressionClassifier are very similar to train_nb and NaiveBayesClassifier.

(i) (15 points) First, implement a logistic regression model without regularization. And then train your model with unigram and bigram features:

```
python sentiment_classifier.py --model LogisticRegression --feature unigram
python sentiment_classifier.py --model LogisticRegression --feature bigram
```

Report both training and development accuracy on the dataset. How do you compare the results of the unigram and bigram models? How do you compare them with the Naive Bayes results? Hint: You might find that batch optimization is too slow. Try to use stochastic gradient descent or (mini-batch) stochastic gradient descent!

(ii) (5 points) Next, we would like to experiment with \( l^2 \) regularization \( R(\theta) = \alpha \|\theta\|^2 \). Plot the accuracy on train and development sets as a function of \( \alpha = \{0, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\} \). You only need to experiment with unigram features for this part. Explain what you have observed. Set \( \alpha \) to the optimal value you find in this problem and we will evaluate your model on the test set.

(c) (Features) In the last part, you’ll explore and implement a more sophisticated set of features. All you need to modify is the class CustomizedFeatureExtractor in models.py. Here are some common strategies (you are welcome to implement some of them but try to come up with more!):

• Remove stopwords (e.g. a, the, in).

• Clip the feature count at one (see “binarized naive Bayes” in Problem 1).

• Use a mixture of unigrams, bigrams or trigrams.

• Negation is also a very important aspect in sentiment analysis. For example, sentences I really like this movie and I don’t like this movie only differ in one word, but express opposite meanings. A simple baseline is to prepend the prefix NOT_ to every word after a token of logical negation (n’t, not, no, never) until the next punctuation mark. Thus the phrase I don’t like the movie , but becomes I don’t NOT_like NOT_the NOT_movie , but.
Use your creativity for this problem and try to obtain an accuracy as high as possible on the development set! After you implement CustomizedFeatureExtractor, run:

```bash
python sentiment_classifier.py --model NaiveBayes --feature customized
python sentiment_classifier.py --model LogisticRegression --feature customized
```

Describe the features that you have implemented and report your development accuracy for both models. We’ll evaluate your two models on the test set. You will receive up to 10 bonus points: up to 5 points based on the novel features you try and the rest based on how well your models perform compared to other submissions:

\[
\text{Bonus} = 5 + 5 \times \frac{1}{\text{rank}}
\]

e.g. if you rank first in the class, you will receive the full bonus point!