HW2: Probability

Re-submit Assignment

Due Tuesday by 2:30pm **Points** 100 **Submitting** a file upload **File Types** zip
Available Feb 2 at 4pm - Feb 9 at 6pm 7 days

Summary

In this assignment, you'll be using your knowledge of probability to tackle the problem of document classification. You are given the skeleton code **classify.py** , and you will need to implement the functions as required**.**

This assignment will use the probability background introduced in lecture in order to make predictions in uncertain situations. You'll apply concepts like conditional probability, priors, and conditional independence. You'll also improve your Python skills.

Document Classification

We'll be reading in a corpus (a collection of documents) with two possible labels and training a classifier to determine which label a query document is more likely to have.

Here's the twist: the corpus is created from CS 540 essays about AI from 2016 and 2020 *on the same topic*. Based on training data from each, you'll be predicting whether an essay was written in 2020 or 2016. (Your classifier will probably be bad at this! It's okay, we're looking for a very subtle difference here.)

- You will need **hw2.zip**, which includes 2 directories: corpus and EasyFiles.

There are 3 main steps you will complete.

1. Loading the data into a convenient representation.
    - This part will involve 3 functions. The main one, **load_training_data**, will give the final representation. You will only need to implement the helper function, **create_bow**, the code for the other two is already provided. However, please read the information about all of them since you'll need to use all 3. High-level descriptions of the three functions are:
        - **create_vocabulary(directory, cutoff)** -- create and **return** a vocabulary as a list of word types with counts >= cutoff in the training directory
        - **create_bow(vocab, filepath)** -- create and **return** a bag of words Python dictionary from a single document

        - **load_training_data(vocab, directory)** -- create and **return** training set (bag of words Python dictionary + label) from the files in a training directory
    - We will define bag of words and give more detailed descriptions of each function in the sections that follow.
2. Computing probabilities from the constructed representation.
    - This part will involve two functions. You must implement both.
        - **prior(training_data, label_list)** -- given a training set, estimate and **return** the prior probability *P(label)* of each label
        - **p_word_given_label(vocab, training_data, label)** -- given a training set and a vocabulary, estimate and return the class conditional distribution $P\left(word \mid label\right)$ over all words for the given label using smoothing
    - We will define smoothing and give more detailed descriptions of each function in the sections that follow.
3. Use the probabilities computed to create a prediction model and use this model to classify test data.
    - This part will involve two functions. You must implement both. Note, **train** will use all of the functions from the previous parts.
        - **train(training_directory, cutoff)** -- load the training data, estimate the prior distribution *P(label)* and class conditional distributions $P\left(word \mid label\right)$, **return** the trained model
        - **classify(model, filepath)** -- given a trained model, predict the label for the test document (see below for implementation details including the **return** value) --- Note, this high-level function should also use **create_bow(vocab, filepath)**

You are allowed to write additional helper functions. Your submitted code **should not produce any additional printed output**.

Toy Example

For some of the smaller helper functions, we'll be using a very simple version of a training data directory. The top level directory is called EasyFiles/. It contains two subdirectories, like your actual training data directory will, called 2020/ and 2016/. EasyFiles/2016/ contains two files, 0.txt (hello world) and 1.txt (a dog chases a cat.). EasyFiles/2020/ contains one file, 2.txt (it is february 19, 2020.). Each of these files has been pre-processed like the corpus, so all words are in lower case and all tokens (including punctuation) are on different lines:

```
it
is
february
19
,
2020
.
```

You may wish to create a similar directory structure on your computer.

Step1 (Data Representation)

Create Vocabulary (Given to you)

Our training directory structure as provided in the corpus is intentional: under train/ are two subdirectories, 2016/ and 2020/, each of which function as the labels for the files they contain. We've pre-processed the corpus, so that every line of the provided files contains a **single token**. Tokens are also referred to as word types in these instructions.

To create the vocabulary (list of relevant words) for your classifier, traverse **BOTH** of these subdirectories under train/ **(note: do not include test/)** and count the number of times a word type appears in any file in either directory.

As a design choice, we will exclude any word types which appear at a frequency **strictly less than** the cutoff argument (cutoff = 1 means retain all word types you encounter). Return a sorted list of these word types.

```
>>> create_vocabulary('./EasyFiles/', 1)
=> [',', '.', '19', '2020', 'a', 'cat', 'chases', 'dog', 'february', 'hello', 'is', 'it', 'world']
>>> create_vocabulary('./EasyFiles/', 2)
=> ['.', 'a']
```

Create Bag of Words (Implement)

A *bag of words*, is a simple way to represent a document, **d**, using a python dictionary. Specifically, a bag of words representation keeps track of each word in the document and the total number of occurrences of each word. If **bow** is your python dictionary, it should be the case that each word in the document is a key in this dictionary. Then, for a word **w** appearing in the document, you should have that **bow[w] = c(w,d)** where **c(w,d)** is the count of times **w** appears in the document, **d**. In other words, the dictionary has (key,value)-pair **(w,c(w,d))** for each word **w** in the document. For this assignment, we will only keep track of words that appear more times than the given **cutoff**. The constructed vocabulary gives exactly the words appearing enough times to warrant being a key in the **bow** dictionary.

This function (create_bow) takes a path to a text file (assume a valid format, one token per line), reads the file in, creates a bag-of-words representation based on the vocabulary, and returns the bag-of-words in dictionary format. Give all counts of word types not in the vocabulary to **out of vocabulary** (OOV) (see below).

```
>>> vocab = create_vocabulary('./EasyFiles/', 1)
>>> create_bow(vocab, './EasyFiles/2016/1.txt')
=> {'a': 2, 'dog': 1, 'chases': 1, 'cat': 1, '.': 1}
>>> create_bow(vocab, './EasyFiles/2020/2.txt')
=> {'it': 1, 'is': 1, 'february': 1, '19': 1, ',': 1, '2020': 1, '.': 1}
```

If you encounter a word type that **does not** appear in the provided vocabulary, add the non-string value `None` as a special key to represent OOV. Collect counts for any such OOV words.

```
>>> vocab = create_vocabulary('./EasyFiles/', 2)
>>> create_bow(vocab, './EasyFiles/2016/1.txt')
=> {'a': 2, None: 3, '.': 1}
```

Load Training Data (Given to you)

Once you can create a bag-of-words representation for a single text document, load the entire contents of the training directory into such Python dictionaries, label them with their corresponding subdirectory label ('2016' or '2020' as strings) and return them in a list of length n=number of training documents.

Python's os module (https://docs.python.org/3/library/os.html) will be helpful here, particularly its `listdir()` function.

```
>>> vocab = create_vocabulary('./EasyFiles/', 1)
>>> load_training_data(vocab,'./EasyFiles/')
=> [{'label': '2020', 'bow': {'it': 1, 'is': 1, 'february': 1, '19': 1, ',': 1, '2020': 1, '.': 1}},
    {'label': '2016', 'bow': {'hello': 1, 'world': 1}},
    {'label': '2016', 'bow': {'a': 2, 'dog': 1, 'chases': 1, 'cat': 1, '.': 1}}]
```

The dictionaries in this list do **not** need to be in any particular order. You should notice that the directory string **will always** include a trailing '/' as shown here.

NOTE: All subsequent functions that have a `training_data` parameter should expect it to be in the format of the output of this function, a list of two-element dictionaries with a label and a bag-of-words.

Step2 (Compute probabilities)

Log Prior Probability

This method should return the log probability $\log P(label)$ of the labels in the training set. In order to calculate these, you will need to count the number of documents with each label in the training data, found in the `training/` subdirectory.

```
>>> vocab = create_vocabulary('./corpus/training/', 2)
>>> training_data = load_training_data(vocab,'./corpus/training/')
>>> prior(training_data, ['2020', '2016'])
=> {'2020': -0.32171182103809226, '2016': -1.2906462863976689}
```

You are required to use **add-1 smoothing** (Laplace smoothing) here. Since we only have two labels here, the equation is:

$P(label = y) = \frac{N_{FilesWithLabel=y}+1}{TotalNumOfFiles+2}$

Where the **N** above indicates "the number of". Note that the return values are the **natural log** of the probability. In the code, we must contend with the possibility of *underflow*: this can occur when we take the product of very small floating point values. As such, all our probabilities in this program will be **log probabilities**, to avoid this issue.

Log Probability of a Word, Given a Label

This function (p_word_given_label) returns a dictionary consisting of the log conditional probability of all word types in a vocabulary (plus OOV) given a particular class label, $logP(word|label)$. To compute this probability, you will use **add-1 smoothing** to avoid zero probability.

The quanitity $P\left(word \mid label\right)$ measures how likely we see this word given that the document is one with this label. Using the definition of conditional probability, this is the total probability of word in documents with this label, divided by the probability of that label. Doing some simplifications, we get the formula:

$$P\left(word \mid label\right) \ = \ \frac{c(word)+1}{wc\ +|V|+1}$$

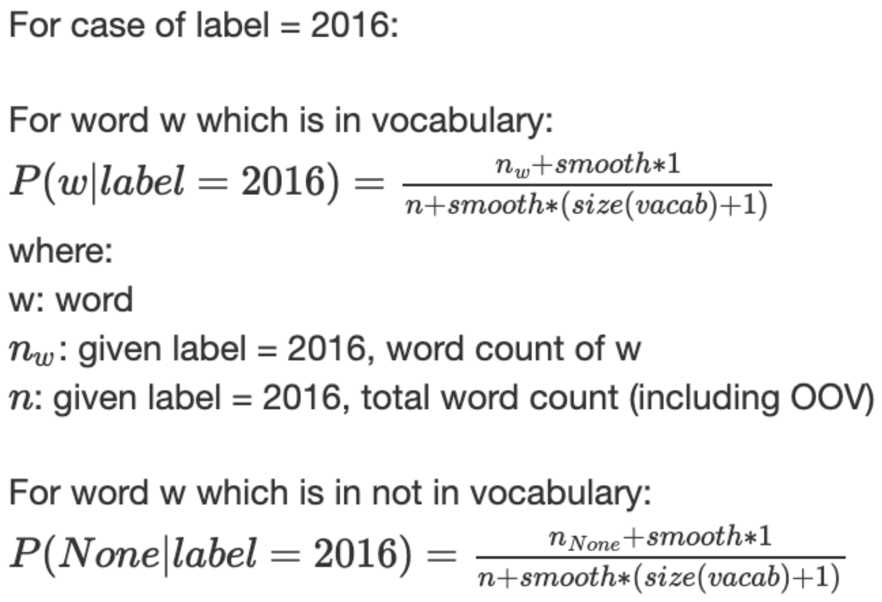
where **c(word)** is the total word count over all documents of the given label, i.e.

$$c\left(word\right) \ = \ \sum{}_{label(d)\ =\ label}\ c\left(word, d\right)$$

wc is the total word count, $wc \ = \ \sum_{label(d)\ =\ label} \sum_{w\ \in d} c\left(w, d\right)$, note $w\ \in d$ ranges over all words appearing in document d (regardless of being in the vocabulary or not). Note, label(d) = *label* means range over all documents d with the label, *label.*

|V| is the size of the vocabulary.

The formulas above apply to any word even if out of the vocabulary. However, all OOV words should be treated as one word, None, as mentioned before. See the Note below. Here's an example of the formulas written in a different way (here smooth = 1):



For case of label = 2016:

For word w which is in vocabulary:

$$P(w|label = 2016) = \frac{n_w + smooth * 1}{n + smooth * (size(vacab) + 1)}$$

where:
w: word
n_w: given label = 2016, word count of w
n: given label = 2016, total word count (including OOV)

For word w which is in not in vocabulary:

$$P(None|label = 2016) = \frac{n_{None} + smooth * 1}{n + smooth * (size(vacab) + 1)}$$

```
>>> vocab = create_vocabulary('./EasyFiles/', 1)
>>> training_data = load_training_data(vocab, './EasyFiles/')
>>> p_word_given_label(vocab, training_data, '2020')
=> {',': -2.3513752571634776, '.': -2.3513752571634776, '19': -2.3513752571634776, '2020': -2.351375257163477
6, 'a': -3.044522437723423, 'cat': -3.044522437723423, 'chases': -3.044522437723423, 'dog': -3.04452243772342
3, 'february': -2.3513752571634776, 'hello': -3.044522437723423, 'is': -2.3513752571634776, 'it': -2.35137525
71634776, 'world': -3.044522437723423, None: -3.044522437723423}
>>> p_word_given_label(vocab, training_data, '2016')
=> {',': -3.091042453358316, '.': -2.3978952727983707, '19': -3.091042453358316, '2020': -3.091042453358316,
 'a': -1.9924301646902063, 'cat': -2.3978952727983707, 'chases': -2.3978952727983707, 'dog': -2.3978952727983
707, 'february': -3.091042453358316, 'hello': -2.3978952727983707, 'is': -3.091042453358316, 'it': -3.0910424
53358316, 'world': -2.3978952727983707, None: -3.091042453358316}
```

```
>>> vocab = create_vocabulary('./EasyFiles/', 2)
>>> training_data = load_training_data(vocab, './EasyFiles/')
>>> p_word_given_label(vocab, training_data, '2020')
=> {'.': -1.6094379124341005, 'a': -2.302585092994046, None: -0.35667494393873267}
>>> p_word_given_label(vocab, training_data, '2016')
=> {'.': -1.7047480922384253, 'a': -1.2992829841302609, None: -0.6061358035703157}
```

Note: In this simple case, we have no words in our training set that are out-of-vocabulary. With the cutoff of 2 in the real set, we will see a number of words in the training set which are still out-of-vocabulary and map to `None`. These counts *should be used* when calculating $P(\text{None}|y = \text{label})$, and the existence of an "out-of-vocabulary" word type should be used when calculating all probabilities.

Also, some properties of logs may make these computations easier!

Step3 (Training and Classifying)

Train

Given the location of the training directory and a cutoff value for the training set vocabulary, use the previous set of helper functions to create the following trained model structure in a Python dictionary:

```
{
  'vocabulary': <the training set vocabulary>,
  'log prior': <the output of prior()>,
  'log p(w|y=2016)': <the output of p_word_given_label() for 2016>,
  'log p(w|y=2020)': <the output of p_word_given_label() for 2020>
}
```

For the EasyFiles data and a cutoff of 2, this would give (formatted for readability):

```
>>> train('./EasyFiles/', 2)
=> {'vocabulary': ['.', 'a'],
    'log prior': {'2020': -0.916290731874155, '2016': -0.5108256237659905},
    'log p(w|y=2020)': {'.': -1.6094379124341005, 'a': -2.302585092994046, None: -0.35667494393873267},
    'log p(w|y=2016)': {'.': -1.7047480922384253, 'a': -1.2992829841302609, None: -0.6061358035703157}}
```

The values for `None` are so high in this case because the majority of our training words are below the cutoff and are therefore out-of-vocabulary.

Classify

Given a trained model, this function will analyze a single test document and give its prediction as to the label for the document. The return value for the function must have the Python dictionary format

```
{
 'predicted y': <'2016' or '2020'>,
 'log p(y=2016|x)': <log probability of 2016 label for the document>,
 'log p(y=2020|x)': <log probability of 2020 label for the document>
}
```

where log p(y=2016|x) denotes the log probability of file x being label 2016. Using Baye's rule and conditional independence, we can compute these probabilities using the conditionals computed in Step2. Specifically, we get the formula:

$$p\,(y\ =\ label \mid x) = P\,(label) \prod_{w\ \in x} P\,(w \mid label)$$

The label for a test document *x (the predicted y above)* is the argmax of the following estimate:

$label_x = argmax_{label \in \{2016, 2020\}} P(label) \prod_i P(word_i|label)$, where $word_i$ corresponds to each word in the file x.

Recall: use **log probabilities**! When you take the log of a product, you should sum the logs of the operands. So the formula you should use is:

$$label_x = argmax_{label \in \{2016, 2020\}} (P(label) + \sum_i P(word_i|label))$$

and similarly for computing $p\,(y \;=\; label \mid x)$. Where all the **P** functions refer to the log probabilities you computed in the previous parts. You will not have to take log of these **P**'s again since they should already logs in them.

```
>>> model = train('./corpus/training/', 2)
>>> classify(model, './corpus/test/2016/0.txt')
=> {'log p(y=2020|x)': -3906.351945884105, 'log p(y=2016|x)': -3916.458747858926, 'predicted y': '2020'}
```

Deliverables

Please submit your files in a zip file named **hw2_<netid>.zip**, where you replace <netid> with your netID (your wisc.edu login). Inside your zip file, there should be **only** one file named: **classify.py**. Do not submit a Jupyter notebook .ipynb file. Make sure you use python3!

Note:

1. The directory path will always **end with a '/'**.

2. The grading will be conducted **automatically**, please follow the guidelines strictly. We grade on computers with python3.

HW2

Criteria	Ratings		Pts
Bag of Words	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
Priors	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
Conditionals	**30 pts** **Full Marks**	**0 pts** **No Marks**	30 pts
Train	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
Classify	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
			Total Points: 100