# HW1

UNI: rl3161

November 2020

## Problem 1 - Text Classification with Naive Bayes

1. Based on this data, estimate the prior probability for a random email to be spam or ham if we don't know anything about its content?
   Solution: $P(Spam) = \frac{Count(Spam)}{\sum_{y_i \in Y} Count(y_i)} = \frac{3}{5}$. Thus, $P(Ham) = 1 - P(Spam) = \frac{2}{5}$.

2. Based on this data, estimate the conditional probability distributions for each word given the class.
   Solution:

| Word | Spam | Ham |
|---|---|---|
| buy | 1/3 | 0 |
| car | 1/3 | 1/2 |
| Nigeria | 2/3 | 1/2 |
| profit | 2/3 | 0 |
| money | 1/3 | 1/2 |
| home | 1/3 | 1 |
| bank | 2/3 | 1/2 |
| check,wire | 1/3 | 0 |
| fly | 0 | 1/2 |

3. Using Naive Bayes' approach and your probability estimates, what is the predicted class label for each of the following emails? Show your calculation.
   $C$ is a positive constant of $P(Word)$.
   Nigeria:
   $P(Spam|Nigeria) = P(Spam) * P(Nigeria|Spam) * C = \frac{3}{5} * \frac{2}{3} * C = 0.4C$
   $P(Ham|Nigeria) = P(Ham) * P(Nigeria|Ham) * C = \frac{2}{5} * \frac{1}{2} * C = 0.2C$
   So, it is Spam email.
   Nigeria home:
   $P(Spam|Nigeria, home) = P(Spam) * P(Nigeria|Spam) * P(home|Spam) = \frac{3}{5} * \frac{2}{3} * \frac{1}{3} * C = \frac{2}{15}C = \frac{4}{30}C$
   $P(Ham|Nigeria, home) = P(Ham) * P(Nigeria|Ham) * P(home|Ham) =$

$\frac{2}{5} * \frac{1}{2} * 1 * C = \frac{2}{10}C = \frac{6}{30}C$
So, it is Ham email.
home bank money:
$P(Spam|home, bank, money) = P(Spam)*P(home|Spam)*P(bank|Spam)*$
$P(money|Spam) = \frac{3}{5} * \frac{1}{3} * \frac{2}{3} * \frac{1}{3} * C = \frac{2}{45}C = \frac{4}{90}C$
$P(Ham|home, bank, money) = P(Ham)*P(home|Ham)*P(bank|Ham)*$
$P(money|Ham) = \frac{2}{5} * 1 * \frac{1}{2} * \frac{1}{2} * C = \frac{1}{10}C = \frac{9}{90}C$
So, it is Ham email.

## Problem 2 - Bigram Models

Proof: Use induction.
Suppose $\sum_{w_1,...,w_n} P(w_1, ..., w_n) = 1$. If we can prove $\sum_{w_1,...,w_{n+1}} P(w_1, ..., w_{n+1}) = 1$, then it is true.
We know $\sum_{w_1,...,w_n} P(w_1|START)*P(w_1|w_2)*...*P(w_n|w_{n-1}) = 1$. $\sum_{w_n,w_n+1} P(w_{n+1}, w_n) = 1$ and $\sum_{w_n} P(w_n) = 1$ is ground truth. Since we just sum up all possibilities, the sum must be 1. Then,

$$\begin{aligned}
&\sum_{w_1,...,w_{n+1}} P(w_1, ..., w_{n+1}) \\
&= \sum_{w_1,...,w_n} P(w_1|START) * P(w_1|w_2) * ... * P(w_n|w_{n-1}) * P(w_{n+1}|w_n) \\
&= 1 * \sum_{w_n,w_n+1} P(w_{n+1}|w_n) \\
&= \frac{\sum_{w_n,w_n+1} P(w_{n+1}, w_n)}{\sum_{w_n} P(w_n)} \\
&= 1
\end{aligned} \tag{1}$$

Thus, $\sum_{w_1,...,w_n} P(w_1, ..., w_n) = 1$ is true for all n with a vocabulary size of V.

## Programming Component - Building a Trigram Language Model

```
import sys
from collections import defaultdict
import math
import random
import os
import os.path
import numpy as np

"""
```

```
COMS W4705 - Natural Language Processing - Fall B 2020
Homework 1 - Programming Component: Trigram Language
    Models
Yassine Benajiba
"""


def corpus_reader(corpusfile, lexicon=None):
    with open(corpusfile, 'r') as corpus:
        for line in corpus:
            if line.strip():
                sequence = line.lower().strip().split()
                if lexicon:
                    yield [word if word in lexicon else "
                        UNK" for word in sequence]
                else:
                    yield sequence


def get_lexicon(corpus):
    word_counts = defaultdict(int)
    for sentence in corpus:
        for word in sentence:
            word_counts[word] += 1
    return set(word for word in word_counts if
        word_counts[word] > 1)


def get_ngrams(sequence, n):
    """
    COMPLETE THIS FUNCTION (PART 1)
    Given a sequence, this function should return a list
        of n-grams, where each n-gram is a Python tuple.
    This should work for arbitrary values of 1 <= n < len
        (sequence).
    """
    final_ngrams = []
    sequence = ['START'] + sequence + ['STOP'] if n == 1
        else ['START'] * (n-1) + sequence + ['STOP']
    for i in range(0, len(sequence) - n + 1):
        final_ngrams.append(tuple(sequence[i:i + n]))

    return final_ngrams


class TrigramModel(object):
```

```python
def __init__(self, corpusfile):

    # Iterate through the corpus once to build a
        lexicon
    generator = corpus_reader(corpusfile)
    self.lexicon = get_lexicon(generator)
    self.lexicon.add("UNK")
    self.lexicon.add("START")
    self.lexicon.add("STOP")

    # Now iterate through the corpus again and count
        ngrams
    generator = corpus_reader(corpusfile, self.
        lexicon)
    self.count_ngrams(generator)

def count_ngrams(self, corpus):
    """
    COMPLETE THIS METHOD (PART 2)
    Given a corpus iterator, populate dictionaries of
         unigram, bigram,
    and trigram counts.
    """

    self.unigramcounts = defaultdict(int)  # might
        want to use defaultdict or Counter instead
    self.bigramcounts = defaultdict(int)
    self.trigramcounts = defaultdict(int)

    ##Your code here
    for corp in corpus:
        for idx, grams in enumerate([get_ngrams(corp,
             1), get_ngrams(corp, 2), get_ngrams(corp,
             3)]):
            for gram in grams:
                if idx == 0:
                    self.unigramcounts[gram] += 1
                elif idx == 1:
                    self.bigramcounts[gram] += 1
                else:
                    self.trigramcounts[gram] += 1
                    if gram[:2] == ('START', 'START')
                        :
                         self.bigramcounts[('START', '
                            START')] += 1
```

```
        return

    def raw_trigram_probability(self, trigram):
        """
        COMPLETE THIS METHOD (PART 3)
        Returns the raw (unsmoothed) trigram probability
        """
        if self.bigramcounts[trigram[0:2]] == 0:
            return 0.0
        return float(self.trigramcounts[trigram] / self.
            bigramcounts[trigram[0:2]])

    def raw_bigram_probability(self, bigram):
        """
        COMPLETE THIS METHOD (PART 3)
        Returns the raw (unsmoothed) bigram probability
        """
        if self.unigramcounts[bigram[0:1]] == 0:
            return 0.0
        return float(self.bigramcounts[bigram] / self.
            unigramcounts[bigram[0:1]])

    def raw_unigram_probability(self, unigram):
        """
        COMPLETE THIS METHOD (PART 3)
        Returns the raw (unsmoothed) unigram probability.
        """
        if not hasattr(self, 'totalWordCount'):
            self.totalWordCount = sum(self.unigramcounts.
                values())
            self.totalWordCount -= self.unigramcounts[('
                Start',)] + self.unigramcounts[('STOP',)]

        # hint: recomputing the denominator every time
            the method is called
        # can be slow! You might want to compute the
            total number of words once,
        # store in the TrigramModel instance, and then re
            -use it.
        return float(self.unigramcounts[unigram] / self.
            totalWordCount)

    def generate_sentence(self, t=20):
        """
        COMPLETE THIS METHOD (OPTIONAL)
```

```python
        Generate a random sentence from the trigram model
            . t specifies the
        max length, but the sentence may be shorter if
            STOP is reached.
        """
        start = (None, 'START', 'START')
        sentence = []
        i = 0

        while i < t and start[2] != 'STOP':
            i += 1
            candidates = [gram for gram in self.
                trigramcounts.keys() if gram[:2] == start
                [1:]]
            possibility = [self.raw_trigram_probability(
                trigram) for trigram in candidates]

            next_word = np.random.choice(a=[trigram[2]
                for trigram in candidates], size=1, p=
                possibility)[0]
            start = (start[1], start[2], next_word)
            sentence.append(next_word)

        return sentence

    def smoothed_trigram_probability(self, trigram):
        """
        COMPLETE THIS METHOD (PART 4)
        Returns the smoothed trigram probability (using
            linear interpolation).
        """
        lambda1 = 1 / 3.0
        lambda2 = 1 / 3.0
        lambda3 = 1 / 3.0
        tri = self.raw_trigram_probability(trigram)
        bi = self.raw_bigram_probability(trigram[1:3])
        uni = self.raw_unigram_probability(trigram[2:3])
        return (lambda1 * tri + lambda2 * bi + lambda3 *
            uni)

    def sentence_logprob(self, sentence):
        """
        COMPLETE THIS METHOD (PART 5)
        Returns the log probability of an entire sequence
            .
        """
```

```python
        trigrams = get_ngrams(sentence, 3)
        probability = 0
        for trigram in trigrams:
            probability += math.log2(self.
                smoothed_trigram_probability(trigram))
        return probability

    def perplexity(self, corpus):
        """
        COMPLETE THIS METHOD (PART 6)
        Returns the log probability of an entire sequence
            .
        """
        l = 0
        M = 0
        for sentence in corpus:
            l += self.sentence_logprob(sentence)
            M += len(sentence)
        l /= M
        return 2**(-l)


def essay_scoring_experiment(training_file1,
    training_file2, testdir1, testdir2):
    model1 = TrigramModel(training_file1)
    model2 = TrigramModel(training_file2)

    total = 0
    correct = 0

    for f in os.listdir(testdir1):
        pp1 = model1.perplexity(corpus_reader(os.path.
            join(testdir1, f), model1.lexicon))
        pp2 = model2.perplexity(corpus_reader(os.path.
            join(testdir1, f), model2.lexicon))
        total += 1
        if pp1 < pp2:
            correct += 1
        # ..

    for f in os.listdir(testdir2):
        pp2 = model2.perplexity(corpus_reader(os.path.
            join(testdir2, f), model2.lexicon))
        pp1 = model1.perplexity(corpus_reader(os.path.
            join(testdir2, f), model1.lexicon))
        total += 1
```

```
        if pp1 > pp2:
            correct += 1
        # ..

    return float(correct / total)


if __name__ == "__main__":
    model = TrigramModel(sys.argv[1])

    # put test code here...
    # or run the script from the command line with
    # $ python -i trigram_model.py [corpus_file]
    # >>>
    #
    # you can then call methods on the model instance in
        the interactive
    # Python prompt.

    # Testing perplexity:
    # dev_corpus = corpus_reader(sys.argv[2], model.
        lexicon)
    # pp = model.perplexity(dev_corpus)
    # print(pp)

    # Essay scoring experiment:
    root_dir = 'hw1_data/ets_toefl_data/'
    acc = essay_scoring_experiment(root_dir+'train_high.
       txt', root_dir+'train_low.txt', root_dir+"
       test_high",
                                          root_dir+"test_low")
    print(acc)

    print(model.generate_sentence(20))
```