CIS519: Applied Machine Learning	**Spring 2018**
Homework 4	
Handed Out: April 3^{rd}, 2018	*Due: April* 14^{th}, 2018, *11:59 PM*

1 Document Classification [60 points]

In this problem, you will implement several text classification systems using the naive Bayes algorithm and semi-supervised Learning. In addition to the algorithmic aspects – you will implement a naive Bayes classifier and semi-supervised learning protocols on top of it – you will also get a taste of data preprocessing and feature extraction needed to do text classification.

Please note that we are leaving some of the implementation details to you. In particular, you can decide how to representation the data internally, so that your implementation of the algorithm is as efficient as possible. **Note, however, that you are required to implement the algorithm yourself, and not use existing implementations, unless we specify it explicitly.**

1.1 Dataset

For the experiments, you will use the *20 newsgroups text* dataset[1]. It consists of ∼18000 newsgroups posts on 20 topics. We have provided 2 splits of the data. In the first split, `20news-bydate`, we split the training and testing data by time, i.e. you will train a model on the data dated before a specified time and test that model on the data dated after that time. This split is realistic since in a real-world scenario, you will train your model on your current data and classify future incoming posts.

In the second split, `20news-random`, we have sampled the training/testing data uniformly at random.

1.2 Preprocessing [15 points]

In the first step, you need to preprocess all the documents and represent it in a form that can be used later by your classifier. We will use three ways to represent the data:

- Binary Bag of Words (B-BoW)
- Count Bag of Words (C-BoW)
- TF-IDF

[1] http://qwone.com/~jason/20Newsgroups/

We define these representations below. In each case, we define it as a matrix, where rows correspond to documents in the collection and columns correspond to words in the training data (details below).

However, since the vocabulary size is too large, it will not be feasible to store the whole matrix in memory. We suggest to use the fact that this matrix is really sparse, and store the document representation as a dictionary mapping from word index to the appropriate value, as we define below. For example, `{'doc1': {'word1': val1, 'word2': val2,...},...}`

We would like you to do the preprocessing yourself, following the directions below. Do not use existing tokenization tools.

1.2.1 Binary Bag of Words (B-BoW) Model [5 points]

1. Extract a case-insensitive (that is, "Extract" will be represented as "extract") vocabulary set, $\mathcal{V}$, from the document collection $\mathcal{D}$ in the training data. Come up with a tokenization scheme - you can use simple space separation or more advanced Regex patterns to do this. You may also want to lemmatize the tokens to extract the root word, and use a list of "stopwords" to ignore words like *the*, *a*, *an*, etc.

2. The set $\mathcal{V}$ of vocabulary extracted from the training data is now the set of features for your training. You will represent each document $d \in \mathcal{D}$ as a vector of all the tokens in $\mathcal{V}$ that appear in d. Specifically, you can think of representing the collection as a matrix $f[d, i]$, defined as follows:

$$f[d,i] = \begin{Bmatrix} 1, & \text{if } \mathcal{V}[i] \in d \\ 0, & \text{else} \end{Bmatrix} \forall i \in [0, |\mathcal{V}|), \forall d \in \mathcal{D}$$

1.2.2 Count Bag of Words (C-BoW) Model [5 points]

1. The first part of vocabulary extraction is the same as above.

2. Instead of using just the binary presence, you will represent each document $d \in \mathcal{D}$ as a vector of all the tokens in $\mathcal{V}$ that appear in d, along with their counts. Specifically, you can think of representing the collection as a matrix $f[d, i]$, defined as follows:

$$f[d,i] = tf(d,i), \forall i \in [0, |\mathcal{V}|), \forall d \in \mathcal{D},$$

where, $tf(d, i)$ is the *Term-Frequency*, that is, number of times the word $\mathcal{V}[i]$ occurs in document d.

1.2.3 TF-IDF Model [5 points]

1. The first part of vocabulary extraction is the same as above.

2. Given the Document collection $\mathcal{D}$, calculate the Inverse Document Frequency (IDF) for each word in the vocabulary $\mathcal{V}$. The IDF of the word w is defined as the log (use

base 10) of the multiplicative inverse of the fraction of documents in $\mathcal{D}$ that contain w. That is:

$$idf(w) = log\frac{|\mathcal{D}|}{|\{d \in \mathcal{D}; w \in d\}|}.$$

3. Similar to the representation above, you will represent each document $d \in \mathcal{D}$ as a vector of all the tokens in $\mathcal{V}$ that appear in d, along with their *tf idf* value. Specifically, you can think of representing the collection as a matrix $f[d, i]$, defined as follows:

$$f[d, i] = tf(d, i) * idf(i, \mathcal{D}), \forall i \in [0, |\mathcal{V}|), \forall d \in \mathcal{D},$$

where, $tf(.)$ is the *Term-Frequency*, and $idf(.)$ is the *Inverse Document-Frequency* as defined above.

1.3 Experiment 1 [10 points]

In this experiment, you will implement a simple (multiclass) Naive Bayes Classifier That is, you will use the training document to learn a model and then compute the most likely label among the 20 labels for a new document, using the Naive Bayes assumption. You will do it for all three document representations defined earlier.

Note that, using Bayes rule, your prediction should be:

$$\hat{y}_d = \underset{y}{\text{argmax}} P(d|y) * P(y),$$

where y ranges over the 20 candidate labels.

Since we are using the Naive Bayes model, that is, we assume the independence of the features (words in a document) given the label (document type), we get:

$$P(d|y) = \Pi_{i \in d} P(i|y)$$

where i is a word in document d and y is the label of document d.

The question is now how to estimate the coordinate-wise conditional probabilities $P(i|y)$. We have suggested to use three different representations of the document, but in all cases, estimate this conditional probability as:

$$P(i|y) = \frac{\sum_{\text{docs } d' \text{ of class } y} f[d', i]}{\sum_{j \in [0,|\mathcal{V}|)} \sum_{\text{docs } d' \text{ of class } y} f[d', j]}$$

Notice that in class we used the same estimation for the **B-BOW** model, and here we generalize it to the other two representations.

To do k**-laplace smoothing** (choose a suitable value of k), modify the above formula as follows: (You can think of 1-laplace smoothing as adding another document of class y which contains all the words in the vocabulary.)

$$P(i|y) = \frac{\sum_{\text{docs } d' \text{ of class } y} f[d', i] + k}{(\sum_{j \in [0,|\mathcal{V}|)} \sum_{\text{docs } d' \text{ of class } y} f[d', j]) + |\mathcal{V}|k}.$$

You will also need to estimate the prior probability of each label, as:

$$P(y) = \frac{\#\text{ of documents in } \mathcal{D} \text{ with label } y}{|\mathcal{D}|}$$

In this experiment you will run the Naive Bayes algorithm for both datasets and all three the representations and provide an analysis of your results. Use *accuracy* to measure the performance.

Important Implementation Detail: Since we are multiplying probabilities, numerical underflow may occur while computing the resultant values. To avoid this, you should do your computation in **log space**.

1.4 Experiment 2 [15 points]

In this experiment, you will use Semi-Supervised Learning to do document classification. The underlying algorithm you will use is Naive Bayes with the **B-BOW** representation, as defined in Experiment 1.

Follow the steps in Algorithm 1 below. You will split the training data into 2 parts, the Supervised (S) part, and the Unsupervised (U) part. You will use S part as labeled and the U part as the unlabeled data. According to the Algorithm you will train on the labeled data S, predict on U, and then augment S with some of the (now labeled) examples from U.

You will use two separate criteria to determine how to choose the examples in U that will augment S in each round of the algorithm. You will also initialize the algorithm with three different sizes of S – 5%, 10% and 50% of the training data.

Algorithm Semi-Supervised Classifier

procedure SEMI-SUPERVISED CLASSIFIER
  $S_X, S_Y \leftarrow$ p% of Training data ($p = 5\%, 10\%, 50\%$ of the training data)
  $U_X \leftarrow$ X's of remaining 100-p% of Training data
loop:
  $model \leftarrow train\_naive\_bayes\_classifier(S_X, S_Y)$
  $U_Y, P_Y \leftarrow predict(model, U_X)$
  $S_X^{new}, S_Y^{new} \leftarrow filter(U_X, U_Y, P_Y)$ (Choose a subset of the labeled U to augment S)
  $S_X \leftarrow S_X \cup S_X^{new}$
  $S_Y \leftarrow S_Y \cup S_Y^{new}$
  $U_X \leftarrow U_X \setminus S_X^{new}$
  if $|U_X| = 0$ **then**
    return model
  end if
  goto *loop*
end procedure

You will use two options for `filter(.)`. In both cases, the decision will be made according to the probability Naive Bayes assigns to the winning label.

1. **Top-k:** Filter the top-k instances and augment them to the labeled data.

2. **Threshold:** Set a threshold. Augment those instances to the labeled data with confidence higher than this threshold. You have to be careful here to make sure that the program terminates. If the confidence is never higher than the threshold, then the procedure will take forever to terminate. You can choose to terminate the program if there are 5 consecutive iterations where no confidence exceeded the threshold value.

Run the algorithm with both filtering techniques for both the datasets, and all three initializations, but only the representation **B-BOW**, and provide an analysis of your results. Use *accuracy* to measure the performance.

1.5 (Optional: Extra Credit) Experiment 3 [5 points]

For experiment 2, initialize as suggested but, instead of the filtering, run EM. That is, for each data point in U, label it fractionally – label data point d with label l, that has weight $p(l|d)$. Then, add all the (weighted) examples to S. Now use Naive Bayes to learn again on the augmented data set (but now each data point has a weight! That is, when you compute $P(f[d,i]|y)$, rather than simply counting all the documents that have label y, now *all* the documents have "fractions" of the label y), and use the model to relabel it (again, fractionally, as defined above.) Iterate, and determine a stopping criteria.

1.6 What to Report: [20 points]

You have to submit the following items on **Canvas** only:

1. `code.py` [3 points] - well-commented code written in Python3. If you have multiple files, please document them appropriately.

2. `README` [2 points] - Your name, penn key and clear instructions to run your code.

3. `writeup.pdf` [15 points] - For each of the above experiments, train it on both the datasets provided and report the accuracies achieved on the test data. Which test dataset achieved higher performance? Which representation model helped achieve a higher score? Which tokenization scheme(s) did you try for vocabulary extraction. How did they affect the performance? How did the stopwords removal affect your classifier performance? Are there any labels for which the classifier often make mistakes? Provide a detailed analysis of your results. For experiment 2, report the accuracy of your classifier on the test data for each iteration.

2 Theory [40 points]

2.1 Multivariate Poisson naïve Bayes [30 points]

In this question, we consider the problem of classifying piazza posts (Y) into two categories: student posts (A), and instructor posts (B). For every post, we have two attributes: number

of words (X_1), and number of mathematical symbols (X_2). We assume that each attribute ($X_i, i = 1, 2$) is related to a post category (A/B) via a Poisson distribution[2] with a particular mean (λ_i^A/λ_i^B). That is

$$Pr[X_i = x|Y = A] = \frac{e^{-\lambda_i^A}(\lambda_i^A)^x}{x!} \quad \text{and} \quad Pr[X_i = x|Y = B] = \frac{e^{-\lambda_i^B}(\lambda_i^B)^x}{x!} \text{ for } i = 1, 2$$

X_1	X_2	Y
0	3	A
4	8	A
2	4	A
6	2	B
3	5	B
2	1	B
5	4	B

Table 1: Dataset for Poisson naïve Bayes

Assume that the given data in Table 1 is generated by a Poisson naïve Bayes model. You will use this data to develop a naïve Bayes predictor over the Poisson distribution.

$\Pr(Y=A) =$	$\Pr(Y=B) =$
$\lambda_1^A =$	$\lambda_1^B =$
$\lambda_2^A =$	$\lambda_2^B =$

Table 2: Parameters for Poisson naïve Bayes

1. **[10 points]** Compute the prior probabilities and parameter values, i.e., fill out Table 2. Please show all the intermediate steps and results. [Hint: Use MLE to compute the λ's].

2. **[10 points]** Based on the parameter values from Table 2, compute

$$\frac{\Pr(X_1=2, X_2=3 \mid Y=A)}{\Pr(X_1=2, X_2=3 \mid Y=B)}$$

3. **[5 points]** Derive an algebraic expression for the Poisson naïve Bayes predictor for Y in terms of the parameters estimated from the data.

4. **[5 points]** Use the parameters estimated from the data given in Table 1 to create a Poisson naïve Bayes classifier. What will the classifier predict as the value of Y, given the data point: $X_1=2, X_2=3$?

[2] http://en.wikipedia.org/wiki/Poisson_distribution

2.2 Dice Roll [10 points]

Consider a scheme to generate a series of numbers as follows: For each element in the series, first a dice is rolled. If it comes up as one of the numbers from 1 to 5, it is shown to the user. On the other hand, if the dice roll comes up as 6, then the dice is rolled for the second time, and the outcome of this roll is shown to the user.

Assume that the probability of a dice roll coming up as 6 is p. Also assume if a dice roll doesn't come up as 6, then the remaining numbers are equally likely. Suppose you see a sequence `3463661622` generated based on the scheme given above. What is the most likely value of p for this given sequence?