已通过 Google 文档发布

# Homework 3 - Text Classification

You may complete this assignment **either as an individual or in pairs**.

In this assignment, you will implement a text classification system for sentiment analysis. I will provide training data in the form of positive and negative reviews. Use this training data to develop a system that takes reviews as test data and categorizes them as positive or negative.

Turn in:

- `textclassify_model.py`
  - We will be running your code with the command:
    - `python textclassify_model.py training-file.txt dev-file.txt`
  - We will be running unit tests on textclassify_model.py.
    - Do not change the function signatures of any provided functions.
  - You should not have any code that is outside of classes, functions or an `if __name__ == "__main__":` conditional guard.
    - (the same as with the previous homework)
  - Submit your work via Gradescope as a python script. (note! Do not submit an ipython notebook/jupyter notebook file)
- analysis.pdf

Learning objectives:

- Analyze a sentiment analysis tool in terms of value sensitive design and potential biases that may be

introduced from this kind of text classification
- Fully implement a probabilistic classifier (Naive Bayes (and possibly also Logistic Regression))
- Deal with live data, including addressing issues of text normalization and feature engineering
- Refine and update your classifier for better performance

Allowed python modules
- you may not use nltk for this assignment, except for section 3, where you may use word lists from nltk
- you may use numpy, matplotlib, all built-in libraries like math, collections, sys, etc
- if you have a question about a particular library, please ask or post on piazza!
    - (if you think that using a certain library would make the problem very easy, then you probably aren't allowed to use it)

Provided files are located on Canvas.

## 1: Analysis, part 1 (10 points)

You have received an intriguing message! Bacefook International, a global social media network centered around letting anyone anywhere share tiny witticisms (a.k.a "bweets") of 140 characters or less, wants *you* to build them the hottest new tool on the market—an automatic system for tracking the opinions of their users! They send you an email offering $$$$ if you'll build them the *best* tool out there *fast*. Bacefook needs something and needs it yesterday. If only you could help them they would immediately wire you all the $$$$! With more than 100,000 users, this is sure to be your way into the burgeoning world of social media!

Since you've got just a few days, you look around for a dataset that you already have access to in order to train up a text classification tool for sentiment analysis. At least this will give them an idea of what their users are thinking, right? At your fingertips—a small corpus of short hotel reviews. This will be an excellent starting point!

You are about to sit down and start cranking out the code when you pause and remember: who even is Bacefook? What do they want to do and why? Who are their users?

Before you start coding a prototype, write a one-page document conducting a value sensitive design analysis for this project. For this analysis, make sure to identify 1) the stakeholders involved 2) their interests and 3) conflicts that might occur. Refer to your notes from Prof. Magnani's guest lecture here.

Here are some examples of the data that you'll be working with:

### Training data

The training data for this task consists of a collection of short hotel reviews. The data is formatted as one review per line. Each line starts with a unique identifier for the review (as in ID-2001) followed by a tab, the text of the review, and finally one more tab and a label (1 for positive, 0 for negative) for the review. The reviews are not tokenized or sentence segmented in any way (the words are space separated). The positive reviews and negative reviews appear in the same file.

Here are two examples:
ID-1274 Absolutely a great hotel for families! The rooms are spacious, restaurants are very kid-friendly, and the pool area is gorgeous. My children felt like they were at a waterpark, not just another hotel. We will definitely return for another stay here! 1

ID-1021 This hotel is located at a busy traffic circle near the interstate. It isn't a walking distance to downtown. The hotel restaurant was average. My wife complained about the quality of the sheets; they were pretty rough. 0

### Development data

The development data consists of short hotel reviews. It is formatted in the same way as the training data, and, in fact, was created by removing 10% of the data from the training set. This is the data that you should use to improve your model. In general, improvement on the development set will correspond to improvement on the test set, but be careful about overfitting!

### Test data

The test set will consist of a single file with mixed positive and negative reviews. You will not have access to the testing data as an incentive to not simply engineer your model to overfit it. Our autograder will be running your models against the test set so that you can see where you land relative to each other and the baseline model.

## 2: Writing your base prototype (45 points)

### 2.1: Utility functions (5 points)

You must implement the helper functions that are outlined in the starter code. Feel free to use any code that you've written during the lecture to help yourself out here!

You may have other methods and functions as you'd like. You should not have any code not within a function/class or guarded by "`if __name__ == "__main__":`", just as in the previous homeworks.

## 2.2: TextClassify baseline class (40 points)

Start by implementing the Naïve Bayes classifier that we went over in class (Chapter 4). Recall that this means building a unigram language model for each class and then assessing the probability of each test case with respect to each of the classes when we are using a "plain" bag of words as features. Then, the highest probability calculated class will be the assigned label. Correctly implementing Naïve Bayes and the general functionality of this program is sufficient to get credit for this component of the assignment. To be specific, Naïve Bayes with add-1 smoothing and a bag of words for features (with no text normalization) will be the baseline that we will use to assess performance.

A few reminders/notes here:

- Ignore unseen words in examples at score time
- Your vocabulary size should include all unique tokens for all classes
- Again, do not normalize the text in any way beyond splitting it with str.split()
- Use the numpy library's np.log function to take logs (you want a natural log)
- For score, you'll use e ^ (log(p|c) + sum(log(p(w_i|c))) for the probabilities of each class

> Your class **must** provide methods outlined in the starter code.

## 2.3 Overall Functionality, Style, Comments (10 points)

When we run your file with the command:
python textclassify_model.py training-file.txt dev-file.txt
Your script should:

1. train all of your models on the given training file
2. classify each example in the given testing file for each model
3. Print the precision, recall, and f1 score of the given test data for each of your models

6 points will be allocated for overall functionality

4 points will be allocated for style and commenting. Make sure to comment all your functions/methods, use good variable names, and avoid writing overly redundant code.

## 3: Improve your model! (TextClassifyImproved class) (20 points)

Next, attempt to improve your performance on this task over your Naïve Bayes baseline from the previous task. To do this, you must experiment with at least 2 different modifications from the following list.

If you are in CS 6120, you **must** implement a logistic regression classifier for your TextClassifyImproved class.

If you are in CS 4120, you **may** either implement a logistic regression classifier or make any two of the following three modifications to your Naïve Bayes classifier.

- Normalize your text in at least two ways
- Pre-process your text to add other information (e.g. information about negation)
- Incorporate a word list or other outside resource into your features

**All**: for each modification/feature (if implementing a logistic regression classifier) that you experiment with, you'll be documenting its effects in your analysis, so be sure to note its effects on the performance of the model as you do your work!

- If you find that any modification reduces performance, leave your code present, but commented out or

> “turned off” with a flag. Make sure to leave your description and report of results in your analysis document!

15 points are allocated to successful implementation and description of these modifications.

5 points are allocated to whether or not your modifications actually improved your performance on the hidden test set according to f1 score. Our baseline’s f1 score on the hidden test set is 0.75.

You may incorporate resources such as word lists and training data as you wish, but you must cite your sources in your comments and they should be accessed via relative paths in your code. (Make sure to submit them along with your code so that the autograder has access to them as well).

You may use any tokenizers and stemmers from nltk that you would like to for this portion of the assignment. If you have a question about using a specific tool that does not explicitly say that it is a tokenizer or a stemmer, ask on Piazza and we'll let you know, as well as adding it to the following list of accepted other tools.

Use the provided development set to analyze your improvements. See the extra credit options if you wish to use k-fold cross validation instead.

## 4: Analysis, part 2 (25 points)

Now that you have a working prototype, you're ready to send your proof of concept to Bacefook international! Finish the report that you started in part 1 by adding the following information. Of course, as you've been working on this project, you've been fully aware that most of Bacefook International's users are not posting hotel reviews, so this is the time to set their expectations appropriately and highlight any concerns about their project that you might have.

Add the following information:

1) **Characterize the dataset that you used**
    a) Describe the dataset that you used to train your model (how big? label distribution? what is the model being trained on?)
    b) Describe the likely effects of training a Naïve Bayes (and Logistic Regression) model on this data and then using it on Bacefook International's data
2) **What are your results?**
    a) Document the results of your TextClassifyImproved class on your dev set along (include a table with a breakdown of the effects of the various modifications/features that you tested out)
3) **Propose an alternate training set that, given more time, you could use to train your model**
    a) Describe the data that you'd like to use
    b) Describe the likely effects in terms of performance that using this data would have on your model.
4) **Raise any concerns of bias that may be learned by the model from using the data that you proposed in #2**

## Hints/Clarifications

- Start by implementing a Naïve Bayes classifier with a bag of words as features. This is a multinomial (not binarized) classifier.
- Use the provided test cases and starter code to help ensure correct implementation of a Naïve Bayes classifier as well as correct implementation of your class's interface.
- If you end up with all classes having equal probability, you should break

ties by returning the first label with that probability ("0" in this case).

- Interpreting the baseline tests on the autograder:
    - test_sentimentbaselinef1_dev (test_baseline.TestBaselineSystem) - tests the performance of your baseline classifier trained on the provided train set and tested on the provided dev set. There are also corresponding precision and recall tests.
        - If you are failing these, this means that you are labeling an example differently than we did. We recommend starting by looking at how you are breaking ties (see above).
    - test_sentimentbaselinef1 (test_baseline.TestBaselineSystem) - tests the performance of your baseline classifier trained on the provided train test and tested on the hidden test set. There are also corresponding precision and recall tests.
- Our implementation of the required components, including blank lines and comments but not including the TextClassifyImproved class, was approximately 200 lines of code.

## Extra Credit 1 (all students): k-fold validation (up to +2 points possible)

Read section 4.8 of the textbook.

Implement the function:

```
def k_fold(all_examples, k):
        # all_examples is a list of
tuples of strings formatted [(id,
example_text, label), (id,
example_text, label)....]
        # containing all examples
from the train and dev sets
```

```
        # return a list of lists
containing k sublists where each
sublist is one "fold" in the given
data
```

Now, combine your training data with the given development data (from the command line parameters) and train & evaluate your model on 10 folds. Report the precision, recall, and f1-measure for each fold when your program is run.

## Extra Credit 2 (all students) - Dealing with multinomial data (up to +10 points possible)

Implement multi-class precision, recall, and f1 functions, and ensure that your TextClassify and TextClassifyImproved classes function correctly. Turn in a multi-class dataset that you used to test your model. When we run your code, after training and testing on the command line argument-provided files, you should train and evaluate your model on your multi-class data. These paths should be hard-coded, but they'll need to be relative so that when you turn them in, the autograder can appropriately access them.

Begin any output from your multiclass model with a clear demarcation, such as:
`"********** MULTI-CLASS MODEL OUTPUT **********"`

If you choose to implement this improvement, add the the following functions:

```
def
precision_multiclass(gold_labels,
classified_labels):
      # gold labels is a list of
strings of the true labels
      # classified labels is a list
of strings of the labels assigned
by the classifier
        return the precision as a
float
```

```
def recall_multi(gold_labels,
classified_labels):
      # gold labels is a list of
strings of the true labels
      # classified labels is a list
of strings of the labels assigned
by the classifier
        return the recall as a
float

def f1_multi(gold_labels,
classified_labels):
      # gold labels is a list of
strings of the true labels
      # classified labels is a list
of strings of the labels assigned
by the classifier
        return the f1 score as a
float
```