CS 440: Project 2
Document Classification
Due April 28

Scott Wallace

Abstract

This handout describes Project 2: A Document Classification Task. The project will be executed in a team environment. Each student will implement at least solution strategies for the problem and then evaluate their strategies with a set of experiments. A written report will document both the implementation and the experimental evaluation.

Overview

This project is to be carried out in roughly 4 person teams. As with the previous project, the goals are threefold: first, to further develop your AI toolkit; second, to help develop your technical writing skills; and third, to give you some research savvy (this will come in handy regardless of whether you decide to head off to graduate school).

For this project, you will implement a document classifier using: (1) a set of 3 pre-specified strategies (one of these is trivial); and (2) strategies defined by your group. Each group member should implement and report on approximately two strategies. This means, each group member must have some coding and some writing responsibilities.

The project has three basic parts: solution design/implementation, evaluation and write-up.

Document Classification

The document classification task can be framed as follows: given a set of documents $d_0, d_1, \ldots, d_n$ and a set of labels $c_0, c_1, \ldots, c_n$ assigned from some pre-specified set C, create a classifier that can take a new document, d, and predict the appropriate class ($c \in C$) to which this document belongs.

In this project, you are given a set of documents from three classes: Deeds of Trust (DT), Liens (L), and Deeds of Reconveyance (DR). Each document is stored as a text file that is the raw OCR output after processing a scanned image of the original document. The text files have been placed in folders based on their class membership.

Solution Design and Implementation

A solution to this project consists of a program that can be written in the language of your choice. A solution should be crafted with three conceptual parts. These are described below.

Training Set Processing

The first part of your project should deal with processing the training set. There are two aspects that must be dealt with here:

1. Partitioning the documents into training and testing sets. The data comes pre-partitioned into a training set and test set. It's perfectly acceptable to use this partitioning for all of your experiments. For extra credit (and to help improve your overall results), you may want to consider using 10-fold cross validation as described in class (and as explained in the book). Cross validation is the defacto standard for measuring classifier performance on smallish datasets like the one you will be given.
2. Processing the text from each document to create a set of features. For the initial strategies, you should generate a standardized set of features. However, you will probably want to change these features to improve your classifier's performance once a baseline has been identified. To be clear, processing may take place in two steps: during *training* you may select features based on examining the training set; but during *testing* you can only apply/measure the features which have already been determined.

Initial Strategies

Each group should implement a set of standardized strategies to identify baseline document classification performance. It is expected that each team member will implement approximately one of these strategies (although obviously some groups will be larger or smaller).

For each of the baseline strategies, you should perform initial document processing as follows:

1. Read in the text documents as a string
2. Replace non-alphabetic characters with spaces.
3. Replace uppercase ascii characters with lowercase ascii characters
4. Collapse multiple adjacent spaces into a single space.

These steps will be referred to as document preprocessing.

Intelli-Grep The Intelli-Grep strategy is extremely simple and is a clear strawman. Indeed, this approach is so simple there is no need for explicit training. Instead you can write the code and operate immediately on all the data. Your code should: 1) perform initial preprocessing on the training text as described above; 2) for each document d search for the number of times each of the following strings occurs: "deed of trust", "deed of reconveyance" and "lien"; 3) classify the document based on the most frequently occurring search string; break ties randomly.

Naive Bayes For your naive bayes implementation you must begin by taking preprocessed text and extracting a set of "features" that you will use to compute conditional probabilities.

Begin with preprocessed *training* set and for each class c determine the relative frequency of each word (that is, count the occurrences of w from all documents in c and divide by the total number of words in c). Call this value $P(w|c)$. Take the top 20 terms (based on $P(w|c)$, breaking ties with alphabetic ordering) from each class and use these as your feature set. Note that there may be duplicate terms across classes, so you may end up with less than 60 features.

Now that you've selected features, process each document d in class c and determine whether each feature occurs in d. That is, create a boolean "Bag of Words" model for each document. Finally, estimate the likelihood of each feature w in each class c by computing $P(w|c)$ as the number of documents in c containing w divided by the number of documents in c. If it happens to the be case that w does not appear in any documents in c, set $P(w|c)$ to 1 divided by the total number of documents in c. Essentially, this acts as though you saw the word once, eliminates problematic 0 probabilities. As you're processing the data, be sure to also compute $P(c)$ by determining the fraction of training documents that belong to c.

Next, compute feature vectors for each of the documents in the *test* set by checking if each word in the feature vector occurs. That is, create a boolean "Bag of Words" model for each document. Use naive bayes (multi-variate bernoulli) to find the most appropriate class for each document in the test set.

Perceptrons Build the feature set just as for Naive Bayes. Then, for all the documents in the *training* set build a "Bag of Words" model with relative frequencies. That is for each document d compute $P(w|d)$ for each word w in the feature set. Use these values, along with a bias, to train your perceptron. Use a hard threshold function with α initially set to .1. Decay α by 1 percent after each pass through the training data and run for at least 100 iterations.

Note that the perceptron discussed in class performs only a boolean classification. It has one output only. Since there are three classes, you should create three distinct perceptrons – one for each class. You can view these as voting machines that will each cast a 0,1 vote for their respective classes. Note that you train each perceptron on *all* the training data, not just the data from the class you want it to recognize.

The trained perceptrons should then be used to classify the *test* data by: 1) computing $P(w|d)$ for each document d in the *test* set (note the words have already been defined during training); 2) running each perceptron and allowing it to cast a vote; 3) breaking ties randomly if necessary.

Evaluation

Once the initial strategies have been implemented, you should evaluate their performance using the partitioned *training* and *test* sets. you can also use 10-fold cross validation, or some other scheme, but you *must* report results from the partitions provided. After the experiments have run, you should look for ways to improve each strategy. If you can identify sources of weakness in the initial approaches, this will improve your writeup and your subsequent results.

Improved Strategies

Given the results form the initial experiments, now it is time to implement an improvement to your initial strategies. You should implement enough strategies in this stage so that each member from the group has been involved with two strategies total.[1]

An improved strategy can either be a new/better way of pre processing the data, creating features, or a new classifier approach altogether. For full credit, however, you must be able to convey a reasonable motivation for why the strategy holds promise. Your motivation may come from some underlying theory you have about classification in general (classifier x is better than classifier y), or about something you've observed in the data. It is acceptable if your "improved" strategy does not, in fact, improve your results provided that your reasoning is well motivated. However, since the end goal is to classify the data accurately, you should strive to substantially improve from the baseline.

Some of the potential mechanisms we've talked about in class are:

- Allowing only english language words to remain in the document text after preprocessing
- Allowing capitalization to remain after preprocessing
- Removing words that exist in all/most of the documents
- Changing from boolean valued Bag of Words model to a model that incorporates the number of occurrences of each feature/word.
- Modifying the manner in which the naive bayes calculation is performed.
- Changing the classifier altogether.

You can earn full credit simply by reporting classifier accuracy on the provided partitioning.

Write Up

To complete the project you must submit a write up that documents your approach along with the executable code.

The writeup should be 4–6 or more pages in AAAI two-column format. This is the standard format used in publications and will equal roughly 10 pages in microsoft words

[1]The person who got away with implementing intelli-grep should probably buy the donuts.

default format. It is also the format that this document is written in.

The report should contain the following sections:

Abstract A brief summarization the paper in a single paragraph.

Introduction This section should describe the problem and identify the roles of each group member.

Initial Strategies This section should describe each initial approach in your own words and report on the accuracy of each classifier (the fraction of correct classifications) on the partitioned data.

Improved Strategies This should not be a single section, but rather one section per improved strategy. In each section, you should layout the motivation for your improvement. Next, describe the implementation from feature processing through classification in enough detail for me to recreate the experiments. Finally, finish the section with a discussion of results and indicate whether your work on this improvement led to any downstream effects (i.e., to other improved strategies).

Conclusion Based on your results, argue which of your classifiers you expect to perform the best on the hidden data set and what elements of that classifier you view as particularly strong/effective.

Turning it in

You need to turn in three things for this assignment:

1. The write up for your implementation and experiments as described above.
2. A executable file (or python source code) that can be run using on the lab machines (assuming I have the Anaconda python stack and sci-kit learn installed). I should be able to run the executable from the command line by specify four directories: one for each class of training data, and one for the test data. The executable should then build classifiers for each initial strategy and improved strategy and produce an output file that consists of one entry per strategy/test document pair in this format:
   strategy name, *test document name*, *class*
   so given classes: *DT*, *DR*, and *L*, test documents *1.txt*, and *2.txt* and strategies *A* and *B*, an output file might look like:
   ```
   A,1.txt,DT
   A,2.txt,DR
   B,1.txt,DT
   B,2.txt,L
   ```
3. A directory labeled `src` containing all of your source code along with compilation instructions if applicable.

Put all of these items into a folder with the last names of your group members (no spaces please), zip it up and turn it in via BlackBoard.