ASSIGNMENT 1:
QUESTION CLASSIFICATION VIA NAIVE BAYES

COMPUTER SCIENCE 187

Sections 1-2 of this assignment are due Monday, February 11, 2013 by 11:59 pm. Sections 3-5 of this assignment are due Wednesday, February 13, 2013 by 11:59 pm.

In this assignment, you will tackle the task of question classification. Question classification is useful for downstream tasks in question-answering systems, since the initial classification can be utilized to constrain subsequent tasks. The end goal will be to classify the questions in a dataset provided by Li and Roth [2002] with the aim of assigning each test question to one of the following six coarse classes: Abbreviation, Entity, Description, Human, Location, and Numeric Value.

The data files necessary to complete this assignment are available on the course web site as `hw1.zip`. *The scripts that you write below should assume that the data files and the scripts are all in the same directory.*

We recommend that you read through this handout to the end prior to starting. If you plan ahead and make your code sufficiently general, you may well find yourself writing only relatively small amounts of new code after Section 1.

1. State of the Union Dataset: Bernoulli Naive Bayes

As a preliminary task, you will develop and apply your models to an example dataset of words extracted from some recent Presidential State of the Union Addresses.

1.1. **Data.** The file `state_of_union_train_positional.txt` contains the training set, and the file `state_of_union_test_positional.txt` contains the test set. The training set consists of the

actual word tokens[1] extracted from (in the following order) George Walker Bush's State of the Union addresses from 2005, 2006, and 2007, and Barack Obama's State of the Union addresses from 2009 and 2010. The test set consists of actual words from (in the following order) George Walker Bush's State of the Union address from 2008 and Barack Obama's State of the Union address from 2012.

Each line of the text files (for both training and test) take the following form:

```
CLASS_LABEL:YEAR Words from the speech
```

The corpus consists of the tokens of only the following word types: "freedom", "honor", "amnesty", "mortgage", "offensive", "guided", and "terrorist". All other words in the speeches were dropped.

1.2. **Classification via Naive Bayes.** Recall that in using Naive Bayes, we aim to predict the class $c \in C$ of a document $w = w_1 \dots w_{|w|}$ (e.g., a speech or a question) by choosing the class with the maximum a posteriori (MAP) probability. In this case, there are two classes: Bush and Obama. We calculate the MAP class c_{MAP} as follows:

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} \widehat{\Pr}(w \mid c) \cdot \widehat{\Pr}(c),$$

where the class conditional probability of a document $\widehat{\Pr}(w \mid c)$ is calculated differently in the multinomial Naive Bayes and Bernoulli Naive Bayes models. As such, even though in both cases we will be using word types as features, the abstract document representation used in the two models differs in subtle but important ways. We will start by considering the Bernoulli Naive Bayes model, which ignores multiple occurrences of a word type in a document.

1.3. **Document Representation and MLE Estimates with Bernoulli Naive Bayes.** In the Bernoulli model, each document is represented as a set of binary features B_i for each word type t_i in the dataset indicating the presence or absence of the word type in the document.

[1]Singular and plural words were treated as the same word type in constructing this example.

With the State of the Union dataset, there will be seven such features. For example, ordering the features as in the order of word types presented above, we could represent the first speech in the training set as

$$\langle \mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true} \rangle$$

and the last speech in the training set as

$$\langle \mathsf{true}, \mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true} \rangle \quad .$$

As you consider the design of your code, keep in mind that you will also need to retain an association between a speech and its class label.

1.4. **MLE Estimates for the Class Priors.** The unsmoothed maximum likelihood estimates for the class priors (for both multinomial Naive Bayes and Bernoulli Naive Bayes) correspond to the relative number of occurrences of the particular class in the dataset. In other words,

$$\widehat{\Pr}(c_i) = \frac{\sharp(c_i)}{N} \quad ,$$

where the numerator corresponds to the number of speeches in class c_i and the denominator N corresponds to the total number of speeches in the training set.

Problem 1. *Looking at the training data, calculate by hand the maximum likelihood estimate for* $\widehat{\Pr}(\mathsf{Bush})$ *without smoothing. Also, calculate the maximum likelihood estimate for* $\widehat{\Pr}(\mathsf{Obama})$ *without smoothing.*

Problem 2. *In Python, write code that reads in the training data and calculates these class priors. Ensure that the results match those that you calculated by hand in Problem 1.*

1.5. **MLE Estimates for the Class Conditional Probability of a Word Type.** The MLE estimate for the class conditional probability of a word type in the Bernoulli model can be calculated as

$$\widehat{\Pr}(B_i = \mathsf{true} \mid c) = \frac{\sharp(B_i = \mathsf{true}, c)}{\sharp(c)},$$

where the numerator is the number of documents in class c containing term t_i at least once.

Problem 3. *With a small number of classes and a reasonably sized training set, the unsmoothed MLE estimates for the class priors may well be adequate. This is less likely to be the case in the above calculation of the class conditional probabilities of word types, and these unsmoothed estimates for the conditional probabilities will often be inadequate for calculating the maximum a posteriori (MAP) probability. Why?*

For the reasons you noted in your answer to Problem 3, we will use add-one (or Laplace) smoothing, as follows:

$$\widehat{\Pr}(B_i = \mathsf{true} \,|\, c) = \frac{\sharp(B_i = \mathsf{true}, c) + 1}{\sharp(c) + 2} \quad .$$

For example, in the training set,

$$\widehat{\Pr}(\mathsf{freedom} \,|\, \mathsf{Obama}) = \frac{2+1}{2+2} = 0.75 \quad ,$$

and

$$\widehat{\Pr}(\mathsf{freedom} \,|\, \mathsf{Bush}) = \frac{3+1}{3+2} = 0.8 \quad .$$

Problem 4. *Building on your Python code started above, write code that calculates these class conditional probabilities using add-one smoothing. In your writeup, include a table that contains all 14 smoothed conditional probabilities for the State of the Union training dataset.*

1.6. **Calculating the MAP class.** In the Bernoulli model,

$$\widehat{\Pr}(w \,|\, c) = \widehat{\Pr}(\langle b_1, ..., b_{|w|} \rangle \,|\, c) \quad .$$

After making the conditional independence assumption, we can determine the MAP class using the following:

$$c_{MAP} = \operatorname*{argmax}_{c \in C} \widehat{\Pr}(c) \cdot \widehat{\Pr}(w \,|\, c) = \operatorname*{argmax}_{c \in C} \widehat{\Pr}(c) \prod_{i=1}^{|w|} \widehat{\Pr}(B_i = b_i \,|\, c) \quad .$$

To illustrate this further, in the State of the Union test set,

$$P(\mathsf{Bush} \,|\, 2008 \text{ speech}) \propto \frac{3}{5} \cdot \prod_{i=i}^{7} \widehat{\Pr}(B_i = b_i \,|\, \mathsf{Bush}) \approx 0.00147 \quad .$$

Problem 5. *In practice, instead of the above expression, we use the following:*

$$c_{MAP} = \operatorname*{argmax}_{c \in C} \left(\log \widehat{\Pr}(c) + \sum_{i=1}^{|w|} \log \widehat{\Pr}(B_i = b_i \,|\, c) \right) \quad .$$

Briefly explain why this latter approach is wise to use in practice.

Problem 6. *Building on your code above, write code that identifies the "best" class for each of the test speeches using the MAP estimate.*

1.7. **Bernoulli Naive Bayes: Putting It All Together.** You should now have code that reads in a training and test set (in the format specified above) and calculates the predicted class label for each speech in the test set using the Bernoulli Naive Bayes model. Save your script using the following naming convention: *lastname*_`hw1_section1.py`. When we run your script with

```
% python lastname_hw1_section1.py
```

we should see the following output:

```
Accuracy of prediction: xxx
Accuracy by class c:
     bush  yyy
     obama zzz
```

where the *xxx*, *yyy*, and *zzz* are replaced by the computed values.

By"accuracy for class *c*" we mean the fraction of documents in the test set in class *c* that are correctly labeled by the method as being in class *c*.

Problem 7. *Run your script and generate the output in the format above.*

1.8. **Section 1 Deliverables.** Upload the following to the dropbox on the course web site:

(1) Your Python code: *lastname*_`hw1_section1.py`.
(2) A PDF file containing your numbered answers to all of the numbered problems in Section 1. Use the following naming convention: *lastname*_`hw1_writeup_section1.pdf`.

2. State of the Union Dataset: Multinomial Naive Bayes

With text classification, better results can often be obtained using multinomial Naive Bayes compared to Bernoulli Naive Bayes. With the multinomial model, the MAP class for a particular test document w is determined using the following:

$$c_{MAP} = \operatorname*{argmax}_{c \in C} \widehat{\Pr}(c) \cdot \widehat{\Pr}(w \mid c) = \operatorname*{argmax}_{c \in C} \widehat{\Pr}(c) \prod_{i=1}^{|w|} \widehat{\Pr}(V = t_{w_i} \mid c),$$

where V is a random variable over word types, and $\widehat{\Pr}(V = t \mid C = c)$ is an estimate of the probability distribution measuring the probability that a given word token in a document of class c is of type t.

2.1. **MLE Estimates for $\widehat{\Pr}(t_i \mid c)$.** The MLE estimate for the class conditional probability $\widehat{\Pr}(V \mid C)$ of a word type in the multinomial model can be calculated as follows using add-one smoothing:

$$\widehat{\Pr}(V = t \mid C = c) = \frac{\sharp(V = t, C = c) + 1}{\left[\sum_{j=1}^{|V|} \sharp(V = t_j, C = c)\right] + |V|}$$

where $|V|$ is the number of word types in the corpus. In other words, the numerator is the number of occurrences of word type t_i in class c (plus 1), and the denominator is the total number of occurrences of word types in class c plus the total number of word types in the corpus. For example, in the training set,

$$\widehat{\Pr}(\mathsf{freedom} \mid \mathsf{Obama}) = \frac{2+1}{13+7} = \frac{3}{20},$$

and

$$\widehat{\Pr}(\mathsf{freedom} \mid \mathsf{Bush}) = \frac{41+1}{113+7} = \frac{7}{20}.$$

Problem 8. *Write code that calculates these class conditional probabilities using add-one smoothing. In your writeup, include a table that contains all 14 conditional probabilities for the State of the Union training dataset.*

2.2. **MAP.** Using the Multinomial model, in the State of the Union test set,

$$P(\mathsf{Bush} \mid \text{2012 speech}) \propto \frac{3}{5} \cdot \frac{7}{20} \cdot \left(\frac{1}{120}\right)^5 \approx 8.44e - 12.$$

Analogous to the Bernoulli case, in practice, log probabilities are typically used:

$$\begin{aligned} c_{MAP} &= \operatorname*{argmax}_{c \in C} \widehat{\Pr}(c) \cdot \widehat{\Pr}(w \mid c) \\ &= \operatorname*{argmax}_{c \in C} \log \widehat{\Pr}(c) + \sum_{i=1}^{|w|} \log \widehat{\Pr}(V = t_i \mid c) \qquad . \end{aligned}$$

Problem 9. *Building on your code above, write code that identifies the "best" class for each of the test speeches using the MAP estimate.*

2.3. **Multinomial Naive Bayes: Putting It All Together.** You should now have code that reads in a training and test set (in the format specified above) and calculates the predicted class label for each speech in the test set using the Multinomial Naive Bayes model. Save your script using the following naming convention: *lastname*`_hw1_section2.py`. Your script should generate output in the same format as above.

Problem 10. *Run your script and generate the output in the format above.*

2.4. **Comparing the Multinomial and Bernoulli Models.**

Problem 11. *Based on your empirical data on this (very) small dataset, which model yielded a higher accuracy in predicting the class labels of the test set? What about accuracy by class?*

2.5. **Section 2 Deliverables.** Upload the following to the course web site:

(1) Your Python code: *lastname*`_hw1_section2.py`.

(2) A PDF file containing your numbered answers to all of the numbered problems in Section 2. Your PDF should also include the output of a single run of your script. Use the following naming convention: *lastname*_`hw1_writeup_section2.pdf`.

3. Question Classification: Bernoulli Naive Bayes

We now turn to the task of question classification.

3.1. **Data.** The file `train_5500.label` contains the training set, and the file `TREC_10.label` contains the test set. Each line of the text files (for both training and test) take the following form:

```
COARSE_LABEL:FINE_LABEL Question text
```

We are only concerned with the `COARSE_LABEL`'s. Your code can ignore the `FINE_LABEL`'s.

3.2. **Bernoulli Naive Bayes: Python Implementation.** In Python, implement Bernoulli Naive Bayes using all available word types in the training set as features. (Your code should simply ignore any word types that occur in the test set but not in the training set.) Define a word type as in Assignment 0: using a regular expression, split by whitespace and punctuation (with the exception that internal punctuation, as in contractions such as "don't", are retained). Additionally, disregard capitalization (i.e., ensure all word types are lowercase).

Save your script using the following naming convention: *lastname*_`hw1_section3.py`. When we run your script with

```
% python lastname_hw1_section3.py
```

we should see output in the same format as above, providing both an overall accuracy and accuracy per class.

Problem 12. *Run your script and generate the output in the format above.*

3.3. **Most Common Class.** We'll use a standard simple baseline for prediction accuracy, the most common class.

Problem 13. *What would be the accuracy on the test set of a model that simply always guessed the most common class that occurs in the training set?*

3.4. **Section 3 Deliverables.** Upload the following to the course web site:

(1) Your Python code: *lastname*`_hw1_section3.py`.
(2) A PDF file containing your numbered answers to the numbered questions in Section 3. Your PDF should also include the output of a single run of your script. Use the following naming convention: *lastname*`_hw1_writeup_section3.pdf`.

4. Question Classification: Multinomial Naive Bayes

Using the same word type features as above, apply Multinomial Naive Bayes to the training and test sets.

Save your script using the following naming convention: *lastname*`_hw1_section4.py`. When we run your script with

```
% python lastname_hw1_section4.py
```

we should see output in the same format as above, providing both an overall accuracy and accuracy per class.

Problem 14. *Run your script and generate the output in the format above.*

4.1. **Comparing the Multinomial and Bernoulli Models.**

Problem 15. *Based on your empirical data on this dataset, which model yielded a higher accuracy in predicting the class labels of the test set? What about accuracy by class?*

Problem 16. *Based on your knowledge of the two models, which would you expect, ceteris paribus, to generate a higher class conditional probability for a very common word?*

4.2. **Section 4 Deliverables.** Upload the following to the course web site:

(1) Your Python code: *lastname*_hw1_section4.py.
(2) A PDF file containing your numbered answers to the numbered questions in Section 4, including the output of a single run of your script. Use the following naming convention: *lastname*_hw1_writeup_section4.pdf.

5. Question Classification: Cross Validation

For simplicity, in the above sections, we have used a particular split of the data into training and test sets. In class, we discussed the value of performing cross validation.

Problem 17. *Perform 10-fold cross validation using the multinomial model. The format of the script output (for each run) should be the same as above. Include a copy of the output of all 10 runs.*

Problem 18. *How do your results compare to those you obtained above? Compare the average accuracy on the 10 folds to the accuracy on the single split from Section 4.*

5.1. **Section 5 Deliverables.** Upload the following to the course web site:

(1) Your Python code: *lastname*_hw1_section5.py.
(2) A PDF file containing your answers to the questions in Section 5. Your PDF should also include the output of the 10 runs and the average accuracy rate. Use the following naming convention: *lastname*_hw1_writeup_section5.pdf.

References

Xin Li and Dan Roth. Learning question classifiers. In *COLING'02*, Aug 2002. URL http://l2r.cs.uiuc.edu/~danr/Papers/qc-coling02.pdf.