COS 116
The Computational Universe
Laboratory 11: Machine Learning

In last Tuesday's lecture, we surveyed many machine learning algorithms and their applications. In this lab, you will explore algorithms for two of those applications in greater detail: spam filtering and text generation.

Lab submission: Submit by Tuesday, May 3 by e-mail to TA.
Turn in answers to all the questions posed in the body of the lab and in the "Additional Questions" section.

Part 1: Introduction to Spam Filtering

Modern spam filters use statistical inference to classify an email as spam or "ham" (i.e. non-spam). These filters require access to a large corpus of emails, containing both spam and ham, with each email in the corpus labeled appropriately. The larger and more diverse the corpus, the better the performance of the filter. Even the simplest of these kinds of filters can detect more than 90% of spam. Here is the procedure that a simple spam filter follows to classify an email:

- Step 1: Split the email to be classified into individual words. Ex. "Your loan request approved!" becomes 'your', 'loan', 'request', and 'approved'.

- Step 2: Compute the spam score for each word in the email. The formula for the spam score of a word is:

$$\text{SpamScore}(\textit{word}) = \frac{\text{Fraction of spam emails in the corpus that contain } \textit{word}}{\text{Fraction of ham emails in the corpus that contain } \textit{word}}$$

  Remark: If *word* is much more prevalent in spam than in ham, then SpamScore(*word*) will be a big number. Conversely, if *word* is much more prevalent in ham than in spam, then SpamScore(*word*) will be a small number. So the spam score correlates with the "spammy-ness" of a word.

- Step 3: Multiply the spam scores for all the words in the email to get a spam score for the email itself. For example:

SpamScore("Your loan request approved!") =
SpamScore('your') × SpamScore('loan') × SpamScore('request') × SpamScore('approved').

Remark: The spam score for an email will be large if contains many "spammy" words, and small otherwise.

- Step 4: Classify the email as spam if its spam score is above a certain threshold, and classify it as ham otherwise. The value of the threshold controls how aggressively spam is filtered. A lower threshold will cause more email to be classified as spam.

Review the preceding procedure carefully, as understanding it is necessary for completing the lab. Ask your TA for help if you have any questions.

Part 2: Experiments with Spam Filtering

1. **Open this web page:**

http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab12/spam1.html

This is a web interface to a spam filter. The spam filter is connected to a corpus of roughly 8,000 emails – 6,000 ham and 2,000 spam.

2. **Select "First Data Set", and then click "Classify" (Don't adjust the value of "Threshold" yet.)** This data set contains ten emails. When you click "Classify", the procedure described in Part 1 is performed for each email in the data set, and the results are displayed.

3. **Read the emails that were misclassified, and comment in your report why you think each one was misclassified. Identify some words in each email that you suspect misled the spam filter.**

4. **Adjust "Threshold" so that all the spam emails are correctly classified while minimizing the number of misclassified ham emails. Note this threshold in your report.** Click "Classify" after each adjustment to "Threshold". Recall that an email is classified as spam if its spam score is above the threshold and as ham otherwise.

5. **Adjust "Threshold" so that all the ham emails are correctly classified while minimizing the number of misclassified spam emails. Note this threshold in your report.**

6. **Which of the previous approaches to setting the threshold would you prefer to use for your own inbox? Explain your answer.**

Part 3: More Experiments with Spam Filtering

1. **Open this web page:**

http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab12/spam2.html

This web interface allows you to enter your own emails and have them classified by the spam filter. You can enter any text you like; it doesn't need to be in the format of an email (i.e. with the 'From:', 'To:', etc. at the top). You can even enter exactly one word to determine the "spammy-ness" of that word.

2. **Enter each of the following words *individually* into the text area, and click "Classify" for each word:**
    a. Spam words: viagra, potency, money-back, mortgage, lender, mega
    b. Ham words: blog, management, dialog, ouch, alumni, administrivia

3. **From Part 2, Step 3, obtain the words you identified as having misled the spam filter. Paste each of these words individually into the text area, and classify them. Compared to the words in the previous step, are they as spammy/hammy as you suspected? Report your findings.**

4. ***From your own inbox*, copy a ham and spam email, and classify each of them. (If you are concerned about privacy, know that the web page does not record anything you enter.) Does the spam filter correctly give the spam email a higher spam score than the ham email? Put the text of both emails in your report.**

5. **Try writing a couple of emails that defeat the spam filter.**
    a. **Write an email that conveys a legitimate message, but nonetheless has a high spam score. Try and make the spam score as high as you can.** Ex: "Let's meet for lunch at Frist, I am mega hungry. I hear the chili has a lot of potency."
    b. **Write an email that conveys a spammy message, but nonetheless has a low spam score. Try and make the spam score as low as you can. Can you force the score below 1.0?** Ex: "How about some medicine that rhymes with a famous waterfall?"

    Put the text of both emails in your report.

Part 4: Introduction to Text Generation

It is surprisingly easy to generate novel, semantically-plausible text from a small amount of sample text. For example, from the 2007 State of the Union address, one can automatically generate text like the following:

> "This war is more competitive by strengthening math and science skills. The lives of our nation was attacked, I ask you to make the same standards, and a prompt up-or-down vote on the work we've done and reduce gasoline usage in the NBA."

Below is a simple procedure for generating this kind of text from a sample text. The procedure outputs one word at a time. There is a single parameter, *k*, which controls how similar the generated text should be to the sample text. Higher values of *k* result in greater similarity.

- Step 1: Let "w_1 w_2 … w_k" be the last *k* words outputted.
- Step 2: From among all the occurrences of "w_1 w_2 … w_k" in the sample text, choose one at random.
- Step 3: Output the word in the sample text that immediately follows this occurrence.
- Step 4: Go to Step 1. Repeat as long as you like.

Let's trace through the procedure for the State of the Union example given above. That text was generated by letting $k = 2$. The procedure iterated through Steps 1-3 repeatedly. Below is a "transcript" of the first several iterations. The underlined words are the ones that were used in Steps 1-2, and the italicized word is the one that was outputted in Step 3.

1st iteration: "*This*" [see (a) below]
2nd iteration: "This *war*" [see (a) below]
3rd iteration: "This war *is*"
4th iteration: "This war is *more*"
5th iteration: "This war is more *competitive*"
6th iteration: "This war is more competitive *by*"
7th iteration: "This war is more competitive by *strengthening*"

Observe that this procedure generates text that mimics the sample text at a 'small scale'. For example, if the word "the" is usually followed by a noun in the sample text, this will also be true in the generated text. If the phrase "war on" is usually followed by "terror" in the sample text, this will also be true in the generated text (assuming $k >= 2$).

The text generator has a few minor details that have been elided thus far. For completeness, they are discussed below:

a) In Step 1, if fewer than *k* words have been outputted so far, then all of the words that have been outputted so far are used instead. If no words have yet been outputted, the procedure chooses a word at random from the sample text.

b) In Step 2, if there are no occurrences of "w_1 w_2 … w_k" in the sample text, the procedure looks for occurrences of the last *k-1* words outputted, then the last *k-2* words outputted, and so on. This will eventually succeed, since there is always at least one occurrence of the last word outputted, because every word outputted appears in the sample text.

c) Punctuation is dealt with on a case-by-case basis. Good results can be achieved by treating commas, semi-colons and periods just like whole words, and ignoring all other punctuation.

Part 5: Experiments with Text Generation

1. **Open this web page:**

http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab12/generate.html

This is a web interface to the text generator described in Part 4.

2. **Also open this web page:**

http://www.cs.princeton.edu/courses/archive/spring07/cos116/lab12/texts.html

This web page has links to sample texts.

Note: When copying text, save it to a file first, then open the file and copy. Don't copy from the web page directly.

3. **Copy the 2007 State of the Union address into the text area of the Text Generator, and click "Generate". (Don't change the "Style" option yet.) Do this for several values of *k*. What is the lowest *k* that still produces sensible sounding text? How high can you make *k* before the generated text becomes a nearly verbatim copy of the sample text?**

4. **Copy the 1997 State of the Union address into the text area of the Text Generator, and click "Generate". Do this for several values of *k*. Is it easy to tell text generated from the 1997 SOTU apart from text generated from the 2007 SOTU? Is there any value of *k* for which it is difficult?**

5. **Copy *both* the 1997 and 2007 SOTU addresses into the text area of the Text Generator, and click "Generate". Choose a value for *k* that produces sensible sounding text. Find a funny sentence (e.g. an odd juxtaposition of ideas, an un-Presidential choice of words, etc.), and include it in your report.**

The procedure described in Part 4 is easily modified to produce text one *letter* at a time, instead of one word at a time. In Step 1, let "$l_1 l_2 \ldots l_k$" be the last *k* letters outputted, and modify the remainder of the procedure analogously. Spaces and punctuation are treated as letters. The effect is to produce words that are pronounceable, but don't actually mean anything

6. **Copy the soliloquy from Hamlet into the text area of the Text Generator. Select "Letter" for the "Style" option, and click "Generate". Do this for several values of *k*. What is the lowest *k* that still produces English-like words?**

7. **Copy the English news article on the death of Boris Yeltsin into the Text Generator. Select "Letter" for the "Style" option, and click "Generate". Do this for several values of *k*. Repeat the process for the French article. Are you able to determine the language of the sample text from the generated text?**

For which values of *k* is this difficult? Note that these are two different articles; one is not a translation of the other.

8. **Just for fun: If you have handy access to an essay/term paper that you've written, try feeding it to the Text Generator. Try both the "Letter" and "Word" styles, and different values of *k*. Are you able to recognize the generated text as your own writing?**

Part 6: Complex Text Generation

1. **Open this web page:**
   http://pdos.csail.mit.edu/scigen/

2. **Read the description of what the program on this web page does. Then generate an academic paper in which you are the author. Find a sentence or few sentences in the paper that illustrate that the SCIgen algorithm for text generation is more sophisticated than the simple algorithm described in Part 4. (Hint: The SCIgen algorithm can produce text with large scale structure spanning several paragraphs.) Include these sentences in your report.**

Additional Questions

1. Many email clients (e.g. Outlook, Gmail) prevent images from being displayed. This is because displaying an image makes it possible for the sender to determine whether his email reached the inbox, and therefore was not flagged by the spam filter. Considering the procedure you undertook in Part 3, Step 5, do you understand why images are not usually displayed? Explain.

2. A technique for weakening spam filters is called "poisoning". It takes considerable effort: a spammer sends you many spam messages, but each spam message also contains a few legitimate hammy words. Assuming you mark each of these messages as spam, how will this degrade the effectiveness of your spam filter – specifically, will it make the filter more prone to incorrectly classify ham, or spam? How would you adjust the threshold to compensate for this type of error? And if you make this adjustment, how will that make it easier for spam to get through the filter?