Modeling Problems

Outline

Naive Bayes

Understanding Inputs and Outputs

Organizing Programs

Todo List

Quiz

Naive Bayes Classifier

You probably get a lot of email. I do too. Most of it is *spam*: unsolicited, unwanted advertisements.

This problem came up in the 90s and *spam filters*, largely based on probabilistic machine learning techniques, became popular.

We'll discuss one simple implementation of such a filter, the *naive Bayes classifier*, built on Bayes's rule.

Discrete Probability

Bayes's Rule is an important formula in discrete probability theory; you'll talk about that in CS54.

For this lecture, it's enough to think of probability in this way:

- We have some set of events that can happen
- Each event E has some probability
- 6-sided die: 6 possible events, each equally likely (1/6 chance)
- Course grade: 5 possible letters, some more likely than others (F is very unlikely)

Conditional Probability

- Some events are more or less likely if another event also happens
  - Or: if we know that event A happened, it may tell us event B is more or less likely
- For example: Most days the grass is dry, so wet grass is unlikely. But!
  - If we know it rained last night, it is very likely the grass will be wet
  - Or if we know the sprinklers ran this morning, it is very likely the grass will be wet
  - We can write this as $p(\mathrm{wet}|\mathrm{sprinkler})$, the probability of the grass being wet given that we know the sprinklers ran.

Conditional Probability

- Rain itself is very unlikely, and sprinklers are a 1/7 or 2/7 chance.
  - But knowing the grass is wet changes what we think about these too!
- If you know...
  - the grass is wet, is it more or less likely the sprinklers ran?
  - the grass is wet, is it more or less likely it rained?

Conditional Probability

- In spam detection, advertising emails tend to contain certain words.
  - "sale", "free", "50%", "offer"
  - Many more besides but trying to keep this lecture SFW
- Say you have a big collection of spam and a big collection of legitimate mail
  - We can count how often words appear in both sets
- "sale" is more likely to appear in spam, and less likely to appear in legitimate mail
  - So if we have an unknown email containing "sale"...
  - $p(\text{spam}|\text{sale}) > p(\text{legit}|\text{sale})$

Product Rule

- We can also compute the probability of multiple events happening.
  - For a coin flip to come out heads three times in a row:
    - Each flip is independent of the others, so we multiply their probabilities:
    - $p(\mathrm{H}_1, \mathrm{H}_2, \mathrm{H}_3) = p(\mathrm{H}_1) * p(\mathrm{H}_2) * p(\mathrm{H}_3)$
- It is actually the case that a word like "50%" will likely occur alongside "offer"
  - But to keep the math simple, we'll assume that each word is conditioned only on the class (spam or legit)
  - This is what makes it "naive" Bayes
- So, the probability of an email of k distinct words being spam is:
  - $p(\mathrm{spam}) =$
    $p(\mathrm{words}_0|\mathrm{spam}) * p(\mathrm{words}_1|\mathrm{spam}) * \ldots * p(\mathrm{words}_k|\mathrm{spam})$
  - If the likelihood of spam is bigger than the likelihood of being legit, it's probably spam!

Implementing Naive Bayes

- That's all the theory we need.
- In this assignment we'll be working with whether a review of a thing (a movie, a product, etc) has positive or negative *sentiment*
- We'll go through the outline of the assignment now

Understanding Inputs and Outputs

- The output of this program is clear: a prediction of whether a review is positive or negative
- What is the input of this program? A *review* and ...
  - A "model" of probabilities?
  - The positive and negative sets?
- Is it just one program?

Understanding Inputs and Outputs

- We're used to thinking of programs in two ways:
  1. A giant program like Adobe Photoshop or Microsoft Word
  2. A tiny program like our movie hangman game
- Certainly these are both programs, but one of them is much easier to reason about!
  - At a large scale, Photoshop can be thought of as many small programs
    - One for the flood fill tool, one for the sepiatone filter, etc
  - To manage complexity we have to draw boundaries carefully

Understanding Inputs and Outputs

Our Bayes classifier is kind of three "programs":

1. Train a *model* from positive and negative example files
2. Make a prediction for a single review from this model
3. Evaluate our model's accuracy on a test set of positive and negative examples

Each sub-program should have clear inputs and outputs.

Events are Words

- Our probabilistic model is based on events
- We've seen for spam detection we can use the occurrence of words as events
- So we need a way to turn our positive and negative files into words

Files Contain Words

- We can think of files as a bunch of lines
  - Each line is a bunch of words
- So we have another subprogram that *maps* a file to a bunch of words
- We can count how many times each word appears in the file
  - Using dictionaries!

Organizing Programs

Refer to these slides as you work through the assignment this week!

Counting Words

- The writeup suggests working one function at a time
- First, you'll implement a function to construct a dictionary of word counts
  - Keys: words (strings)
  - Values: counts (integers)
- Split a string on spaces...
- Iterate through each word...
  - What to do for new words?
  - What to do for words you've seen before?

Testing

- Create a file `test_bayes.py`
  - `from bayes import *`
- Write an assert like `assert count_words("one one two") == {"one":2, "two":1}`
- Write more asserts if you need to!

Computing Word Probabilities

The probability of an event is the number of times it happened divided by the number of observations.

If you flipped a coin a million times and it came up heads once and tails 999,999 times, you'd be justified in computing $p(\mathrm{H}) = 0.000001$ for this (unfair!) coin.

So the probability of "one" appearing in the example before is 2/3.

You'll need to write a function that maps from a dictionary of words and counts to a dictionary of words and probabilities, by dividing through the number of total words in the file.

Testing

Write a test case for this too!

Computing Review Probabilities

The probability of a review being positive is the product of the probabilities of each word individually appearing in a positive review. Likewise for a negative review.

Taking it Piece by Piece

In this example, the writeup already has broken down the program into simple sub-programs. Let's do it ourselves with a different example.

Todo List Problem

Design a command-line todo-list program, writing out the key functions and their inputs and outputs.
You should be able to add todos (by typing the text of the todo), mark todos as done (by number), delete todos (by number, but with an "are you sure?" prompt), and show only the todos that are not-done, only those that are done, or all of them.

Todo List Problem

Sounds hard. How do we get started?

Program state

One idea is to think about the state of the program: both the durable, core state as well as the state of the user interface. Brainstorm a bit, putting the state into one or the other “bin”.

How can the core state change?

What actions can the user take to change the durable state? Does that suggest that you'll need certain functions?

How can the UI state change?

Is there more state you need to track to support interactions like addition, deletion, or filtering? How can that be changed? Does that suggest certain functions?

How should the user interact?

What should the rendering of the todo list and the user input be like? How can you structure the overall program to call the right functions in the right places?

A further complication

Your users love your todo app. Two common requests show up again and again:

- Priority levels on todos, and the ability to sort on priority
- Deadlines on todos, and the ability to sort on upcoming deadlines

What durable state do you need to add? What UI state do you need to add? What actions have we added here? How do these two features interact?

Quiz