Nov 23 Classifiers and NLP

Last time we learned:

1. Active Reinforcement nb. We'll play around with the Python 3 package "gym" on Wednesday!

Learning Q versus U

$U(s) = \max_{a \in A(s)} Q(s,a)$ tracks (state, action) tuples

Q-learning and temporal-difference learning are built on the same idea:

$Q-$learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a) \right]$$

$Q(s, \text{"left"})$ vs. $U^{\pi}(s)$

(Passive) temporal-difference (TD): policy π controls many actions.

$$U^{\pi}_{i+1}(s) = U^{\pi}_i(s) + \alpha \left[R(s) + \gamma U^{\pi}_i(s') - U^{\pi}_i(s) \right]$$

The difference is mostly nominal, and lies in how we interpret the outputs, and what we might save into memory. The TD model estimates the utility of a *given* policy π as a whole that may span multiple states. A Q-learning agent is trying to pick the policy as it goes, so it will actively maintain a full dictionary of values of $(state, action)$ tuples.

when we choose action: $Q(s,a_1)$ vs $Q(s,a_2)$ vs $Q(s,a_3)$

Big Logistic Regression

Last time in lecture we discussed *classification* as a possible result of a linear regression model. In particular, we could generalize on multiple linear regression's estimators to create a classifier. All we needed was a *link* function that would take a continuous (∞, ∞) output and shove it into a $(0, 1)$ probability space. The *sigmoid* does this.

A simple logistic regression model is then $P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$.

In reality, we will have many features or predictors. For example:

Predict the probability of precipitation or a storm

Given temperature, pressure, humidity, whether it rained yesterday (or a week ago!), etc.

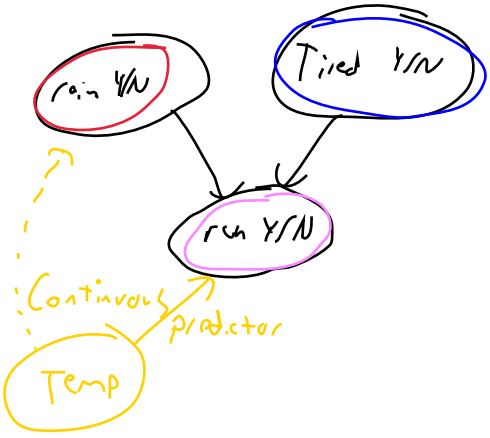
Ties to Our Class

We've seen a lot of models in this class that can make use of this!
In *all* of our Markov Models, we might have time series data to begin with. This allows us to create a model of the conditional probabilities involved:

What is the probability of classifying the day the "does it rain?" on day 1 *as a function* of the weather on day 0? This is a linear model where the response variable is comes from the *features* prior to it in the Bayes Net or HMM.

Similarly, we can do this with an MDP: what kind of underlying probability do we want on our conditional probability *given an action* tables? If the randomness should depend on lots of factors at once - especially continuous ones - we may want a linear model underlying them!

Creating a design matrix: Bayes Nets



Classification Underview

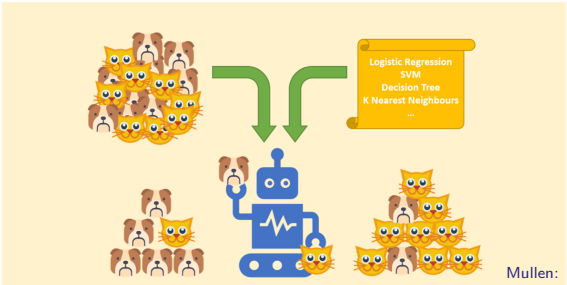
At their roots, many classification problems are about *discretization.* We take processes and measurements that might be continuous, and often turn them into discrete sets of cutoffs. Other examples we've see that might call for a linear model:

1. Is the weather *bad enough* to bring an Umbrella? Or to go for a run?
2. Which set of a discrete set of moves should we take... but trying to apply it to e.g. an autonomous vehicle.
3. Heuristic modeling: sometimes linear or *predictive* models can provide incredibly accurate heuristics (at the possible cost of admissibility/consistency!)

NLP Classifiers

At their roots, many classification problems are about *discretization.* We take processes and measurements that might be continuous, and often turn them into discrete sets of cutoffs. Other examples we've see that might call for a linear model: Now we will do a pair of lectures regarding how classifiers and models work with *Natural Language Processors* (NLP).

It turns out we've already seen one of the first tools for our toolbox in NLP, and it is an alternative to the logistic regression classification model.



NLP Classifiers

NLP is includes at least two major classification tasks:

1. *Text categorization*: the task of assigning a label or category to an entire text or document.
2. *Sentiment categorization*: the extraction of sentiment, argument, logic, or the positive or negative orientation that an author expresses toward some subject or object.

Simple Classifiers

An example we touched on earlier in the semester was *e-mail* classification as either **SPAM** or **HAM**. This was a *binary classification* of the text document, and we used Bayes' theorem to set up some preliminary classifications of $P(HAM|\,words)$ and $P(SPAM|\,words)$.

Imagine working for a review aggregator like Yelp or Rotten Tomatoes. A categorization can be done for general sentiments, by categorizing words as positive or negative:

Positive $(+)$	**Negative** $(-)$
... zany characters and richly applied, satire, and some great plot twists	It was pathetic. The worst part about it was the boxing scenes
awesome caramel sauce and sweet toasty almonds. I love this place!	awful pizza and ridiculously overpriced

unrealistic/under difficult

Supervision Classifiers

Machine learning tasks are often divided into two sub-types:

1. **Supervised Learning**, where a model is constructed using data whose categories are *known*. The model is then applied to other data sets (or a *holdout* from the original set) to test its efficacy or accuracy.
2. **Unsupervised Learning**, where a model must try to create categories based on *features* of the data set without having any a priori knowledge of what separates the categories.

Supervised NLP

Suppose we are provided with a set of N documents that have been pre-labelled with classes, so we have a set of tuples $(document_i, class_i)$. Our goal might be to:

1. Describe the *features* (words, sentences, logical thoughts) that make a documnet *more likely* to belong to a specific class
2. Be able to assign a probability to a *new* or unobserved theoretical document belonging to the classes in the original corpus.

GOAL: P(class given data)

Definition: A *generative classifier* like the Naive Bayes spam filter builds a model of how a class could generate input data. P(words | HAM), P(words | SPAM)

Definition: A *discriminative classifier* like the logistic regression model learns what features from the input are most useful to differentiate between classes.

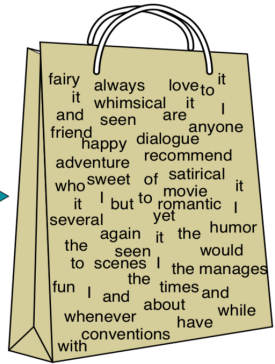
P(class | data)

Naive Bayes

counts/bag of words

ordered text

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Order doesn't matter

Figure 4.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

from "Speech and Language Processing" by Dan Jurafsky and James H. Martin

Naive Bayes

Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Our goal is to create a model that estimate the "probability of words" *given* the document classification and then flip it to find the "probability of classification" given the words in the document.

It's pretty daunting to come up with a model that does the full "probability of words" and includes things like structured thought, so the Naive Bayes approach:

1. "Bag-of-words" assumption: Assume position of words doesn't matter.
2. Naive Bayes Assumption: Conditional independence assumption.

More advanced models start to relax these!

P(A AND B) = P(A) P(B)

P(word 1, word 2 | class) ≈ P(word 1 | class) · P(word 2 | class)

Naive Bayes

Bayes' theorem goal: find $P(class)$ of the ith document given some list of words f_i.

$$P(c|f_i) = \frac{P(f_i|c)P(c)}{P(f_i)}$$

So we have to estimate a few things:

1. For the document *prior* $P(c)$, we could ask what percentage of documents in our training set are in each class c:

$$\hat{P(c)} = \frac{N_c}{N_{doc}}$$

prop. of each class

2. For the probabilities $P(f_i|c)$, we can tally up whether or not a word exists in each document's "bag-of-words:"

$$\hat{P(w_i|c)} = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

c both word & class

p all words in that class

where the numerator is the overlap of the word and the class, and the denominator sum spans all of the *words* in our total vocabulary V.

Smoothing Naive Bayes

Tallying only observed words can be awkward.
Example: Suppose we're trying to estimate the likelihood of the word "fantastic" in the *positive* association class, but we don't actually have any observations of "fantastic" in a known positive document. Then we'd have:

$$P(\text{“}fantastic\text{”}|+) = \frac{count(\text{“}fantastic\text{”}, +)}{\sum_{w \in V} count(w, +)} = 0$$

which is probably *too* strong of an inference. As a result, it's common to use "Add-one" or *Laplace* smoothing, and estimate:

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w, c) + 1} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

Naive Bayes Underview

The Naive Bayes can have some issues:

1. **Unknown words**: words in the new data but not in a predefined vocabulary are hard to classify
2. **Stop words** like "the," or "and" don't actually convey sentiment, but will make up for a large proportion of the sample data. They should be predefined and omitted, or else you might think that the word "and" is predictive just because some of your training data used it more in one class than another!

Log rule: most frequent words occur exponentially/log more often

function TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

for each class $c \in C$ # Calculate $P(c)$ terms
  N_{doc} = number of documents in D
  N_c = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class c
  for each word w in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of w in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
return $logprior$, $loglikelihood$, V

function TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) **returns** best c

for each class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  for each position i in $testdoc$
    $word \leftarrow testdoc[i]$
    if $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
return $\text{argmax}_c\ sum[c]$

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add-α smoothing instead, change the $+1$ to $+\alpha$ for loglikelihood counts in training.

NB Example

order / structure / sequence not matching

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

Example: classify "predictable with no fun" as "positive" or "negative."

Grams

That was a little awkward. Our classifier found "fun" in the positive sentiments, and increased $P(+)$ in our classifier, despite the sentence being "no fun." One way to work around this is an $N-$gram or a single.

Definition: a sequence of written symbols of length N is an $N-$gram.

1. In NLP, an $N-$ gram could be a sequence of words: "no fun" is a **bigram**, since it's $N = 2$. "just plain boring" is a **trigram**.
2. We could also create grams out of individual letter or character sequences. For example, $boring$ includes the *letter* trigrams "bor," "ori," "rin," and "ing."

Taking a document of e.g. 1000 symbols and creating 998 trigrams is known as *shingling*.

Grams

tf: higher = unique to that document

Gram or shingle models start to open up numerous more possibilities.

1. Measures of *document similarity*. Comparing the frequency of grams in a set of documents allows us to compare the documents. For example, if we set N to be 20 and words, documents only "share" grams if there are *exactly copied* sequences of 20 words between them. Plagiarism, indeed!

2. A measure common for shingling and grams are *term frequency* tf and *inverse document frequency* idf. The resulting measures:

$$tf(t, d) = f_{t,d} := \text{times term t appeared in document d}$$

$$idf(t, D) := \text{measures the times term t appeared in all documents D}$$

$$\text{tf-idf}(t, d, D) = tf(t, d) \cdot idf(t, D) = f_{t,d} \cdot \frac{|D|}{count_{d \in D}(\{t \in d\})}$$

← overall frequency

tf-idf tracks whether *important* words are observed.

Sentences and logic

In the second overview of NLP, we'll talk about the next big model: how to create better descriptions of sequences or words. In particular:

1. a gram model is often a *Markov Model*, since certain terms are much more likely to be followed by other terms. This also lets us tracking something like $P(\text{“}nofun\text{”})$ by looking at the bigrams that begin with “no.”
2. Modern AI has to read documents to figure out underlying predicate logic. For example, with a dictionary of words available, a modern AI could take the sentence “Zach likes math” and generate all kinds of *predicate logical* conclusions.

Suppose we classify “like” as a verb that needs both an object and a subject: $L(x, y) :=$ “x likes y.” Then we have statements such as
$\exists x L(x, \text{“}math\text{”})$, and $\exists y L(\text{“}Zach\text{”}, y)$,
and so forth: we can open up the whole cadre of predicate statements from discrete logic!

Moving Forward

- Coming up:
  1. Some notebooks Friday
  2. We talk more AI classifiers! NLP, Perceptrons, and then NNs!