NAIVE BAYES AND SENTIMENT CLASSIFICATION

Introduction

- Classification lies at the heart of both human and machine intelligence.
  - Deciding what letter, word, or image has been presented to our senses
  - recognizing faces or voices
  - sorting mail
  - assigning grades to homework
- these are all examples of assigning a category to an input.

Introduction

- text categorization, the task of assigning a label or category to an entire text or document.
- sentiment analysis, the extraction of sentiment, the positive or negative orientation that a writer expresses toward some object.
- The simplest version of sentiment analysis is a binary classification task, and the words of the review provide excellent cues.
- Consider, for example, the following phrases extracted from positive and negative reviews of movies and restaurants.:

Introduction

- + ...many 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...

- Words like great, richly, awesome, and pathetic, and awful and ridiculously are very informative cues:

Introduction

- Spam detection is another important commercial application, the binary classification task of assigning an email to one of the two classes spam or not-spam.

- Many lexical and other features can be used to perform this classification.

- For example you might quite reasonably be suspicious of an email containing phrases like “online pharmaceutical” or “WITHOUT ANY COST” or “Dear Winner”.

Introduction

- One of the oldest tasks in text classification is assigning a library subject category or topic label to a text.
- Deciding whether a research paper concerns epidemiology or embryology, is an important component of information retrieval.
- Even language modeling can be viewed as classification: each word can be thought of as a class, and so predicting the next word is classifying the context-so-far into a class for each next word.
- A part-of-speech tagger classifies each occurrence of a word in a sentence as, e.g., a noun or a verb.

Introduction

- The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes.

- One method for classifying text is to use handwritten rules.

- There are many areas of language processing where handwritten rule-based classifiers constitute a state-of-the-art system, or at least part of it.

Introduction

- Rules can be fragile, however, as situations or data change over time, and for some tasks humans aren't necessarily good at coming up with the rules.
- Most cases of classification in language processing are instead done via supervised machine learning.
- In supervised learning, we have a data set of input observations, each associated with some correct output (a 'supervision signal').
- The goal of the algorithm is to learn how to map from a new observation to a correct output.

Introduction

- Formally, the task of supervised classification is to take an input x and a fixed set of output classes Y = {y1, y2,..., yM} and return a predicted class y ∈ Y.

- In the supervised situation we have a training set of N documents that have each been hand-labeled with a class: {(d1, c1),....,(dN, cN)}.

- Our goal is to learn a classifier that is capable of mapping from a new document d to its correct class c ∈ C, where C is some set of useful document classes.

- A probabilistic classifier additionally will tell us the probability of the observation being in the class

Introduction

- Generative classifiers like naïve Bayes build a model of how a class could generate some input data.
- Given an observation, they return the class most likely to have generated the observation.
- Discriminative classifiers like logistic regression instead learn what features from the input are most useful to discriminate between the different possible classes.

Naive Bayes Classifiers

- We represent a text document bag of words as if it were a bag of words, that is, an unordered set of words with their position ignored, keeping only their frequency in the document.

Naive Bayes Classifiers

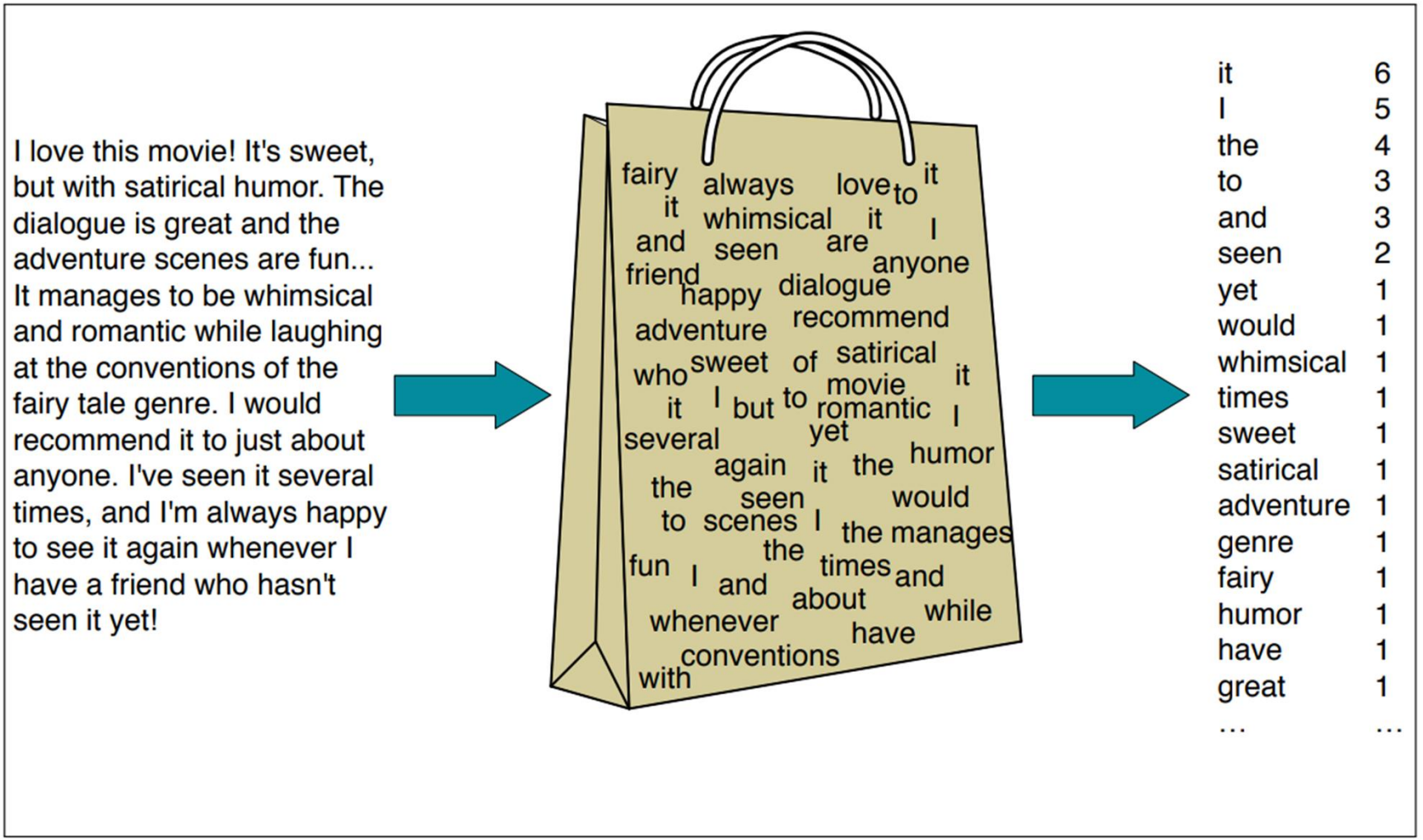


Figure 4.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag-of-words* assumption) and we make use of the frequency of each word.

Naive Bayes Classifiers

- Naive Bayes is a probabilistic classifier, meaning that for a document d, out of all classes c ∈ C the classifier returns the class $\hat{c}$ which has the maximum posterior probability given the document.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

Naive Bayes Classifiers

- Bayes' rule gives us a way to break down any conditional probability P(x|y) into three other probabilities

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Substituting in the previous equation,

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

Naive Bayes Classifiers

- We can simplify the Eq. by dropping the denominator P(d). This is possible because P(d) doesn't change for each class and hence will not give any contribution in deciding argmax.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Naive Bayes Classifiers

- To return to classification: we compute the most probable class $\hat{c}$ given some document d by choosing the class which has the highest product of two probabilities:
  - the prior probability of the class P(c) and
  - the likelihood of the document P(d|c):

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

Naive Bayes Classifiers

- we can represent a document d as a set of features f1, f2,..., fn

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(f_1, f_2, \ldots, f_n | c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

- without some simplifying assumptions, estimating the probability of every possible combination of features (for example, every possible set of words and positions) would require huge numbers of parameters and impossibly large training sets.

Naive Bayes Classifiers

- Naive Bayes classifiers therefore make two simplifying assumptions.
  - The first is the bag-of-words.
    - we assume position doesn't matter, and that the word "good" has the same effect on classification whether it occurs as the 1st, 20th, or last word in the document.
    - Thus we assume that the features f1, f2,..., fn only encode word identity and not position.

Naive Bayes Classifiers

- Naive Bayes classifiers therefore make two simplifying assumptions.
  - The second is commonly called the naive Bayes assumption:
    - this is the conditional independence assumption that the probabilities P(fi|c) are independent given the class c and hence can be 'naively' multiplied as follows

$$P(f_1, f_2, ..., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c)$$

Naive Bayes Classifiers

- The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

- Applying NB to text

$$\text{positions} \leftarrow \text{all word positions in test document}$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

Naive Bayes Classifiers

- Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed. Thus Eq. is generally instead expressed as

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

- Classifiers that use a linear combination of the inputs to make a classification decision —like naive Bayes are called linear classifiers.

Training Naive Bayes Classifiers

- How can we learn the probabilities P(c) and P(fi |c)?
- For the class prior P(c) we ask what percentage of the documents in our training set are in each class c.
- Let Nc be the number of documents in our training data with class c and
- Ndoc be the total number of documents. Then:

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

Training Naive Bayes Classifiers

- To learn the probability P(fi |c), we'll assume a feature is just the existence of a word in the document's bag of words.
- P(wi |c) is the fraction of times the word wi appears among all words in all documents of topic c.
- We first concatenate all documents with category c into one big "category c" text.
- Then we use the frequency of wi in this concatenated document to give a maximum likelihood estimate of the probability:

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

Training Naive Bayes Classifiers

- Imagine we are trying to estimate the likelihood of the word "fantastic" given class positive, but suppose there are no training documents that both contain the word "fantastic" and are classified as positive.
- Perhaps the word "fantastic" happens to occur (sarcastically?) in the class negative.
- In such a case the probability for this feature will be zero:

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

Training Naive Bayes Classifiers

- But since naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term for any class will cause the probability of the class to be zero, no matter the other evidence!

- The simplest solution is the add-one (Laplace) smoothing

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

Naive Bayes Algorithm

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 $\text{argmax}_c\ sum[c]$

Naive Bayes Algorithm example

	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

- Predict the class for the test document.
- (Solved in the class)

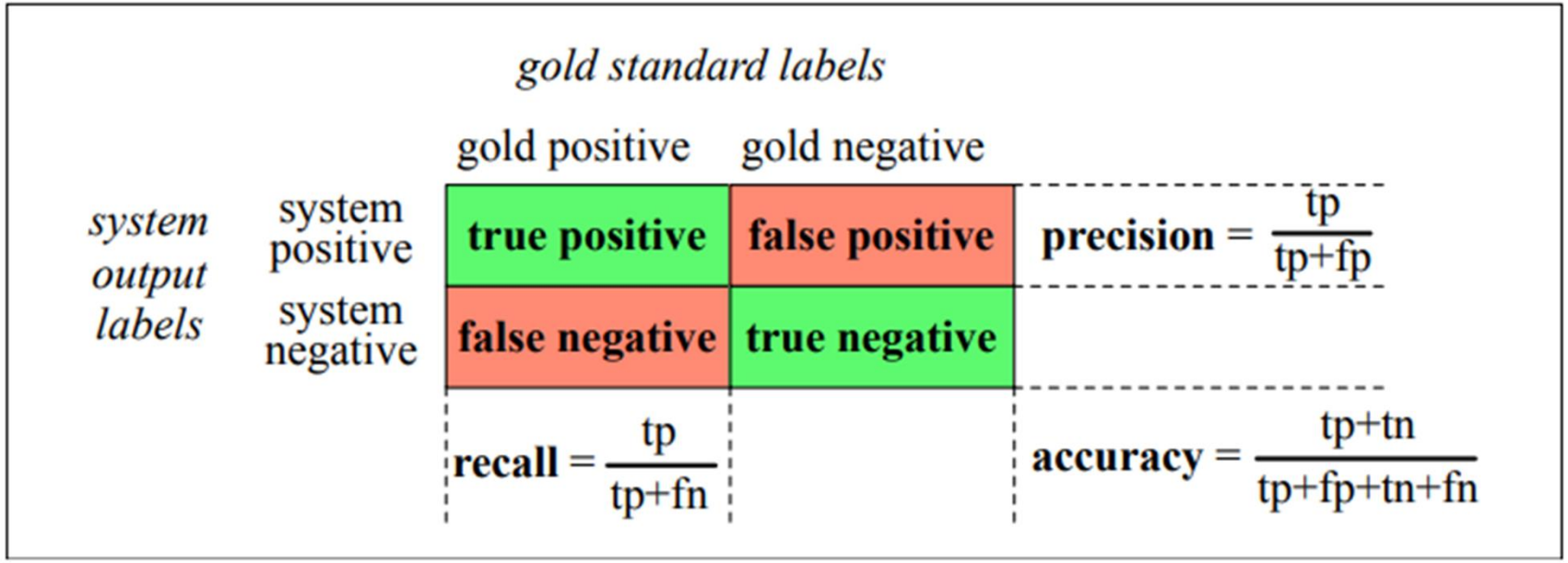
Evaluation: Precision, Recall, F-measure

- Let's consider spam detection example
- Here, our goal is to label every text as being in the spam category ("positive") or not in the spam category ("negative").
- For each item (email document) we therefore need to know whether our system called it spam or not.
- We also need to know whether the email is actually spam or not, i.e. the human-defined labels for each document that we are trying to match.
- We will refer to these human labels as the gold labels

Evaluation: Precision, Recall, F-measure

- To evaluate, we start by building a confusion matrix
- A confusion matrix is a table for visualizing how an algorithm performs with respect to the human gold labels, using two dimensions (system output and gold labels), and each cell labeling a set of possible outcomes.
- In the spam detection case, for example, true positives are documents that are indeed spam (indicated by human-created gold labels) that our system correctly said were spam.
- False negatives are documents that are indeed spam but our system incorrectly labeled as non-spam

Evaluation: Precision, Recall, F-measure



Evaluation: Precision, Recall, F-measure

- Although accuracy might seem a natural metric, we generally don't use it for text classification tasks. That's because accuracy doesn't work well when the classes are unbalanced.

- Imagine that we looked at a million tweets, and let's say that only 100 of them are discussing their love (or hatred) for our pie while the other 999,900 are tweets about something completely unrelated.

- Imagine a simple classifier that stupidly classified every tweet as "not about pie".

Evaluation: Precision, Recall, F-measure

- This classifier would have 999,900 true negatives and only 100 false negatives for an accuracy of 999,900/1,000,000 or 99.99%!

- This classifier would be completely useless, since it wouldn't find a single one of the customer comments we are looking for.

- In other words, accuracy is not a good metric when the goal is to discover something that is rare, or at least not completely balanced in frequency, which is a very common situation in the world.

Evaluation: Precision, Recall, F-measure

- That's why instead of accuracy we generally turn to precision and recall.
- Precision measures the percentage of the items that the system detected (i.e., the system labeled as positive) that are in fact positive (i.e., are positive according to the human gold labels).
- Precision is defined as:

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluation: Precision, Recall, F-measure

- Recall measures the percentage of items actually present in the input that were correctly identified by the system.
- Recall is defined as :

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- Precision and recall, unlike accuracy, emphasize true positives: finding the things that we are supposed to be looking for.
- F-measure is a weighted harmonic mean of precision and recall.

$$F_1 = \frac{2PR}{P+R}$$