Text Categorization Applications

- Web pages
  - Recommending
  - Yahoo-like classification
- Newsgroup/Blog Messages
  - Recommending
  - spam filtering
  - Sentiment analysis for marketing
- News articles
  - Personalized newspaper
- Email messages
  - Routing
  - Prioritizing
  - Folderizing
  - spam filtering
  - Advertising on Gmail

Text Categorization Methods

- Representations of text are very high dimensional (one feature for each word).
- Vectors are sparse since most words are rare.
  - Zipf’s law and heavy-tailed distributions
- High-bias algorithms that prevent overfitting in high-dimensional space are best.
  - SVMs maximize margin to avoid over-fitting in hi-D
- For most text categorization tasks, there are many irrelevant and many relevant features.
- Methods that sum evidence from many or all features (e.g. naïve Bayes, KNN, neural-net, SVM) tend to work better than ones that try to isolate just a few relevant features (decision-tree or rule induction).

Naïve Bayes for Text

- Modeled as generating a bag of words for a document in a given category by repeatedly sampling with replacement from a vocabulary \( V = \{ w_1, w_2, \ldots, w_m \} \) based on the probabilities \( P(w_j | c_i) \).
- Smooth probability estimates with Laplace \( m \)-estimates assuming a uniform distribution over all words \( (p = 1/|V|) \) and \( m = |V| \)
  - Equivalent to a virtual sample of seeing each word in each category exactly once.

Naïve Bayes Generative Model for Text

Naïve Bayes Classification
Text Naïve Bayes Algorithm

(Train)

Let $V$ be the vocabulary of all words in the documents in $D$
For each category $c_i \in C$
   Let $D_i$ be the subset of documents in $D$ in category $c_i$
   Let $P(c_i) = |D_i| / |D|
   Let $T_i$ be the concatenation of all the documents in $D_i$
   Let $n_i$ be the total number of word occurrences in $T_i$
   For each word $w_j \in V$
      Let $n_{ij}$ be the number of occurrences of $w_j$ in $T_i$
      Let $P(w_j | c_i) = (n_{ij} + 1) / (n_i + |V|)$

Text Naïve Bayes Algorithm

(Test)

Given a test document $X$
Let $n$ be the number of word occurrences in $X$
Return the category:

$$\arg\max_{c_i \in C} \prod_{i=1}^{n} P(a_i | c_i)$$

where $a_i$ is the word occurring the $i$th position in $X$

Underflow Prevention

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.
- Class with highest final un-normalized log probability score is still the most probable.

Naïve Bayes Posterior Probabilities

- Classification results of naïve Bayes (the class with maximum posterior probability) are usually fairly accurate.
- However, due to the inadequacy of the conditional independence assumption, the actual posterior-probability numerical estimates are not.
  - Output probabilities are generally very close to 0 or 1.

Textual Similarity Metrics

- Measuring similarity of two texts is a well-studied problem.
- Standard metrics are based on a “bag of words” model of a document that ignores word order and syntactic structure.
- May involve removing common “stop words” and stemming to reduce words to their root form.
- Vector-space model from Information Retrieval (IR) is the standard approach.
- Other metrics (e.g. edit-distance) are also used.

The Vector-Space Model

- Assume $t$ distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These “orthogonal” terms form a vector space.
  Dimension = $t$ = |vocabulary|
- Each term, $i$, in a document or query, $j$, is given a real-valued weight, $w_{ij}$
- Both documents and queries are expressed as $t$-dimensional vectors:
  $$d_j = (w_{1j}, w_{2j}, ..., w_{tj})$$
Graphic Representation

Example:

- $D_1 = 2T_1 + 3T_2 + 5T_3$
- $D_2 = 3T_1 + 7T_2 + T_3$
- $Q = 6T_1 + 0T_2 + 2T_3$

- Is $D_1$ or $D_2$ more similar to $Q$?
- How to measure the degree of similarity? Distance? Angle? Projection?

Document Collection

- A collection of $n$ documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the “weight” of a term in the document; zero means the term has no significance in the document or it simply doesn’t exist in the document.

<table>
<thead>
<tr>
<th>$T_j$</th>
<th>$T_2$</th>
<th>...</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>$w_{11}$</td>
<td>$w_{12}$</td>
<td>...</td>
</tr>
<tr>
<td>$D_2$</td>
<td>$w_{21}$</td>
<td>$w_{22}$</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$D_n$</td>
<td>$w_{n1}$</td>
<td>$w_{n2}$</td>
<td>...</td>
</tr>
</tbody>
</table>

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.
  - $f_{ij} =$ frequency of term $i$ in document $j$
- May want to normalize term frequency ($tf$) by dividing by the frequency of the most common term in the document:
  - $tf_{ij} = f_{ij} / \max_i(f_{ij})$

Term Weights: Inverse Document Frequency

- Terms that appear in many different documents are less indicative of overall topic.
  - $df_i = \text{document frequency of term } i$
    - $= \text{number of documents containing term } i$
  - $idf_i = \text{inverse document frequency of term } i$
    - $= \log_2(N/df_i)$
    - (N: total number of documents)
- An indication of a term’s discrimination power.
- Log used to dampen the effect relative to $tf$

TF-IDF Weighting

- A typical combined term importance indicator is $tf-idf$ weighting:
  - $w_{ij} = tf_{ij} \cdot idf_i = tf_{ij} \cdot \log_2(N/df_i)$
- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, $tf-idf$ has been found to work well.

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.
  - $\text{CosSim}(d, q) = \frac{\vec{d} \cdot \vec{q}}{||\vec{d}|| \cdot ||\vec{q}||} = \frac{\sum_{i} w_{d_i} \cdot w_{q_i}}{\sqrt{\sum_{i} w_{d_i}^2} \cdot \sqrt{\sum_{i} w_{q_i}^2}}$

- $D_1 = 2T_1 + 3T_2 + 5T_3$
  - $\text{CosSim}(D_1, Q) = 10 / \sqrt{14 \cdot 25} = 0.81$
- $D_2 = 3T_1 + 7T_2 + T_3$
  - $\text{CosSim}(D_2, Q) = 2 / \sqrt{14 \cdot 25} = 0.13$
- $Q = 6T_1 + 0T_2 + 2T_3$

$D_1$ is 6 times better than $D_2$ using cosine similarity but only 5 times better using inner product.
Relevance Feedback in IR

- After initial retrieval results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
- Use this feedback information to reformulate the query.
- Produce new results based on reformulated query.
- Allows more interactive, multi-pass process.

Using Relevance Feedback (Rocchio)

- Relevance feedback methods can be adapted for text categorization.
- Use standard TF/IDF weighted vectors to represent text documents (normalized by maximum term frequency).
- For each category, compute a prototype vector by summing the vectors of the training documents in the category.
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

Rocchio Text Categorization Algorithm (Training)

Assume the set of categories is \( \{c_1, c_2, \ldots, c_n\} \)
For \( i \) from 1 to \( n \) let \( p_i = 0, 0, \ldots, 0 \) (init. prototype vectors)
For each training example \( s, c(s) \in D \)
Let \( d \) be the frequency normalized TF/IDF term vector for doc \( s \)
Let \( i = j : (c_j = c(s)) \) (sum all the document vectors in \( c_i \) to get \( p_i \))
Let \( p_i = p_i + d \)

Rocchio Text Categorization Algorithm (Test)

Given test document \( x \)
Let \( d \) be the TF/IDF weighted term vector for \( x \)
Let \( m = -2 \) (init. maximum \( \cosSim \))
For \( i \) from 1 to \( n \) (compute similarity to prototype vector)
Let \( s = \cosSim(d, p_i) \)
if \( s > m \)
let \( m = s \)
let \( r = c_i \) (update most similar class prototype)
Return class \( r \)
Rocchio Properties

- Does not guarantee a consistent hypothesis.
- Forms a simple generalization of the examples in each class (a prototype).
- Prototype vector does not need to be averaged or otherwise normalized for length since cosine similarity is insensitive to vector length.
- Classification is based on similarity to class prototypes.

Rocchio Anomaly

- Prototype models have problems with polymorphic (disjunctive) categories.

Illustration of 3 Nearest Neighbor for Text

K Nearest Neighbor for Text

**Training:**
For each training example \( <x, c(x)> \in D \)
Compute the corresponding TF-IDF vector, \( d_x \), for document \( x \)

**Test instance \( y \):**
Compute TF-IDF vector \( d \) for document \( y \)
For each \( <x, c(x)> \in D \)
Let \( s = \cosSim(d, d_x) \)
Sort examples, \( x, \in D \) by decreasing value of \( s \)
Let \( N \) be the first \( k \) examples in \( D \) \( (\text{get most similar neighbors}) \)
Return the majority class of examples in \( N \)

3 Nearest Neighbor Comparison

- Nearest Neighbor tends to handle polymorphic categories well.

Inverted Index

- Linear search through training texts is not scalable.
- An index that points from words to documents that contain them allows more rapid retrieval of similar documents.
- Once stop-words are eliminated, the remaining words are rare, so an inverted index narrows attention to a relatively small number of documents that share meaningful vocabulary with the test document.
Conclusions

- Many important applications of classification to text.
- Requires an approach that works well with large, sparse features vectors, since typically each word is a feature and most words are rare.
  - Naïve Bayes
  - kNN with cosine similarity
  - SVMs