Text Categorization

Categorization

- Given:
  - A description of an instance, \( x \in X \), where \( X \) is the instance language or instance space.
  - A fixed set of categories: \( C = \{c_1, c_2, \ldots, c_n\} \)
- Determine:
  - The category of \( x \): \( c(x) \in C \), where \( c(x) \) is a categorization function whose domain is \( X \) and whose range is \( C \).

Learning for Categorization

- A training example is an instance \( x \in X \), paired with its correct category \( c(x) \): \( <x, c(x)> \) for an unknown categorization function, \( c \).
- Given a set of training examples, \( D \).
- Find a hypothesized categorization function, \( h(x) \), such that:
  \[ \forall <x, c(x)> \in D : h(x) = c(x) \]
  *Consistency*

Sample Category Learning Problem

- Instance language: \(<\text{size}, \text{color}, \text{shape}>\)
  - size \( \in \{\text{small}, \text{medium}, \text{large}\} \)
  - color \( \in \{\text{red}, \text{blue}, \text{green}\} \)
  - shape \( \in \{\text{square}, \text{circle}, \text{triangle}\} \)
- \( C = \{\text{positive, negative}\} \)
- \( D: \)

<table>
<thead>
<tr>
<th>Example</th>
<th>Size</th>
<th>Color</th>
<th>Shape</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>small</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>2</td>
<td>large</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>3</td>
<td>small</td>
<td>red</td>
<td>triangle</td>
<td>negative</td>
</tr>
<tr>
<td>4</td>
<td>large</td>
<td>blue</td>
<td>circle</td>
<td>negative</td>
</tr>
</tbody>
</table>

General Learning Issues

- Many hypotheses are usually consistent with the training data.
- Bias
  - Any criteria other than consistency with the training data that is used to select a hypothesis.
- Classification accuracy (\% of instances classified correctly).
  - Measured on independent test data.
- Training time (efficiency of training algorithm).
- Testing time (efficiency of subsequent classification).

Generalization

- Hypotheses must generalize to correctly classify instances not in the training data.
- Simply memorizing training examples is a consistent hypothesis that does not generalize.
- Occam’s razor:
  - Finding a simple hypothesis helps ensure generalization.
Text Categorization

- Assigning documents to a fixed set of categories.
- Applications:
  - Web pages
    - Recommending
  - Newsgroup Messages
    - Recommending
    - Spam filtering
  - News articles
    - Personalized newspaper
  - Email messages
    - Routing
    - Prioritizing
    - Folderizing
    - Spam filtering

Learning for Text Categorization

- Manual development of text categorization functions is difficult.
- Learning Algorithms:
  - Bayesian (naïve)
  - Neural network
  - Relevance Feedback (Rocchio)
  - Rule based (Ripper)
  - Nearest Neighbor (case based)
  - Support Vector Machines (SVM)

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 \(<x, c(x)> \in D\)
Let \( d \) be the frequency normalized TF/IDF term vector for doc \( x \)
Let \( i = f: (c_i = c(x)) \)
(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 \( s = \text{cosSim}(d, p_i) \)
  if \( s > m \)
    let \( m = s \)
    let \( r = c_i \) (update most similar class prototype)
Return class \( r \)

Illustration of Rocchio Text Categorization
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 Time Complexity

- **Note:** The time to add two sparse vectors is proportional to minimum number of non-zero entries in the two vectors.
- **Training Time:** \( O(D(L_D + |V_D|)) = O(D L_D) \) where \( L_D \) is the average length of a document in \( D \) and \( V_D \) is the average vocabulary size for a document in \( D \).
- **Test Time:** \( O(L_t + |C||V_t|) \) where \( L_t \) is the average length of a test document and \( |V_t| \) is the average vocabulary size for a test document.
  - Assumes lengths of \( p \) vectors are computed and stored during training, allowing \( \text{cosSim}(d, p) \) to be computed in time proportional to the number of non-zero entries in \( d \) (i.e. \( |F_d| \))

Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in \( D \).
- Testing instance \( x \):
  - Compute similarity between \( x \) and all examples in \( D \).
  - Assign \( x \) the category of the most similar example in \( D \).
- Does not explicitly compute a generalization or category prototypes.
- Also called:
  - Case-based
  - Memory-based
  - Lazy learning

K Nearest-Neighbor

- Using only the closest example to determine categorization is subject to errors due to:
  - A single atypical example.
  - Noise (i.e. error) in the category label of a single training example.
- More robust alternative is to find the \( k \) most-similar examples and return the majority category of these \( k \) examples.
- Value of \( k \) is typically odd to avoid ties, 3 and 5 are most common.

Similarity Metrics

- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous \( m \)-dimensional instance space is Euclidian distance.
- Simplest for \( m \)-dimensional binary instance space is Hamming distance (number of feature values that differ).
- For text, cosine similarity of TF-IDF weighted vectors is typically most effective.

3 Nearest Neighbor Illustration (Euclidian Distance)
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 \(s_x = \cosSim(d, d_x)\)
Sort examples, \(x\), in \(D\) by decreasing value of \(s_x\)
Let \(N\) be the first \(k\) examples in \(D\) (get most similar neighbors)
Return the majority class of examples in \(N\)

Illustration of 3 Nearest Neighbor for Text

Rocchio Anomaly

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

3 Nearest Neighbor Comparison

• Nearest Neighbor tends to handle polymorphic categories better.

Nearest Neighbor Time Complexity

• Training Time: \(O(|D|L_d)\) to compose TF-IDF vectors.
• Testing Time: \(O(L_t + |D||V_t|)\) to compare to all training vectors.
  – Assumes lengths of \(d\) vectors are computed and stored during training, allowing \(\cosSim(d, d_x)\) to be computed in time proportional to the number of non-zero entries in \(d\) (i.e. \(|V_t|\))
• Testing time can be high for large training sets.

Nearest Neighbor with Inverted Index

• Determining \(k\) nearest neighbors is the same as determining the \(k\) best retrievals using the test document as a query to a database of training documents.
• Testing Time: \(O(B|V_t|)\)
  where \(B\) is the average number of training documents in which a test-document word appears.
• Therefore, overall classification is \(O(L_t + B|V_t|)\)
  – Typically \(B \ll |D|\)
Bayesian Methods

- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- Uses prior probability of each category given no information about an item.
- Categorization produces a posterior probability distribution over the possible categories given a description of an item.

Axioms of Probability Theory

- All probabilities between 0 and 1
  \[ 0 \leq P(A) \leq 1 \]
- True proposition has probability 1, false has probability 0.
  \[ P(\text{true}) = 1 \quad \text{P(false)} = 0. \]
- The probability of disjunction is:
  \[ P(A \lor B) = P(A) + P(B) - P(A \land B) \]

Conditional Probability

- \( P(A \mid B) \) is the probability of \( A \) given \( B \)
- Assumes that \( B \) is all and only information known.
- Defined by:
  \[ P(A \mid B) = \frac{P(A \land B)}{P(B)} \]

Independence

- \( A \) and \( B \) are independent iff:
  \[ P(A \mid B) = P(A) \quad \text{P(B \mid A) = P(B)} \]
  These two constraints are logically equivalent
- Therefore, if \( A \) and \( B \) are independent:
  \[ P(A \mid B) = \frac{P(A \land B)}{P(B)} = \frac{P(A)}{P(B)} \]
  \[ P(A \land B) = P(A)P(B) \]

Bayes Theorem

- Let set of categories be \( \{c_1, c_2, \ldots, c_n\} \)
- Let \( E \) be description of an instance.
- Determine category of \( E \) by determining for each \( c_j \)
  \[ P(c_j \mid E) = \frac{P(E \mid c_j)P(c_j)}{P(E)} \]
- \( P(E) \) can be determined since categories are complete and disjoint.
  \[ \sum_{j=1}^{n} P(c_j \mid E) = \sum_{j=1}^{n} \frac{P(E \mid c_j)P(c_j)}{P(E)} = 1 \]
  \[ P(E) = \sum_{j=1}^{n} P(c_j)P(E \mid c_j) \]
Bayesian Categorization (cont.)

- Need to know:
  - Priors: \( P(c_i) \)
  - Conditionals: \( P(E | c_i) \)
- \( P(c_i) \) are easily estimated from data.
- If \( n_i \) of the examples in \( D \) are in \( c_i \), then \( P(c_i) = n_i / |D| \)
- Assume instance is a conjunction of binary features:
  \( E = e_1 \land e_2 \land \cdots \land e_m \)
- Too many possible instances (exponential in \( m \)) to estimate all \( P(E | c_i) \)

Naïve Bayesian Categorization

- If we assume features of an instance are independent given the category \( (c_i) \)
  (conditionally independent).
  \( P(E | c_i) = P(e_1 \land e_2 \land \cdots \land e_m | c_i) = \prod_{j=1}^{m} P(e_j | c_i) \)
- Therefore, we then only need to know \( P(e_j | c_i) \) for each feature and category.

Naïve Bayes Example

- \( C = \{ \text{allergy, cold, well} \} \)
- \( e_1 = \text{sneeze}; e_2 = \text{cough}; e_3 = \text{fever} \)
- \( E = \{ \text{sneeze, cough, ¬fever} \} \)

<table>
<thead>
<tr>
<th></th>
<th>Well</th>
<th>Cold</th>
<th>Allergy</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(c_1) )</td>
<td>0.9</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>( P(\text{sneeze}</td>
<td>c_1) )</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>( P(\text{cough}</td>
<td>c_1) )</td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>( P(\text{fever}</td>
<td>c_1) )</td>
<td>0.01</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Smoothing

- To account for estimation from small samples, probability estimates are adjusted or smoothed.
- Laplace smoothing using an \( m \)-estimate assumes that each feature is given a prior probability, \( p \), that is assumed to have been previously observed in a “virtual” sample of size \( m \).
  \( P(e_j | c_i) = \frac{n_j + mp}{n_i + m} \)
- For binary features, \( p \) is simply assumed to be 0.5.
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.

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 $c_i$.
  - $P(c_i) = |D_i| / |D|$.
  - 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 $T_i$.
    - $P(w_j | c_i) = (n_{ij} + 1) / (n_i + |V|)$.

(Test)

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

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

where $a_j$ is the word occurring the $i$th position in $X$.

Naïve Bayes Time Complexity

- Training Time: $O(|D|/L_d + |C|/|V|)$
  - Assumes $|V|$ and all $D_i$, $n_i$, and $n_{ij}$ pre-computed in $O(|D|/L_d)$ time during one pass through all of the data.
  - Generally just $O(|D|/L_d)$ since usually $|C|/|V| < |D|/L_d$.
- Test Time: $O(|C|/L_t)$
  - $L_t$ is the average length of a test document.
  - Very efficient overall, linearly proportional to the time needed to just read in all the data.
  - Similar to Rocchio time complexity.

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.
Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).
- **Classification accuracy** $c/n$ where $n$ is the total number of test instances and $c$ is the number of test instances correctly classified by the system.
- Results can vary based on sampling error due to different training and test sets.
- Average results over multiple training and test sets (splits of the overall data) for the best results.

N-Fold Cross-Validation

- Ideally, test and training sets are independent on each trial.
  - But this would require too much labeled data.
- Partition data into $N$ equal-sized disjoint segments.
- Run $N$ trials, each time using a different segment of the data for testing, and training on the remaining $N-1$ segments.
- This way, at least test-sets are independent.
- Report average classification accuracy over the $N$ trials.
  - Typically, $N = 10$.

Learning Curves

- In practice, labeled data is usually rare and expensive.
- Would like to know how performance varies with the number of training instances.
- **Learning curves** plot classification accuracy on independent test data ($Y$ axis) versus number of training examples ($X$ axis).

N-Fold Learning Curves

- Want learning curves averaged over multiple trials.
- Use $N$-fold cross validation to generate $N$ full training and test sets.
- For each trial, train on increasing fractions of the training set, measuring accuracy on the test data for each point on the desired learning curve.

References