

Text Classification

CS 277, Data Mining

Padhraic Smyth

Department of Computer Science

Bren School of Information and Computer Sciences

University of California, Irvine

Acknowledgments:

Some slides in this presentations were adapted from *Introduction to Information Retrieval*, by Manning, Raghavan, and Schutze, Cambridge University Press, 2009, as well as from slide materials from Ray Mooney, U Texas Austin

Lectures on Text Analysis

- Text Classification
 - Preprocessing
 - Classification methods
 - Evaluation
- Topic Models and Clustering
 - Principles of topic models
 - Comparisons to clustering
 - Applications
- Other Topics
 - Information extraction, named entity recognition

Further Reading on Text Classification

- General references on text and language modeling
 - *Introduction to Information Retrieval*, C. Manning, P. Raghavan, and H. Schutze, Cambridge University Press, 2008
 - *Foundations of Statistical Language Processing*, C. Manning and H. Schutze, MIT Press, 1999.
 - *Speech and Language Processing: An Introduction to Natural Language Processing*, Dan Jurafsky and James Martin, Prentice Hall, 2000.
 - *The Text Mining Handbook: Advanced Approaches to Analyzing Unstructured Data*, Feldman and Sanger, Cambridge University Press, 2007
- Web-related text mining in general
 - S. Chakrabarti, *Mining the Web: Discovering Knowledge from Hypertext Data*, Morgan Kaufmann, 2003.
 - See chapter 5 for discussion of text classification
- SVMs for text classification
 - T. Joachims, *Learning to Classify Text using Support Vector Machines: Methods, Theory and Algorithms*, Kluwer, 2002

Text Classification

- Text classification has many applications
 - Spam email detection
 - Tagging of news articles, e.g., Google News
 - Classifying Web pages into categories
- Data Representation
 - “Bag of words” most commonly used: either counts or binary
 - Can also use “phrases” (e.g., bigrams) for commonly occurring combinations of words
- Classification Methods
 - Naïve Bayes widely used (e.g., for spam email)
 - Fast and reasonably accurate
 - Support vector machines (SVMs)
 - Often the most accurate method in research studies
 - But complex computationally, so not widely used in practice
 - Logistic Regression
 - Widely used in industry, can be competitive with SVMs
 - Feature selection and regularization are also very important

Types of Labels/Categories/Classes

- Labels for documents or web-pages
 - Labels are often general categories
 - e.g., for news articles
 - *"finance," "sports," "news>world>asia>business"*
 - e.g., for biomedical articles
 - *"gene expression", "microarray", "lung cancer"*
- Labels may be genres
 - *"editorials" "movie-reviews" "news"*
- Labels may be opinion on a person/product
 - *"like", "hate", "neutral"*
- Labels may be domain-specific
 - *"interesting-to-me" : "not-interesting-to-me"*
 - *"contains adult language" : "doesn't"*
 - *language identification: English, French, Chinese, ...*
 - *"link spam" : "not link spam"*

Where do Document Labels come from?

- Manually assigned
 - Predefined dictionary of labels
 - Human labelers read all or part of the article and assigning the most likely label
 - Who are the labelers?
 - Domain experts
 - Librarians/editors (e.g., for the New York Times)
 - Low-paid labelers, e.g., via Amazon Turk
 - This is a subjective process
 - Even domain experts will disagree on some labels
 - In many cases there is no absolute “right” or “wrong” labeling
- Semi-automated process
 - e.g., domain experts define selected keywords for each label
 - Keyword matching used to return documents with most keyword matches for each label
 - Experts then label these returned documents
 - Classifier trained on these labeled documents

Other Aspects of Document Labels

- Large numbers of label values
 - Many applications have a very large number of possible class labels (thousands)
 - Distribution of labels is often highly skewed
 - Some labels very common, other labels very rare
- Multi-Label versus Single-Label documents
 - Multi-Label: each document can have multiple labels
 - Single-Label: each document is assigned a single label
 - The multi-label problem is more complex to handle
 - E.g., the model needs to decide how many labels to assign to each document (we will assume single-label for now, return to multi-label later)
- Hierarchical labels
 - Common in real-world applications that labels are related hierarchically in a tree
 - e.g., *"news>world>asia>business"*
 - Classifiers that use this hierarchy will generally perform better than classifiers that ignore it

Toy Example of Bag-of-Words Document-Term Matrix

	database	SQL	index	regression	likelihood	linear
d1	24	21	9	0	0	3
d2	32	10	5	0	3	0
d3	12	16	5	0	0	0
d4	6	7	2	0	0	0
d5	43	31	20	0	3	0
d6	2	0	0	18	7	16
d7	0	0	1	32	12	0
d8	3	0	0	22	4	2
d9	1	0	0	34	27	25
d10	6	0	0	17	4	23

Practical Document Preprocessing Issues

- Tokenization
 - Convert document to word counts = “bag of words”
 - word token = “any nonempty sequence of characters”
 - for HTML (etc) need to remove formatting
 - Sounds trivial, but can be tricky depending on the document format
- Canonical forms, Stopwords, Stemming
 - Remove capitalization
 - Stopwords:
 - remove very frequent words (a, the, and...) – can use standard stopwords list
 - Can also remove very rare words, e.g., words that occur in k or fewer documents, e.g., $k = 5$
 - Stemming (next slide)
- Data representation
 - e.g., sparse 3 column for bag of words: <docid termid count>
 - can use inverted indices, etc

Stemming

- Want to reduce all morphological variants of a word to a single term
 - Variants of word such as *fish* and *fisher* and *fishing*
 - Remove plurals
- Stemming - reduce words to their root form
 - e.g. *fish* – becomes a new term
- Porter stemming algorithm (1980)
 - relies on a preconstructed suffix list with associated rules
 - e.g. if suffix=IZATION and prefix contains at least one vowel followed by a consonant, replace with suffix=IZE
 - BINARIZATION => BINARIZE
 - Not always desirable: e.g., {university, universal} -> univers (in Porter's)
- WordNet: dictionary-based approach
- Can help classification performance, but not always
 - May be more useful for information retrieval/search than classification

Generating Multi-Word Terms

- Consider multi-word terms like “New York”
 - Would rather treat this as one word “New York” rather than “New” and “York”
 - We can extend our vocabulary to include multi-word terms (or ngrams)
 - Ngrams with $n=1,2,3,4,\dots$ e.g., “University of California Irvine” ($n=4$)
 - Finding candidate n-grams
 - Space of possible multi-word combinations is huge
 - W word tokens: W^2 bigrams, W^3 trigrams, etc. (W order of 10^5)
 - General approach: select ngrams that occur frequently
 - Keep track of all k -frequent ngrams in the corpus (e.g., $k=10$)
 - Use feature selection (e.g., mutual information) to select best
 - Can also use other filters to find good terms,
 - e.g., use a parser to automatically extract noun-phrases
- The big dog jumped over the lazy brown cat

Feature Selection using Mutual Information

Average mutual information between (a) C , the class label and (b) W_t , the presence or absence of a term in a document, defined as

$$\begin{aligned}
 I(C; W_t) &= H(C) - H(C|W_t) & (8) \\
 &= - \sum_{c \in \mathcal{C}} P(c) \log(P(c)) \\
 &\quad + \sum_{f_t \in \{0,1\}} P(f_t) \sum_{c \in \mathcal{C}} P(c|f_t) \log(P(c|f_t)) \\
 &= \sum_{c \in \mathcal{C}} \sum_{f_t \in \{0,1\}} P(c, f_t) \log \left(\frac{P(c, f_t)}{P(c)P(f_t)} \right),
 \end{aligned}$$

From McCallum and Nigam, 1998

Where here c is the class and f_t indicates the presence or absence of term t

Typical approach: compute for all terms, include the top T terms in the classifier, and optimize the value of T via cross-validation

Naïve Bayes and Multinomial Classifiers

Notation

- **Data:**
 - N documents, d terms
 - N x d sparse matrix of counts
 - Each row is a bag of words
- **Variables:**
 - C = class variable, takes one of K values c_k
 - \underline{x} is the d-dimensional vector of counts for a document
- **Probabilistic Classification:**
 - Learn a model for $P(c_k | \underline{x})$, conditional probability of class c_k given \underline{x}
 - Note that by Bayes rule, $P(c_k | \underline{x}) = P(\underline{x} | c_k) P(c_k) / P(\underline{x})$

Probabilistic Classifiers

- Model $P(\underline{x} \mid c_k)$ for each class and perform classification via Bayes rule,

For 0-1 loss select the class c_k that maximizes $P(c_k \mid \underline{x})$

By Bayes rule this is the same c_k that maximizes $P(\underline{x} \mid c_k) P(c_k)$

- How can we model $P(\underline{x} \mid c_k)$?

$P(\underline{x} \mid c_k)$ = probability of a “bag of words” \underline{x} given a class c_k

Two commonly used approaches (for text):

- Naïve Bayes: treat each term x_j as being conditionally independent, given c_k
- Multinomial: model a document with N words as N tosses of a d -sided die

Other models possible but less common,

- E.g., model word order by using a Markov chain for $P(\underline{x} \mid c_k)$

Conditional Independence Assumption

In building probability models it is often convenient to make conditional independence assumptions

Conditional Independence Property:

A is conditionally independent of B, given C, if and only if

$$P(A = a \mid B = b, C = c) = P(A = a \mid C = c) \quad (\text{for all values } a, b, c)$$

In other words, once we know C, then B contains no additional information about A

Example:

A = today's max temp in Irvine, B = today's max temp in Moscow, and C = month of year

Notes:

- Conditional independence does not imply independence
- This is a convenient assumption: often not true, but can capture dependencies to first-order

Naïve Bayes Classifiers

Naïve Bayes classifier is based on a conditional independence model

Assume features x_j are conditionally independent given the class:

$$P(\underline{x} | c_k) = \prod P(x_j | c_k)$$

Product is over all d features, $x_1 \dots x_d$

Model the probability that a word appears in a document, given the document class
Assumes that each word appears conditionally independently of the others

Typical naïve Bayes approach assumes that each x_j is binary, i.e.,
only model whether a word occurs or not in a document, ignore counts
(So the data matrix is a binary matrix)

This binary model is sometimes referred to as **Bernoulli Naïve Bayes** model

Naïve Bayes with Count Data

Non-binary terms (counts) (far less common)

$$P(\underline{x} \mid c_k) = \prod P(x_j = k \mid c_k)$$

We could use a parametric model (e.g., Poisson) or non-parametric model (e.g., histogram) for $P(x_j = k \mid c_k)$ distributions.

Training a Naïve Bayes Classification Algorithm

Data is a matrix of counts, for N documents x d terms

For each class

For each feature

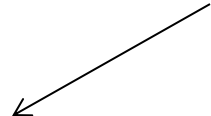
$$P(x_j = 1 \mid c_k) = n_{jk} / n_k$$

end

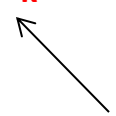
$$P(c_k) = n_k / N$$

end

Number of times term j
occurs in documents labeled as class k



Number of documents
labeled as class k



Output = K sets of d probabilities, one set per class

Complexity? Linear in T where T = total number of word tokens, $O(T + Kd)$

Simple Example: Naïve Bayes (binary)

Vocabulary = {complexity, bound, cluster, classifier}

Two labels: class 1 = algorithms, class 2 = data mining

Examples of $P(\mathbf{x}_j = 1 \mid c_k)$ probabilities for a naïve Bayes classifier:

	complexity	bound	regression	classifier
Algorithms	0.9	0.7	0.1	0.1
Data Mining	0.2	0.1	0.7	0.8

Importance of Smoothing in Probability Estimates

In practice it is useful to use smoothing in probability estimates

Replace frequency counts in probability estimates with

$$P(x_j = 1 \mid c_k) = (n_{jk} + \alpha) / (n_k + \alpha + \beta)$$

where the α and β are smoothing constants

Has the effect of “smoothing” our probability estimates,
away from 0 towards $\alpha / (\alpha + \beta)$

Example:

$$\alpha = 1 \text{ and } \alpha + \beta = 1000$$

This is equivalent to adding 1 additional “pseudocount” of each word in each document, and “pulling” probabilities towards $1/1000$

Smoothing Example

	database	SQL	index	regression	likelihood	linear	Class
d1	1	1	1	0	0	1	1
d2	1	1	1	0	1	0	1
d3	1	1	1	0	0	0	1
d4	1	1	1	0	0	0	1
d5	1	1	1	0	1	0	1
d6	1	0	0	1	1	1	2
d7	0	0	1	1	1	0	2
d8	1	0	0	1	1	1	2
d9	1	0	0	1	1	1	2
d10	1	0	0	1	1	1	2

Unsmoothed estimate: $P(\text{"SQL"} = 1 \mid c_2) = n_{jk} / n_k = 0 / 5 = 0$

Note that this effectively states that it is impossible to see the word "SQL" in documents from class 2

A model that uses this unsmoothed estimate will classify all documents that contain "SQL" as having zero probability of being from class 2 (not desirable in general given that future documents may not look exactly like our finite training sample of documents)

Smoothing Example

	database	SQL	index	regression	likelihood	linear	Class
d1	1	1	1	0	0	1	1
d2	1	1	1	0	1	0	1
d3	1	1	1	0	0	0	1
d4	1	1	1	0	0	0	1
d5	1	1	1	0	1	0	1
d6	1	0	0	1	1	1	2
d7	0	0	1	1	1	0	2
d8	1	0	0	1	1	1	2
d9	1	0	0	1	1	1	2
d10	1	0	0	1	1	1	2

Smoothed estimate:

$$P(\text{"SQL"} = 1 \mid c_2) = (n_{jk} + \alpha) / (n_k + \alpha + \beta) = (0 + 1) / (5 + 1000) \sim 0.001$$

Now future documents with the term “SQL” are unlikely (low probability) to belong to class 2 (according to the model), but are not impossible

If there are enough other words with high probability for class 2, they can counter the “low evidence” of the presence of the “SQL” term.

This model will generalize better to new data, e.g., noisy new documents that have combinations of words we did not see in the training data

Naïve Bayes and Spam Email Classification

Naïve Bayes widely used in spam filtering

See Paul Graham's *A Plan for Spam* (on the Web)

- Naive Bayes-like classifier with unusual parameter estimation

Widely used in early work on spam filters

- Naive Bayes works well when appropriately used
- Very easy to update the model: just update counts

But also many other aspects to spam filtering

- black lists, HTML features, etc.

Multinomial Classifier for Text

Note: in the Manning et al. book they refer to this as a Naïve Bayes Multinomial classifier.

Multinomial Classification model

Assume that the data are generated by a d-sided die (multinomial model)

$$p(\underline{\mathbf{x}} \mid c_k) \propto \prod_{j=1}^{n_j} p(x_j \mid c_k)$$

where product is over all terms in the vocabulary

n_j = number of times term j occurs in the document

Here we have a single random variable for each class, and the $p(x_j \mid c_k)$ probabilities sum to 1 over i (i.e., a multinomial model)

Note that high correlation can not be modeled well compared to naïve Bayes

- e.g., two words that should almost always appear in a document for a given class

Probabilities typically only defined and evaluated for non-zero counts

- But “zero counts” could also be modeled if desired
- This would be equivalent to a Naïve Bayes model with a geometric distribution on counts

Simple Example: Multinomial

Vocabulary = {complexity, bound, cluster, classifier}

Two classes: class 1 = algorithms, class 2 = data mining

Multinomial model:

- Probability of next word in a document, given the document class
- Assumes that words are “memoryless”

Examples of $P(\mathbf{x}_j \mid \mathbf{c}_k)$ probabilities for a multinomial classifier:

	complexity	bound	regression	classifier
Algorithms	0.5	0.4	0.05	0.05
Data Mining	0.1	0.05	0.3	0.55

Note that unlike the naïve Bayes Bernoulli model, here the probabilities across the terms sum to 1

High Probability Terms in Multinomial Distributions

Examples from WebKB data set

Faculty

associate	0.00417
chair	0.00303
member	0.00288
ph	0.00287
director	0.00282
fax	0.00279
journal	0.00271
recent	0.00260
received	0.00258
award	0.00250

Students

resume	0.00516
advisor	0.00456
student	0.00387
working	0.00361
stuff	0.00359
links	0.00355
homepage	0.00345
interests	0.00332
personal	0.00332
favorite	0.00310

Courses

homework	0.00413
syllabus	0.00399
assignments	0.00388
exam	0.00385
grading	0.00381
midterm	0.00374
pm	0.00371
instructor	0.00370
due	0.00364
final	0.00355

Departments

departmental	0.01246
colloquia	0.01076
epartment	0.01045
seminars	0.00997
schedules	0.00879
webmaster	0.00879
events	0.00826
facilities	0.00807
eople	0.00772
postgraduate	0.00764

Research Projects

investigators	0.00256
group	0.00250
members	0.00242
researchers	0.00241
laboratory	0.00238
develop	0.00201
related	0.00200
arpa	0.00187
affiliated	0.00184
project	0.00183

Others

type	0.00164
jan	0.00148
enter	0.00145
random	0.00142
program	0.00136
net	0.00128
time	0.00128
format	0.00124
access	0.00117
begin	0.00116

Predicting Class Labels for Test Documents

To predict a new document, given its term vector \underline{x} , we need to compute $p(\mathbf{c}_k \mid \underline{x})$ which is proportional (as a function of k) to $p(\underline{x} \mid \mathbf{c}_k) p(\mathbf{c}_k)$

For both the naïve Bayes Bernoulli and the multinomial models, the first term on the right is a product of many probabilities
..... so we may get numerical underflow

Predicting Class Labels for Test Documents

To predict a new document, given its term vector \underline{x} , we need to compute $p(\mathbf{c}_k \mid \underline{x})$ which is proportional (as a function of k) to $p(\underline{x} \mid \mathbf{c}_k) p(\mathbf{c}_k)$

For both the naïve Bayes Bernoulli and the multinomial models, the first term on the right is a product of many probabilities
..... so we may get numerical underflow

In practice we can work with

$\log p(\underline{x} \mid \mathbf{c}_k) + \log p(\mathbf{c}_k)$ which is proportional to $\log p(\mathbf{c}_k \mid \underline{x})$
(since the log is a monotonic function)

Because of the independence assumptions in both models, the first term above, $\log p(\underline{x} \mid \mathbf{c}_k)$, is the sum of log-probabilities over individual terms

Score Functions for Prediction

Naïve Bayes

$$\begin{aligned}\log [p(\underline{x} \mid c) p(c)] &= \log [\prod p(x_j \mid c)] + \log p(c) \\ &= \sum \log p(x_j \mid c) + \log p(c)\end{aligned}$$

Sum over all words in vocabulary

Score Functions for Prediction

Naïve Bayes

$$\begin{aligned}\log [p(\underline{x} | c) p(c)] &= \log [\prod p(x_j | c)] + \log p(c) \\ &= \sum \log p(x_j | c) + \log p(c)\end{aligned}$$

Sum over all words in vocabulary

Multinomial

$$\begin{aligned}\log [p(\underline{x} | c) p(c)] &= \log [\prod p(x_j | c)^{n_j}] + \log p(c) \\ &= \sum n_j \log p(x_j | c) + \log p(c)\end{aligned}$$

Sum over all words that occur in document vector \underline{x}

Comments on Probabilistic Text Classifiers

(Comments applicable to both Naïve Bayes and Multinomial classifiers)

- Simple and fast => popular in practice
 - e.g., linear in d , N , K for both training and prediction
 - e.g., easy to use in situations where classifier needs to be updated regularly (e.g., for spam email)
- Numerical issues
 - Typically work with $\log p(c_k | \underline{x})$, etc., to avoid numerical underflow
 - Useful trick for naïve Bayes classifier:
 - when computing $\sum \log p(x_j | c_k)$, for sparse data, it may be much faster to
 - precompute $\sum \log p(x_j = 0 | c_k)$
 - and then subtract off the $\log p(x_j = 1 | c_k)$ terms
- Note: both models are “wrong”: but for classification are often sufficient

Experiments and Evaluation

Confusion Matrix and Accuracy

	True Class 1	True Class 2	True Class 3
Predicted Class 1	80	60	10
Predicted Class 2	10	90	0
Predicted Class 3	10	30	110

Accuracy = fraction of examples classified correctly
= 280/400
= 70%

Precision

Precision for class k

= fraction predicted as class k that belong to class k

= 90% for class 2 below

	True Class 1	True Class 2	True Class 3
Predicted Class 1	80	60	10
Predicted Class 2	10	90	0
Predicted Class 3	10	30	110

Recall

Recall for class k

= fraction that belong to class k predicted as class k

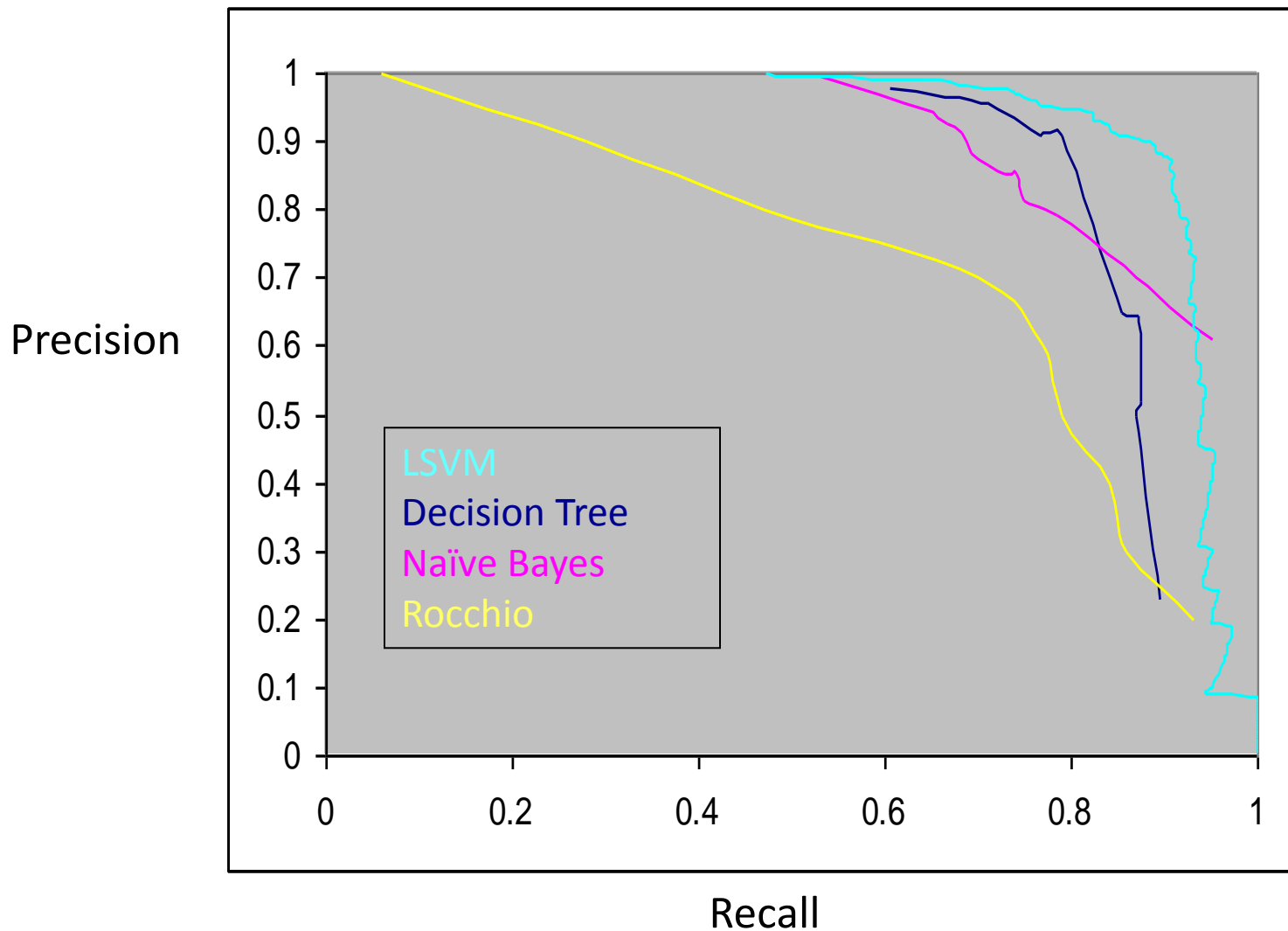
= 50% for class 2 below

	True Class 1	True Class 2	True Class 3
Predicted Class 1	80	60	10
Predicted Class 2	10	90	0
Predicted Class 3	10	30	110

Precision-Recall Curves

- Rank our test documents by predicted probability of belonging to class k
 - Sort the ranked list
 - For $t = 1 \dots N$ (N = total number of test documents)
 - Select top t documents on the list
 - Classify documents above as class k , documents below as not class k
 - Compute precision-recall number
 - Generates a set of precision-recall values we can plot
 - Perfect performance: precision = 1, recall = 1
 - Similar concept to ROC

Precision-Recall for Reuters Class Label = Crude Oil



From
Dumais
(1998)

F-measure

- F-measure = weighted harmonic mean of precision and recall
(at some fixed operating threshold for the classifier)

$$F_1 = 1 / (0.5/P + 0.5/R) = 2PR / (P + R)$$

Useful as a simple single summary measure of performance

Sometimes “breakeven” F_1 used, i.e., measured when $P = R$

Common Data Sets used for Research Evaluation

- Reuters
 - 10700 labeled documents
 - 10% documents with multiple class labels
- Yahoo! Science Hierarchy
 - 95 disjoint classes with 13,598 pages
- 20 Newsgroups data
 - 18800 labeled USENET postings
 - 20 leaf classes, 5 root level classes
- WebKB
 - 8300 documents in 7 categories such as “faculty”, “course”, “student”.
- Industry
 - 6449 home pages of companies partitioned into 71 classes

Note that these data sets are very small compared to “industry-scale” text data

WebKB Data Set

Classify webpages from CS departments into:

- student, faculty, course, project

Train on 5000 hand-labeled web pages

- Cornell, Washington, U.Texas, Wisconsin

Crawl and classify a new site (CMU)

Typical Results with naïve Bayes



	Student	Faculty	Person	Project	Course	Department
Extracted	180	66	246	99	28	1
Correct	130	28	194	72	25	1
Accuracy:	72%	42%	79%	73%	89%	100%

Reuters-21578 Data Set

Most (over)used data set

21578 documents

9603 training, 3299 test articles (ModApte/Lewis split)

118 categories

- An article can be in more than one category
- Learn 118 binary category distinctions

Average document: about 90 types, 200 tokens

Average number of classes assigned

- 1.24 for docs with at least one category

Only about 10 out of 118 categories are large

Common categories
(#train, #test)

- | | |
|----------------------------|-----------------------|
| • Earn (2877, 1087) | • Trade (369,119) |
| • Acquisitions (1650, 179) | • Interest (347, 131) |
| • Money-fx (538, 179) | • Ship (197, 89) |
| • Grain (433, 149) | • Wheat (212, 71) |
| • Crude (389, 189) | • Corn (182, 56) |

Example of a Reuters Document

<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="12981" NEWID="798">

<DATE> 2-MAR-1987 16:51:43.42</DATE>

<TOPICS><D>livestock</D><D>hog</D></TOPICS>

<TITLE>AMERICAN PORK CONGRESS KICKS OFF TOMORROW</TITLE>

<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork Congress kicks off tomorrow, March 3, in Indianapolis with 160 of the nations pork producers from 44 member states determining industry positions on a number of issues, according to the National Pork Producers Council, NPPC.

Delegates to the three day Congress will be considering 26 resolutions concerning various issues, including the future direction of farm policy and the tax law as it applies to the agriculture sector. The delegates will also debate whether to endorse concepts of a national PRV (pseudorabies virus) control and eradication program, the NPPC said.

A large trade show, in conjunction with the congress, will feature the latest in technology in all areas of the industry, the NPPC added. Reuter

</BODY></TEXT></REUTERS>

Comparing Naïve Bayes and Multinomial models

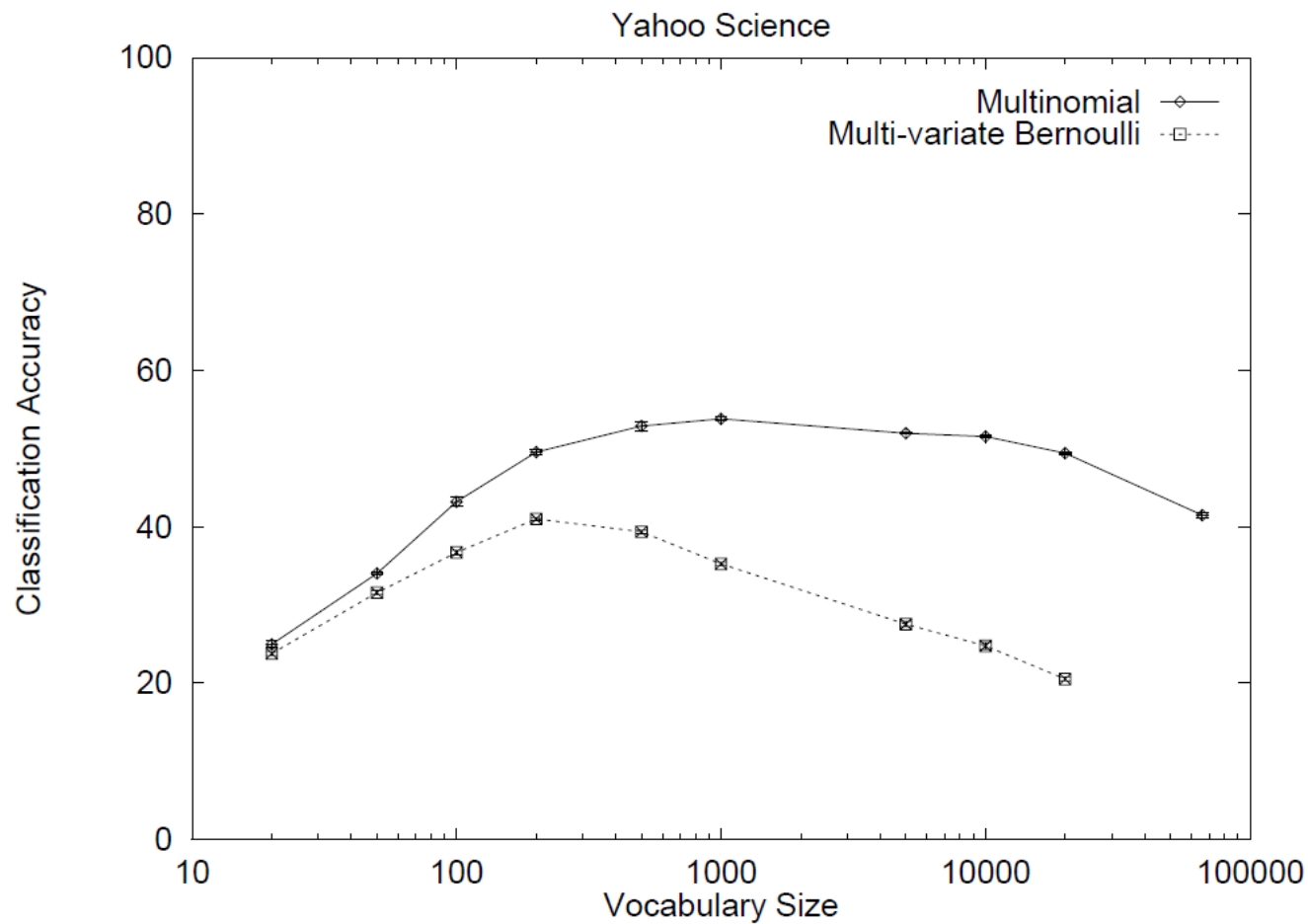
McCallum and Nigam (1998):

- Conducted experiments on a variety of text classification data sets
- Main conclusion was that the multinomial (with counts) tended to have higher accuracy than the naïve Bayes binary/Bernoulli model
- The naïve Bayes binary/Bernoulli model tended to not do as well as the vocabulary size grows

Note on terminology in the literature

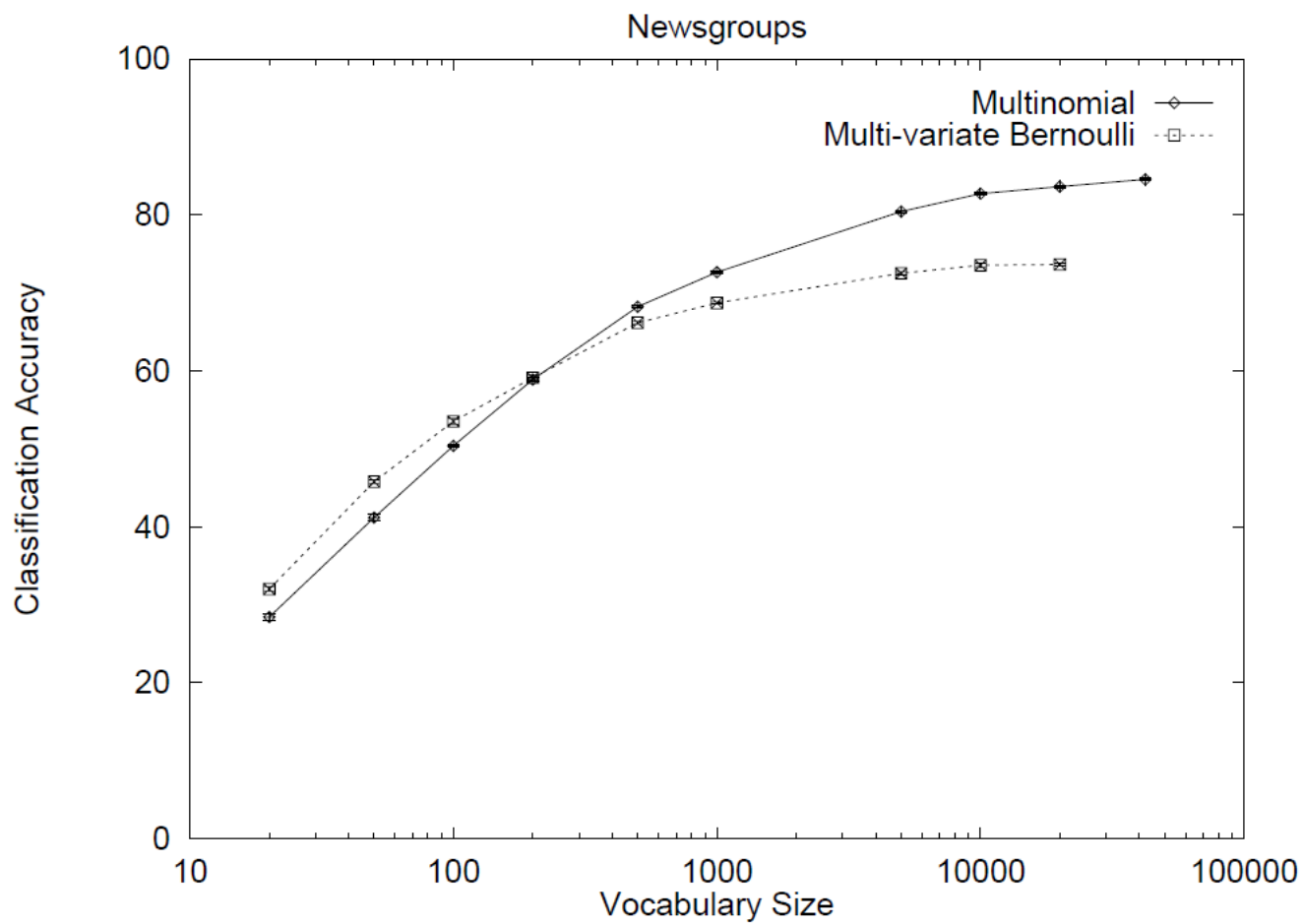
- Bernoulli (or multivariate Bernoulli) sometimes used for binary version of Naïve Bayes model
- multinomial model is also referred to as “unigram” model
- multinomial model is also sometimes (confusingly) referred to as naïve Bayes

Results from classifying 13,589 Yahoo! Web pages in Science subtree
of hierarchy into 95 different topics



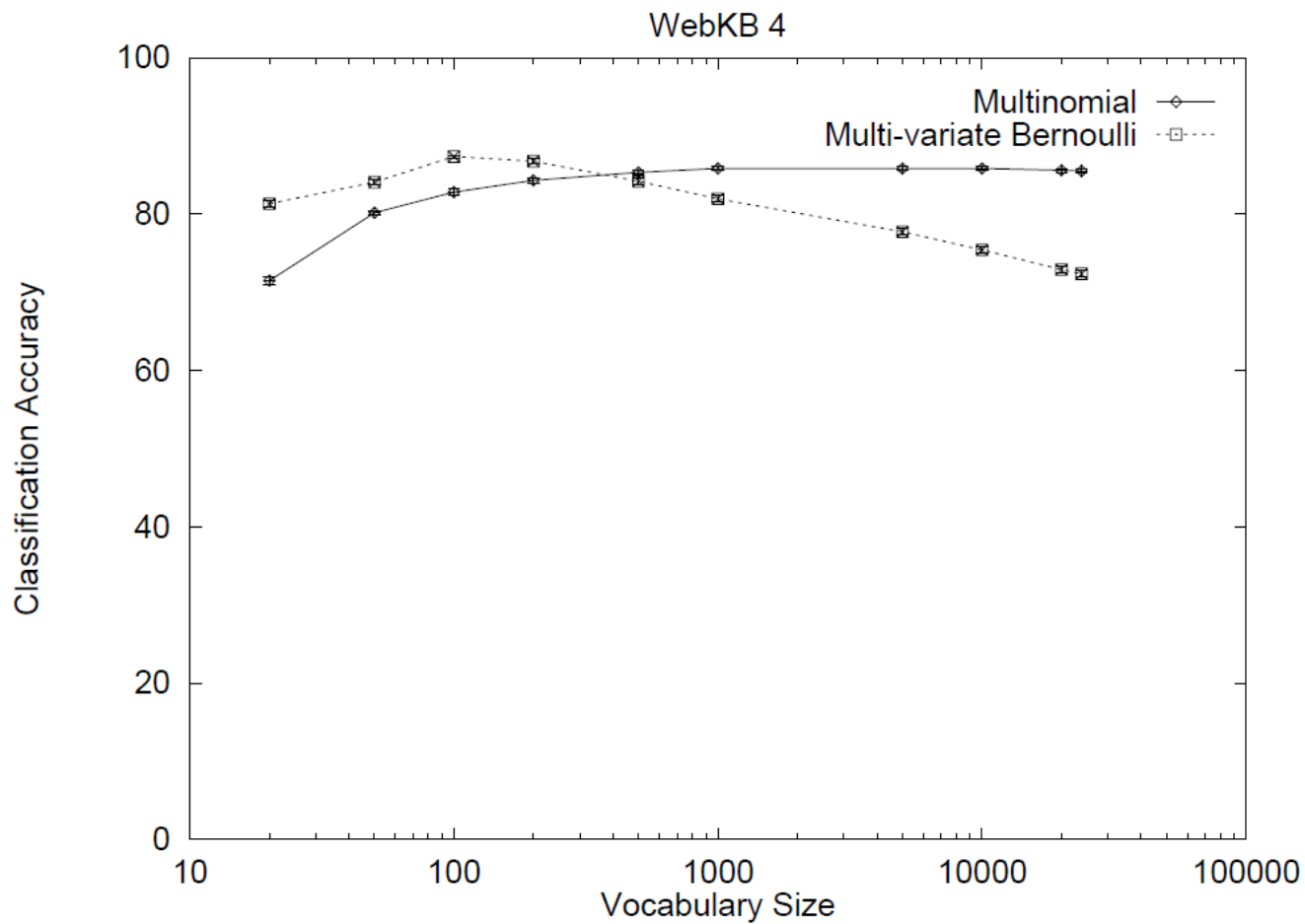
From McCallum and Nigam, 1998

Results from classifying 18,800 Newsgroup postings with 20 class labels



From McCallum and Nigam, 1998

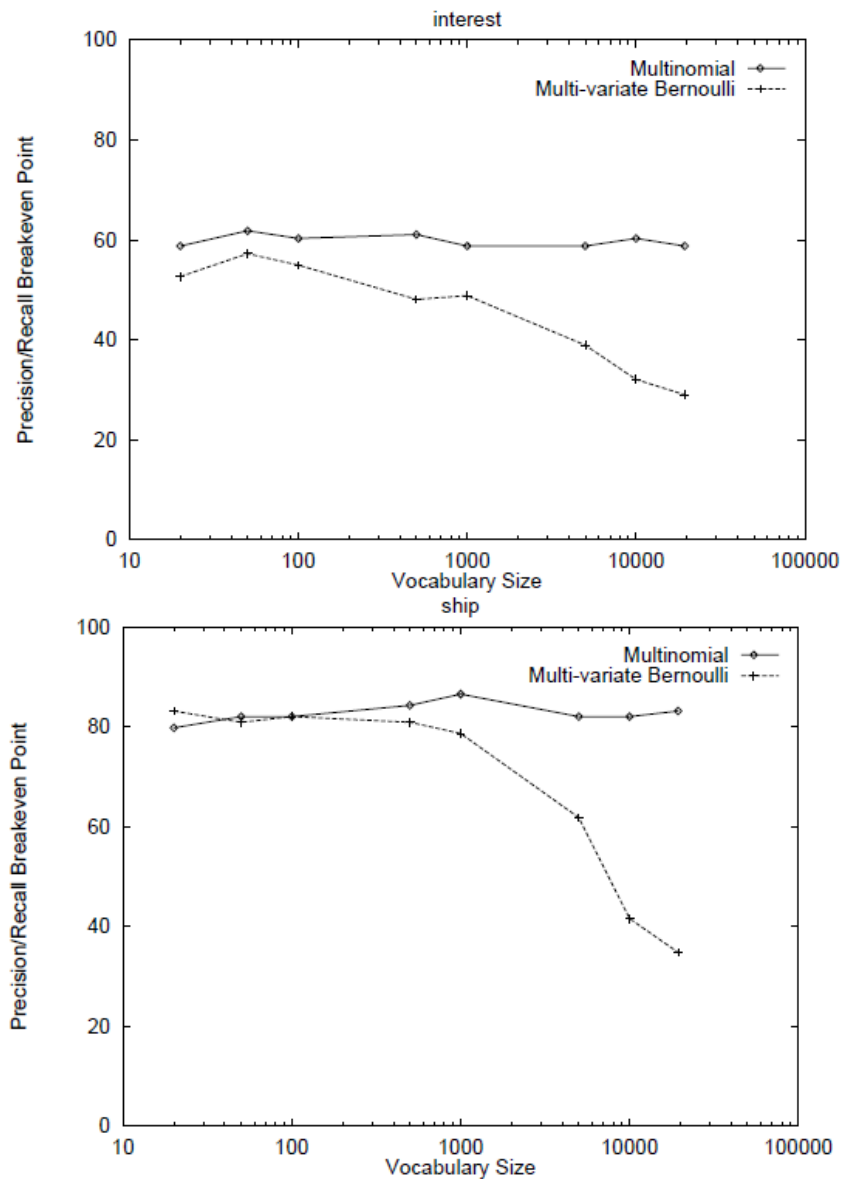
Results from classifying 8300 university Web pages with 7 class labels



From McCallum and Nigam, 1998

Two of the classification tasks for the Reuters Data Set

From McCallum and Nigam, 1998



Feature Selection for Text Classification

- Performance of text classification algorithms can be optimized by selecting only a subset of the discriminative terms
 - Particularly important for Naïve Bayes (see previous slides)
 - Multinomial classifier is less sensitive
- Greedy search
 - Start from empty set or full set and add/delete one at a time
 - Heuristics for adding/deleting
 - Information gain (mutual information of term with class)
 - Chi-square
 - Other ideas

Methods tend not to be particularly sensitive to the specific heuristic used for feature selection, but some form of feature selection often improves performance

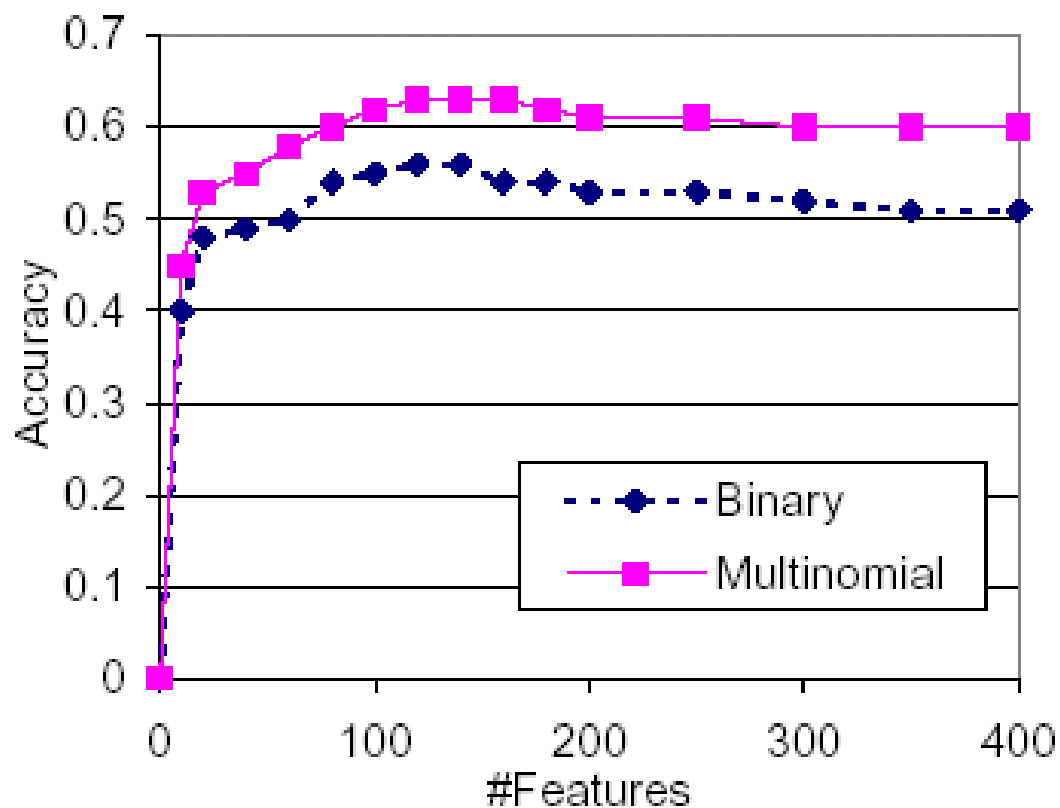
Example of Feature Selection with Multiple Classes

- For each class we build a list of K most discriminating terms
 - Use the union of these terms in our classifier
 - K selected heuristically, or via cross-validation
- For example (on 20 Newsgroups), using mutual information
 - ***sci.electronics***: circuit, voltage, amp, ground, copy, battery, electronics, cooling, ...
 - ***rec.autos***: car, cars, engine, ford, dealer, mustang, oil, collision, autos, tires, toyota, ...
- Greedy: does not account for correlations between terms

Example of Feature Selection

(from Chakrabarti, Chapter 5)

9600 documents from US Patent database
20,000 raw features (terms)



Logistic Regression and Related Ideas

Reading (on class Website)

- *Introduction to Information Retrieval*, C. Manning, P. Raghavan, and H. Schutze, Cambridge University Press, 2008
 - Chapter 13: Naïve Bayes classifiers for text
 - Chapter 15: Support vector machine classifiers for text
- Notes on logistic regression by Charles Elkan
- Paper on large-scale Bayesian logistic regression

Linear Classifiers for Text Documents

- Linear classifiers use a weighted sum of the inputs
 - For d terms we learn d weights, one per input
- Examples
 - Logistic Regression
 - Support Vector Machines
 - Extensions: Neural Networks
- In practice the inputs can be weighted
 - Instead of raw counts it can be helpful to use TF-IDF weights

$\text{TF-IDF}(t, d) = (\text{relative frequency of term } t \text{ in doc } d) * \log (N / \text{number of docs containing term } t)$

The Logistic Regression Classification Model

Notation:

- d -dimensional feature vector \underline{x} (e.g., word counts for a document)
- we assume one of the components of \underline{x} is set to all 1's (to give us an intercept term in the model)
- $c \in \{0, 1\}$ is a binary class label

Logistic regression model with parameter weights β_1, \dots, β_d :

$$P(c_i = 1 | \underline{x}) = \frac{1}{1 + e^{-\sum_{j=1}^d \beta_j x_j}}$$

The Logistic Regression Classification Model

Notation:

- d -dimensional feature vector \underline{x} (e.g., word counts for a document)
- we assume one of the components of \underline{x} is set to all 1's (to give us an intercept term in the model)
- $c \in \{0, 1\}$ is a binary class label

Logistic regression model with parameter weights β_1, \dots, β_d :

$$P(c_i = 1 | \underline{x}) = \frac{1}{1 + e^{-\sum_{j=1}^d \beta_j x_j}}$$

We can interpret this model as a linear weighted sum of the inputs

$$z(\underline{x}) = \sum_{j=1}^d \beta_j x_j$$

where $z(\underline{x})$ is then put through a "squashing" logistic function, $\frac{1}{1 + \exp(z(-\underline{x}))}$ to ensure that its values stay between 0 and 1.

Logistic Regression as a Linear Classifier

We can also rewrite the equation for the logistic model in log-odds form:

$$\frac{P(c_i = 1|\underline{x})}{1 - P(c_i = 1|\underline{x})} = \sum_{j=1}^d \beta_j x_j$$

Because this is a linear function of the input variables \underline{x} , the boundary defined by setting $\frac{P(c_i=1|\underline{x})}{1-P(c_i=1|\underline{x})}$ equal to a constant is a **linear decision boundary** in \underline{x} space. So the logistic regression model is a linear classifier.

The Log-Loss Objective Function

- Training data of N pairs, $\underline{x}_i, c_i, i = 1, \dots, N$
- d -dimensional feature vector \underline{x}_i (e.g., word counts for document i)
- $c_i \in \{0, 1\}$ is the binary class label for example i

Log-Loss Objective Function:

$$L = \sum_{i:c_i=1} \log \frac{1}{p_i} + \sum_{i:c_i=0} \log \frac{1}{1 - p_i}$$

where

$$p_i = p(c_i = 1 | \underline{x}_i, \theta)$$

“theta” here represents the vector of parameters β_1, \dots, β_d

is the prediction of our model, expressed as the probability of class 1 given input \underline{x}_i , with parameters $\theta = \beta_1, \dots, \beta_d$.

Note that minimizing the log-loss function L amounts to

- producing probabilities close to 1 for examples with $c_i = 1$
- producing probabilities close to 0 for examples with $c_i = 0$

Recall: Gradient Descent Algorithm

Iteratively move “downhill” on our loss function by taking small steps in parameter space in the direction of the gradient.

1. Initialize all parameters β_1, \dots, β_d to some initial values (e.g., randomly or heuristically)
2. Given the current parameter vector θ , compute gradient vector of partial derivatives
3. Update each parameter j :

$$\beta_j^{\text{new}} = \beta_j^{\text{old}} - \gamma \frac{\partial L}{\partial \beta_j}$$

4. Check if the algorithm has converged, e.g., if $L^{\text{new}} - L^{\text{old}} < \epsilon$
5. If not converged, return to step 2

Gradient Descent Algorithm for Logistic Regression

One can show (e.g., see Elkan notes on Website) that

$$\gamma \frac{\partial L}{\partial \beta_j} = \sum_{i=1}^N (p_i - c_i) x_{ij}$$

where x_{ij} is the value of the j th feature of feature vector \underline{x}_i .

So the gradient updates are:

$$\beta_j^{\text{new}} = \beta_j^{\text{old}} - \gamma \sum_{i=1}^N (p_i - c_i) x_{ij}, \quad j = 1, \dots, d$$

where γ is a learning rate

In general one update of all parameters takes $O(Nd)$ operations, i.e., 1 iteration through the N training data examples.

Stochastic Gradient Algorithm for Logistic Regression

- Select an example \underline{x}_i randomly from the training data
- Update each parameter based on just on this example:

$$\beta_j^{\text{new}} = \beta_j^{\text{old}} - \gamma(p_i - c_i)x_{ij}, \quad j = 1, \dots, d$$

- Continue to select examples \underline{x}_i at random (and updating each time), until the loss function L appears to have converged

Usually the training examples are presorted in random order and the algorithm cycles through this ordering multiple times, updating parameters after each example is visited.

Useful to prescale the data so that all features are on the same scale (e.g., pre-scale to all have unit variance)

Stochastic Gradient with Sparse Data

Recall the gradient update is:

$$\beta_j^{\text{new}} = \beta_j^{\text{old}} - \gamma(p_i - c_i)x_{ij}, \quad j = 1, \dots, d$$

Note that if $x_{ij} = 0$ then we can simply skip over dimension j since the update term is 0.

Equivalently, we just need to update the weights β_j that are non-zero for example \underline{x}_i .

Say the vocabulary size $= d = 100,000$.

Say the average number of unique words in a document is $f = 20$.

Our complexity per update is now $O(f)$ (very fast) instead of $O(d)$

Examples of Fast Stochastic Gradient Convergence

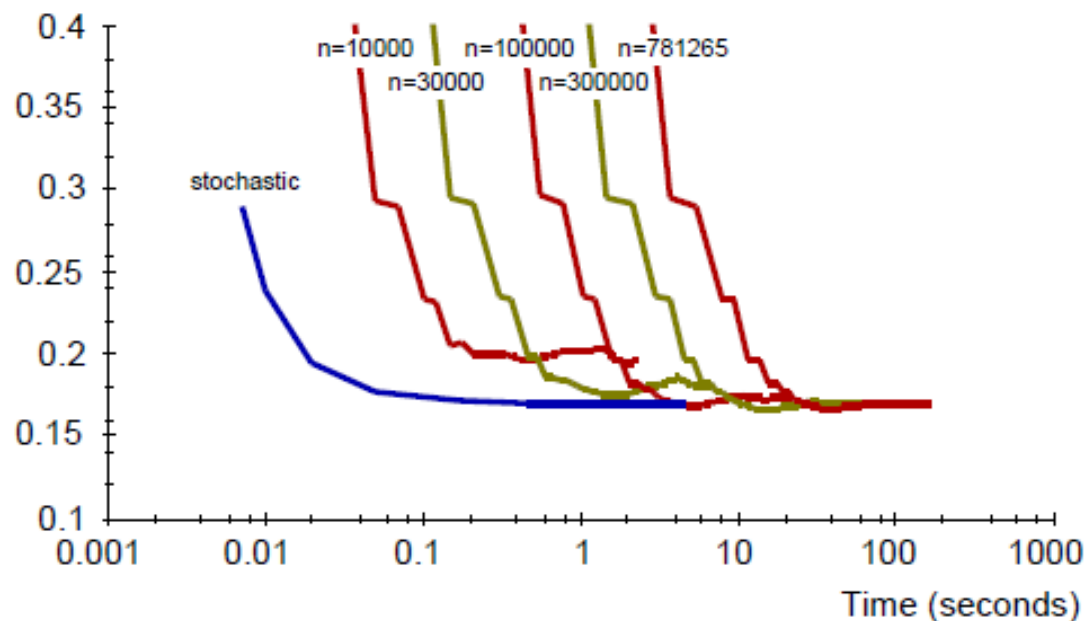
Binary Text Classification Problem

$d = 47k$ word features

$N = 781k$ documents

Various forms of linear classifiers with regularization

Average Test Loss



Log-loss problem

Batch Conjugate
Gradient on various
training set sizes

Stochastic Gradient
on the full set

From Leon Bottou, Stochastic Gradient Learning, MLSS Summer School 2011, Purdue University,
http://learning.stat.purdue.edu/mlss/_media/mlss/bottou.pdf

Regularization in Logistic Regression Learning

With a large number of weights we may overfit

Regularization is a useful way to prevent overfitting

Basic idea is to add a penalty function that penalizes large weights, e.g.,

$$\sum_{j=1}^d \beta_j^2$$

We now want to find the β 's to minimize:

$$\sum_{i:c_i=1} \log \frac{1}{p_i} + \sum_{i:c_i=0} \log \frac{1}{1-p_i} + \lambda \sum_{j=1}^d \beta_j^2$$

where λ is typically selected via cross-validation.

This results in updated gradient equations of the form:

$$\beta_j^{\text{new}} = \beta_j^{\text{old}} - \gamma \left((p_i - c_i) x_{ij} - 2\lambda \beta_j \right), \quad j = 1, \dots, d$$

Different Forms of Regularization

- Sum of squares of weights = Ridge Regression
- Absolute sum of weights = Lasso Regression
 - Has been found to be more effective than sum of squares on some problems
- See Genkin, Madigan, Lewis, 2007, Technometrics for a systematic comparison of different regularization techniques

Experiments Comparing Different Regularizers

From Genkin, Lewis, Madigan, Large-scale Bayesian logistic regression for text categorization, Technometrics, 2007

Table 1. Text categorization effectiveness (macroaveraged F1) on five collections

Algorithm	Threshold	Test collection				
		ModApte	RCV1-v2	OHSUMED	WebKB	20 NG
Lasso	Default	48.64	54.75	47.23	46.96	73.24
Ridge	Default	37.63	47.72	36.09	45.73	71.47
SVM	Default	37.69	44.87	25.28	44.14	75.15
Lasso	Tuned	52.03	57.66	51.30	47.28	75.13
Ridge	Tuned	39.71	52.42	42.99	46.03	73.67
SVM	Tuned	52.09	57.26	49.35	39.50	81.19

Experiments Comparing Different Classifiers

From Zhang and Oles, Text categorization based on regularized linear classification methods

Table 2. Binary classification performance on Reuters (all 118 classes).

	Naive Bayes	Lin Reg	Mod Least Squares	Logistic Reg	SVM	Mod SVM
Precision	77.0	87.1	89.2	88.0	89.2	89.4
Recall	76.9	84.9	85.3	84.9	84.0	83.7
F_1	77.0	86.0	87.2	86.4	86.5	86.5
BEP	75.8	86.3	86.9	86.9	86.5	86.7

Number of Features Selected by the Lasso Method

From Genkin, Lewis, Madigan, Large-scale Bayesian logistic regression for text categorization, Technometrics, 2007

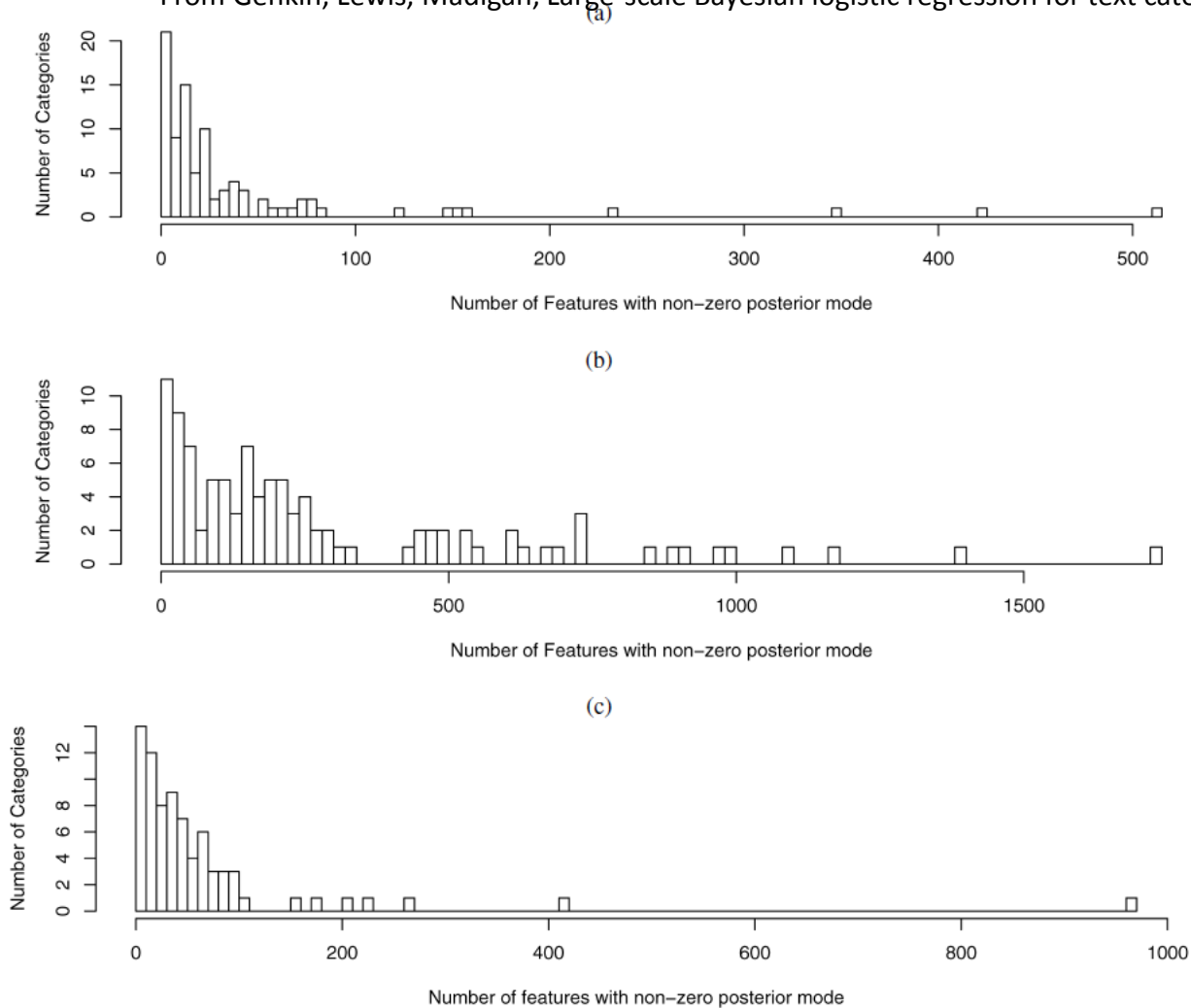


Figure 6. Number of features selected for each category in three test collections by lasso logistic regression (a) ModApte (21,989 features); (b) RCV1-v2 (47,152 features); (c) OHSUMED (122,076 features). Hyperparameter values were chosen by training set cross-validation.

Additional Topics

- Using “document zones”
 - Weighting of terms different zones, e.g., upweighting words in titles, abstracts
 - Separate features/term-counts for different sections of a document
 - E.g., patent abstract v claims
- Features beyond term-counts or term presence
 - Using TF-IDF for term-weighting
$$\text{TF-IDF}(t, d) = (\text{relative frequency of term } t \text{ in doc } d) * \log (N / \text{number of docs containing term } t)$$
 - Grouping terms based on prior knowledge (e.g., lists of positive and negative sentiment words)
 - Using parts-of-speech information (e.g., number of adjectives)
- Semi-supervised learning
 - Probabilistic algorithms (such as Expectation-Maximization) that can learn from both labeled and unlabeled examples
 - Still more of a research topic than a practical technique

[See Manning et al, Chapter 15, Section 15.3 for further discussion]

Accuracy as a Function of Data Size

With enough data the choice of classifier may not matter much

Scaling to very very large corpora for natural language disambiguation
Banko and Brill, Annual Meeting of the ACL, 2001

