Lecture 12: Text Catergorization

Xu Ruifeng

Harbin Institute of Technology, Shenzhen

Last Time

- Chomsky Hierarchy and Grammar
- Parsing Approaches for Context-Free Grammar (CFG)
- Probabilistic Context-Free Grammar (PCFG)
- Lexicalized PCFG
- Dependency Parsing
- Parsing Resources

Today's Class

- Text Categorization: Background
- Document Representation
- Feature Selection
- Feature Generation
- Text Categorization Algorithms
  - Rocchio
  - Bayesian
  - K-Nearest Neighbour
  - CNN and RCNN
- Text Categorization Evaluation

Introduction

- Purpose: classification of natural language texts into a set of predefined labels.

Save 15% on Supplements
Everyday!!
Join our Nutritional Supplement
Discount Program and take 15%
Off Supplement Shelf prices every
day of the year.
Quick, easy sign-up & start saving
immediately.

spam? Or legitimate?

Topic Classification/ Webpage Categorization

娱乐 · 体育

- 2010娱乐年终盘点 娱乐攻略演艺市场抢钱行动盘点 施瓦辛格今日卸任美加州州长 16卫视跨年视频汇总 滨崎步已在美国小教堂完婚 黄子华细数娱乐圈血泪史 图集：李玉刚反串美女 钟楚红精神 罗志祥贺好友大婚 言承旭重金送礼宠女友 威廉王子大婚带来商机
- NBA-8时视频播勇士-魔术 10时直播火箭-掘金 视频 AC米兰官方宣布卡萨诺加盟 国米4年合约签下未来磐石 权威俱乐部教练排名:穆帅居首 拜仁签约巴西天才新星 韦迪:中国足球改变需十年 恒大接洽国安亚冠死敌锋将 李霄鹏:大羽该退役帮女足 韩鹏:保证2010中超最干净 CBA江苏签前国王悍将 达喀尔中国车手退赛 更多新闻>>

社会

- 民警接力资助被害人3名孤儿 保姆乘飞机返回偷金条 妈妈照顾脑瘫女儿20年 窃贼赴派出所报案称遇劫(图) 孕妇节食减肥致孩子患病 男生不满女友转学将其捅死 业主堵路抗议涨停车费 肇事司机疑送医途中弃伤者

篮球 · NBA 李宁iRUN社区送积分

- 月最佳教练：热火创最佳战绩 老辣马刺旧貌换新颜 05:55
- 月最佳新秀：给力芬强势当选 沃尔不敌纽约新人王 05:47
- 易建联伤愈参加完整训练课 桑德斯：他还有些生锈 05:35
- 周最佳球员：韦德当仁不让揽最佳 掘金老枪抢风头 04:44
- 官方实力榜：马刺重登榜首热火次席 湖人仅列第7 01:28
- 阿联满腔热血遭兜头冷水 教练不清楚自己明白吗？ 13:23

国际足坛 李宁iRUN社区送积分

- 西甲-迪马利亚造点球C罗拔头筹 皇马半场2-1领先 05:48
- 国米官方宣布国脚铁卫加盟 未来磐石签约至2015年 21:50
- AC米兰官方宣布卡萨诺加盟 金童首秀后终入圣西罗 20:44
- 穆里尼奥：伊瓜因必须手术 有人自命不凡害了皇马 07:23
- 英超近10年最强11人阵容：欧文+亨利 曼联五巨头 05:31
- 卡卡时隔239天正式回归皇马 穆帅：卡卡踢15分钟 04:28

国内足坛 李宁iRUN社区送积分

- 伊万:鲁能1点优势值得各队学习 气质难用金钱解决 12:12
- 鲁能：李金羽退役一事尚在研究 伊万将定大羽去留 05:32
- 谢菲联整体转让已成定局 姚夏落泪：两年不碰足球 04:15
- 热身赛-蒿俊闵闪电战核心导逆转 国足3比2伊拉克 23:21
- 微博专题-挽留永远的29号 李金羽成1个时代的印记 18:27

综合体育 李宁iRUN社区送积分

- 卢宁军退出2011达喀尔拉力赛 1公里未跑遗憾告退 22:28

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.

Examples of Text Categorization

- LABELS=BINARY
  - “spam” / “not spam”
- LABELS=TOPICS
  - “finance” / “sports” / “asia”
- LABELS=OPINION
  - “positive” / “negative” / “neutral”
- LABELS=AUTHOR
  - “Shakespeare” / “Marlowe” / “Ben Jonson”
  - The Federalist papers

Cost of Manual Text Categorization

- Yahoo!
  - 200 (?) people for manual labeling of Web pages
  - using a hierarchy of 500,000 categories
- MEDLINE (National Library of Medicine)
  - $2 million/year for manual indexing of journal articles
  - using MEdical Subject Headings (18,000 categories)
- Mayo Clinic
  - $1.4 million annually for coding patient-record events
  - using the International Classification of Diseases (ICD) for billing insurance companies
- US Census Bureau decennial census (1990: 22 million responses)
  - 232 industry categories and 504 occupation categories
  - $15 million if fully done by hand

What does it take to compete?

- Suppose you were starting a web search company, what would it take to compete with established engines?
  - You need to be able to establish a competing hierarchy *fast*.
  - You will need a relatively *cheap* solution
- Humans can encode knowledge of what constitutes membership in a category.
- This encoding can then be automatically applied by a machine to categorize new examples

Classification types

- Document Membership
  - Single Label
  - Multiple Labels
  - Binary

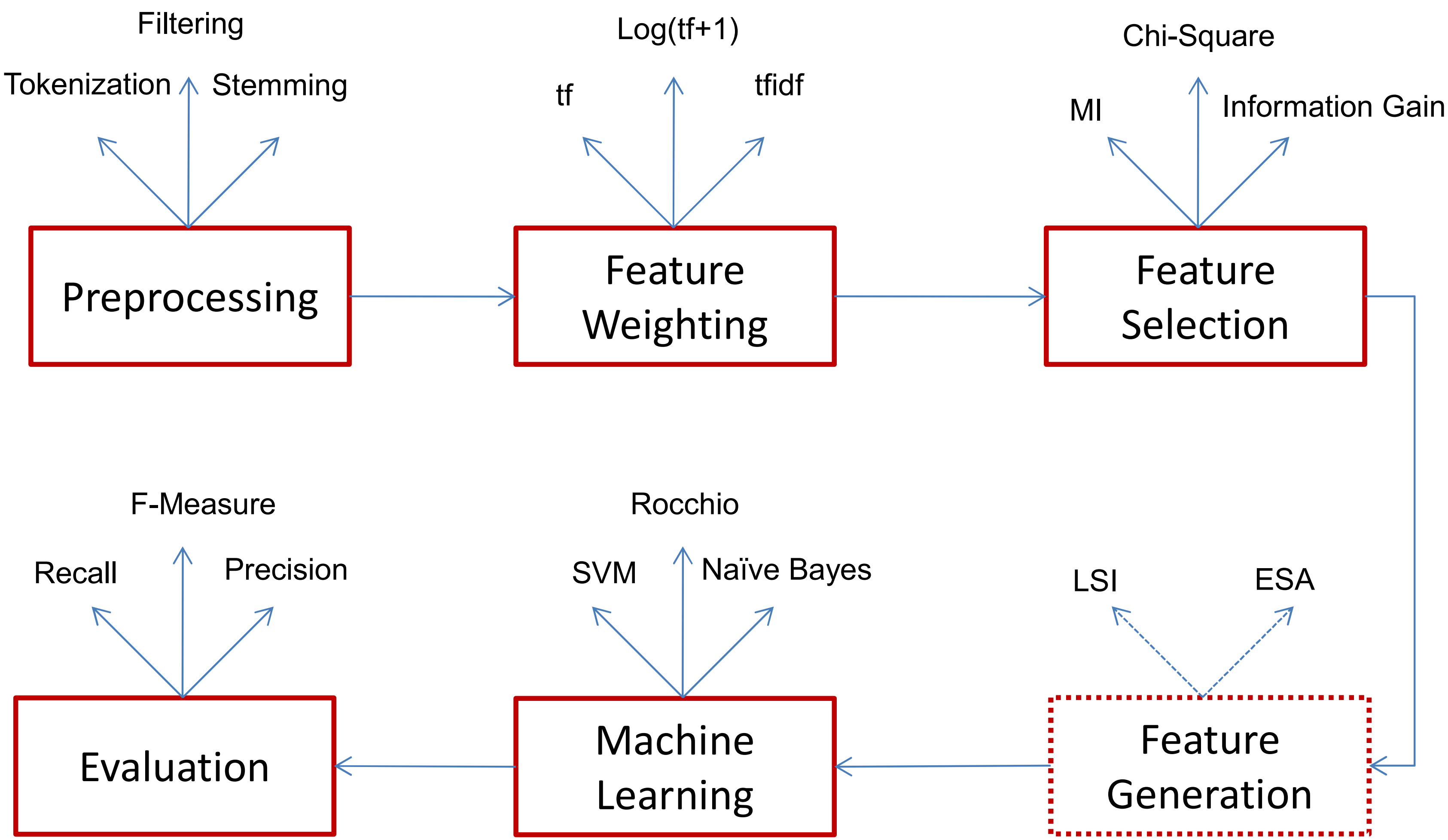
- Hard vs Ranking Classifiers
  - Hard = Decisive!
  - Ranking = Probabilistic

Supervised vs. Unsupervised

- Supervised Learning
  – Training classifier based on set of labeled documents
  – Training set vs. Test set

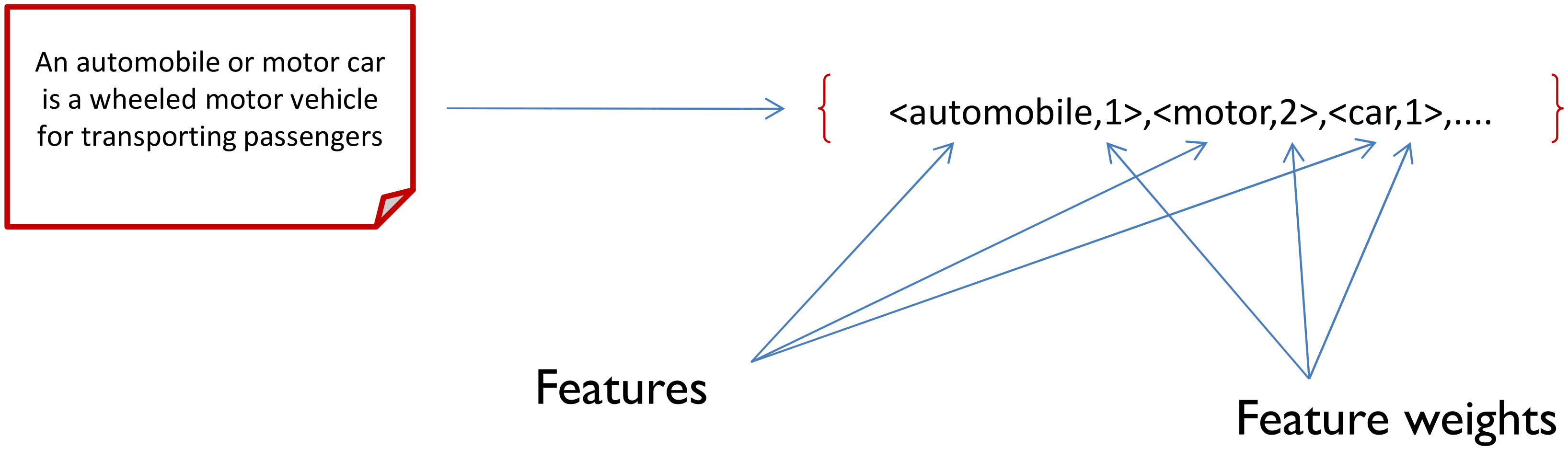
- Unsupervised Learning
  – No labeled examples
  – The system tries to cluster documents based on some heuristics & distance measures

Framework of a Text Categorizer



Document Representation

- The idea is to process the natural language text in a document and transform into a vector



- Document Representation is a vector of term weights
- Each term represents specific information about the original document. Terms are sometimes referred to as features
- Each term usually has an associated weight which represents its contribution to the document

Terms

- Simplest approach: a term is a word
  – Bag of Words
- Preprocessing
  – Stop word removal (“a”,”the”,”of”,”and”)
  – Stemming (“stemming”,”stemmed”,”stemmer”)
- Ignore word order
- Sophisticated Approaches
- Higher Order statistics
- Phrases (how to define?)
  – Syntactically – according to grammar (Noun phrases)
  – Statistically – strongly occurring patterns of words

Weights

- Term frequency $$tf(t_k) = \sum_i \frac{n_k}{n_i}$$
- Tf-IDF
  - The more often the term appears in a document, the more the representative is it of the document.
  - The more documents the term appears in the less discriminating it is.

$$tf.idf(t_k) = tf(t_k).idf(t_k) \quad idf(t_k) = \log(\frac{N_i}{N_k})$$

- Normalized TF-IDF
  - Normalize the tf.idf values to the range[0,1]

$$w(t_k) = \frac{tf.idf(t_k)}{\sqrt{\sum_{s=1}(tf.idf(t_s))^2}}$$

Dimensionality Reduction

- There are many terms
  - Many learning algorithms don’t deal with extremely high dimensions
  - Over fitting problem
  - Not all terms are equally effective
- Dimensionality Reduction - Feature Selection
  - Idea: find a more efficient document representation, with much fewer dimensions, with a minimal loss of effectiveness (accuracy).
- Local vs. Global Policies
  - Local Policy: For each category, find the best terms.
  - Global Policy: Given all the categories find the best terms.

Term Filtering

- A simple filtering can be done by ignoring rare terms
- Remove terms that occur in less that *n* documents
  - Experiments has shown a good performance
    - Dimensionality reduction factor of 10 without loss in accuracy.
    - Dimensionality reduction factor of 100 with small loss in accuracy.

Term Selection

- Out of original set of terms, *t*, find a much smaller subset, *t′*, that yields high-test effectiveness (accuracy).

- Examples
  - Chi Square
  - Mutual Information
  - Information Gain
  - Information Ratio
  - Odd Ratio

Feature Generation

- Term Clustering
  - Unsupervised
  - Supervised
  - Distributional clustering

- Latent Semantic Indexing
- Explicit Semantic Indexing

Latent Semantic Indexing (LSI)

- Words by themselves are not a good measure.
  - Synonyms (car, automobile)
  - Polysemous (Apple, Jaguar)
- LSI: a method for inferring the contextual similarity of terms
  - Finds the best *m* uncorrelated terms that best describe the original *n* terms.
  - Uncover latent information (synonyms)

Explicit Semantic Analysis

- Expand the terms using concept space (e.g. Wikipedia)
  - Bag of Words

{ American politics } →

Democrats, Republicans, abortion, taxes, homosexuality, guns, etc

  - Explicit Semantic Analysis

{ Car } →

Wikipedia:Car, Wikipedia:Automobile, Wikipedia:BMW, Wikipedia:Railway, etc

More Discussions on Feature Selection and Generation

- Finding appropriate feature set for specific categorization task
  - Granularity
    - Chinese character / Chinese Word
    - Punctuation
    - Pattern
  - Specific task
    - "The *Dream* of the Red Chamber" Cao Xueqin or Gao Er
      - Use stopwords
    - Sentiment Analysis
      - Sentiment oriented words
      - Sentiment words

Rule-based Approach to Text Categorization

- Text in a Web Page

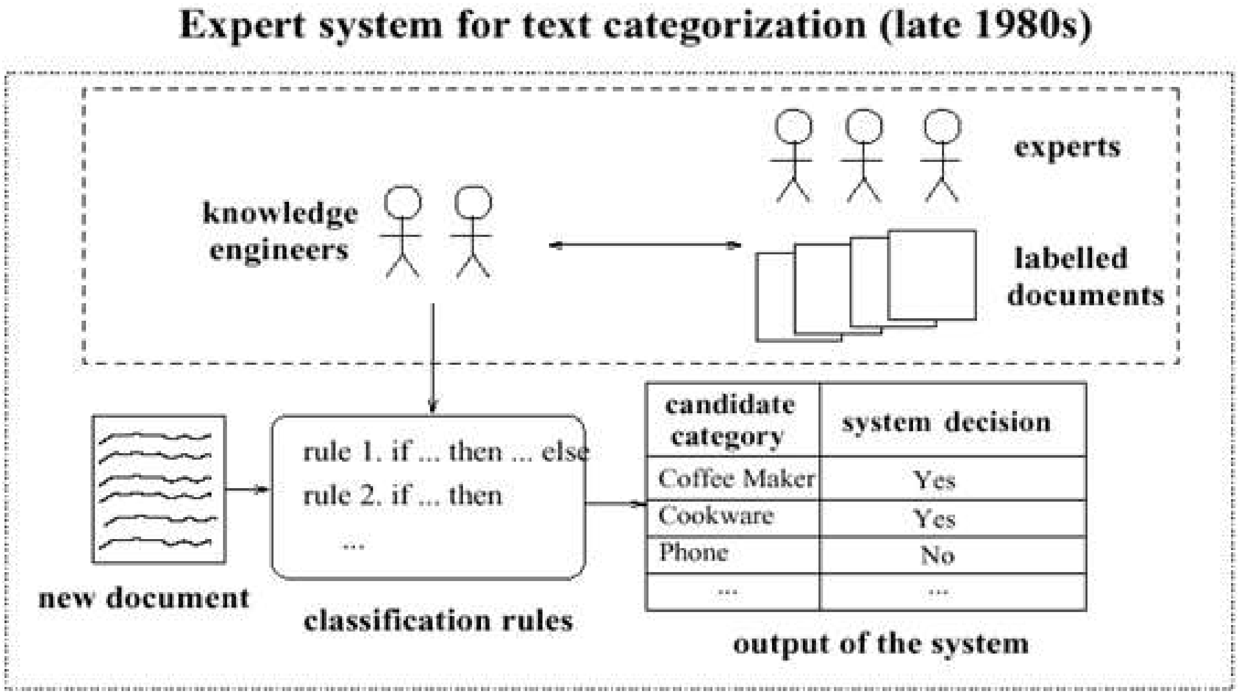
  "Saeco revolutionized *espresso* brewing a decade ago by introducing Saeco SuperAutomatic *machines*, which go from bean to *coffee* at the touch of a button. The all-new Saeco Vienna Super-Automatic home coffee and *cappucino machine* combines top quality with low price!"

- Rules
  - Rule 1.

    (*espresso* **or** *coffee* **or** *cappucino*) **and** *machine** → *Coffee Maker*
  - Rule 2.

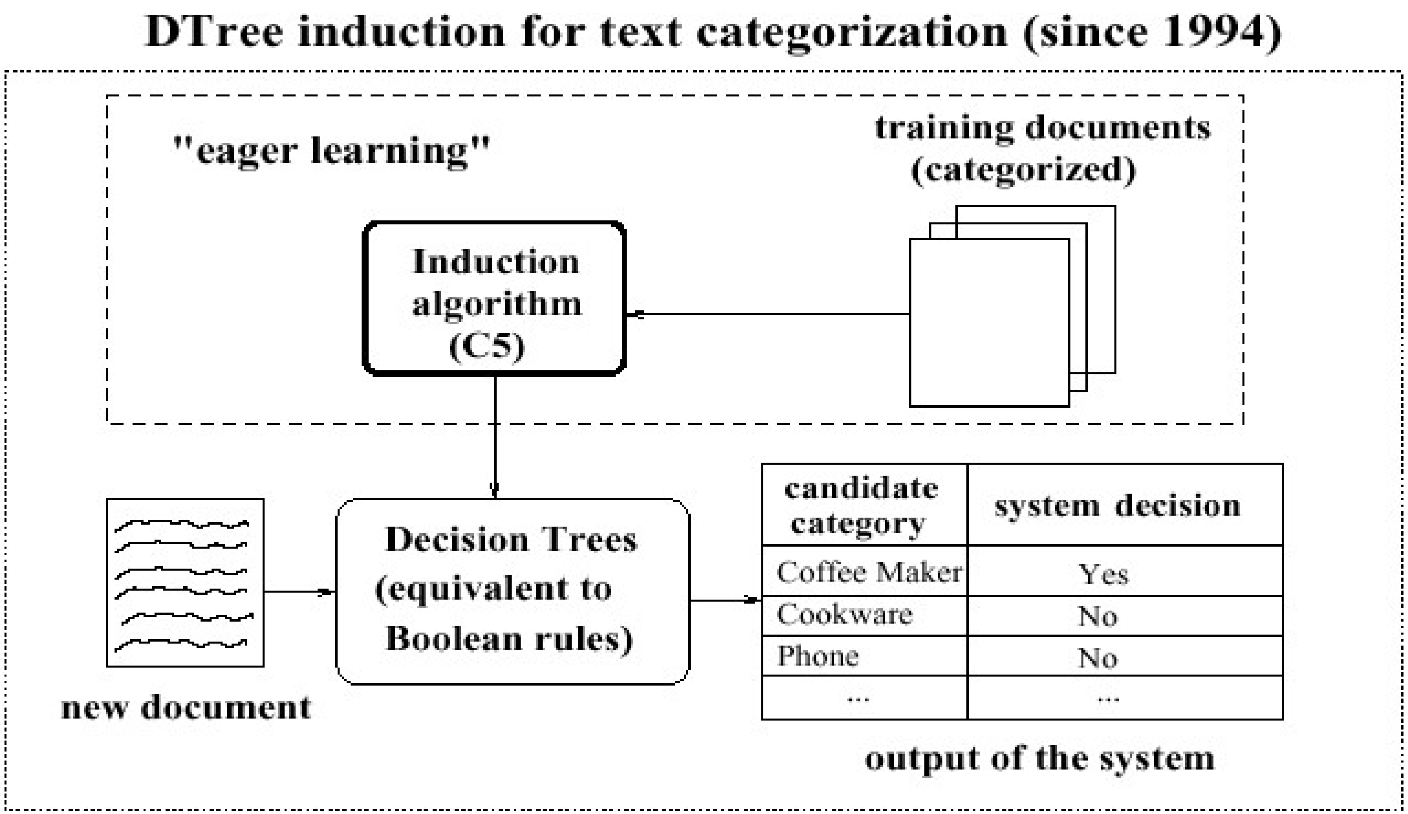
    *automat** **and** *answering* **and** *machine** → *Phone*
  - Rule ...

Expert System for Text Categorization (late 1980s)



- Experience has shown
  - too time consuming
  - too difficult
  - inconsistency issues (as the rule set gets large)

Replace Knowledge Engineering with a Statistical Learner



Knowledge Engineering vs. Statistical Learning

- For US Census Bureau Decennial Census 1990
  – 232 industry categories and 504 occupation categories
  – $15 million if fully done by hand
- Define classification rules manually:
  – Expert System AIOCS
  – Development time: 192 person-months (2 people, 8 years)
  – Accuracy = 47%
- Learn classification function
  – Nearest Neighbor classification (Creecy ’92: 1-NN)
  – Development time: 4 person-months (Thinking Machine)
  – Accuracy = 60%

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

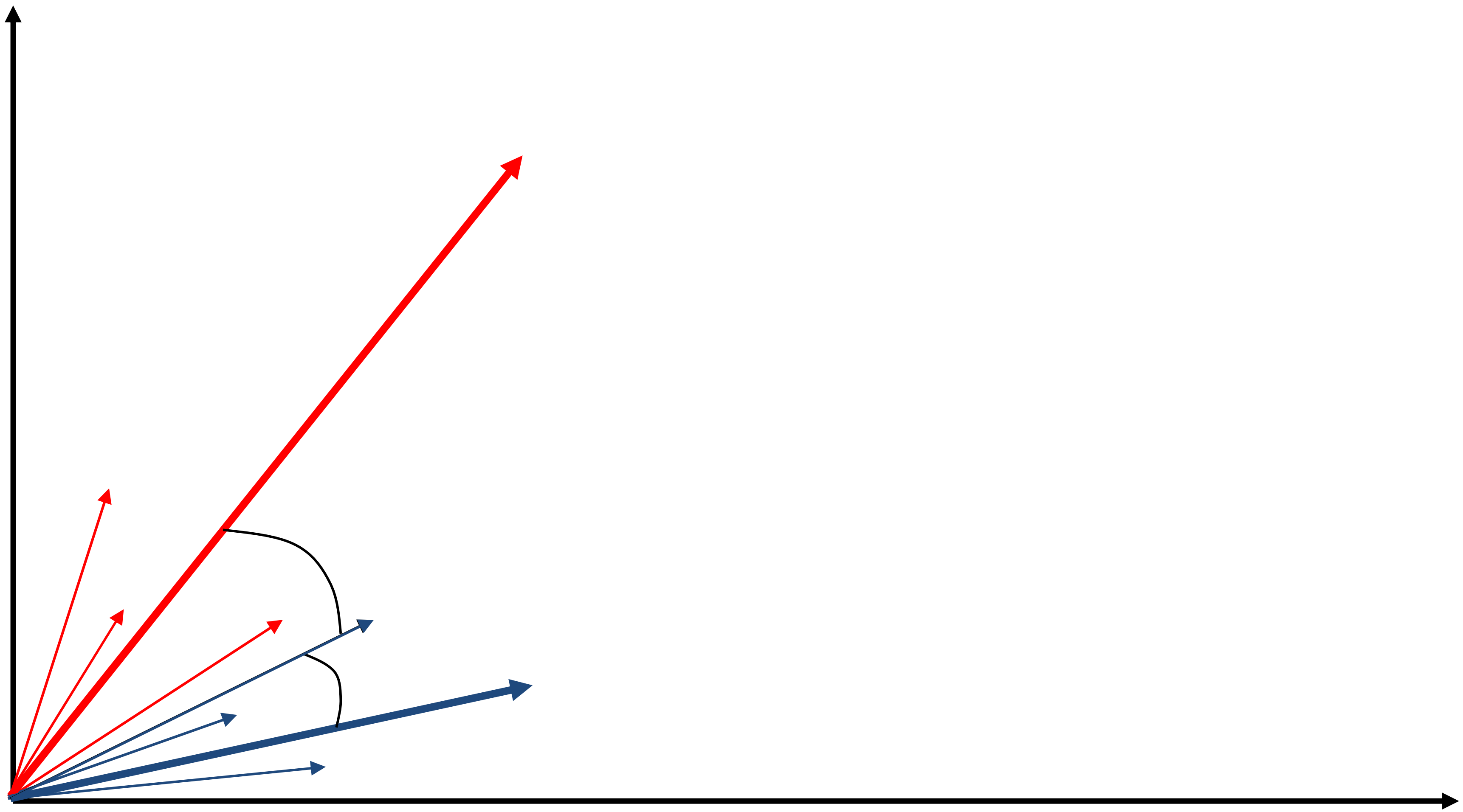
Rocchio Text Categorization Algorithm (Training)

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

Rocchio Text Categorization Algorithm (Test)

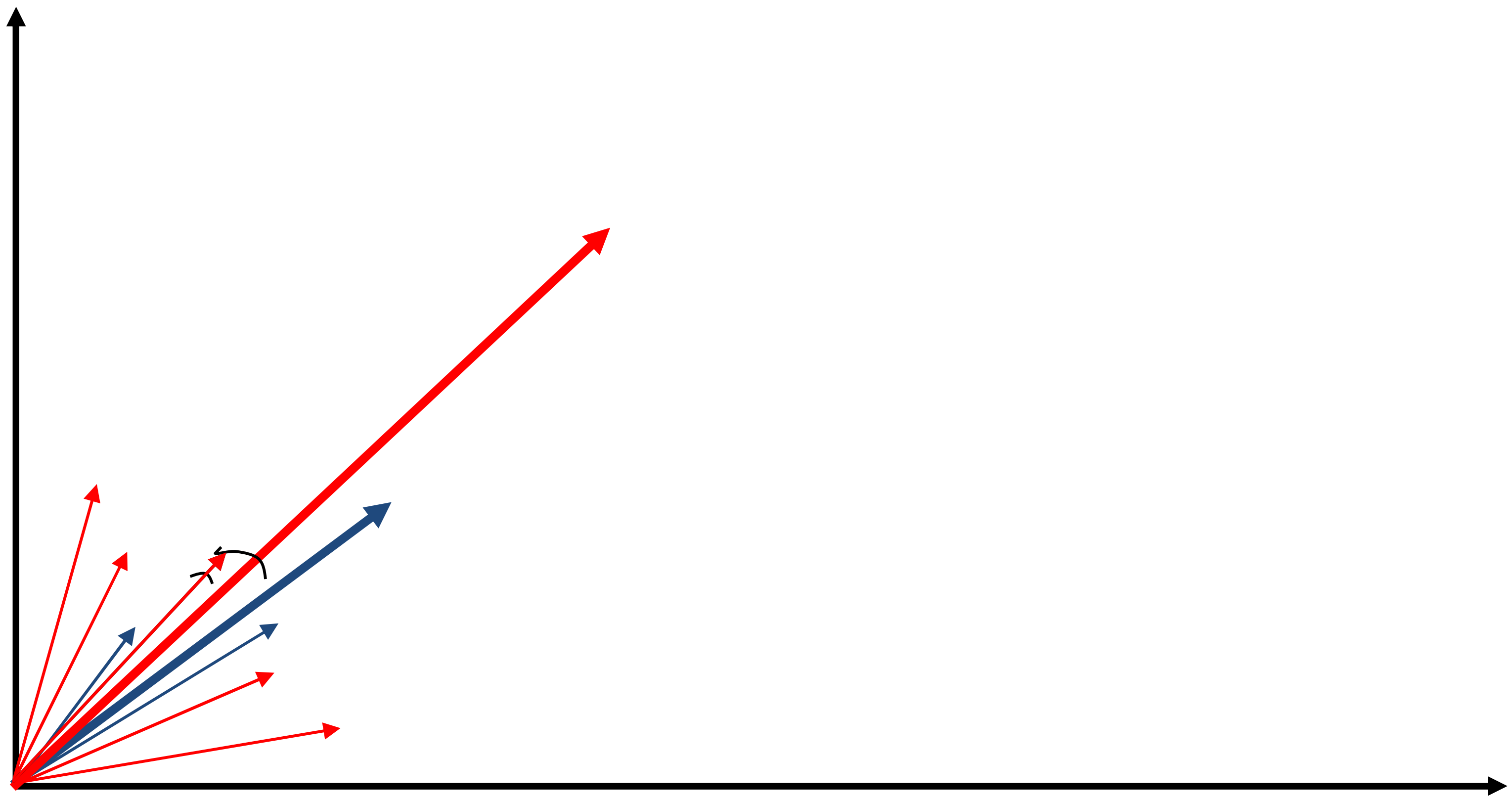
Given test document *x*
Let **d** be the TF/IDF weighted term vector for *x*
Let $m = -2$ (*init. maximum cosSim*)
For *i* from 1 to *n*:
    (*compute similarity to prototype vector*)
    Let $s = \text{cosSim}(\mathbf{d}, \mathbf{p}_i)$
    if $s > m$
        let $m = s$
        let $r = c_i$ (*update most similar class prototype*)
Return class *r*

Illustration of Rocchio Text Categorization



Rocchio Anomoly

- Prototype models have problems with polymorphic (disjunctive) categories



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.

Bayesian Methods

- Learning and classification methods based on probability theory (see spelling / POS)
- Bayes theorem plays a critical role
- Build a *generative model* that approximates how data is produced
- 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.

Bayesian Categorization

- Determine category of x_k by determining for each y_i

$$P(Y = y_i \mid X = x_k) = \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)}$$

- P(X=x_k) can be determined since categories are complete and disjoint.

$$\sum_{i=1}^{m} P(Y = y_i \mid X = x_k) = \sum_{i=1}^{m} \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)} = 1$$

$$P(X = x_k) = \sum_{i=1}^{m} P(Y = y_i)P(X = x_k \mid Y = y_i)$$

- Need to know:
  - Priors: $P(Y=y_i)$
  - Conditionals: $P(X=x_k \mid Y=y_i)$
- $P(Y=y_i)$ are easily estimated from data.
  - If n_i of the examples in D are in y_i then $P(Y=y_i) = n_i / |D|$
- Too many possible instances (e.g. 2^n for binary features) to estimate all $P(X=x_k \mid Y=y_i)$.
- Still need to make some sort of independence assumptions about the features to make learning tractable.

Generative Probabilistic Models

- Assume a simple (usually unrealistic) probabilistic method by which the data was generated.
- For categorization, each category has a different parameterized generative model that characterizes that category.
- **Training**: Use the data for each category to estimate the parameters of the generative model for that category.
  - **Maximum Likelihood Estimation (MLE)**: Set parameters to maximize the probability that the model produced the given training data.
  - If M_λ denotes a model with parameter values λ and D_k is the training data for the kth class, find model parameters for class k (λ_k) that maximize the likelihood of D_k:

$$\lambda_k = \underset{\lambda}{\operatorname{argmax}}\, P(D_k \mid M_\lambda)$$

- **Testing**: Use Bayesian analysis to determine the category model that most likely generated a specific test instance.

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

    Let $P(w_j \mid c_i) = (n_{ij} + 1) / (n_i + |V|)$

Text Naïve Bayes Algorithm (Test)

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

$$\underset{c_i \in C}{\operatorname{argmax}} P(c_i) \prod_{i=1}^{n} P(a_i \mid c_i)$$

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

Naive Bayes Is Not So Naive

- Naïve Bayes: First and Second place in KDD-CUP 97 competition, among 16 (then) state of the art algorithms
  - Robust to Irrelevant Features
  - Irrelevant Features cancel each other without affecting results
  - Instead Decision Trees & Nearest-Neighbor methods can heavily suffer from this.
- Very good in Domains with many <u>equally important</u> features
  - Decision Trees suffer from *fragmentation* in such cases – especially if little data
- A good dependable baseline for text classification (but not the best)!

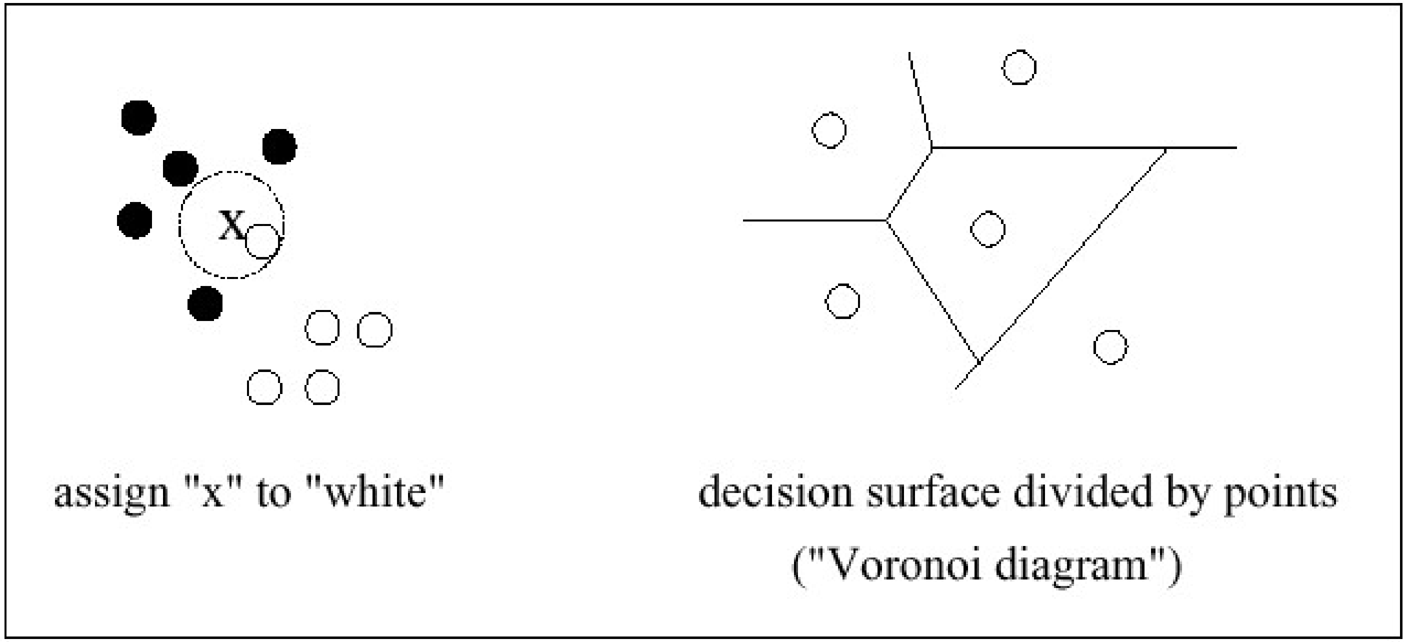
- Optimal if the Independence Assumptions hold:
  - If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
- Very Fast:
  - Learning with one pass over the data; testing linear in the number of attributes, and document collection size
- Low Storage requirements
- Handles Missing Values

Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in *D*.
- Initially by Fix and Hodges (1951)
- Theoretical error bound analysis by Duda & Hart (1957)
- 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

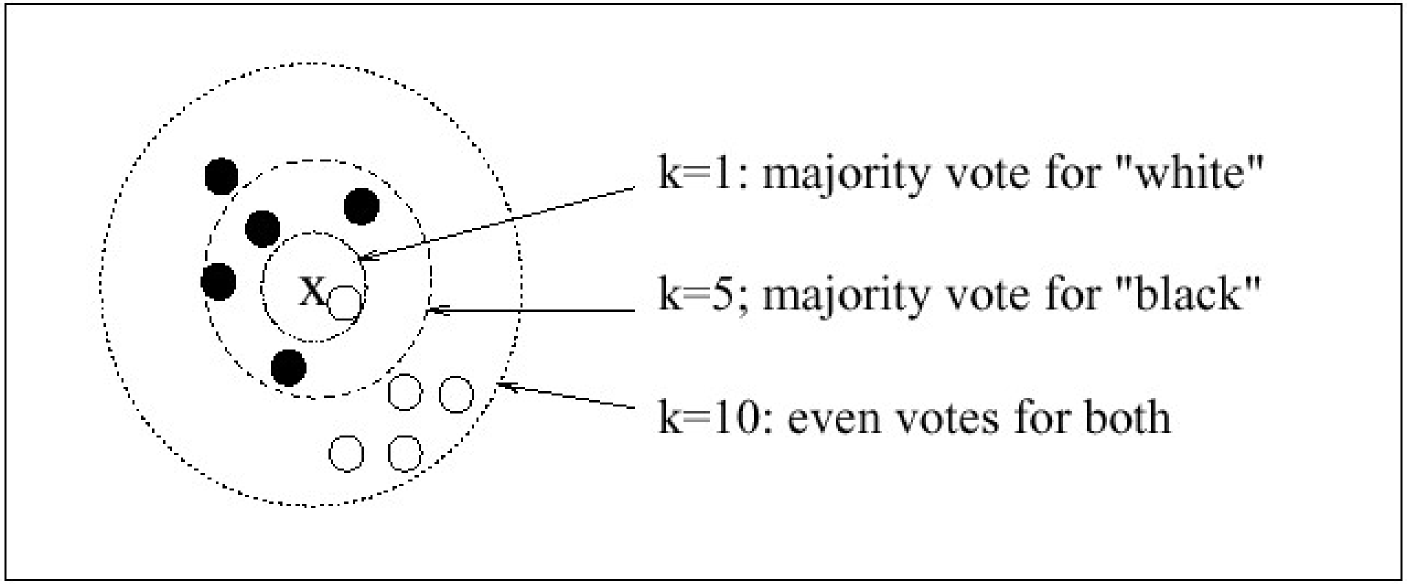
1-Nearest Neighbor

1-NN: assign "x" (new point) to the class of it nearest neighbor



K-Nearest Neighbor using a *majority* voting scheme

K-Nearest Neighbor using a *majority* voting scheme



Category Scoring for Weighted-Sum

- The score for a category is the sum of the similarity scores between the point to be classified and all of its k-neighbors that belong to the given category.

- To restate: $$score(c \mid x) = \sum_{d \in kNN\ of\ x} sim(x, d)\, I(d, c)$$

  where *x* is the new point; *c* is a class (*e.g. black* or *white);*
  d is a classified point among the k-nearest neighbors of *x*;
  sim(x,d) is the similarity between *x* and *d;*
  I(d,c) = 1 iff point *d* belongs to class *c*;
  I(d,c) = 0 otherwise.

kNN for Text Categorization (Yang, SIGIR-1994)

- Represent documents as points (vectors).
- Define a similarity measure for pairwise documents.
- Tune parameter *k* for optimizing classification effectiveness.
- Choose a voting scheme (e.g., weighted sum) for scoring categories
- Threshold on the scores for classification decisions.

K Nearest Neighbor for Text

Training:

For each each training example $<x, c(x)> \in D$

Compute the corresponding TF-IDF vector, $\mathbf{d}_x$, for document x

Test instance *y*:

Compute TF-IDF vector $\mathbf{d}$ for document y

For each $<x, c(x)> \in D$

Let $s_x = \text{cosSim}(\mathbf{d}, \mathbf{d}_x)$

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

Thresholding for Classification Decisions

- Alternative thresholding strategies:
  - Rcut: For each document to be categorized, rank candidate categories by score, and assign YES to the top-*m* categories (where *m* is some fixed number).
  - Pcut: Applies only when we have a whole batch of documents to be categorized. Make the category assignments proportional to the category distribution in the training set (i.e. if 1/4th of the training documents were in the category "Coffee Maker" then we will assign 1/4th of the documents in this batch to the "Coffee Maker" category).
  - Scut: For each category, choose a threshold score (empirically). Any document with a category score that surpasses its respective threshold will be predicted to be a member of that category.

Similarity Measures

- Cosine similarity

$$\cos(\vec{x}, \vec{y}) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \times \sqrt{\sum_i y_i^2}}$$

- 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)
- Kullback-Leibler distance (distance between two probability distributions)
- For text, cosine similarity of TF-IDF weighted vectors is typically most effective

Key Components in kNN

- Functional definition of "similarity"
  - e.g. cos, Euclidean, kernel functions, …
- How many neighbors do we consider?
  - Value of *k* determined *empirically* (see methodology section)
- Does each neighbor get the same weight?
  - Weighted-sum or not
- All categories in neighborhood? Most frequent only? How do we make the final decision?
  - Rcut, Pcut, or Scut

Pros of kNN

- Simple and effective (among top-5 in benchmark evaluations)
  - Non-linear classifier (vs linear)
  - Local estimation (vs global)
  - Non-parametric (very few assumptions about data)
  - Reasonable similarity measures (borrowed from IR)
- Computation (time & space) linear to the size of training data
- Low cost for frequent re-training, i.e., when categories and training documents need to be updated (common in Web environment and e-commerce applications)

Cons of kNN

- Online response is typically slower than *eager learning* algorithms
  - Trade-off between off-line training cost and online search cost
- Scores are not normalized (probabilities)
  - Comparing directly to and combining with scores of other classifiers is an open problem
- Output not good in explaining why a category is relevant
  - Compared to DTree, for example (take this with a grain of salt).

SVM and Hyperplane

- A learning algorithm for classification
  - General for any classification problem (text classification as one example)
- invented by V. Vapnik and his co-workers in 1970s in Russia and became known to the West in 1992
- Binary classification
- Maximizes the margin between the two different classes
- Rigorous theoretical foundation,
  - More accurately than most other methods in applications, especially for high dimensional data.
  - It is perhaps the best classifier for text classification.

SVM and Hyperplane: Basic concepts

- Let the set of training examples *D* be

  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$,

  where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n)$ is an **input vector** in a real-valued space $X \subseteq R^n$ and y_i is its **class label** (output value), $y_i \in \{1, -1\}$.

  1: positive class and -1: negative class.
- SVM finds a linear function of the form (**w**: weight vector, b: bias)

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

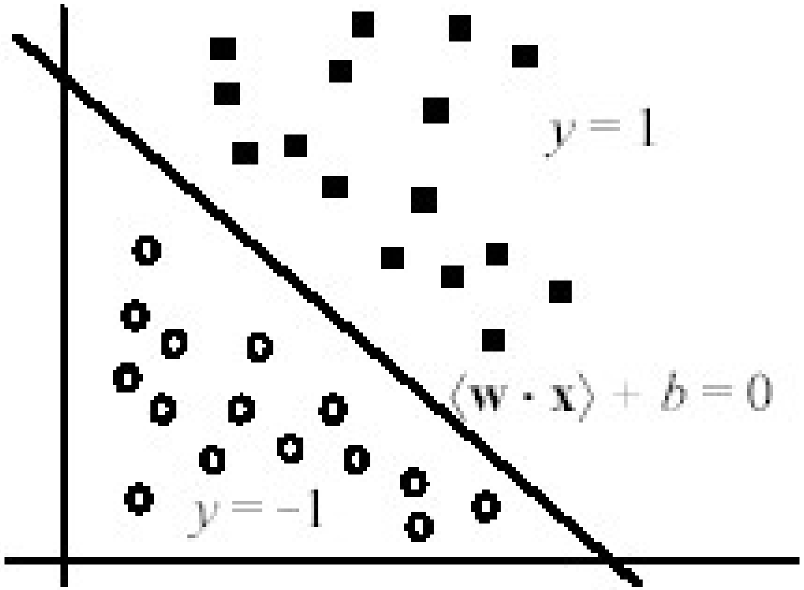
$$y_i = \begin{cases} 1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

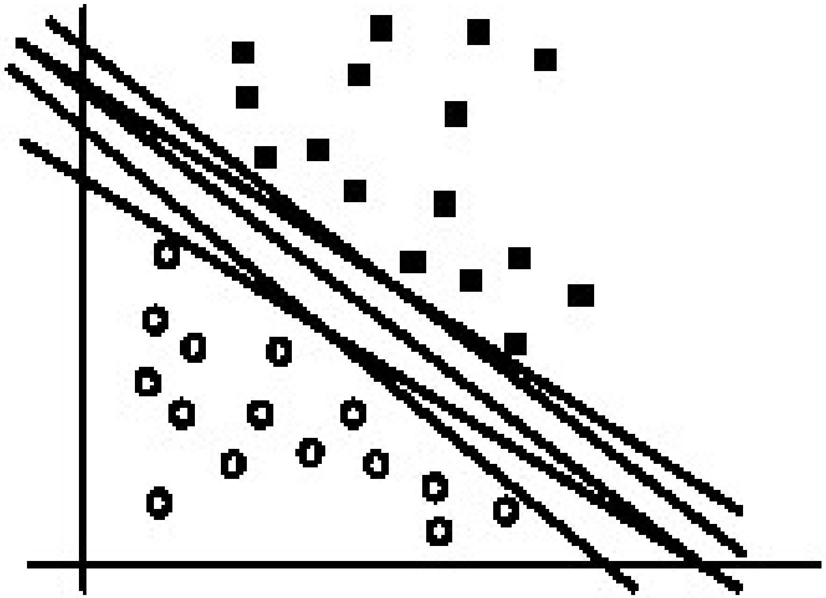
SVM and Hyperplane

- The hyperplane that separates positive and negative training data is

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$$

- It is also called the decision boundary (surface).
- So many possible hyperplanes, which one to choose?

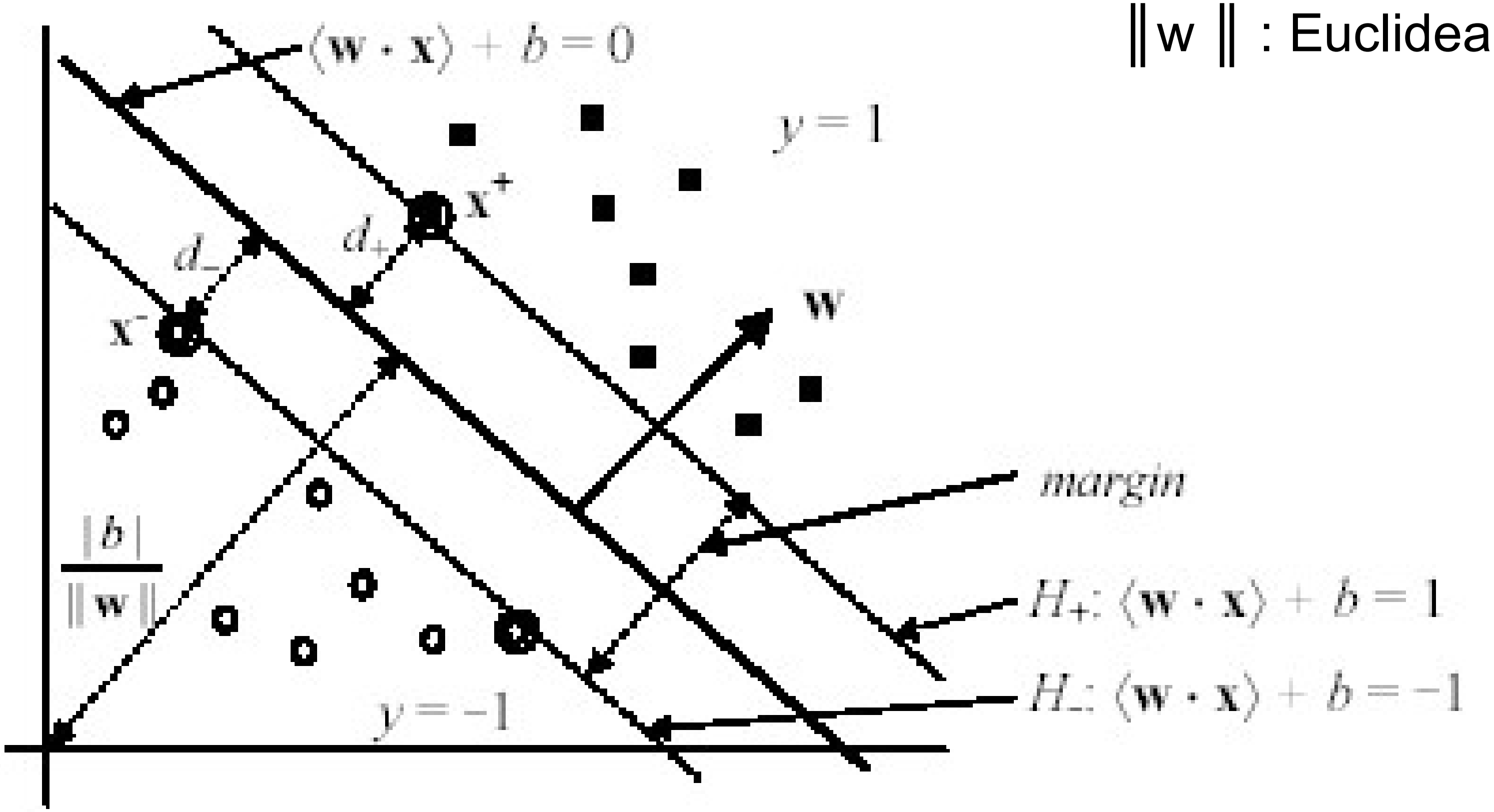




SVM and Hyperplane: Maximal margin hyperplane

- SVM looks for the separating hyperplane with the largest margin.
- Machine learning theory says this hyperplane minimizes the error bound

|B|: scalar value of bias
‖w‖ : Euclidean norm

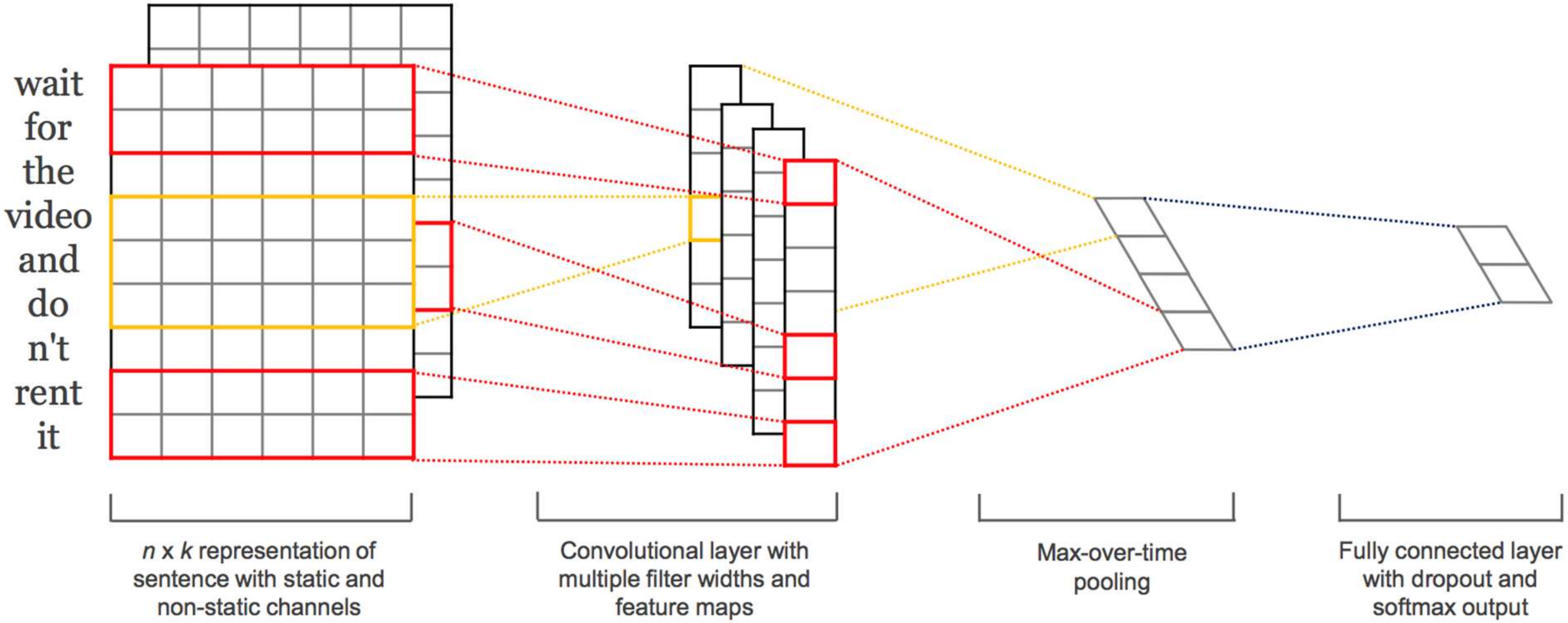


More Approaches to Text Categorization

- Regression based on Least Squares Fit (1991)
- Nearest Neighbor Classification (1992)
- Bayesian Probabilistic Models (1992)
- Symbolic Rule Induction (1994)
- Neural Networks (1995)
- Rocchio approach (traditional IR, 1996)
- Support Vector Machines (1997)
- Boosting or Bagging (1997)
- Hierarchical Language Modeling (1998)
- First-Order-Logic Rule Induction (1999)
- Maximum Entropy (1999)
- Hidden Markov Models (1999)
- Error-Correcting Output Coding (1999)

CNN for Text Categorization (Kim EMNLP-2014)

The model architecture for text categorization is shown below:



Embedding layer

Let $x_i \in R^k$ be the k-dimensional word vector corresponding to the i-th word in the sentence. A sentence of length n (padded where necessary) is represented as:

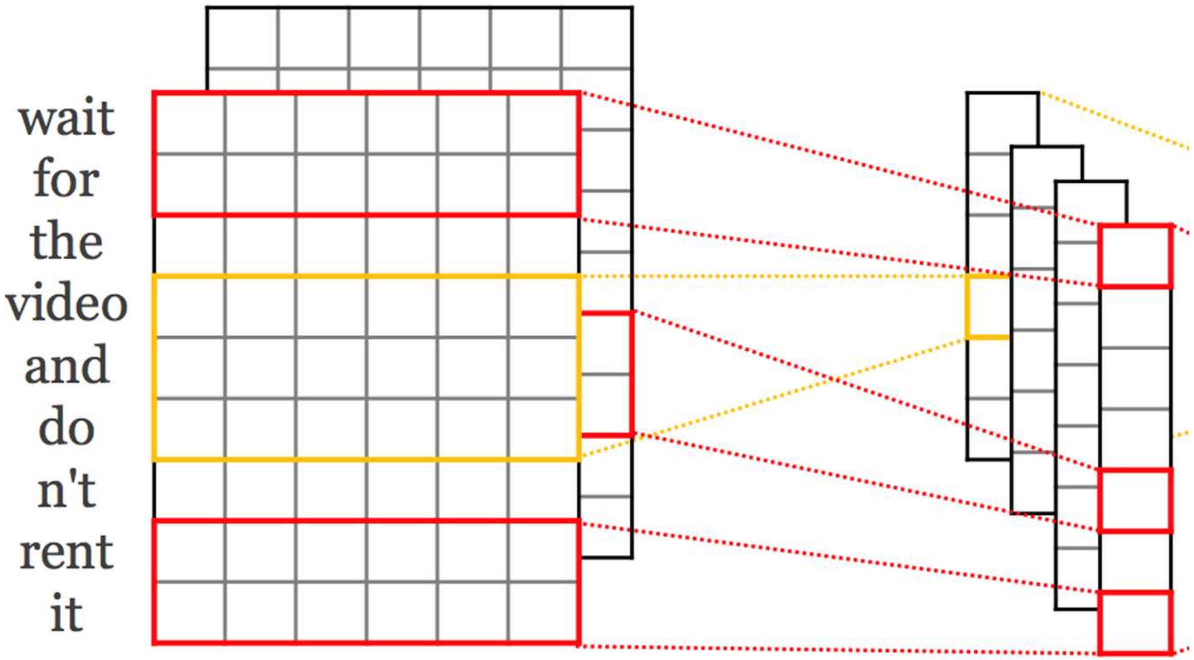
$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n$$

wait	0.99	0.05	0.01	0.80	0.24	-0.1	0.07	-0.3	……
for	0.18	0.23	-0.2	0.92	0.17	0.88	0.43	0.15	……
⋮									
it	0.56	0.08	0.16	-0.5	0.17	0.98	0.32	-0.8	……

Convolutional layer

A convolution operation involves a filter $w \in R^{hk}$, which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by：

$$c_i = f(w \cdot x_{i:i+h-1} + b)$$



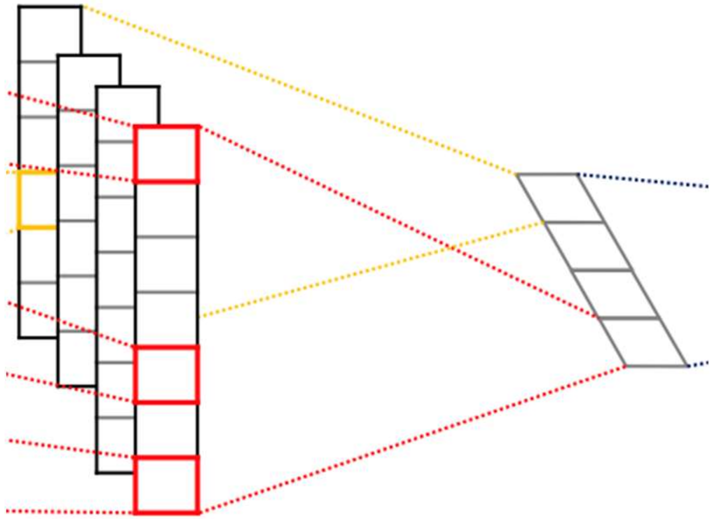
Pooling layer

This filter is applied to each possible window of words in the sentence $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$ to produce a feature map：

$$c = [c_1, c_2, \cdots, c_{n-h+1}]$$

Then apply a max-overtime pooling operation over the feature map and take the maximum value：

$$\hat{c} = \max\{c\}$$



Regularization

Given the penultimate layer $z = [\hat{c}_1, \ldots, \hat{c}_m]$, for output unit *y* in forward propagation, dropout uses:

$$y = w \cdot (z \circ r) + b$$

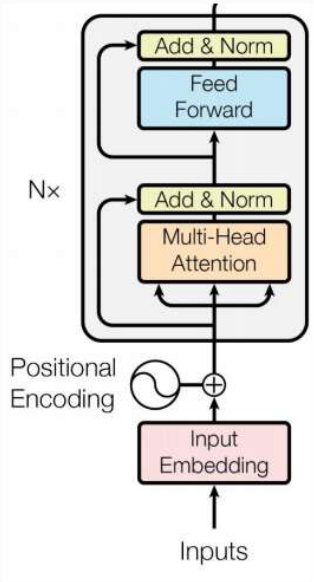
At test time, the learned weight vectors are scaled by p such that $\hat{w} = pw$, and $\hat{w}$ is used (without dropout) to score unseen sentences.

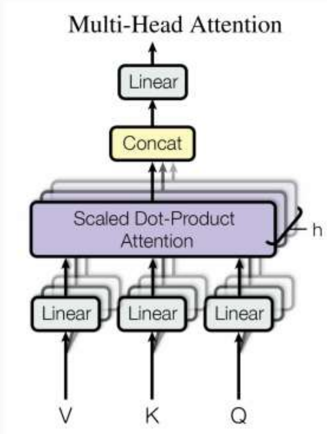
Additionally constrain l_2-norms of the weight vectors by rescaling w to have $||w||_2 = s$ whenever $||w||_2 > s$ after a gradient descent step.

Bert: Pre-training of deep bidirectional transformers for language understanding (*Devlin et al., HLT-NAACL 2018*)

Transformer

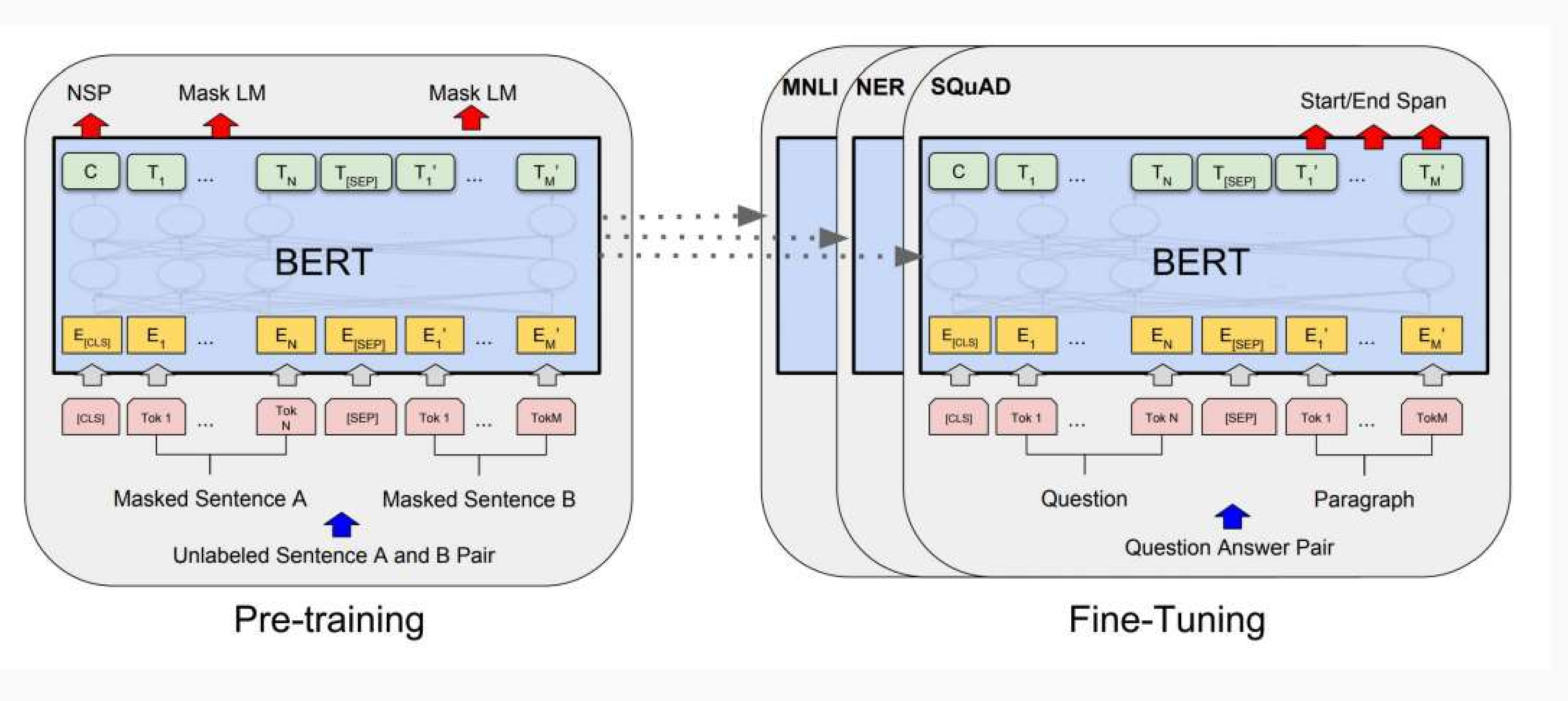
- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning





Bert: Pre-training of deep bidirectional transformers for language understanding (*Devlin et al., HLT-NAACL 2018*)

- Pretraining : Masked LM + Next Sentence Prediction
- Finetuning : Other NLP tasks



How to Fine-Tune BERT for Text Classification? (*Sun et al., CCL 2018*)

- The top layer of BERT is more useful for text classification
- With an appropriate layer-wise decreasing learning rate, BERT can overcome the catastrophic forgetting problem
- Within-task and in-domain further pre-training can significantly boost its performance
- A preceding multi-task fine-tuning is also helpful to the single-task fine-tuning, but its benefit is smaller than further pre-training
- BERT can improve the task with small-size data

RoBERTa: A Robustly Optimized BERT Pretraining Approach (*Liu et al., ICLR 2019*)

Improved masking and pre-training data slightly

Trained BERT for more epochs and/or on more data

- Showed that more epochs alone helps, even on same data
- More data also helps

Voting

- Bagging
  - Train *K* classifiers using one classification method using *K* different training sets
  - Run the K classifiers on test sample
  - Assign the test sample the label that has most votes
- Boosting
  - Train the *K* classifier in such a serial mode that the (*i+1*)-th classifier can correctly classify the those training samples that cannot be correctly classified by the *i*-th classifier
  - AdaBoost

Evaluation Metrics

	Correct=Y	Correct=N
Assigned=Y	a	b
Assigned=N	c	d

- Accuracy = (a+d)/(a+b+c+d)
- Precision = a/(a+b)
- Recall= a/(a+c)
- F-Measure = 2*Precision*Recall/(Precision+Recall)
- Micro/Macro Averaging
- Breakeven (When Precision=Recall)

Evaluation Metrics:

- How to combine P/R from 3 classes
- Macro-averaging:
  - compute the performance for each class, and then average over classes
- Micro-averaging:
  - collect decisions for all classes into one confusion matrix
  - compute precision and recall from that table.

Evaluation Metrics:
Macro-averaging and Micro-averaging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\begin{array}{c}\text{microaverage}\\\text{precision}\end{array} = \frac{268}{268+99} = \mathbf{.73}$$

$$\begin{array}{c}\text{macroaverage}\\\text{precision}\end{array} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

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, $N = 10$.

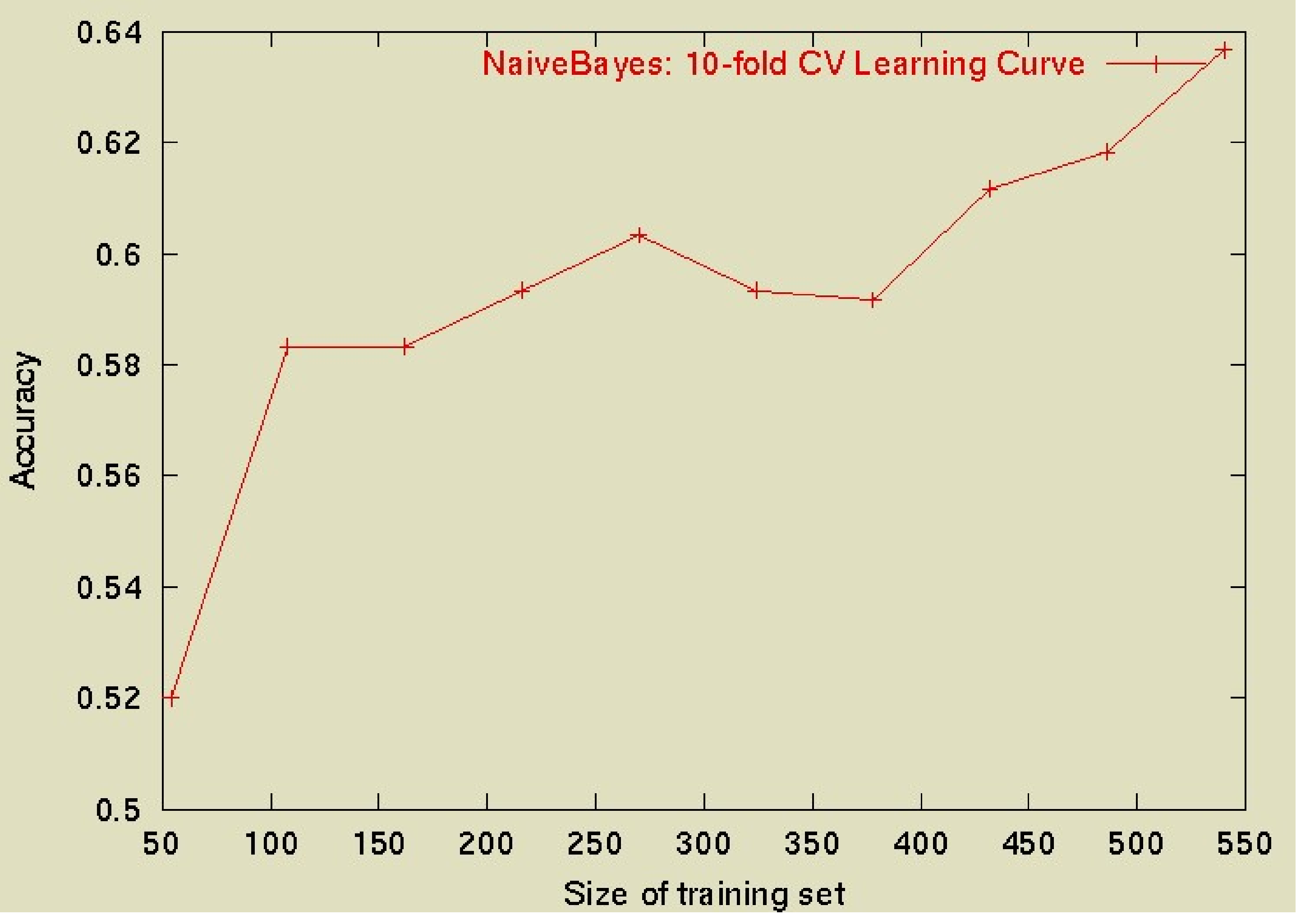
Learning Curves

- 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.

Sample Learning Curve



The Next Lecture

- Lecture 13

  Text Clustering