Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search

Pandu Nayak and Prabhakar Raghavan

Lecture 11: Text Classification;
Vector space classification

[Borrows slides from Ray Mooney]

Recap: Naïve Bayes classifiers

- Classify based on prior weight of class and conditional parameter for what each word says:

$$c_{NB} = \operatorname*{argmax}_{c_j \in C} \left[\log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j) \right]$$

- Training is done by counting and dividing:

$$P(c_j) \leftarrow \frac{N_{c_j}}{N} \qquad P(x_k | c_j) \leftarrow \frac{T_{c_j x_k} + \alpha}{\sum_{x_i \in V} [T_{c_j x_i} + \alpha]}$$

- Don't forget to smooth

The rest of text classification

- Today:
  - Vector space methods for Text Classification
    - Vector space classification using centroids (Rocchio)
    - K Nearest Neighbors
    - Decision boundaries, linear and nonlinear classifiers
    - Dealing with more than 2 classes
- Later in the course
  - More text classification
    - Support Vector Machines
    - Text-specific issues in classification

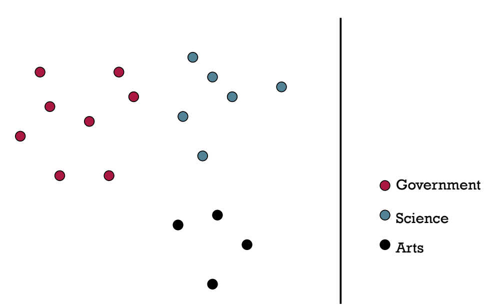
Recall: Vector Space Representation

- Each document is a vector, one component for each term (= word).
- Normally normalize vectors to unit length.
- High-dimensional vector space:
  - Terms are axes
  - 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space
- How can we do classification in this space?

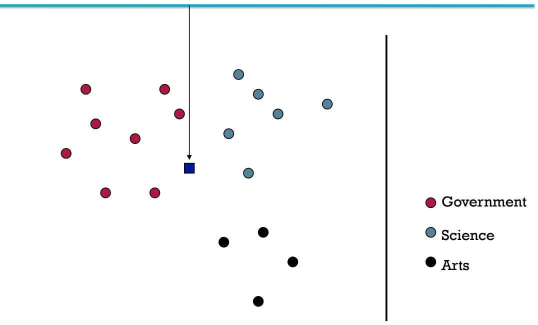
Classification Using Vector Spaces

- As before, the training set is a set of documents, each labeled with its class (e.g., topic)
- In vector space classification, this set corresponds to a labeled set of points (or, equivalently, vectors) in the vector space
- Premise 1: Documents in the same class form a contiguous region of space
- Premise 2: Documents from different classes don't overlap (much)
- We define surfaces to delineate classes in the space

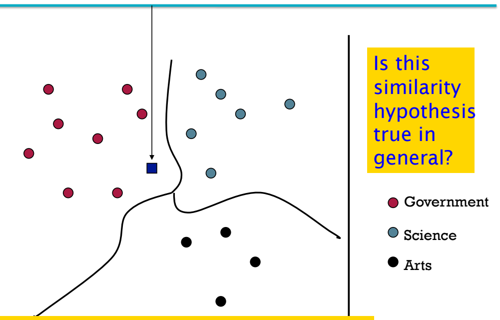
Documents in a Vector Space



Test Document of what class?



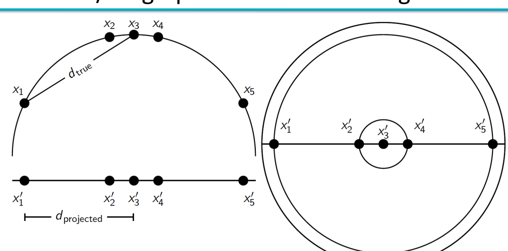
Test Document = Government



Is this similarity hypothesis true in general?

Our main topic today is how to find good separators

Aside: 2D/3D graphs can be misleading

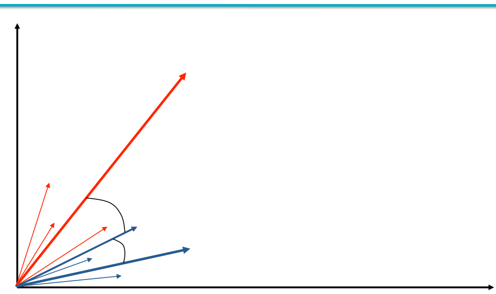


Left: A projection of the 2D semicircle to 1D. For the points x_1, x_2, x_3, x_4, x_5 at x coordinates $-0.9, -0.2, 0, 0.2, 0.9$ the distance $|x_2x_3| \approx 0.201$ only differs by 0.5% from $|x'_2x'_3| = 0.2$; but $|x_1x_3|/|x'_1x'_3| = d_{\text{true}}/d_{\text{projected}} \approx 1.06/0.9 \approx 1.18$ is an example of a large distortion (18%) when projecting a large area. *Right:* The corresponding projection of the 3D hemisphere to 2D.

Using Rocchio for text classification

- Relevance feedback methods can be adapted for text categorization
  - As noted before, relevance feedback can be viewed as 2-class classification
    - Relevant vs. nonrelevant documents
- Use standard tf-idf weighted vectors to represent text documents
- For training documents in each category, compute a prototype vector by summing the vectors of the training documents in the category.
  - Prototype = centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

Illustration of Rocchio Text Categorization



Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

- Where D_c is the set of all documents that belong to class *c* and *v*(*d*) is the vector space representation of *d.*
- *Note that centroid will in general* not *be a unit vector even when the inputs are unit vectors.*

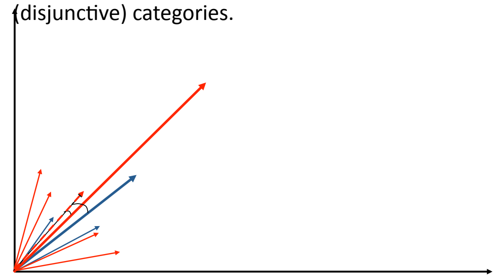
Rocchio Properties

- 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.
- Does not guarantee classifications are consistent with the given training data.

Why not?

Rocchio Anomaly

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



Rocchio classification

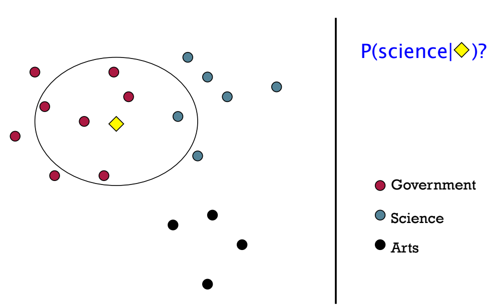
- Rocchio forms a simple representation for each class: the centroid/prototype
- Classification is based on similarity to / distance from the prototype/centroid
- It does not guarantee that classifications are consistent with the given training data
- It is little used outside text classification
  - It has been used quite effectively for text classification
  - But in general worse than Naïve Bayes
- Again, cheap to train and test documents

k Nearest Neighbor Classification

- kNN = k Nearest Neighbor

- To classify a document *d* into class c:
- Define *k*-neighborhood N as *k* nearest neighbors of *d*
- Count number of documents i in N that belong to c
- Estimate P(c|*d*) as i/k
- Choose as class argmax_c P(c|*d*) [= majority class]

Example: k=6 (6NN)



Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in *D*.
- Testing instance *x (under 1NN)*:
  - 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 learning
  - Memory-based learning
  - Lazy learning
- Rationale of kNN: contiguity hypothesis

kNN Is Close to Optimal

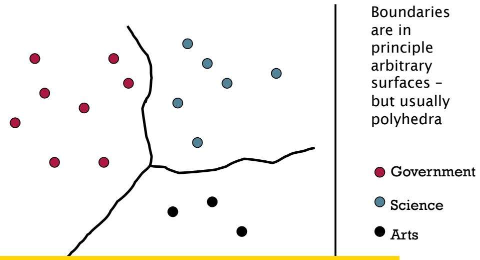
- Cover and Hart (1967)
- Asymptotically, the error rate of 1-nearest-neighbor classification is less than twice the Bayes rate [error rate of classifier knowing model that generated data]
- In particular, asymptotic error rate is 0 if Bayes rate is 0.
- Assume: query point coincides with a training point.
- Both query point and training point contribute error → 2 times Bayes rate

k Nearest Neighbor

- Using only the closest example (1NN) to determine the class is subject to errors due to:
  - A single atypical example.
  - Noise (i.e., an 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.

kNN decision boundaries

Boundaries are in principle arbitrary surfaces – but usually polyhedra

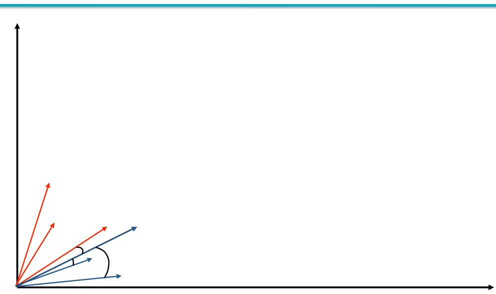


kNN gives locally defined decision boundaries between classes – far away points do not influence each classification decision (unlike in Naïve Bayes, Rocchio, etc.)

Similarity Metrics

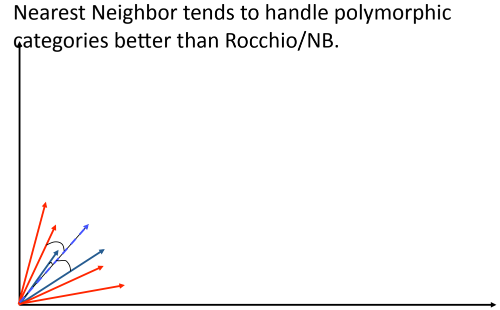
- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous *m*-dimensional instance space is *Euclidean distance*.
- Simplest for *m*-dimensional binary instance space is *Hamming distance* (number of feature values that differ).
- For text, cosine similarity of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

Illustration of 3 Nearest Neighbor for Text Vector Space



3 Nearest Neighbor vs. Rocchio

- Nearest Neighbor tends to handle polymorphic categories better than Rocchio/NB.



Nearest Neighbor with Inverted Index

- Naively, finding nearest neighbors requires a linear search through $|D|$ documents in collection
- But 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.
- Use standard vector space inverted index methods to find the *k* nearest neighbors.
- Testing Time: $O(B|V_t|)$ where *B* is the average number of training documents in which a test-document word appears.
  - Typically $B << |D|$

kNN: Discussion

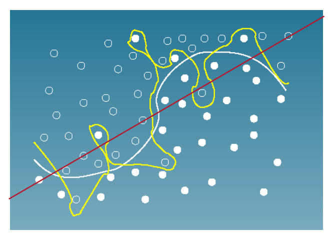
- No feature selection necessary
- Scales well with large number of classes
  - Don't need to train *n* classifiers for *n* classes
- Classes can influence each other
  - Small changes to one class can have ripple effect
- Scores can be hard to convert to probabilities
- No training necessary
  - Actually: perhaps not true. (Data editing, etc.)
- May be expensive at test time
- In most cases it's more accurate than NB or Rocchio

kNN vs. Naive Bayes

- Bias/Variance tradeoff
  - Variance ≈ Capacity
- kNN has high variance and low bias.
  - Infinite memory
- NB has low variance and high bias.
  - Decision surface has to be linear (hyperplane – see later)
- Consider asking a botanist: Is an object a tree?
  - Too much capacity/variance, low bias
    - Botanist who memorizes
    - Will always say "no" to new object (e.g., different # of leaves)
  - Not enough capacity/variance, high bias
    - Lazy botanist
    - Says "yes" if the object is green
  - You want the middle ground

(Example due to C. Burges)

Bias vs. variance: Choosing the correct model capacity

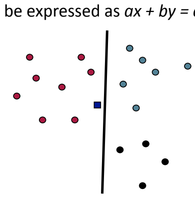


Linear classifiers and binary and multiclass classification

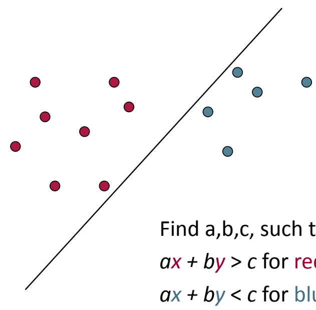
- Consider 2 class problems
  - Deciding between two classes, perhaps, government and non-government
    - One-versus-rest classification
- How do we define (and find) the separating surface?
- How do we decide which region a test doc is in?

Separation by Hyperplanes

- A strong high-bias assumption is *linear separability*:
  - in 2 dimensions, can separate classes by a line
  - in higher dimensions, need hyperplanes
- Can find separating hyperplane by *linear programming* (or can iteratively fit solution via perceptron):
  - separator can be expressed as $ax + by = c$



Linear programming / Perceptron

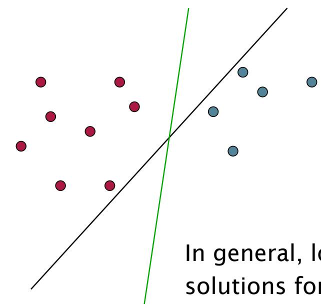


Find a,b,c, such that

$ax + by > c$ for red points

$ax + by < c$ for blue points.

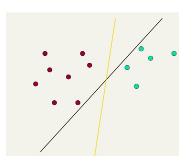
Which Hyperplane?



In general, lots of possible solutions for a,b,c.

Which Hyperplane?

- Lots of possible solutions for *a,b,c.*
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  - E.g., perceptron
- Most methods find an optimal separating hyperplane
- Which points should influence optimality?
  - All points
    - Linear/logistic regression
    - Naïve Bayes
  - Only “difficult points” close to decision boundary
    - Support vector machines



Linear classifier: Example

- Class: “interest” (as in interest rate)
- Example features of a linear classifier

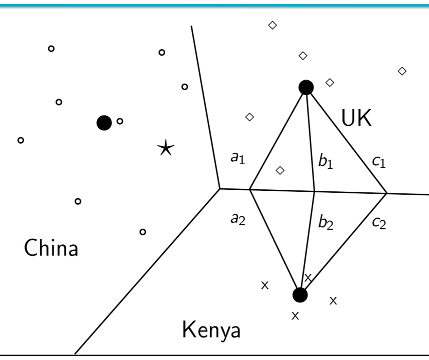
w_i t_i	w_i t_i
0.70 prime	−0.71 dlrs
0.67 rate	−0.35 world
0.63 interest	−0.33 sees
0.60 rates	−0.25 year
0.46 discount	−0.24 group
0.43 bundesbank	−0.24 dlr

- To classify, find dot product of feature vector and weights

Linear Classifiers

- Many common text classifiers are linear classifiers
  - Naïve Bayes
  - Perceptron
  - Rocchio
  - Logistic regression
  - Support vector machines (with linear kernel)
  - Linear regression with threshold
- Despite this similarity, noticeable performance differences
  - For separable problems, there is an infinite number of separating hyperplanes. Which one do you choose?
  - What to do for non-separable problems?
  - Different training methods pick different hyperplanes
- Classifiers more powerful than linear often don’t perform better on text problems. Why?

Rocchio is a linear classifier



Two-class Rocchio as a linear classifier

- Line or hyperplane defined by:

$$\sum_{i=1}^{M} w_i d_i = \theta$$

- For Rocchio, set:

$$\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$$
$$\theta = 0.5 \times (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$$

[Aside for ML/stats people: Rocchio classification is a simplification of the classic Fisher Linear Discriminant where you don't model the variance (or assume it is spherical).]

Naive Bayes is a linear classifier

- Two-class Naive Bayes. We compute:

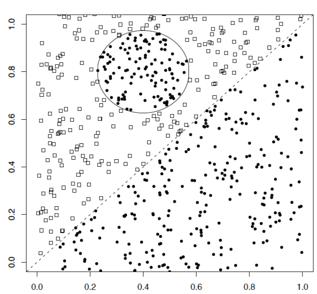
$$\log \frac{P(C|d)}{P(\overline{C}|d)} = \log \frac{P(C)}{P(\overline{C})} + \sum_{w \in d} \log \frac{P(w|C)}{P(w|\overline{C})}$$

- Decide class *C* if the odds is greater than 1, i.e., if the log odds is greater than 0.
- So decision boundary is hyperplane:

$$\alpha + \sum_{w \in V} \beta_w \times n_w = 0 \quad \text{where } \alpha = \log \frac{P(C)}{P(\overline{C})};$$

$$\beta_w = \log \frac{P(w|C)}{P(w|\overline{C})}; \quad n_w = \text{\# of occurrences of } w \text{ in } d$$

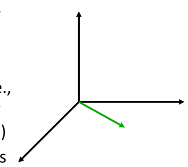
A nonlinear problem



- A linear classifier like Naïve Bayes does badly on this task
- kNN will do very well (assuming enough training data)

High Dimensional Data

- Pictures like the one at right are absolutely misleading!
- Documents are zero along almost all axes
- Most document pairs are very far apart (i.e., not strictly orthogonal, but only share very common words and a few scattered others)
- In classification terms: often document sets are separable, for most any classification
- This is part of why linear classifiers are quite successful in this domain



More Than Two Classes

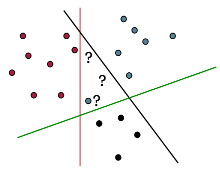
- Any-of or multivalue classification
  - Classes are independent of each other.
  - A document can belong to 0, 1, or >1 classes.
  - Decompose into *n* binary problems
  - Quite common for documents
- One-of or multinomial or polytomous classification
  - Classes are mutually exclusive.
  - Each document belongs to exactly one class
  - E.g., digit recognition is polytomous classification
    - Digits are mutually exclusive

Set of Binary Classifiers: Any of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Apply decision criterion of classifiers independently
- Done
  - Though maybe you could do better by considering dependencies between categories

Set of Binary Classifiers: One of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Assign document to class with:
  - maximum score
  - maximum confidence
  - maximum probability



Summary: Representation of Text Categorization Attributes

- Representations of text are usually very high dimensional (one feature for each word)
- High-bias algorithms that prevent overfitting in high-dimensional space should generally work best*
- For most text categorization tasks, there are many relevant features and many irrelevant ones
- Methods that combine evidence from many or all features (e.g. naive Bayes, kNN) often tend to work better than ones that try to isolate just a few relevant features*

*Although the results are a bit more mixed than often thought

Which classifier do I use for a given text classification problem?

- Is there a learning method that is optimal for all text classification problems?
- No, because there is a tradeoff between bias and variance.
- Factors to take into account:
  - How much training data is available?
  - How simple/complex is the problem? (linear vs. nonlinear decision boundary)
  - How noisy is the data?
  - How stable is the problem over time?
    - For an unstable problem, it's better to use a simple and robust classifier.

Resources for today's lecture

- *IIR* 14
- Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1-47, 2002.
- Yiming Yang & Xin Liu, A re-examination of text categorization methods. *Proceedings of SIGIR*, 1999.
- Trevor Hastie, Robert Tibshirani and Jerome Friedman, *Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer-Verlag, New York.
- Open Calais: Automatic Semantic Tagging
  - Free (but they can keep your data), provided by Thompson/Reuters
- Weka: A data mining software package that includes an implementation of many ML algorithms