Vector Space Classification

Cornelia Caragea

Acknowledgments: Manning, Rai

Department of Computer Science
University of Illinois at Chicago

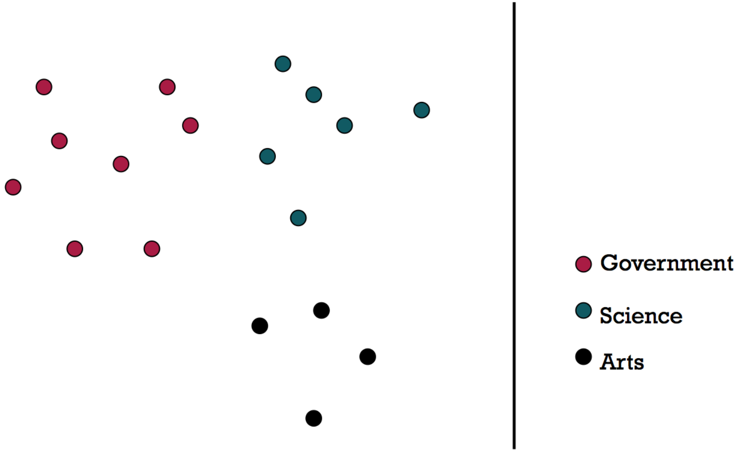
Required Reading

- "Information Retrieval" textbook
  - Chapter 14: Vector Space Classification

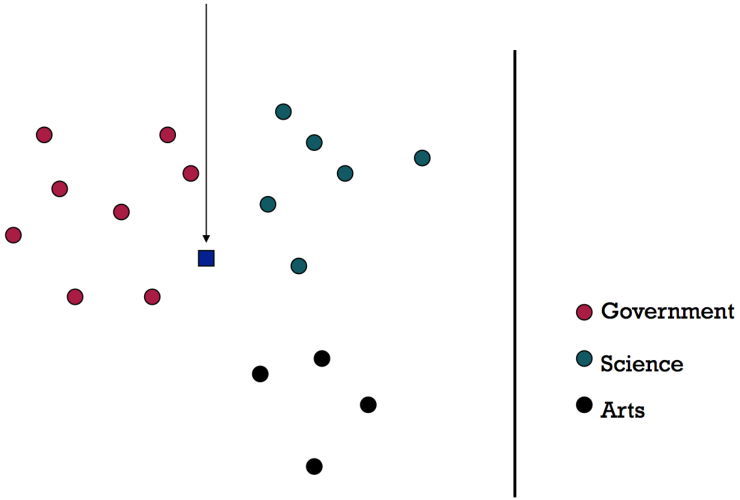
Classification Using Vector Spaces

- In vector space classification, the training set corresponds to a labeled set of points (equivalently, vectors)
- **Premise 1:** Documents in the same class form a contiguous region of space
- **Premise 2:** Documents from different classes do not overlap (much)
- Learning a classifier: build surfaces to delineate classes in the space

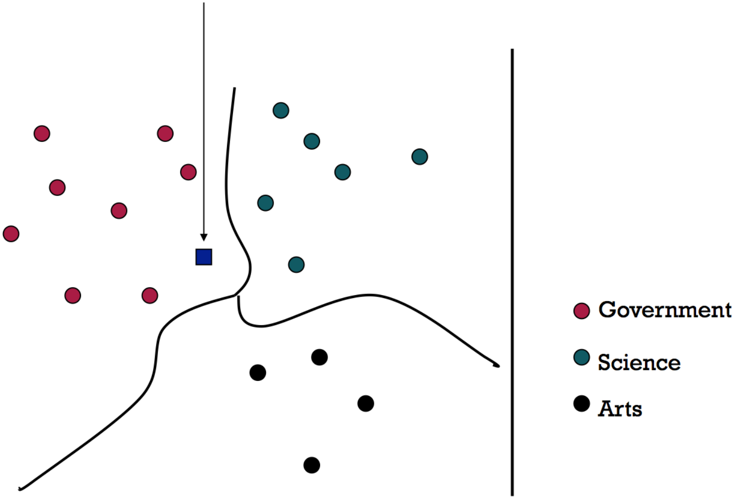
Documents in a Vector Space



Test Document of What Class?



Test Document = Government



Our focus: how to find "good" separators (or decision boundaries)?

Two Vector Space Classification Methods

- Rocchio Classification
  - Divides the vector space into regions centered on centroids
  - Simple and efficient, but inaccurate if classes are not approximately spheres with similar radii.
- K-Nearest Neighbors Classification
  - Assigns the majority class of the k nearest neighbors to a test document
  - Requires no explicit training
  - It is less efficient than other classification methods

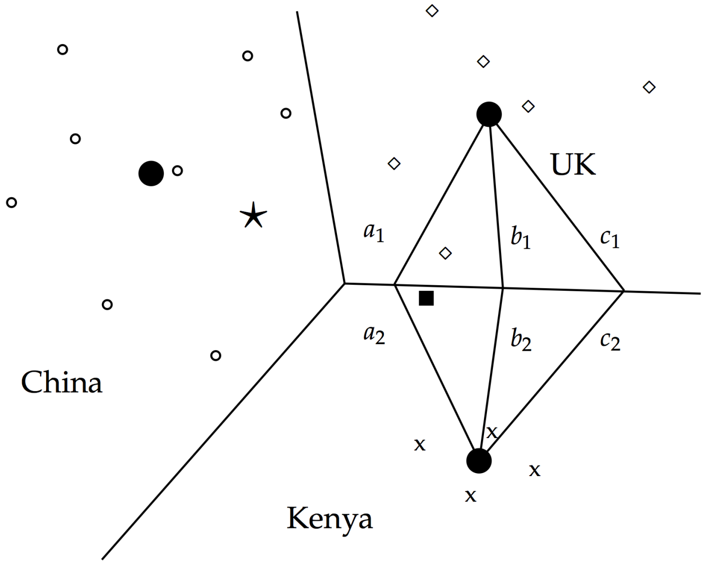
Rocchio Classification

- Uses centroids to define the boundaries
- The centroid of a class c is computed as the vector average:

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

  where D_c is the set of documents in D whose class is c and $v(d)$ is the vector space representation of d.
- The boundary between two classes in Rocchio classification is the set of points with equal distance from the two centroids.

Example



Rocchio Classification: Training and Testing

TrainRocchio($\mathbb{C}, \mathbb{D}$)
1 **for each** $c_j \in \mathbb{C}$
2 **do** $D_j \leftarrow \{d : \langle d, c_j \rangle \in \mathbb{D}\}$
3 $\vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$
4 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}$

ApplyRocchio($\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d$)
1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

The distance to a centroid is computed as the Euclidian distance.

Rocchio Classification

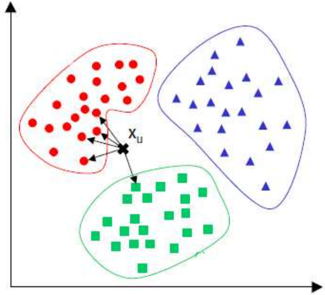
- Rocchio forms a simple representative for each class: the centroid/prototype
- Classification: nearest prototype/centroid
- Cheap to train and test documents

k Nearest Neighbor Classification

- kNN = k Nearest Neighbor
- To classify a document d:
  - Define k-neighborhood as the k nearest neighbors of d
  - Pick the majority class label in the k-neighborhood

K Nearest Neighbor (K-NN)

- Given training $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



  - Assign the majority class label (majority voting)
- The algorithm requires:
  - Parameter K: number of nearest neighbors to look for
  - Distance function: To compute the similarities between examples
- **Special Case**: 1-Nearest Neighbor

K Nearest Neighbors Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification)

Note: K-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, K-Nearest Neighbors does not learn an explicit mapping f from the training data
- It simply uses the training data at test time to make predictions

K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m}$ (make them zero mean, unit variance)
  - $\bar{x}_m = \frac{1}{N}\sum_{i=1}^{N} x_{im}$: empirical mean of m^{th} feature
  - $\sigma_m^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{im} - \bar{x}_m)^2$: empirical variance of m^{th} feature

K-NN: Computing the Distances

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in R^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

K-NN: Some Other Distance Measures

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} I(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree
- Mixed feature types (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part
- Can also assign weights to features: $d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm})$

k Nearest Neighbor

- Using only the closest example (1NN) 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: find the k examples and return the majority category of these k
- k is typically odd to avoid ties; 3 and 5 are most common
- Choosing k
  - Often data dependent and heuristic based
  - Or using cross-validation (using some held-out data)
  - In general, a k too small or too big is bad!

kNN: Discussion

- No training necessary
- Scales well with large number of classes
  - No need to train n classifiers for n classes
- It could be more accurate than NB or Rocchio

K-Nearest Neighbor: Further Discussion

- What is nice
  - Simple and intuitive; easily implementable
- What is not so nice...
  - Store all the training data in memory even at test time
    - Can be memory intensive for large training datasets
    - An example of non-parametric, or memory/instance-based methods
    - Different from parametric, model-based learning models
  - Expensive at test time: O(ND) computations for each test point
    - Have to search through all training data to find nearest neighbors
    - Distance computations with N training points (D features each)

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
- 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?