Deep Belief Networks with applications to Fast Document Retrieval

CSC2515

Existing Methods

- One of the most popular and widely used in practice algorithms for document retrieval tasks is TF-IDF.

- TF-IDF weights each word by:
  - its frequency in the query document (Term Frequency)
  - the logarithm of the reciprocal of its frequency in the whole set of documents (Inverse Document Frequency).

  However, TF-IDF has several limitations:
  - It computes document similarity directly in the word-count space, which may be slow for large vocabularies.
  - It assumes that the counts of different words provide independent evidence of similarity.
  - It makes no use of semantic similarities between words.

Existing Methods

- To overcome these limitations, many models for capturing low-dimensional, latent representations have been proposed.
- One such simple and widely-used method is Latent Semantic Analysis (LSA).
- It extracts low-dimensional semantic structure using SVD to get a low-rank approximation of the word-document co-occurrence matrix:

$$\log(1 + M(doc, w)) \sim USV.$$

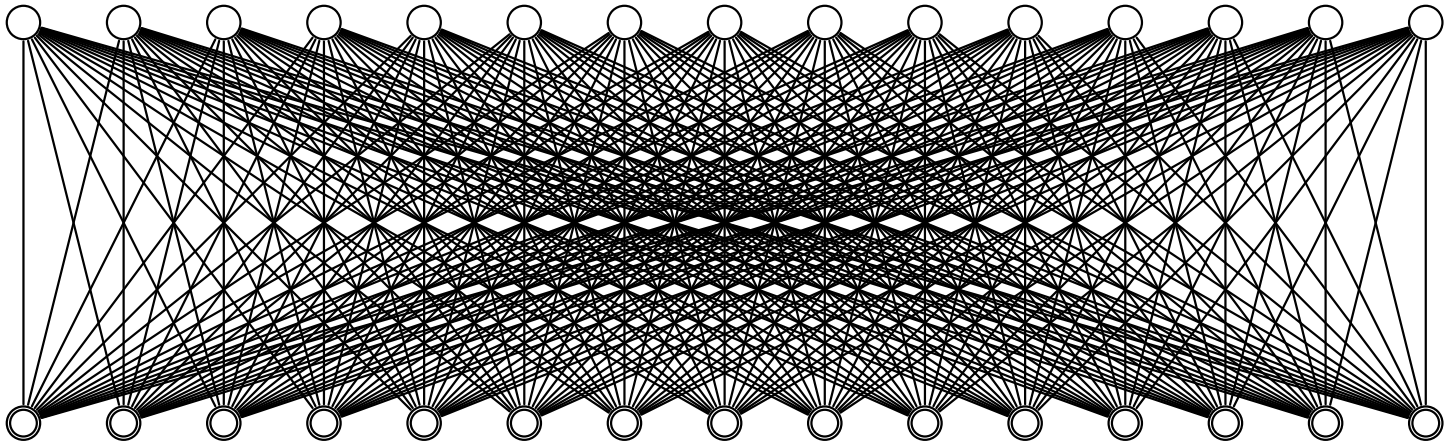
$U = |doc| \times d$, $S = d \times d$, $V = d \times |w|$.

- A test query q is represented as d-dim vector $S^{-1}V \log(1 + q)$.

Drawbacks of Existing Methods

- LSA is a linear method so it can only capture pairwise correlations between words.
- Numerous methods, in particular probabilistic versions of LSA were introduced in the machine learning community.
- These models can be viewed as graphical models in which a single layer of hidden topic variables have directed connections to variables that represent word-counts.
- There are limitations on the types of structure that can be represented efficiently by a single layer of hidden variables.
- We will build a network with multiple hidden layers and with millions of parameters and show that it can discover latent representations that work much better.

RBM's revisited



- A joint configuration $(\mathbf{v}, \mathbf{h})$ has an energy:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i b_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j W_{ij}.$$

- The probability that the model assigns to $\mathbf{v}$ is:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})).$$

RBM's for continuous data

- Hidden units remain binary.
- The visible units are replaced by linear stochastic units that have Gaussian noise.
- The energy becomes:

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j W_{ij}.$$

- Conditional distributions over hidden and visible units are:

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-b_j - \sum_i W_{ij} v_i/\sigma_i)},$$

$$p(v_i = v|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\Big(- \frac{(v - b_i - \sigma_i \sum_j h_j W_{ij})^2}{2\sigma_i^2}\Big).$$

RBM's for count data

- Hidden units remain binary and the visible word counts are modeled by the Poisson model.

- The energy is defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i b_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j W_{ij} + \sum_i \log v_i!.$$

- Conditional distributions over hidden and visible units are:

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-b_j - \sum_i W_{ij} v_i)},$$

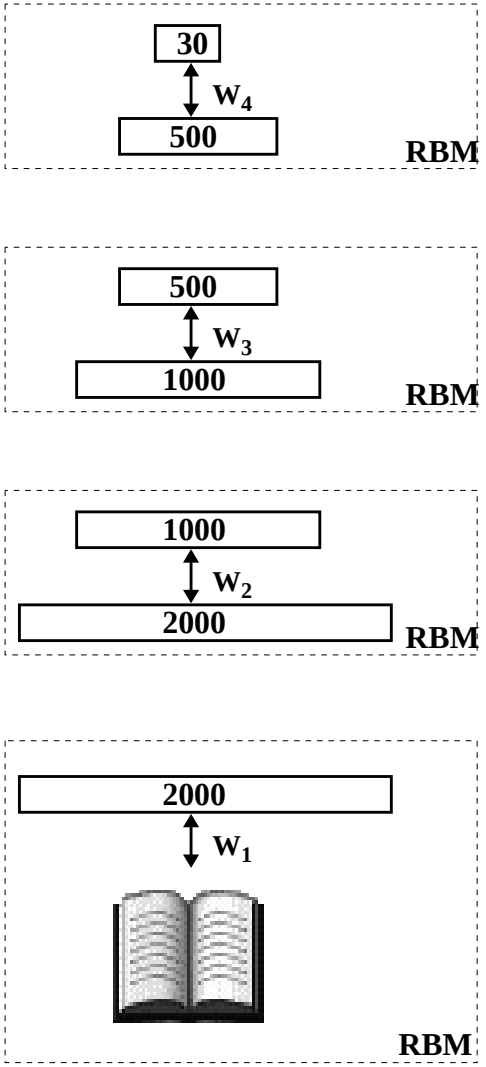
$$p(v_i = n|\mathbf{h}) = \text{Poisson}\left(n, \exp\left(b_i + \sum_j h_j W_{ij}\right)\right),$$

where

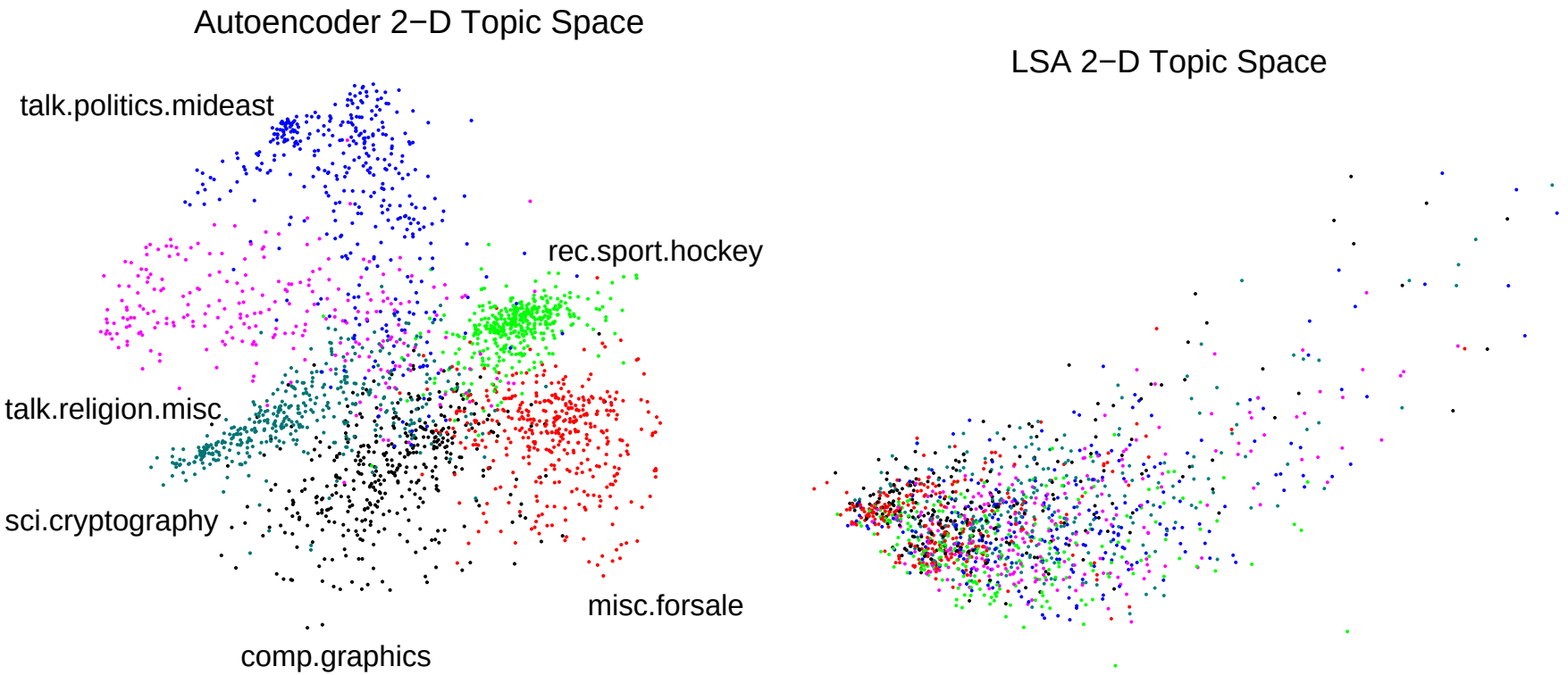
$$\text{Poisson}\left(n, \lambda\right) = e^{-\lambda} \frac{\lambda^n}{n!}.$$

Learning Stacks of RBM's

- Perform greedy, layer-by-layer learning:
  - Learn and Freeze W_1 using Poisson Model.
  - Treat the existing feature detectors as if they were data.
  - Learn and Freeze W_2.
  - Greedily learn many layers.
- Each layer of features captures strong high-order correlations between the activities of units in the layer below.



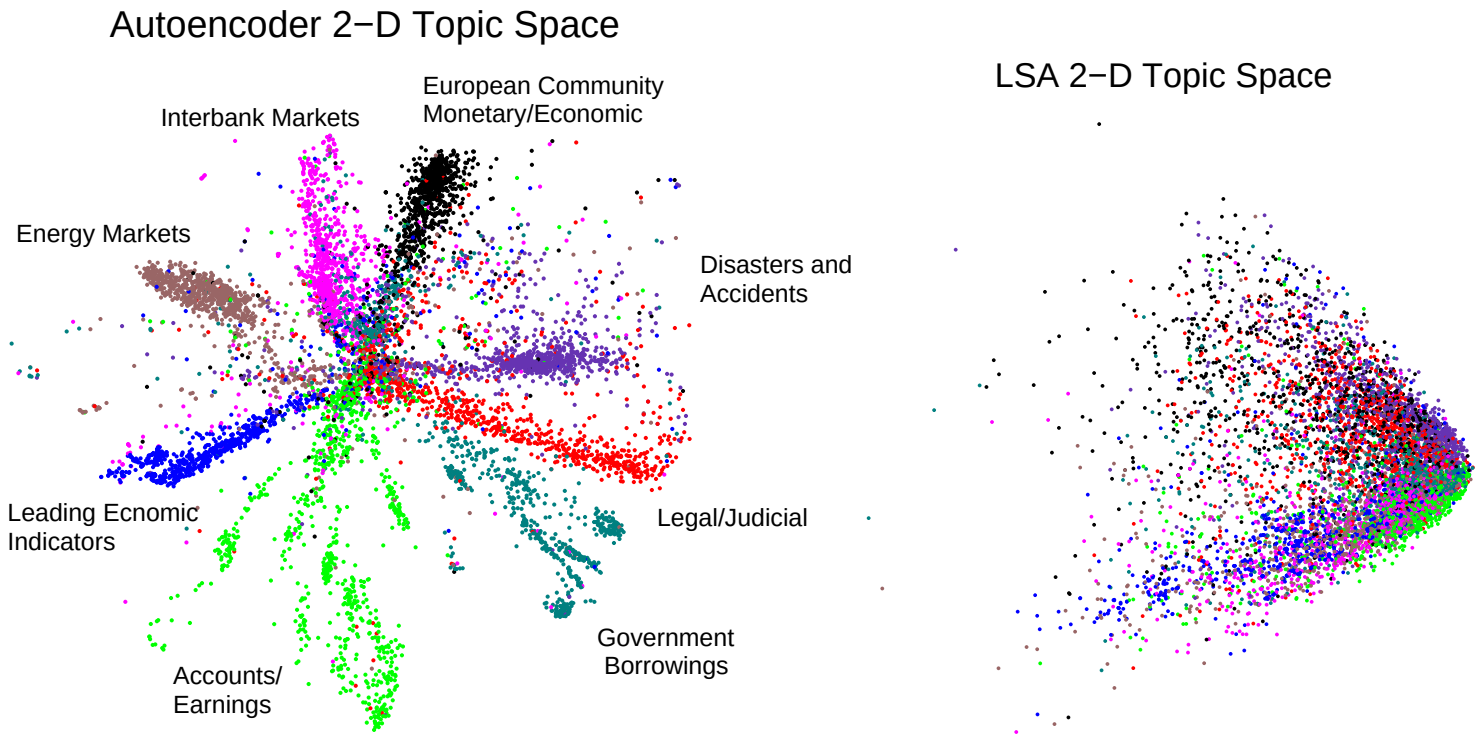
20 newsgroup: 2-D topic space



- The 20 newsgroup corpus contains 18,845 postings (11,314 training and 7,531 test) taken from the Usenet newsgroups.

- We use a 2000-500-250-125-10 autoencoder to convert a document into a low-dimensional code.
- We used a simple "bag-of-words" representation.

Reuters Corpus: 2-D topic space

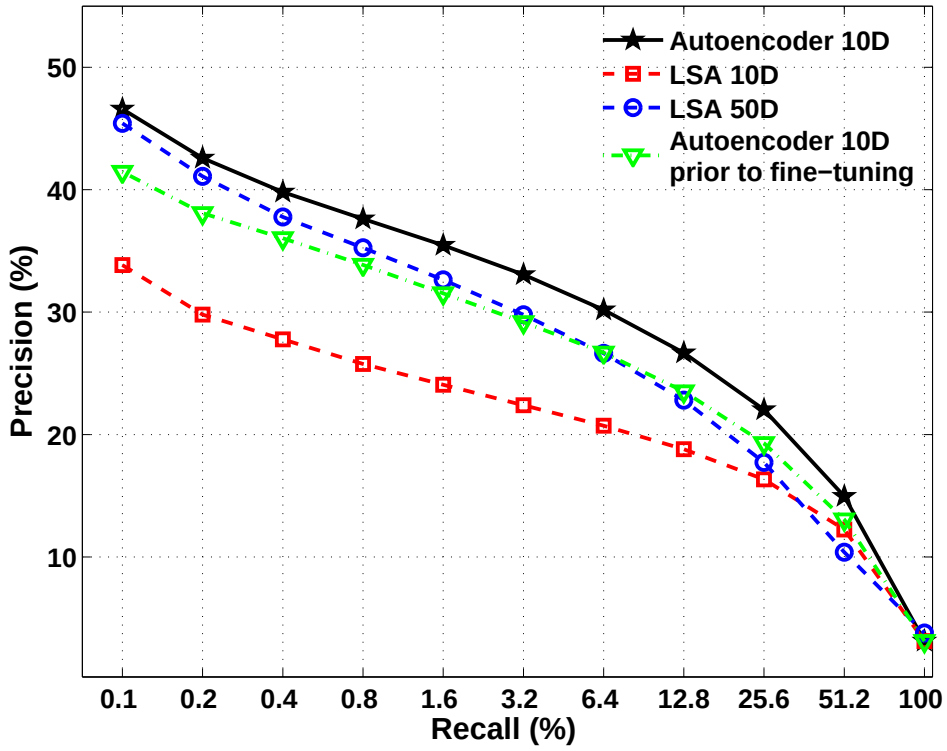


- We use a 2000-500-250-125-2 autoencoder to convert test documents into a two-dimensional code.

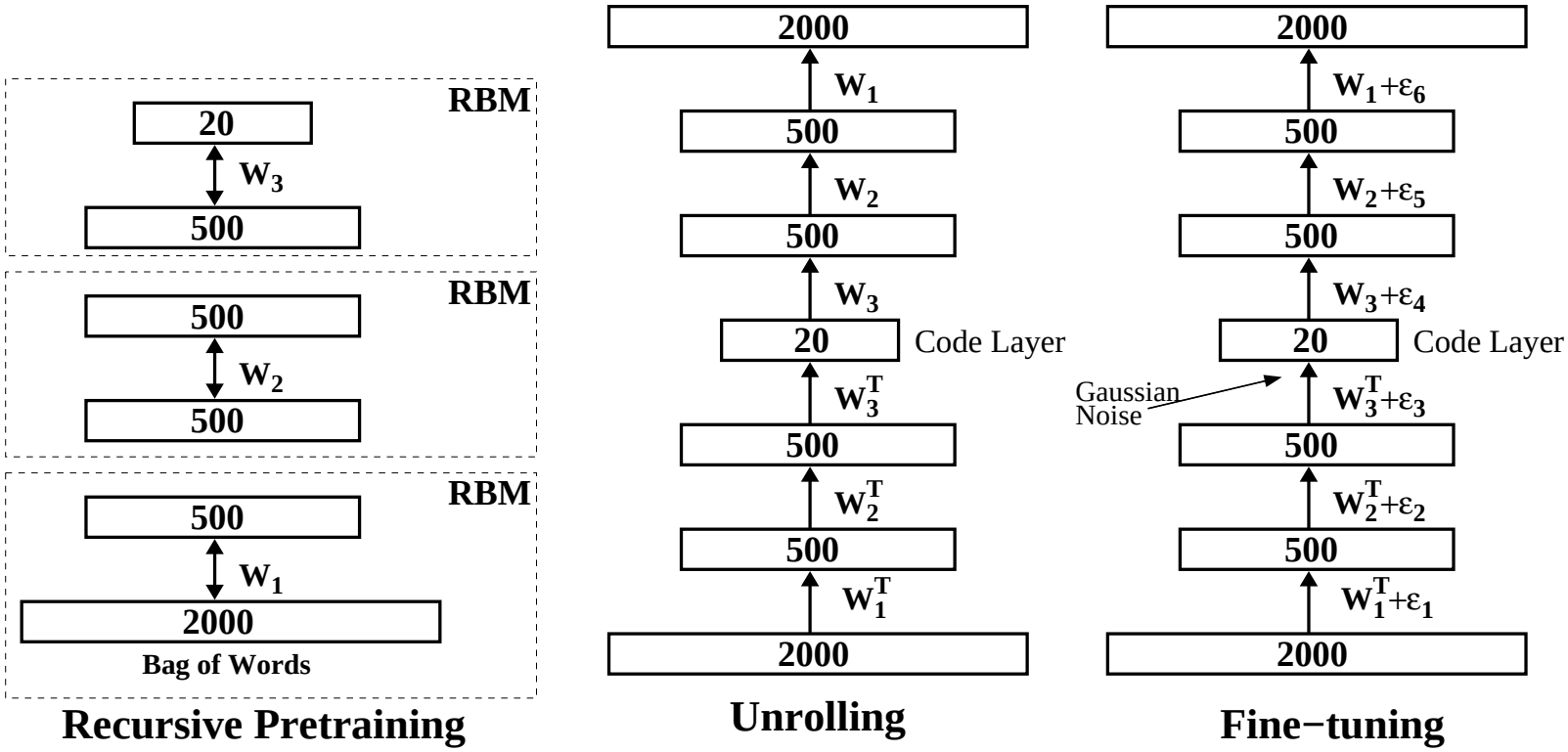
- The Reuters Corpus Volume II contains 804,414 newswire stories (randomly split into **402,207** training and **402,207** test).
- We used a simple "bag-of-words" representation.

Results for 10-D codes

- We use the cosine of the angle between two codes as a measure of similarity.
- Precision-recall curves when a 10-D query document from the test set is used to retrieve other test set documents, averaged over 402,207 possible queries.

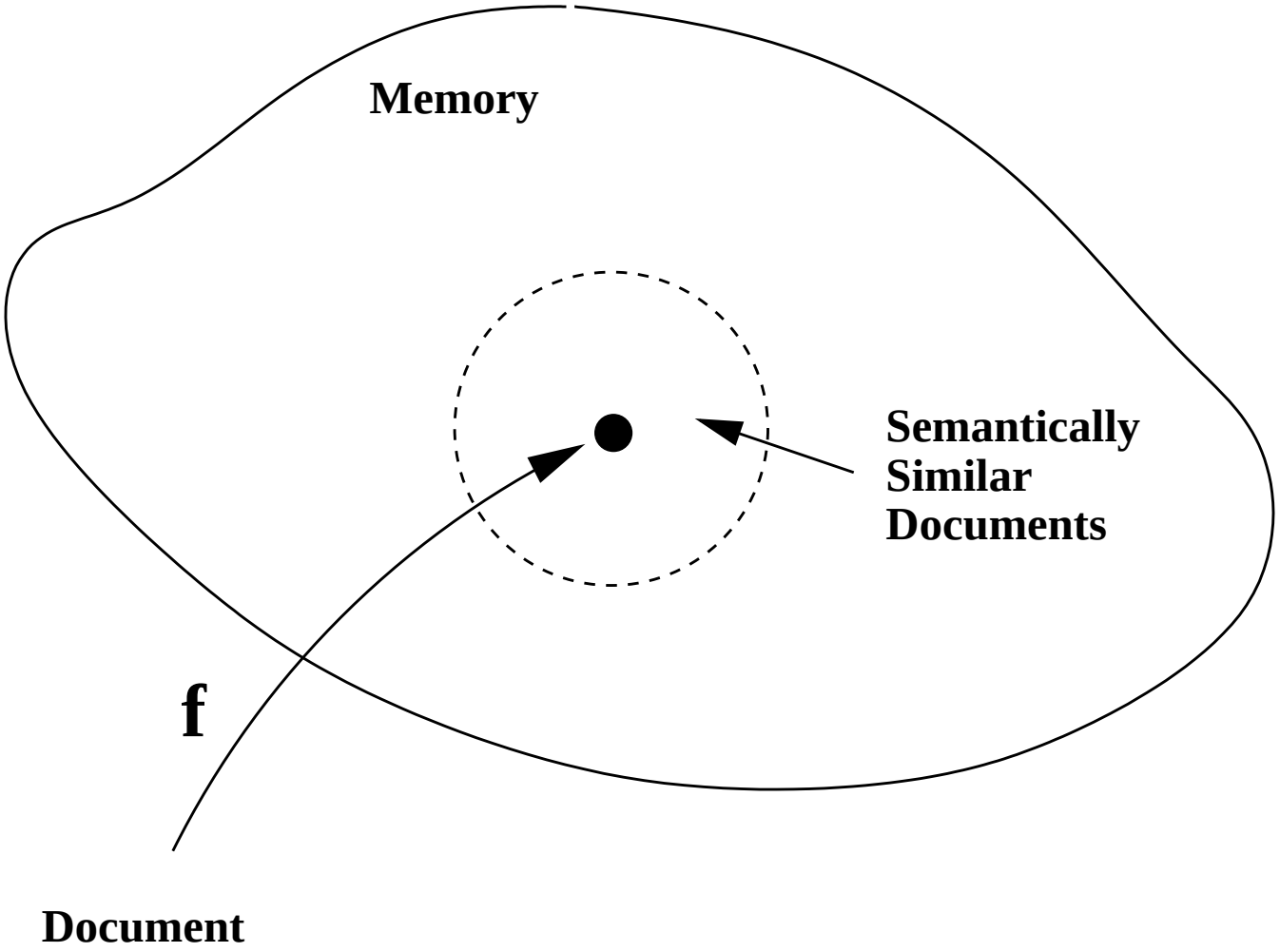


Semantic Hashing

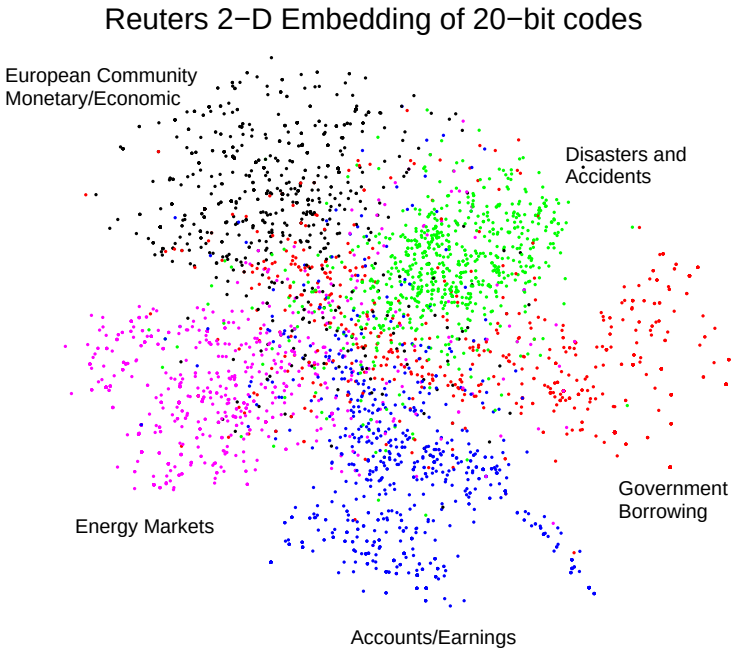


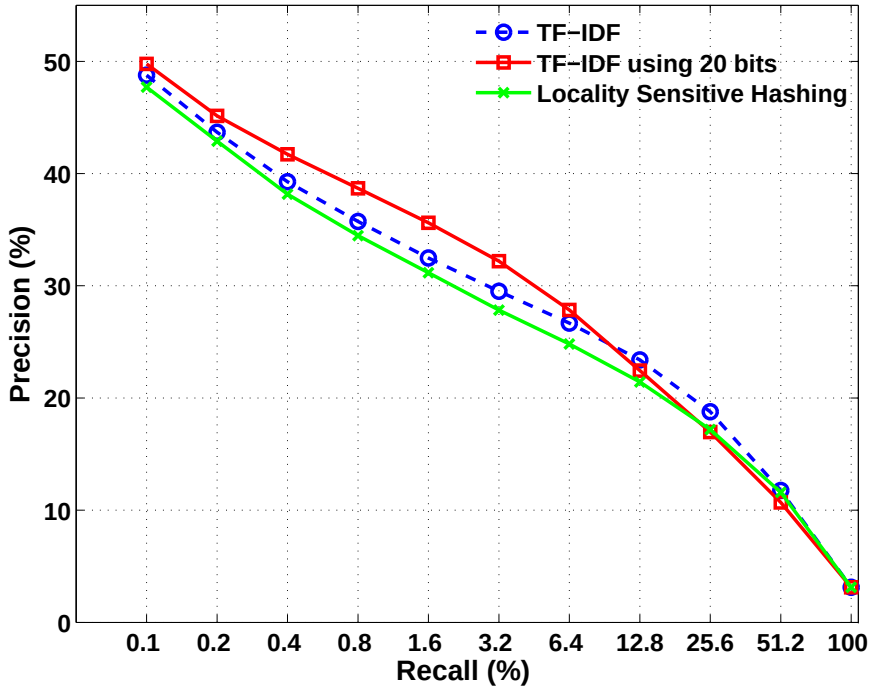
- Learn to map documents into *semantic* 20-D binary code and use these codes as memory addresses.
- We have the ultimate retrieval tool: Given a query document, compute its 20-bit address and retrieve all of the documents stored at the similar addresses **with no search at all**.

The Main Idea of Semantic Hashing



Semantic Hashing

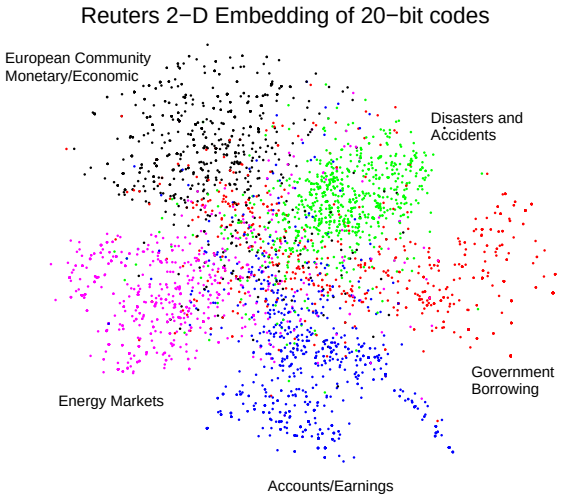


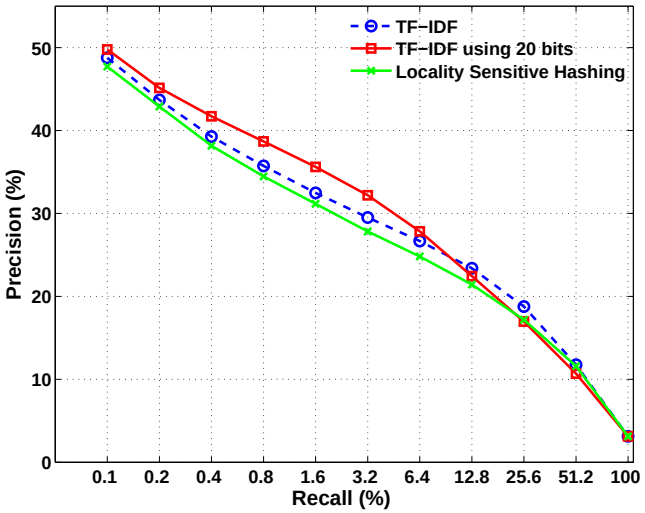


- Left picture shows a 2-dimensional embedding of the learned 20-bit codes using stochastic neighbor embedding.

- Right picture shows Precision-Recall curves when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries.

Semantic Hashing

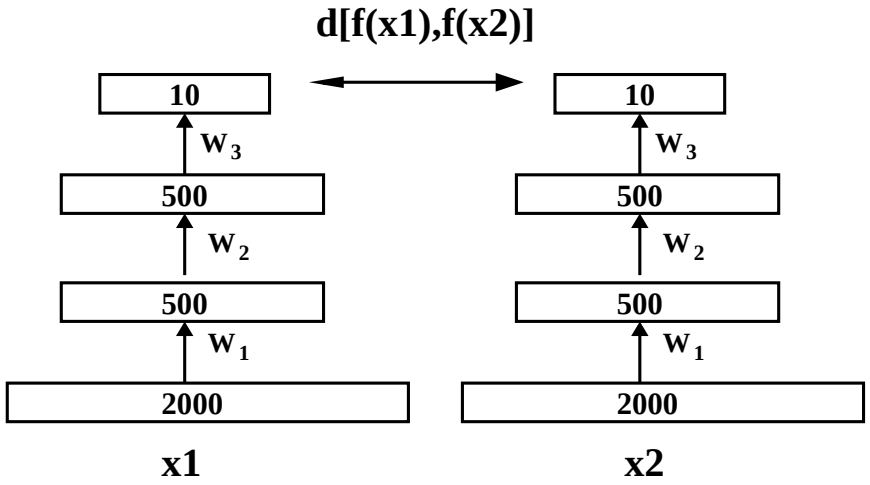




- We used a simple C implementation on Reuters dataset (402,212 training and 402,212 test documents).
- For a given query, it takes about 0.5 milliseconds to create a short-list of about 3,000 semantically similar documents.
- It then takes 10 milliseconds to retrieve the top few matches from that short-list using TF-IDF, and it is more accurate than full TF-IDF.

Learning nonlinear embedding

- Learning a similarity measure over the input space X.
- Given a distance metric D (e.g. Euclidean) we can measure similarity between two input vectors $\mathbf{x}^n$, $\mathbf{x}^k \in X$ by computing $D[f(\mathbf{x}^n|W), f(\mathbf{x}^k|W)]$.
- "Push-Pull" Idea: Pull points belonging to the same class together. Push points belonging to the different classes apart.



Learning Nonlinear NCA

- We are given a set of N labeled training cases $(\mathbf{x}^n, c^n)$, $a = 1, 2, ..., N$, where $\mathbf{x}^n \in R^d$, and $c^n \in \{1, 2, ..., C\}$.

- For each training vector $\mathbf{x}^n$, define the probability that point n selects one of its neighbours k as:

$$p_{nk} = \frac{\exp\left(-D_{nk}\right)}{\sum_{z \neq n} \exp\left(-D_{nz}\right)}, \qquad p_{nn} = 0$$

  where $D_{nk} = \| f(\mathbf{x}^n|W) - f(\mathbf{x}^k|W) \|^2$, and $f(\cdot|W)$ is a multi-layer perceptron.

- Previous algorithms: a simple linear projection $f(x|W) = Wx$.

- The Euclidean distance is then the Mahalanobis distance:

$$D[f(\mathbf{x}^n), f(\mathbf{x}^k)] = (\mathbf{x}^n - \mathbf{x}^k)^T W^T W (\mathbf{x}^n - \mathbf{x}^k).$$

Learning Nonlinear NCA

- Probability that point n belongs to class a is:

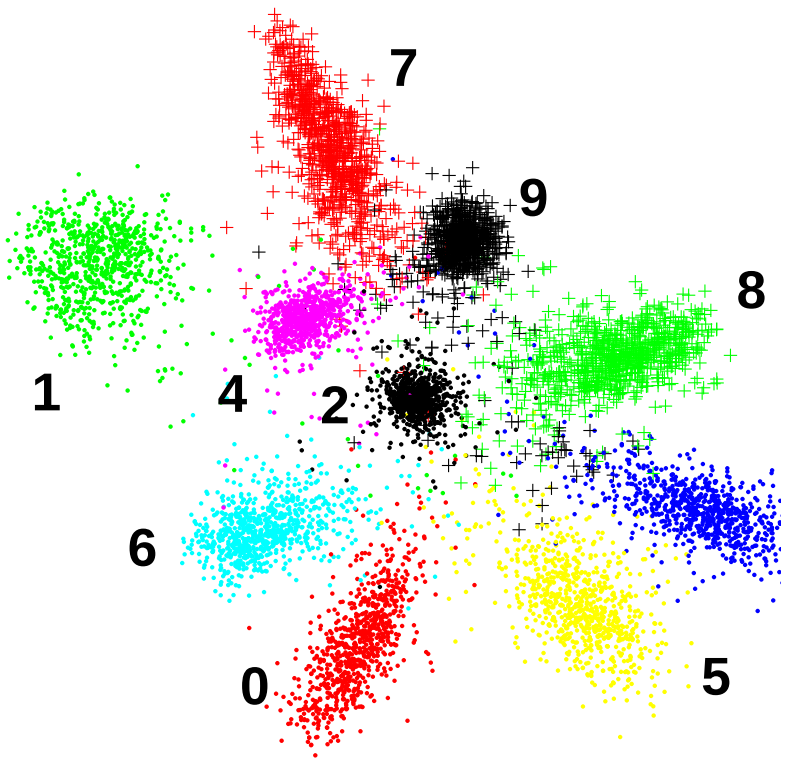
$$p(c^n = a) = \sum_{k:c^k=a} p_{nk}.$$

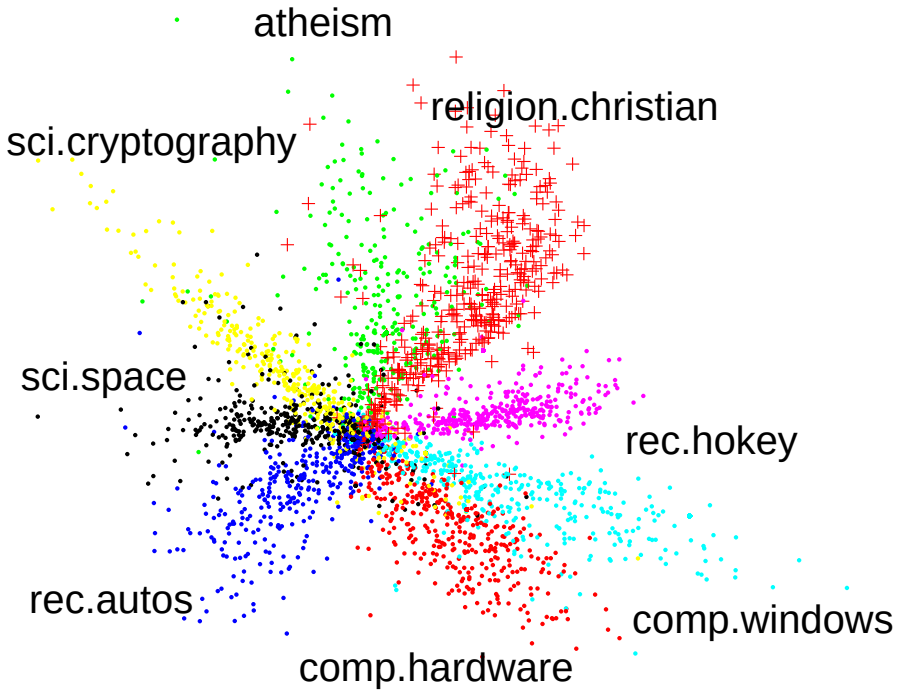
- Maximize the expected number of correctly classified points on the training data:

$$O_{NCA} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k:c^n=c^k} p_{nk}.$$

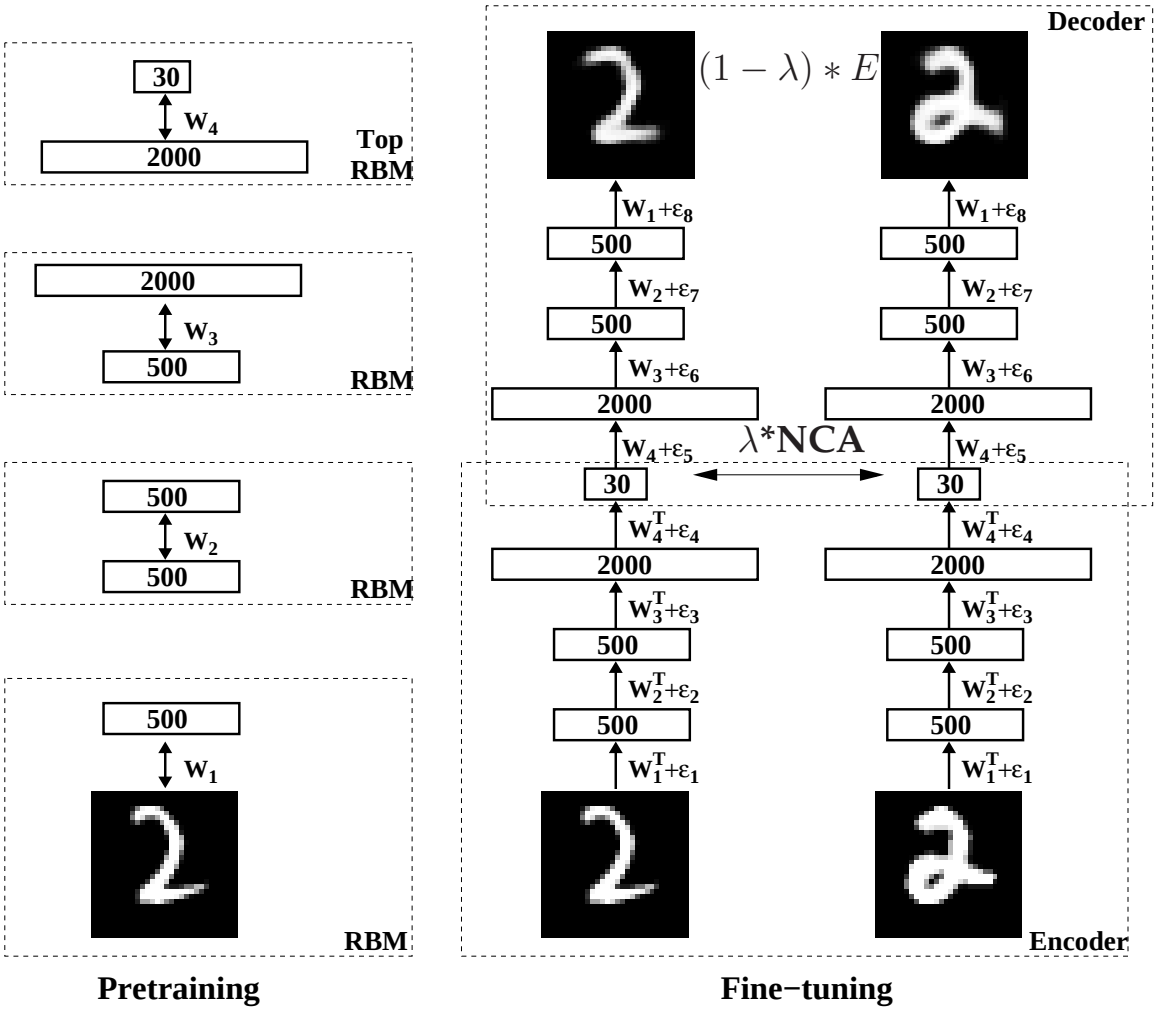
- By considering a linear perceptron we arrive at linear NCA.

2-D codes


7
9
8
1
4
2
6
0
5


atheism
religion.christian
sci.cryptography
sci.space
rec.hokey
rec.autos
comp.windows
comp.hardware

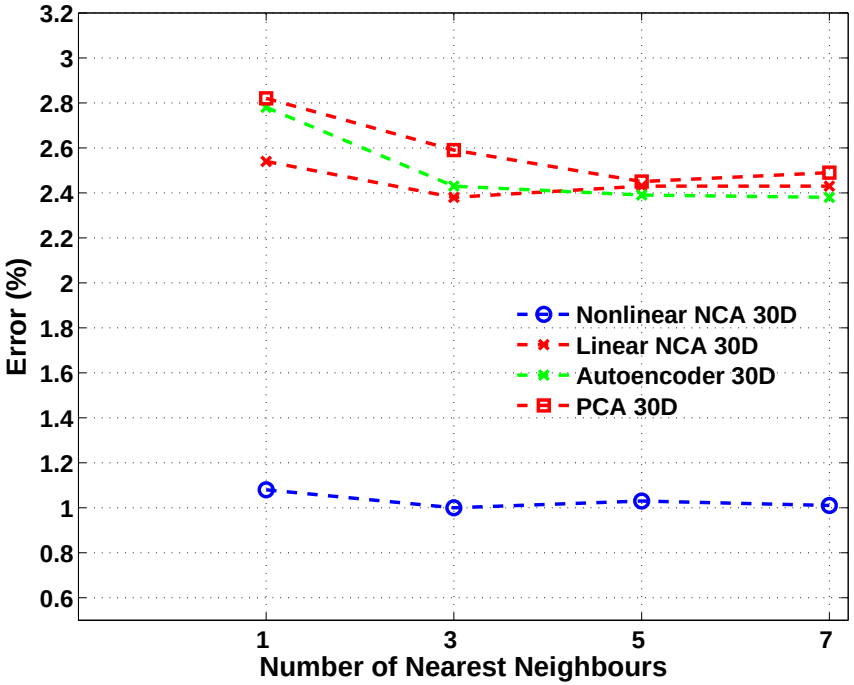
Regularized Nonlinear NCA

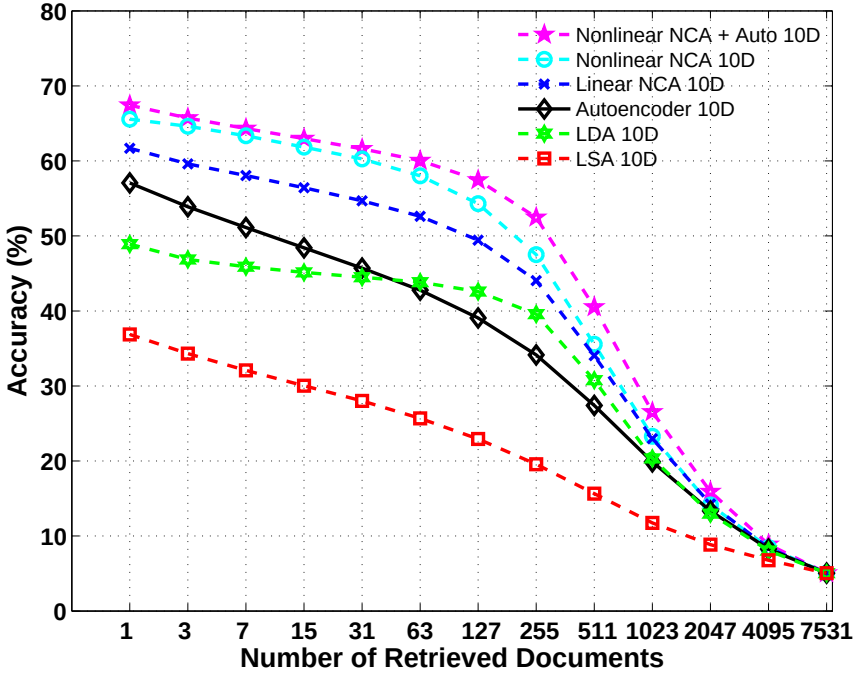


- The NCA objective O_{NCA} is combined with the autoencoder reconstruction error E to maximize:

$$C = \lambda O_{NCA} + (1-\lambda)(-\mathrm{E}).$$

Results





- Left: K nearest neighbours results in the 30-D space on the MNIST test set.
- Right: Precision-recall curves for 20 newsgroup test data.

THE END