**Document Vectorization**

- **Word → Vec**
  - `the`: [0.37 0.9 1.1 ...]
  - `pulls`: [1.2 -7.5 0.1 ...]
  - `man`: [0.11 0.32 0.6 ...]
  - `lever`: [-1.3 -1.1 2.3 ...]

- **Recurrent Neural Network**
  - Man
  - `man` vector: [0.11 0.32 0.6 ...]

- **Document → Vec**
  - 1: [-0.2 1.0 -0.2 ...]
  - 2: [1.0 -0.4 0.9 ...]
  - 3: [0.03 0.92 1.1 ...]
  - 4: [0.06 0.04 2.2 ...]

**Augmented TF-IDF**
- A Term Frequency (TF) vector is constructed for each document.
- Each dimension of a TF vector represents the frequency of a given term within the document.
  - We augment this with word similarities derived from word vectors. Example: the word "lion" might also increase the frequency scores of "lions" and "lioness".
- Inverse Document Frequency (IDF) measures the "rareness" of each term across all documents.

**Querystring Search**
- Like Google: Given a short phrase, find matching documents.
  - Build TF vector for query
  - Compare query TF against document TFs using cosine similarity (dot product)
  - Weight by IDF (matching rarer terms matters more)

**Personalized Recommendations**
- Based on a user’s previously viewed articles, suggest new ones
  - We train a Linear Support Vector Machine to draw a dividing hyperplane (in 2D, this is a line) through the space so all of a user’s previously-liked documents lie on one side of the hyperplane.
  - Mathematically, it minimizes the **hinge loss**.
  - Any unseen documents on the same side of the line are recommended to the user.

**Related Documents**
- Given an article, find other articles that contain similar topics
  - Document vectors already arranged spatially based on similarity
  - Given a query document, just find nearest neighbor using Euclidean distance

**Error Calculation**
- Backpropagate cross-entropy loss to train.
  - Doc vectors become semantically meaningful as a side effect.