Vector Space Model

Feast or Famine

Feast or famine refers to the phenomenon where boolean retrieval models tend to retrieve either (1) A lot of results; or, (2) few results

Jaccard Coefficient

$\text{Jaccard} = \frac{|A \bigcap B|}{|A \bigcup B|}, A \neq \emptyset, B \neq \emptyset$

The Jaccard Coefficient measures the overlap between two sets, A and B. In the context of information retrieval, it ignores term frequency and does not normalize the length of individual documents.

TF-IDF

$w_{t,d} = (1 + log(tf_{t,d})) \cdot log\left(\frac{N}{df_t}\right)$

tf-weighting: The number of times that the term t occurs in the document d.

idf-weighting: Measures the "informativeness" of the term t.

Vector Space Model

Represents each document as a vector $v \in \mathbb{R}^{|V|}$ where v is composed of $tf - idf$ weights. This means the document collection is a real-valued vector space of $|V|$ dimensions. Terms represent axes in this space, documents are points or vectors.

In this space, we can rank queries based on how close they are to the query.

Cosine Similarity

We will rank documents according to their distance from the query in the vector space. To do this, we will rank them according to $cosine(\vec{q}, \vec{d})$. We will normalize the lengths of the documents by dividing each component in each vector by the vector's length.

$$cos(\vec{q}, \vec{d}) = sim(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q} \cdot \vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2 \sum_{i=1}^{|V|} d_i^2}}$$

Different Ways of Encoding

Term Frequency	Document Frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
l (logarithm) $1 + log(tf_{t,d})$	t (idf) $log(\frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2+w_2^2+\cdots+w_M^2}}$
a (augmented) $0.5 + \frac{0.5 \cdot tf_{t,d}}{max_t(tf_{t,d})}$	p (prob idf) $max0, log\frac{N-df_t}{df_t}$	u (pivoted unique) $\frac{1}{u}$
b (boolean) $\begin{cases} 1 & if \text{tf}_{t,d} > 0 \\ 0 & otherwise \end{cases}$		b (byte size) $\frac{1}{CharLength^{\alpha}}, \alpha < 1$
L (log avg) $\frac{1+log(tf_{t,d})}{1+log(ave_{t \in d}(tf_{t,d}))}$		

Complete Search System

Exact Top K Retrieval using Min-Heap

To process a new document d with score s:

Get current minimum hm of heap (O(1))

If $s\prime \leq hm$ skip to next document

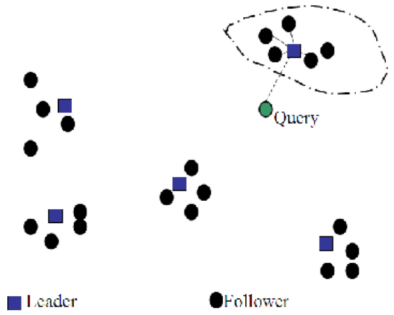
If $s\prime > hm$ heap-delete-root ($O(logk)$)

Heap-add $d/s(O(logk))$

Inexact Top K retrieval

- Order documents based on query-independent model (like PageRank)
- Primarily consider query terms with high *idf* values
- Cluster Pruning (see figure 1)

Figure 1: Cluster pruning to retrieve top K results



Evaluation

Unranked evaluation

Precision (P): The fraction of retrieved documents that are relevant

Recall (R): The fraction of relevant documents that are retrieved

F Score: Allows a trade off between precision and recall

$$F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P+R} \text{ where } \beta^2 = \frac{1-\alpha}{\alpha}$$

and where

$$\alpha \in [0,1] \text{ and } \beta^2 \in [0,\infty]$$

Most frequently, **balanced F** is used where $\alpha = 0.5$ and $\beta = 1$

Accuracy

In IR, striving for accuracy would frequently result in nothing being returned. Instead, we want to return *something*, even if what is returned isn't necessarily accurate.

Ranked evaluation

Precision-Recall Curve: Each point corresponds to a result for the top k ranked hits

Mean Average Precision (MAP): Roughly the area under the precision-recall curve

$$MAP(Q) = \frac{1}{|Q|} = \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk})$$

where

m_j - number of relevant documents for query j.

R_{jk} - number of documents until the relevant document d_k for query j.

Mean Reciprocal Rank (MRR): $MRR(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{\text{rank}_j}$ where rank_j is the position of top correct answer for query j.

Inter-Annotator Agreement

Kappa measure: Measures how much judges agree or disagree

$$\kappa = \frac{P(A)-P(E)}{1-P(E)}$$

where

$P(A)$ = proportion of judges which agree

$P(E)$ = agreement we would get by chance

For these measures, $\kappa \in [\frac{2}{3}, 1]$ is seen as acceptable

Real-world evaluations

A/B testing: Testing a feature by splitting a portion of your user base and introducing the feature to that small portion. Evaluate how well the new feature is recieved by the portion of the userbase with automatic evaluation technology.

Result Summaries

Static Summaries: Summary is always the same regardless of the query (i.e first 50 words, etc.)

Dynamic Summaries: Query dependent. Attempt to explain why the document was retrieved for the query

Criteria for good dynamic summaries

- Maximally informative for query
- Self-contained, easy to read
- Short enough, shouldn't take up real estate

Relevance Feedback & Query Expansion

Centroid

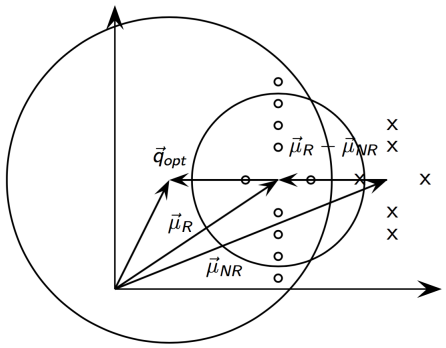
The centroid is the center of mass for a set of points. Since we represent documents as points in a high-dimensional space, we can compute the centroid of a collection of documents.

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

where D is the set of documents and $\vec{v}(d)$ is the vector for the document d.

Rocchio algorithm

Figure 2: An illustration of the Rocchio algorithm



Implements relevance feedback in the vector space model. Rocchio chooses the query $\vec{q}_{opt}$ that maximizes:

$$\vec{q}_{opt} = \arg_{\vec{q}}\max[sim(\vec{q}, \mu(D_r)) - sim(\vec{q}, \mu(D_{nr}))]$$

where D_r is the set of relevant documents and D_{nr} is the set of non-relevant documents.

The vector $\vec{q}_{opt}$ is the vector that separates relevant and non-relevant documents. With some additional assuptions, we can rewrite this as:

$$\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

In practice, the SMART algorithm is used:

$$\vec{q_m} = \alpha\vec{q_0} + \beta\mu(D_r) - \gamma\mu(D_{nr}) = \alpha\vec{q_0} + \frac{\beta}{|D_r|}\sum_{\vec{d_j} \in D_r}\vec{d_j} - \frac{\gamma}{|D_{nr}|}\sum_{\vec{d_j} \in D_{nr}}\vec{d_j}$$

where

- $\vec{q_m}$ - Modified query vector
- $\vec{q_0}$ - Original query vector
- D_r and D_{nr} sets of known relevant and non-relevant documents
- α, β, and γ are weights

Query Expansion Using Global Resources

Some examples of global resources which can be used for query expansion:

- Manual thesaurus maintained by editors
- Automatically derived thesaurus (co-occurence statistics)
- Query-equivalence based on query log mining

Query expansion at search engines

Primary source of query expansion in search engines comes from mining the query logs from those search engines. If queries frequently occur within immediate proximity of one another, then you can begin to draw equivalence between those two queries.

Probabilistic Information Retrieval

Basic probability theory:

Conditional Probability: $P(A|B) = \frac{P(A \bigcap B)}{P(B)}$

Chain Rule: $P(A \bigcap B) = P(A|B) \cdot P(B)$

Bayes' Rule: $P(A|B) = \frac{P(A \bigcap B)}{P(B)} = \frac{P(A)P(B|A)}{P(B)}$

Law of Total Probability: $P(B) = P(A)P(B|A) + P(\overline{A})P(B|\overline{A})$

Odds: $O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1-P(A)}$

Independent Events: A is independent of B if $P(A|B) = P(A)$.

Probability Ranking Principle

If the retrieved documents (with respect to a query) are ranked decreasingly on their probability of relevance, then the effectiveness of the system will be the best that is obtainable

Binary independence model

A document d is represented as a vector $\vec{x} = (\vec{x}_1, \ldots, \vec{x}_M)$ where $x_t = 1$ if t occurs in d and $x_t = 0$ otherwise. We make the assumption of independence between terms, which means that there is no association between terms. Theoretically, this is false, but it works in practice.

	document	relevent ($R = 1$)	nonrelevant ($R = 0$)	Total
Term present	$x_t = 1$	s	$df_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - df_t) - (S - s)$	$N - df_t$
	Total	S	N - S	N

$p_t = s/S$ $u_t = (df_t - s)/(N - S)$

We rank documents using log odds ratios for terms in the query c_t:

$c_t = log\frac{p_t(1-u_t)}{u_t(1-p_t)} = log\frac{p_t}{1-p_t} - log\frac{u_t}{1-u_t}$

The result for c_t can be interpreted like so:

- $c_t = 0$ term has equal odds of appearing in relevant and non-relevant documents
- $c_t > 0$ term has higher odds to appear in relevant documents
- $c_t < 0$ term has higher odds to appear in nonrelevant documents

how to derive the ranking function for terms; the formula for ct, with smoothing; BIM after simplifying assumptions

Okapi BM25

Improve *idf* term by factoring in term frequency and document length:

$RSV_d = \sum_{t \in q} log\left[\frac{N}{df_t}\right] \cdot \frac{(k_1+1)tf_{td}}{k_1((1-b)+b\cdot(L_d/L_{ave}))+tf_{td}}$

where

- tf_{td}: Term frequency in document d
- L_d: and L_{ave}: Length of d and average document length respectively
- k_1: Controlling document term frequency scaling $k_1 > 0$
- b: Controlling scaling by document length $b \in [0, 1]$

Language Models

Smoothed LM Ranking

For a query q, we compute $P(q|d)$ using the following equation:

$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1-\lambda)P(t_k|M_c)$

where

$P(t|M_d) = \frac{tf_{t,d}}{|d|}$

$P(t|M_c) = \frac{cf_t}{T}$

N-Gram Language Models

These models are used in applications where the independence assumption made in language models is impossible, such as text prediction systems.

Text Classification & Naive Bayes

Why Text Classification

Text classification is used for things like automatic language detection, spam detection, sentiment detection, etc.

How to compute P(c | d), smoothing

We compute $P(c|d)$ as:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

where n_d is the length of the document, $P(t_k|c)$ is a measure of how much evidence t_k contributed that c is the correct class, and $P(c)$ is the prior probability of c.

To prevent floating point underflow, what is usually computed in practice is:

$$P(c|d) \propto logP(c) + \prod_{1 \leq k \leq n_d} logP(t_k|c)$$

Multinomial vs. Bernoulli

The Bernoulli model estimates $P(t|c)$ as the fraction of documents of class c that contain term t.

In contrast, the multinomial model estimates $P(t|c)$ as the fraction of tokens or fraction of positions in documents of class c that contain term t

Naive Bayes Pitfalls (3)

- Underflow: Not using log probabilities
- Zero Counts: Not smoothing
- Not doing feature selection

Naive Bayes Assumptions (2)

- Conditional Independence Assumption: We assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(X_k = t_k|c)$.
- $P(X_{k1} = t|c) = P(X_{k2} = t|c)$. The probability of generating a term in the first position is equal to the probability of generating that term in the last position

The combination of these assumption amount of **bag of words**.

Evaluating classification

Macroaveraging:

1. Compute F_1 for C classes
2. Average these C numbers

Microaveraging:

1. Compute TP, FP, and FN for each of the C classes
2. Sum these C numbers
3. Compute F_1 for aggregate TP, FP, and FN.

Feature selection

There are several different ways of picking features:

- Frequency: Select the most frequent terms
- Mutual Information: Select the terms with the highest mutual information
- Chi-Squared: Measure how much expected counts E and observed counts N deviate from each other

Vector Space Classification

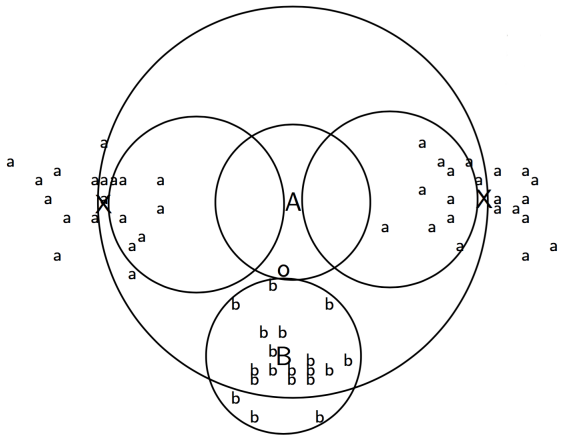
Rocchio

Basic idea of Rocchio:

1. Compute centroid for each class
2. Assign each test document to the class of its closest centroid

In most cases, Rocchio performs worse than Naive Bayes' because it cannot handle nonconvex, multimodal classes correctly (see figure 3):

Figure 3: An example of nonconvex, multimodal classes



K Nearest Neighbors

Assign the document d to the most popular class amongst it's k nearest neighbors. In probabilistic IR, you can represent this as finding the class c that maximizes $P(d|c$ for the document d.

The time complexity to train kNN is $\Theta(|\mathbb{D}|L_{ave})$. For testing, its time complexity is $\Theta(L_a + |\mathbb{D}|M_{ave}M_a) = \Theta(|\mathbb{D}|M_{ave}M_a)$.

Flat Clustering

Classification vs. Clustering

Classification is a form of supervised learning. It requires a series of training data which determine the kinds of classes that documents can be classified into. Clustering is a form of unsupervised learning where clusters are *inferred from the data* without human input.

Applications of Clustering in IR

- Search results to provide more information to the user
- Clustering collections for exploratory browsing
- Cluster-based retrieval for higher efficiency/faster searching

K-means

K-means is an algorithm designed to find a clustering for a collection of documents. It does this by computing centroids and attempting to minimize the

average squared distance from the centroid.

$$\mu(\omega) = \frac{1}{\omega} \sum_{\vec{x} \in \omega} \vec{x} \qquad \text{(Centroid definition for a cluster } \omega\text{)}$$

The process iterates over two steps:

- **Reassignment**: Assign each vector to its closest centroid
- **Recomputation**: Recompute each centroid as the average of the vectors that were assigned to it in reassignment

This is guarenteed to converge on a clustering:

1. Let RSS be the sum of all squared distances between document vectors and their closest centroids
2. RSS decreases during each reassignment step because each vector is moved to a closer centroid
3. RSS decreases during each recomputation step
4. Since there are only a finite number of clusterings, it will eventually converge upon a clustering

Just because K-Means converges does **NOT** mean it converges on the *optimal* clustering. A bad starting seed can result in a terrible clustering.

K-Means++

K-Means++ is an algorithm to initialize K-Means:

1. Choose one center uniformly at random from among the data points.
2. For each data point x, compute D(x), the distance between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to D(x)2.
4. Repeat Steps 2 and 3 until K centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard K-means.

Clustering evaluation

We can measure the **purity** of a clustering by determining how well we can reproduce the classes:

$$\text{purity}(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \bigcap c_j|$$

where $\Omega = \omega_1, \omega_2, \ldots, \omega_k$ is the set of clusters and $C = c_1, c_2, \ldots, c_j$ is the set of classes.

For each cluster ω_k: Find class c_j with most members n_{kj} in ω_k. Sum all n_{kj} and divide by the number of points.

Another criterion is the Rand Index:

$$RI = \frac{TP+TN}{TP+FP+FN+TN}$$

and the F measure:

1. RI assigns the same weight to FPs and FNs.
2. In clustering, FNs are often more important (do not want to miss cluster elements).
3. The F measure can be used then: $\beta > 1$ to add more weight to recall and penalize FNs more strongly.
4. For example, $\beta = 5$

Number of Clusters

A way to determine a valid number of clusters:

1. Start with 1 cluster (K = 1)
2. Keep adding clusters (= keep increasing K)
3. Add a penalty for each new cluster
4. Then trade off cluster penalties against average squared distance from centroid
5. Choose the value of K with the best tradeoff

Link Analysis & Page Rank

Anchor Text

Anchor text is the text contained within links on a web page. These links form a directed graph and the links can be assumed to be a quality signal. The anchor

text in links is also frequently a good summary of the page's content.

We can use the directed graph nature of the web and the "quality signals" of links to rank pages based on how many in-links that they have.

Google Bombs

A search with "bad" results due to maliciously manipulated anchor text. These were mostly fixed by an algorithm change made by Google in 2007.

PageRank

PageRank essentially makes a random walk through the directed graph of the web by following links and occasionally "teleporting". It uses link frequencies to create a probability table for the web.

The algorithm has a random chance to teleport to a random page in the web. This is a hyperparameter for the algorithm.

To converge the probability vectors for PageRank, we can use the power method:

> Algorithm: multiply x by increasing powers of P until convergence.

Issues:

- Real surfers are not random surfers.
- PageRank alone frequently causes bad ranks for most pages