Text Mining

Francesco Farinola

December 2021

Contents

1 Text processing **4**
- 1.1 Building the inverted index 4
  - 1.1.1 Text tokenization 4
  - 1.1.2 Stop Words 5
  - 1.1.3 Word Normalization 5
  - 1.1.4 Uppercase and Lowercase, synonyms and phonetic equivalence 5
  - 1.1.5 Lemmatization 5
  - 1.1.6 Stemming 5
- 1.2 Scoring in Ranked Retrieval Models 6
  - 1.2.1 Jaccard Coefficient 6
- 1.3 Bag of words model 7
  - 1.3.1 TF-IDF 8
  - 1.3.2 Cosine similarity 9
  - 1.3.3 PageRank 10
  - 1.3.4 Meaning of words 10
- 1.4 Evaluation of results 10
  - 1.4.1 Accuracy 11
  - 1.4.2 F1-measure 11
  - 1.4.3 R-Precision 12
  - 1.4.4 Mean Average Precision 12

2 Dimensionality Reduction **13**
- 2.1 Mutual Information 13
- 2.2 χ^2 test (chi-square) 14
- 2.3 Latent Semantic Analysis 16
  - 2.3.1 Trasformation with Singular Value Decomposition SVD - Factorization 16
  - 2.3.2 Low-rank Approximation 19

3 Text Classification **20**
- 3.1 Multi-label text categorization 20
  - 3.1.1 Tranformation to single-label 20
  - 3.1.2 Hybrid solution - Chain of Classifiers (2010) 21
  - 3.1.3 Accuracy measures for Multi-label 21
  - 3.1.4 Search Engines 22
  - 3.1.5 Classification by Feedback of Users 23
- 3.2 Rocchio 23
- 3.3 K-Nearest Neighbor kNN 24
  - 3.3.1 Voting with cosine similarity 24
  - 3.3.2 Nearest Neighbour with Inverted Index 25
- 3.4 Bayesian probabilistic models 25
  - 3.4.1 Bernoulli Multivariate or Binomial Multivariate 27
  - 3.4.2 Multinomial Naive Bayes 28

3.4.3 Multinomial Variant 29
3.4.4 Comparison 30
3.4.5 Computational Complexity 30
3.4.6 Underflow prevention 31
3.5 Accuracy Confidence Interval 31
3.5.1 Comparing two models 32
3.6 Weka 33

4 Natural Language Understanding 33
4.1 POIROT: Phenomena explanation from text 34
4.2 Events for Natural Language Understanding 35
4.3 Graphs and LMs 36

5 Deep Learning 36
5.1 LSTM 36
5.1.1 GRU Gated Recurring Units 37
5.1.2 Bidirectional LSTM 37
5.2 Memory-based DNNs - Neural Turing Machines NTM 37
5.3 Word Embeddings 38
5.3.1 Approaches for Word Representation 38
5.4 Attention 39
5.5 Transformer 39
5.6 BERT - Bidirectional Encoder Representations from Transformers 41
5.7 Variants of BERT 42
5.7.1 RoBERTa - Robustly Optimized 42
5.7.2 BART 42
5.8 Cross-Modal Transformers 43
5.9 Optimizations for Attention 43
5.9.1 Performer 43
5.9.2 BigBird - Sparse Graph Attention 44
5.9.3 Reformer 44
5.9.4 Transformer-XL 45
5.10 Metric Learning 45
5.11 Loss Functions 46
5.11.1 Pairwise Ranking Loss or Contrastive Loss 46
5.11.2 Triplet Loss 46
5.12 Neural Ranking Models 47

1 Text processing

Information Retrieval: methods and algorithms to search for relevant docs wrt user queries in repositories of unstructured docs. It replies to more advanced queries such as: retrieving docs containing terms "Information" adjacent to "Retrieval" or "Relevance" but without "Protocol".

Data selection in data mining is the first step after defining the mining goals. IR offers efficient methods for representation and selection of unstructured data which can be useful for Text Mining.

Boolean representation: in the boolean retrieval model the query expressive power is based on boolean propositions (AND, OR, NOT). **Limits of the vector representation:** with big number of documents and big number of distinct terms we may have a sparse matrix with big memory usage.

Solution: Inverted index. For each term t, it stores a list of all docs containing t. Each doc is identified by a docID. We may use dynamic structures and add terms specifying the insert position. To optimize the query processing, we may visit the lists in increasing order of frequency, starting from the shorter list. When we have more complex Boolean queries, it may be better rewriting using boolean logics (it is more efficient to process intersection before union).

Example of Boolean Query with Inverted Index.

Query: Bruto AND Calpurnia AND Cesare. We select the postings of the 3 terms and apply the AND to the 3 lists (intersection of lists).

1.1 Building the inverted index

- **Generalized Inverted Index GIN**: it contains an index entry for each term, with a compressed list of matching locations. Best for static data because lookups are faster.
- **Generalized Search Tree GiST**: generalization of several traditional indexes, like B-tree, with no limitation in text size, moreover it allows using arbitrary predicates. Best for dynamic data as it is faster to update under 100k terms.

DB Management Systems (DBMS) are equipped with such indexes for speeding up full text searches. **NoSQL** born for text manipulation, but less efficient than Relational. **Relational RDBMS** are incorporating efficient text operations, for instance now PostgreSQL has GIN, GiST, JSON data type like NoSQL.

1.1.1 Text tokenization

A token is a sequence of chars followed by a delimiter, which is one or more chars, usually the delimiter is the space. Generic problems with French (L'ensemble), German (long composed nouns), Chinese, Japanese (no spacing), Arabic (from right to left).

1.1.2 Stop Words

Very frequent terms but with a scarse semantic content: conjunctions, prepositions, articles.

A priori ignored for most of text mining tasks. Sometimes included in NLP for the semantic analysis of sentences. Used in exact match searches.

1.1.3 Word Normalization

Means that different instances of a word are reduced to the same form (U.S.A == USA). A solution is the definition of equivalence classes of terms (e.g. elimination of punctuation and hyphens). Accented words are transformed to terms without accents.

When performing tokenization and normalization we need to identify the language and we may use search expansion as an alternative to equivalence classes (when searching for window we search also for windows and Windows) - usually leads to more expressive power but is less efficient.

1.1.4 Uppercase and Lowercase, synonyms and phonetic equivalence

Usually all uppercase chars are reduced to lowercase. Often is better to reduce all to lowercase as the users, in short texts, don't care much about using uppercase chars.

Treatments of synonyms an homonyms with class of equivalence (*car = automobile*). The equivalence is applied as rewriting of terms (automobile indexed as *car-automobile*) or by expanding the search.

Term equivalence by phonetic heuristics: the idea is to generate for each term a phonetic hash so that terms with a similar sound have the same hash code - **soundex algorithms**.

1.1.5 Lemmatization

Lemmatization is the process of reducing the inflected forms of a word to a single dictionary term called **lemma** or base form. e.g. *democratic, democratization* $\rightarrow$ *democracy*

In this way, searching for words in a given inflected form can also return documents with the same words in different inflected forms.

Differently from other methods, the lemma is always a word belonging to the dictionary - it is employed the morphological analysis.

1.1.6 Stemming

Reduction of terms their "root" called theme where the goal is to reduce co-related words to the same root. e.g. *andare, andai, andò* $\rightarrow$ *and* even if it is not a valid morphological form of the word.

It is based on heuristics that cut suffixes. The reduction rule depend from the language, There are several algorithms.

Porter Algorithm: the most popular algorithm for the English language. Empiric results show that it is as effective as other more complex algorithms - it includes 60 suffixes.

It contains rules and 5 progressive reduction phases. Each phase, sequentially applied contains a set of rules. e.g. *hopefulness* → *hopeful* → *hope*. General rule: in each phase, and for each word, are applied the rules that maximizes the suffix length to be removed.

Phase 1: *sses* → *ss*, *ies* → *i*, *ational* → *ate*, *tional* → *tion*. Rules are sensitive to word lengths.

Lovins algorithm: the longest suffix removal in a single step, it has a dictionary with 294 suffixes, each of them is associated with several exceptions.

In short, given an inflected word, if it ends with one of the existing suffixes **s** in its dictionary - if the word is not an exception wrt the suffix s, then it removes s. The idea is that any exception is a word that contains only apparently the suffix and not as a real morphological constituent.

In IR, not always normalization and stemming help. In English, results are not always consistent. Given the same searches, they generally increase retrieved documents, both relevant and unrelevant. Some searches lose precision, but we get much more benefit in Spanish, German, Finnish.

About Boolean search: often they generate extreme results, either no result or too large. Those are good for expert users who know the text set and are capable of formulating precise searches. Also, expressing complex boolean searches is not easy for most of people and results cannot be ordered. PRO: it is very efficient. **Limits:** results are not ordered/ranked, only exact match, not proximity search - e.g. find GATES near MICROSOFT (requires index terms and their position in docs), no semi-structured searches (find documents where (author=Jim Gray) AND (abstract contains transaction), no searches by frequency (find docs with at least 3 times "Text", wildcards.

1.2 Scoring in Ranked Retrieval Models

The goal is to return a list of documents in an order as relevant as possible wrt the user query. Several methods and algorithms to rank each document in the interval [0,1]. Results with a large number of answers are no longer a problem being ordered by relevance. The more the term is frequent in the document, the more the score of the document should be high.

1.2.1 Jaccard Coefficient

Measures the grade of intersection of 2 sets A and B:

$$Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

$$Jaccard(A,A) = 1 \tag{2}$$

$$Jaccard(A,B) = 0 \quad if A \cap B = 0 \tag{3}$$

The distance $J_d(A,B) = 1 - Jaccard(A,B)$ is a metric and satisfy the the following conditions given x, y, z in X:

- Not-negativity: $d(x,y) \geq 0$
- $d(x,y) = 0$ iff $x = y$
- Simmetry: $d(x,y) = d(y,x)$
- Triangular inequality: $d(x,z) \leq d(x,y) + d(y,z)$

It does not rely on the frequency of terms and rare terms are most informative than frequent ones, but Jaccard ignores this aspect. **Jaccard and trigrams are suited for short texts.**

Example. Query: ides of march. Doc1: Cesare died in march. Doc2: the long march.

$J_d(Query, Doc1) = 1 - 1/6 = 5/6$.

$J_d(Query, Doc2) = 1 - 1/5 = 4/5$ (most relevant)

1.3 Bag of words model

Term-document matrix: produce a matrix where each document is a binary vector or use also the number of occurrences of a term in each document.

John is faster than Mary and *Mary is faster than John* produces identical vectors and ignores the word order. It's a **backward step** wrt to other solutions who takes into account the positional index or uses paragraph vectors in deep learning. This is not suited for Natural Language Understanding.

Frequency of terms

The term frequency $tf_{t,d}$ of a term t in a doc d is the number of occurrences of t in d.

The tf value should be used to determine which documents are more relevant for a given query. The relevance does not grow linerarly with the frequency of terms.

Logarithmic Weight of Frequency

A better measure is the frequency with log weight of a term t in d which is:

$$w_{t,d} = 1 + log_{10} tf_{t,d}, \quad if tf_{t,d} > 0, \quad tf_{t,d} \in N \qquad 0, else \tag{4}$$

The relevance of doc d for a given query q, is the sum of log frequency over the terms t in both q and d:

$$\sum_{t \in q \cap d} w_{t,d} \tag{5}$$

The relevance is zero when query and document do not share any term.

Relevance of Terms
Rare terms are generally more important than frequent terms. However the tf gives instead more importance to much more frequent terms. The more a query term is rare, the more the doc with such term has higher probability to be relevant. We should reweight the tf to give more importance to less frequent terms in the doc repository.

Inverse Document Frequency IDF
df_t is the number of documents in the corpus with the term t.
idf of t is:

$$idf_t = log_{10}(N/df_t) \tag{6}$$

We use the log because the relevance is not inversely proportional wrt frequency. idf does not take into account the repetition of t in each document of the corpus.

1.3.1 TF-IDF

It is a standard in IR since '70s to compute the relevance of each term withing a document according to the corpus. It is the product:

$$w_{t,d} = log(1 + tf_{t,d} * log(N/df_t) \tag{7}$$

In conclusion, the term relevance in any doc *increases* with the number of occurrences in the doc and *decreases* with the number of occurrences of the term in all documents of the corpus.

Relevance of a Doc d wrt a Query q:

$$Rank(q, d) = \sum_{t \in q \cap d} tf - idf_{t,d} \tag{8}$$

There are several versions: tf computed with and without logarightm, weighting of query terms, tfc which takes into account the length of docs, itc that better smoothes the high difference of frequencies, IDF modified by computing the entropy of t.

As a result, we have documents as vectors of reals. It's a multi-dimensional space where each term is an axis and each doc a point in such space, and its coordinates are the tf-idf values. It has **high dimensionality and sparsity.**

We need a ranking method to overcome the boolean model where each match can be only true or false. We may compute the euclidean distance among vector but this depends on the length of vectors.

We need a monotone decreasing function that returns a value between 0 and 1, where 0 means the maximum angle (180) and 1 when angle is 0 which is the maximum similarity.

1.3.2 Cosine similarity

$cos(\overrightarrow{q}, \overrightarrow{d})$ is the cosine of the angle between vector $\overrightarrow{q}$ and $\overrightarrow{d}$:

$$cos(\overrightarrow{q}, \overrightarrow{d}) = \frac{\overrightarrow{q} \bullet \overrightarrow{d}}{||\overrightarrow{q}|| \quad ||\overrightarrow{d}||} \tag{9}$$

where:

$$\overrightarrow{q} \bullet \overrightarrow{d} = \sum_{i=1}^{k} q_i * d_i \tag{10}$$

k = terms of the vector and the Norm L_2 of the vector is:

$$||\overrightarrow{q}|| = \sqrt{\sum_{i=1}^{k} q_i^2} \tag{11}$$

in which q_i can be the tf-idf of term i in the query and d_i can be the tf-idf of the term i in the document.

Vector Normalization
A vector can be normalized diving each component by its norm L_2:

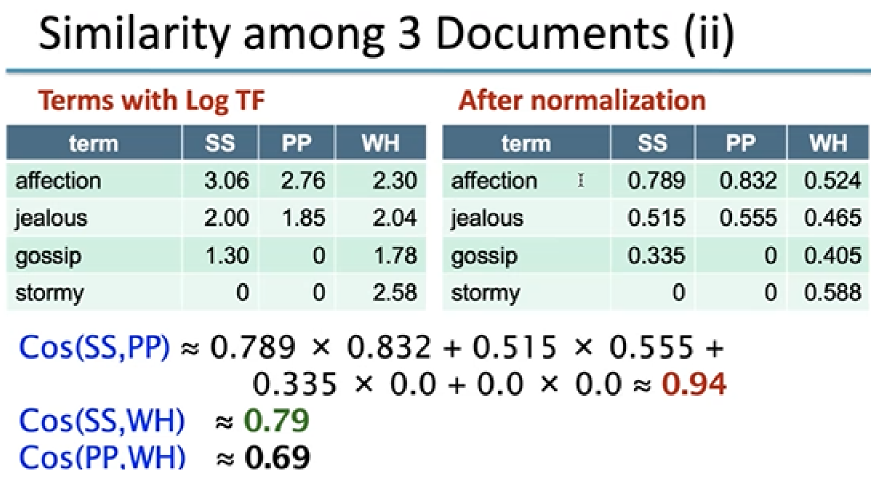
$$||\overrightarrow{x}||_2 = \sqrt{\sum_{i} x_i^2} \tag{12}$$

$$\overrightarrow{x}^u = \frac{\overrightarrow{x}}{||\overrightarrow{x}||_2} \tag{13}$$

The **result of the vector is unitary** and the impact on doc d and d' become equal. *Short and long documents in this wasy can be comparable.*

The similarity based on the cosine of vectors q and d normalized is **simply their scalar product**.

$$cos(\overrightarrow{q}^u, \overrightarrow{d}^u) = \sum_{i=1}^{k} q_i^u d_i^u \tag{14}$$



Similarity among 3 Documents (ii)

Terms with Log TF

term	SS	PP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
stormy	0	0	2.58

After normalization

term	SS	PP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
stormy	0	0	0.588

$\mathrm{Cos}(SS,PP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$

$\mathrm{Cos}(SS,WH) \approx 0.79$

$\mathrm{Cos}(PP.WH) \approx 0.69$

1.3.3 PageRank

The basis algorithm of Google. Measures the relevance of web pages from the number of entry links rather than only their contents. Entry links are in turn weighted according to the PageRank of pages to which links belong to.

It extracts from links among Web pages the implicit knowledge of pages' importance according to the relevance given by the authors that linked them. The knowledge is a probability distribution funded on **random walk**.

1.3.4 Meaning of words

Up to now the similarity among texts has been based on lessical matching of their words.

Definition: the idea that is represented by a word, phrase, the idea that a persion wants to express by using words, signs, the idea expressed in a work of writing, art.

The meaning of a word depends on its relations to other words.

Text similarity using Wordnet

WordNet has a total of 29 kind of relationships (hypernyms, synonyms, merenomy...). Example. painting IS-A graphic_art. car is synonym with auto, automobile, machine, motorcar.

A directed graph where each node is an english word and each edge an oriented relation between 2 words. e.g. sculpture and painting are independent, but instead in wordnet they share a common ancestor which is art.

Drawback: it is computationally expensive and has a limited dictionary. Word embeddings are overcoming such issues.

1.4 Evaluation of results

An approach commonly used to measure the docs relevance wrt a query:

- one or more reference sets of documents
- a reference set of queries
- known binary evaluations for each query and the set of documents in order to determine relevant/irrelevant doc

Human experts evaluate for each query which docs are relevant/irrelevant.

1.4.1 Accuracy

Number of relevant retrieved docs + irrelevant not retrieved) /all docs in the corpus.

Confusion matrix			
	Retrieved	Not Retrieved	doc in the corpus
Relevant	True Positives TP	False Negatives FN	Relev = TP+FN
Irrelevant	False Positives FP	True Negatives TN	Irrelev = FP+TN

Accuracy = (TP+TN)/(TP+TN+FN+FP) but this measure is insufficient with datasets that have classes with a number of unbalanced instances.

1.4.2 F1-measure

Precision: fraction of docs retrieved that are relevant: $P(relevant|retrieved) = tp/(tp + fp)$

Recall: fraction of docs relevant that have been retrieved: $P(retrieved|relevant) = tp/(tp + fn)$

Generally in a good system, the recall tends to decrease as the precision increases and viceversa.

Example. A corpus with 10200 documents of which 100 revelant for a given query, but the query ahcieves 100 docs irrelevant. So we have 100 false negavites and 100 false positives. 0 true positive and 10000 true negative.

Accuracy = (0+10000)/10200 = 0.98

Precision = 0/(0+100) = 0

Recall = 0/(0+100) = 0

A measure that combines them is the F-measure:

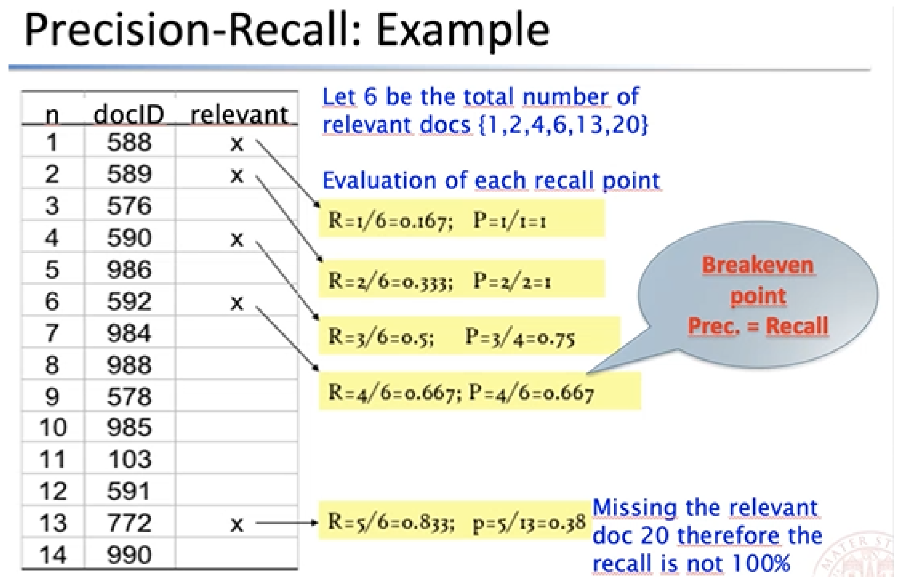
$$F - measure = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{15}$$

in which if $\beta = 1$ becomes the F1-measure known also as the armonic measure:

$$F1 - measure = 2 * \frac{PR}{P + R} \tag{16}$$

Given a system and a set of queries with their results, we plot for each query a graph precision-recall with the first k answers. In general it is plotted the average graph from the graphs of queries. **Standard TREC:** 11 recall levels with 0.1 interpolated increments (from 0 to 1)

The **Breakeven point** of precision-recall is the interpolated value such that precision and recall are identical.



Precision-Recall: Example

n	docID	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	
13	772	x
14	990	

Let 6 be the total number of relevant docs {1,2,4,6,13,20}

Evaluation of each recall point

R=1/6=0.167; P=1/1=1

R=2/6=0.333; P=2/2=1

R=3/6=0.5; P=3/4=0.75

R=4/6=0.667; P=4/6=0.667

Breakeven point Prec. = Recall

R=5/6=0.833; p=5/13=0.38

Missing the relevant doc 20 therefore the recall is not 100%

1.4.3 R-Precision

R-precision: let R be the number of docs relevant for a given query, R-precision is the number of docs relevant in the first R doc retrieved divided by R.

1.4.4 Mean Average Precision

Mean Average Precision MAP: of a query q_j is the average of R-precision from 1 to m_j relevant doc of q_j. The MAP of a set Q of queries is the following:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{17}$$

Let q_1 and q_2 be two queries to which corresponds 10 and 8 relevant docs respectively, in the corpus D. MAP approximates the area average of the precision-recall graph of a set of queries. Each query has the same weight in the MAP, also with large difference in the number of relevant docs among queries.

1	2	3	4	5	6	7	8	9	10
x		x	x	x		x	x		

1	2	3	4	5	6	7	8	9	10
	x		x		x	x			

$$
\begin{aligned}
MAP(\{q_1,q_2\}) = & \frac{1}{2}(\frac{1}{10}*(\frac{1}{1}+\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\frac{4}{5}+\frac{4}{6}+\frac{5}{7}+\frac{6}{8}+\frac{6}{9}+\frac{6}{10})+ \\
& \frac{1}{8}*(\frac{0}{1}+\frac{1}{2}+\frac{1}{3}+\frac{2}{4}+\frac{2}{5}+\frac{3}{6}+\frac{4}{7}+\frac{4}{8})) = \\
& = \frac{0.711+0.413}{2} = 0.562
\end{aligned} \tag{18}
$$

2 Dimensionality Reduction

Reducing the number of features allows the application of greater number of ML tools and algorithms. Less features also reduce the training time. The resulting learning models are less complex and efficient, often more general as well.

Datasets usually contain redundant or irrelevant variables. There are two wide areas of research and technologies: **Feature selection** (select a subset) and **Feature extraction** (transformation of the data representation)

The easiest feature selection in text mining involve using most frequent terms Term Frequency, or most relevant with TF-IDF.

The goal is to estimate the feature utility in the classification task.

2.1 Mutual Information

Measures the reciprocal dependency between two variables, usually the dependence among terms and classes. Calculates how much information a term t and a class share. Is 0 if independent.

U is the random variable related tot he document and if it is $e_t = 1$ then the term t is in the document, else $e_t = 0$. C is the random variable related to the class and if $e_c = 1$ the document is in the class c, else $e_c = 0$

$$
I(U;C) = \sum_{e_t \in 1,0} \sum_{e_c \in 1,0} P(U = e_t, C = e_c) log_2 \frac{P(U = e_t, C = e_c)}{P(E = e_t)P(C = e_c)} \tag{19}
$$

In the denominator we are assuming the two are independent.

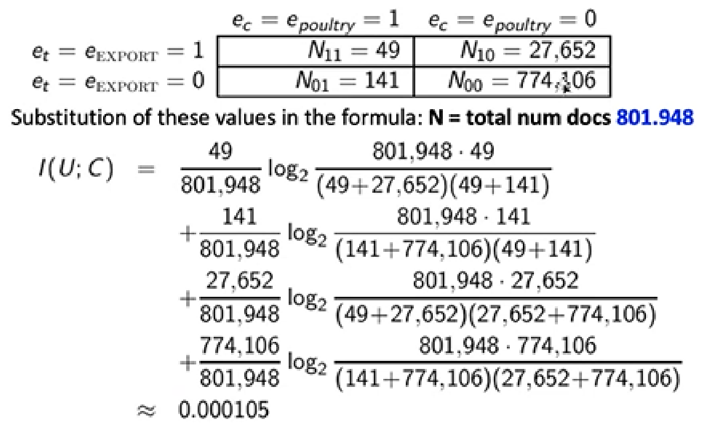
To compute it we have to compute 4 values:

- N_{10} number of docs that contain t ($e_t = 1$) and are not in the class c ($e_c = 0$)
- N_{11} number of docs that contain t ($e_t = 1$) and are in the class c ($e_c = 1$)
- N_{01} number of docs that do not contain t ($e_t = 0$) and are in the class c ($e_c = 1$)
- N_{00} number of docs that do not contain t ($e_t = 0$) and are not in the class c ($e_c = 0$)
- $N = N_{11} + N_{10} + N_{01} + N_{00}$

$$I(U;C) = \frac{N_{11}}{N} log_2 \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N} log_2 \frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N} log_2 \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N} log_2 \frac{NN_{00}}{N_{0.}N_{.0}} \tag{20}$$

where $N_{1.} = N_{10} + N_{11}$ number of doc with the term t independently from any class $e_c \in 0, 1$.

Example. MI from Reuters with the term Export and class Poultry



	$e_c = e_{poultry} = 1$	$e_c = e_{poultry} = 0$
$e_t = e_{\text{EXPORT}} = 1$	$N_{11} = 49$	$N_{10} = 27{,}652$
$e_t = e_{\text{EXPORT}} = 0$	$N_{01} = 141$	$N_{00} = 774{,}106$

Substitution of these values in the formula: N = total num docs 801.948

$$\begin{aligned} I(U;C) &= \frac{49}{801{,}948} \log_2 \frac{801{,}948 \cdot 49}{(49+27{,}652)(49+141)} \\ &+ \frac{141}{801{,}948} \log_2 \frac{801{,}948 \cdot 141}{(141+774{,}106)(49+141)} \\ &+ \frac{27{,}652}{801{,}948} \log_2 \frac{801{,}948 \cdot 27{,}652}{(49+27{,}652)(27{,}652+774{,}106)} \\ &+ \frac{774{,}106}{801{,}948} \log_2 \frac{801{,}948 \cdot 774{,}106}{(141+774{,}106)(27{,}652+774{,}106)} \\ &\approx 0.000105 \end{aligned}$$

2.2 χ^2 test (chi-square)

A popular statistical hypothesis test. Once a confidence level is set, it indicates if the difference between observed and expected data is significant or the result of chance. Each time we have two neural models, we may compare them using the chi-square test.

In feature selection: Let N be the observed frequency and E the expected frequency, we suppose the two events are independent (**null hypothesis**). Let $e_c, e_t \in 0, 1$ have the same meaning of before. The following estimates the difference among the observed and expected values for t and c in D:

$$\chi^2(D, t, c) = \sum_{e_t \in 1,0} \sum_{e_c \in 1,0} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{21}$$

Example of before (MI).

E_{11} is the expected doc number with the term t belonging to the class c, supposing t and c are independent.

$$E_{11} = |D| * P(t) * P(c) = |D| * \frac{N_{11} + N_{10}}{|D|} * \frac{N_{11} + N_{01}}{|D|} \tag{22}$$

and compute the others like this.

$E_{11} \approx 6.6$	$E_{10} \approx 27,694.4$
$E_{01} \approx 183.4$	$E_{00} \approx 774,063.6$

The formula measures the **deviation** among expected and observed value, so by applying the chi-square formula we get $\chi^2(D,t,c) \approx 284$ The greater χ^2, the lower the prob that the hypothesis of independence between class and term holds, a threshold is needed. This inference is possible with 2 normal and independent events because in this case $X^2 \sim \chi^2$, where χ^2 is the chi-square distribution.

Accepting or Rejecting the null hypothesis. First we have to compute the freedom degree: number of columns - 1 * number of rows - 1 (which means the number of classes - 1 * number of terms - 1). Compute a χ^2 distribution table with respective p-value. Before calculating this we set a "certainty" confidence (for medical domain 99%).

We produce with 2 events an $X^2 = 5$ and consult our distribution table. Our chi-value might be between 2 numbers: if our certainty value set was 95%, we had 5 which is ≥ 3.84 which corresponds to the p-value 0.05, then our confidence is $\geq 95\%$ hence the null hypothesis (difference between expected and observed values is random), is rejected and the events are not independent. Otherwise if we had a confidence of 99%, our value is ≤ 6.63 which correspond to 0.01 p-value. So we don't have enough confidence and the null hypothesis is accepted saying the two events are independent and the difference among expected and observed values is random.

If the null hypothesis is rejected, t is a good feature for representing the class c.

When the matrix is 2x2, the χ^2 can be computed as follows.

$$\chi^2(t,c) = \frac{N * (AD - CB)^2}{(A+C) * (B+D) * (A+B) * (C * D)} \tag{23}$$

where $N = A + B + C + D, A = \#(t,c), B = \#(t,-c), C = \#(-t,c), D = \#(-t,-c)$

A new ranking method. The chi-square values computed for each term are normalized, so comparable. The rank of each term corresponds to the weighted average of their χ^2 values with respect to the classes.

$$\chi^2_{avg}(t) \sum_{i=1}^{|C|} P(c_i)\chi^2(D,t,c_i) \tag{24}$$

The second term is the chi value calculated for each class. Or alternatively its maximum value

$$\chi^2_{max}(t) = \max_{i=1}^{|C|}\{\chi^2(D,t,c_i)\} \tag{25}$$

In both cases, the first k terms with larger values are selected.

With χ^2 we can also evaluate whether two sequences of values are or not **homogeneous**. Compare two kind of scores on sentiment analysis from tweets.

Inadequate for small observed and expected values (lesser than 30 or their sum is lower than 200). There are some correction methods: for tables 2x2 there's the **Yates correction**, or Kolmogorov-Smirnov.

As the number of test performed increases, the probability of the total error increases: in 1000 reject hypothesis with 5% error, 50 test are wrong and some features are selected or not wrongly, but this rarely impacts on text classification results. **It is important the ranking (ordering) given by χ^2 to the terms.**

Chi-square with respect to MI gives more importance to rare terms, therefore requires more terms of MI in order to reach the max, but empirically selects better features. Usually, as number of features increases, the classification is less effective.

All those are greedy methods which may select features that do not increase the useful information wrt the features already selected (**redundancy**).

2.3 Latent Semantic Analysis

Document Vector Space Model PROS: lexical matching among terms (with possible partial matching), ranking of results based on similarity features like cosine/jaccard (to easily manage results with a large number of documents), several representations of terms and docs (binary frequency, term weighting, log, tf-idf), enables the use of ML algorithms (even unsupervised), exploitation of geometrical and algebra tools.

Document Vector Space Model has some limits. **Polysemy:** words with different meanings that depend from the context. The vector space model cannot recognize the different meaning and cosine similarity might be greater than the actual so we may have more irrelevant docs with *reduction of precision.* Jaguar is a car, a football team or an animal?

Synonymy: terms lexically different with identical or similar meaning (ship/ boat) - cosine similarity is lower than the actual one and relevant docs are not retrieve with a *reduction of recall.*

LSA transforms the terms-docs matrix by bringing out latent semantic associations (based on lower lexical match) between terms and documents. It maps the matrix in a new vector space with lower dimensions which approximates the original one ignoring the "noise" (irrelevant details).

In the transformed space, terms semantically similar or associated are placed in neighboring positions. It may emerge from co-occurrence of terms in distinct docs.

2.3.1 Trasformation with Singular Value Decomposition SVD - Factorization

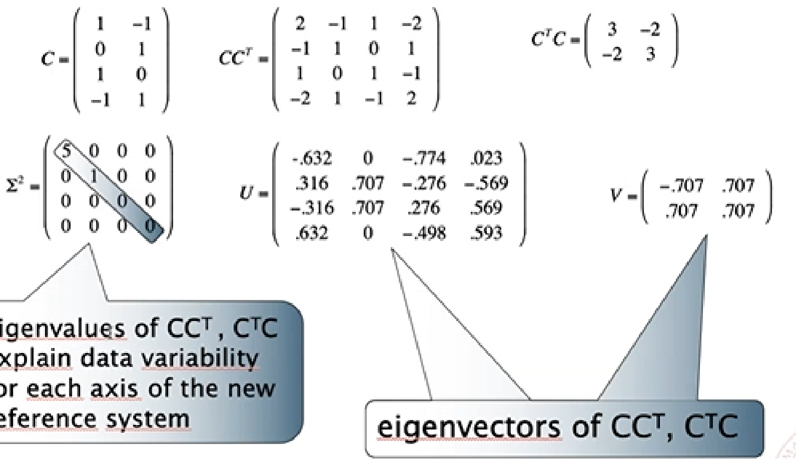
For a matrix C MxN with rank r there exists a factorization as follows:

$$C_{MxN} = U_{MxK}\Sigma_{KxK}V^T_{KxN} \tag{26}$$

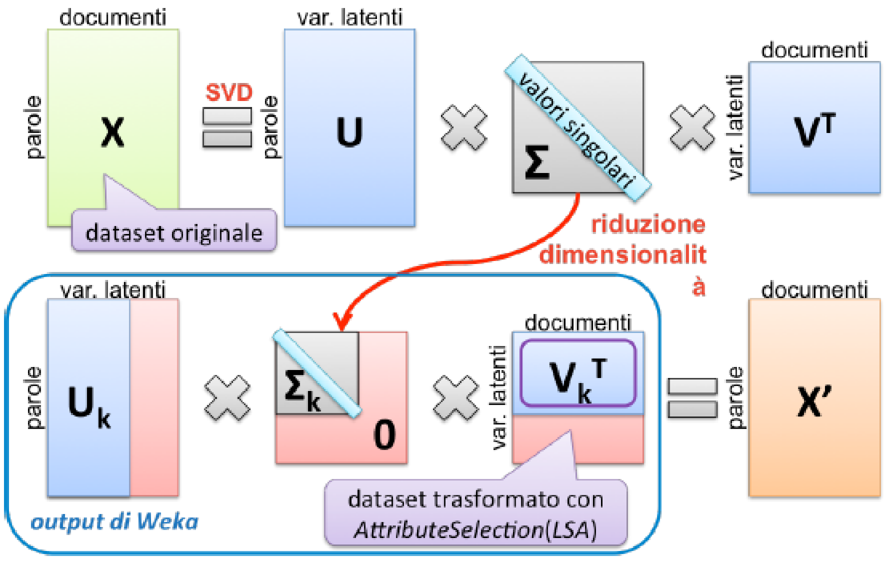
where Σ is the diagonal matrix $K << r$

We may compute the scalar product as a similarity metric and obtain CC^T is the similarity between terms MxM while C^TC is the similarity between documents NxN. When calculating CC^T we obtain $CC^TU = \Sigma^2U$ in which the columns of U are the right eigenvectors of CC^T and the diagonal of Σ^2 contains the eigenvalues λ in descendant order. Analogously the columns of V are the eigenvectors of $C^T C$.

As a result, U and V (eigenvectors) are orthonormal bases of a new multi-dimensional reference system and the sqrt of Σ^2 i.e. $\sigma = \sqrt{\lambda}$ is the data variability of each dimension.


$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}$
$CC^T = \begin{pmatrix} 2 & -1 & 1 & -2 \\ -1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ -2 & 1 & -1 & 2 \end{pmatrix}$
$C^T C = \begin{pmatrix} 3 & -2 \\ -2 & 3 \end{pmatrix}$
$\Sigma^2 = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
$U = \begin{pmatrix} -.632 & 0 & -.774 & .023 \\ .316 & .707 & -.276 & -.569 \\ -.316 & .707 & .276 & .569 \\ .632 & 0 & -.498 & .593 \end{pmatrix}$
$V = \begin{pmatrix} -.707 & .707 \\ .707 & .707 \end{pmatrix}$
eigenvalues of CC^T, C^{T}C explain data variability for each axis of the new reference system
eigenvectors of CC^T, C^{T}C

The singular values of the matrix Σ (eigenvalues) are in descendant order and each of them explains the importance of its related dimension. The first one catch the greater data variation and selecting the first k values allows to remove details to let emerge similarities.

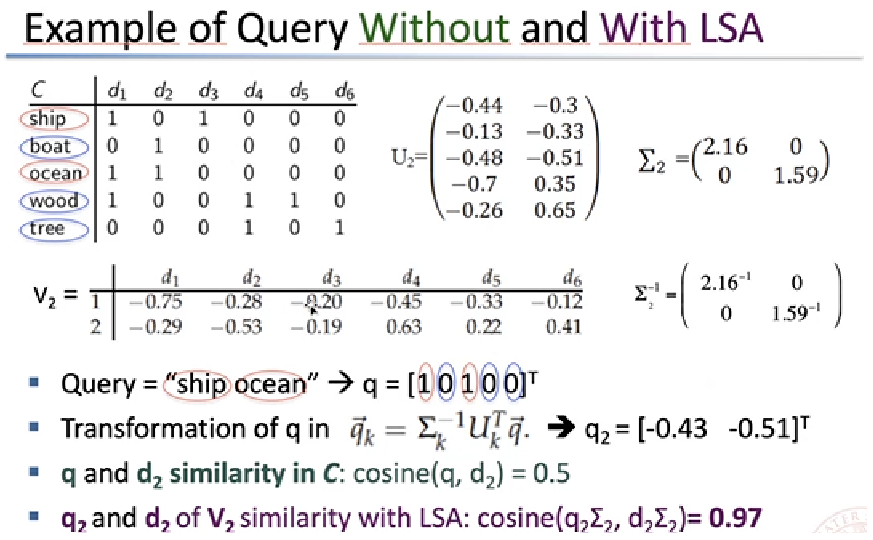


To choose k, we may select a **knee point** which is the curvation radius of the iperbole function interpolating the data of all eigenvalues. $\frac{f''}{(1+(f')^2)^{\frac{3}{2}}}$.

There is no theory to decide which is the best rank, depends on the data and the number of variables in the original space.

Let $X_k = U\Sigma_k V^T$ be the rebuilt terms-docs with rank k. Similarity between:

- Documents v_i, v_j is $cos(v_i\Sigma_k, v_j\Sigma_k$ in which sigma is just a vector (low memory requirement)
- Terms: $(U\Sigma_k)(U\Sigma_k)^T$
- a query q (terms vector in X) and a doc is computed transforming q in a new doc $q_k = \Sigma_k^{-1}U_k^T q$. The similarity between q_k and v_i is: $cos(q_k\Sigma_k, v_i\Sigma_k) == cos(q^T U_k, v_i\Sigma_k)$. The second formula doesn't need a transformation of the query, we can use the original one.
- a term u_i and a doc v_j is given by $cos(u_i\Sigma_k^{1/2}, \Sigma_k^{1/2}v_j)$
- a term u_i and a query q is achieved by transforming the query and doing the same as before: $cos(u_i\Sigma_k^{1/2}, \Sigma_k^{-1/2}U_k^T q)$



Example of Query Without and With LSA

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

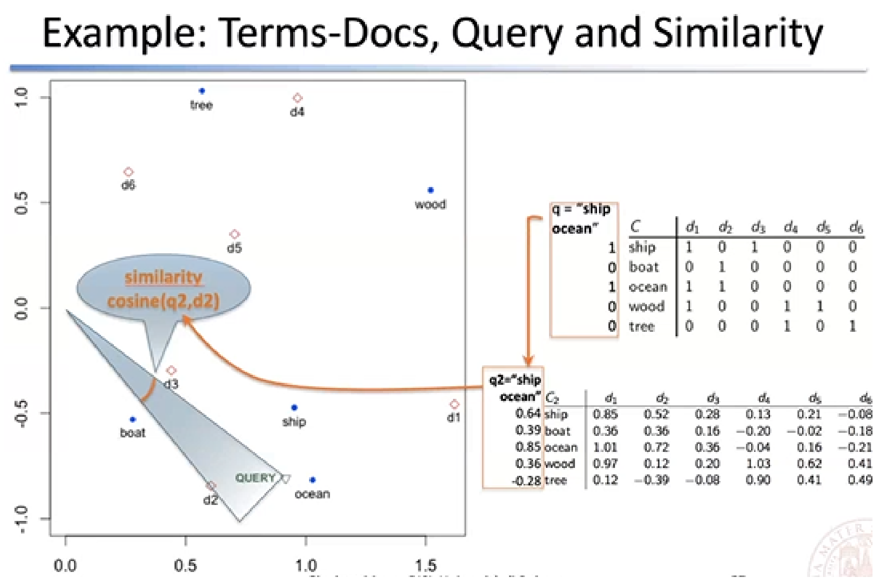
$$U_2 = \begin{pmatrix} -0.44 & -0.3 \\ -0.13 & -0.33 \\ -0.48 & -0.51 \\ -0.7 & 0.35 \\ -0.26 & 0.65 \end{pmatrix} \qquad \Sigma_2 = \begin{pmatrix} 2.16 & 0 \\ 0 & 1.59 \end{pmatrix}$$

$V_2 =$

	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41

$$\Sigma_2^{-1} = \begin{pmatrix} 2.16^{-1} & 0 \\ 0 & 1.59^{-1} \end{pmatrix}$$

- Query = "ship ocean" → q = [1 0 1 0 0]^T
- Transformation of q in $\vec{q}_k = \Sigma_k^{-1} U_k^T \vec{q}$. → q_2 = [-0.43 -0.51]^T
- **q and d_2 similarity in *C***: cosine(q, d_2) = 0.5
- **q_2** and **d_2** of **V_2** similarity with LSA: cosine($q_2\Sigma_2$, $d_2\Sigma_2$)= **0.97**



Example: Terms-Docs, Query and Similarity

q = "ship ocean"	C	d_1	d_2	d_3	d_4	d_5	d_6
1	ship	1	0	1	0	0	0
0	boat	0	1	0	0	0	0
1	ocean	1	1	0	0	0	0
0	wood	1	0	0	1	1	0
0	tree	0	0	0	1	0	1

q2="ship ocean"	C_2	d_1	d_2	d_3	d_4	d_5	d_6
0.64	ship	0.85	0.52	0.28	0.13	0.21	−0.08
0.39	boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
0.85	ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
0.36	wood	0.97	0.12	0.20	1.03	0.62	0.41
-0.28	tree	0.12	−0.39	−0.08	0.90	0.41	0.49

It can be applied in Collaborative Filtering and Recommendation System (understanding and predicting users' interests from a large dataset of ther behaviours, purchases, choices), Opinion mining Sentiment Analysis, Speech to text and Natural Language Understanding.

Limits: cannot directly catch the polysemy since each term leads to a single vector in the LSA space, therefore one meaning. But the resulting term vector is the average of its different meanings of the word in the corpus. Not a real drawback in practice. Sometimes there is a predominant meaning over the others and usually they are positioned in the middle between clouds of topics.

The computational cost is that of a matrix MxN: $O(min(mn^2, m^2n))$ since there are two methods, if $n << m$ then $O(mn^2)$ otherwise $O(m^2n)$.

Randomized SVD has a *nlogn* complexity in Python and R. Expressivity limited only by the underlying bag of words model. INCREDIBLE GAIN with same accuracy.

2.3.2 Low-rank Approximation

Given a matrix C, MxN with rank r and a positive integer $k < r$, finding the matrix C_k MxN with rank at most k that minimizes the difference $X = C - C_k$ according to the Frobenius (for matrixes) norm:

$$||X||_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} X_{ij}^2} \tag{27}$$

If it was $k = r$, the Frobenius norm of the difference matrix X would be zero. When $k < r$, C_k is the k optimal low-rank approximation. **With SVD, the $C_k = U\Sigma V^T$ is the optimal k low-rank approximation of C.**

3 Text Classification

To collect updated news, we should periodically execute the same searches and distinguish relevant from irrelevant documents: these are called **standing queries** or **continuous queries** in DBMS. Google Alert is a recent good example of standing queries or spam filtering.

Given a destructured doc, such as tweets, we have to determany the category of d: $\Phi(d) \in C$ where $\Phi(d)$ is the classification function with domain X (set of istances) and co-domain C (set of classes).

It is a supervised problem where we need pre-classified training set of documents and methods to represent destructured documents and methods to extract features.

We may have different scenarios:

- **Binary classification:** Yes or No
- **Multi-class classification:** one category out of many
- **Multi-label classification:** a subset of categories.

This can be applied to scientific sectors, e-mails (different domains), laws, diagnosis of different diseases, multimedia data (music genres and object recognition), bioinformatics.

3.1 Multi-label text categorization

Formalization. Document space $X = \{d_1, d_2, \ldots, d_m\}$. Classes $C = \{c_1, c_2, \ldots, c_n\}$. Space of labels $Y = \{0, 1\}^n$. The set of labels for each d_i is: $y_i = [c_1, \ldots, c_j, \ldots, c_n] \in Y$ and $c_j = 1$ if the label j is relevant for d_i. We have a training set XxY and a target function $\Phi : X \to Y$. The classification of a test instance $d'_i : y'_i = \Phi(d'_i)$.

We have different solutions (not an easy task):

- Transform the problem in a single-label classification in order to use existing multi-class algorithm or use efficiently binary classification algorithms.
- Ad hoc Multi-label algorithms: Neural networks with multi-value output, generalization of AdaBoost, boosting algorithms or hybrid approaches with chain of classifiers.

3.1.1 Tranformation to single-label

- Each existing multi-label combination in the training set becomes a class. We have $2^{|C|}$ possible classes.

  ***Example*:** if we have a doc classified as *fiction* and *romance*, we create a new class called *fiction romance* and assign it to it.
- Transform the original dataset in $|C|$ datasets each with two classes c_i and $-c_i$ and then train $|C|$ binary classifiers. At the end perform an union of classifications.

- Decompose each instance (d_i, y_i) in $|y_i|$ instances, one for each category. The classifier should return a ranking (probabilities) for categories, so we classify and choose categories according to a ranking threshold (hyperparameter)

  Example. If an instance has 2 categories *fiction* and *romance*, we create two instances: one with *fiction* and one with *romance*.

3.1.2 Hybrid solution - Chain of Classifiers (2010)

It is an incremental classification that, for each instance, determines sequentially the possible categories decomposing the problem for each category in two subproblems. The number of classifiers coincide with the number of classes and the i-th classifier knows the preceding outcomes.

Example. The first classifier C_1 checks whether X belong to *romance* or $-romance$. $C_1 : x \rightarrow \{romance, !romance\} = romance$

The second classifier C_2 gets in input X and the output of preceding classifier to check wheter X belongs to *horror* or $-horror$. $C_2 : x \cup romance \rightarrow \{horror, !horror\} = !horror$ and so on.

3.1.3 Accuracy measures for Multi-label

Denote:

- D the set of test documents
- C the set of classes
- $L(d)$ the set of actual categories of the doc d
- $L(d, c) = 1$ if c is among the actual categories of d, else 0.
- $M(d)$ the set of predicted categories of d
- $M(d, c) = 1$ if the model classifies d as the class c, else 0

Accuracy: for each doc the number of categories correctly predicted, divided by the number of correct + wrong categories:

$$accuracy = \frac{1}{|D|} \sum_{d \in D} \frac{|M(d) \wedge L(d)|}{|M(d) \vee L(d)|} \tag{28}$$

Exact match: the number of documents correctly classified in each categories, divided by the total number of documents:

$$EM = \frac{|\{d \in D : M(d) = L(d)\}|}{|D|} \tag{29}$$

Hamming accuracy: the percentage of couples doc-category correctly predicted. It is in the middle of accuracy and EM. Focuses on C (set of classes) not D.

$$Hamming = \frac{1}{|D| * |C|} \sum_{d \in D} |\{c \in C : M(d, c) = L(d, c)\}| \tag{30}$$

Precision: the number of documents correctly predicted as c divided by all documents classified as c, followed by the average precision among all classes:

$$precision(c) = \pi(c) = \frac{|\{d \in D : c \in M(d) \wedge L(d)\}|}{|\{d \in D : c \in M(d)\}|} \tag{31}$$

$$precision = \pi = \frac{1}{|D|} \sum_{d \in D} \frac{|M(d) \wedge L(d)|}{|M(d)|} \tag{32}$$

Recall: number of documents correcly predicted as c divided by all docs belonging to c, followed by the average recall among all classes:

$$recall(c) = \rho(c) = \frac{|\{d \in D : c \in M(d) \wedge L(d)\}|}{|\{d \in D : c \in L(d)\}|} \tag{33}$$

$$recall = \rho = \frac{1}{|D|} \sum_{d \in D} \frac{|M(d) \wedge L(d)|}{|L(d)|} \tag{34}$$

F-measure: we have two different F1: macroaveraged and microaveraged. **Macroaveraged F1** is the arithmetic average of F1-measure classes, while **microaveraged** is the average of F1 measures of classes weighted with their cardinality.

In macroaverage, larger and smaller classes have the same importance since they are normalized. In microaverage, classes do not depend on size and importance is not given to the size of classes.

$$macroaveraged = \frac{1}{|C|} \sum_{c \in C} \frac{2\rho(c)\pi(c)}{\rho(c) + \pi(c)} \tag{35}$$

$$microaveraged = \frac{2\rho\pi}{\rho + \pi} \tag{36}$$

3.1.4 Search Engines

Search engines find categories of the information they index, such as docs, web pages. Each category usually is a topic organized both hierarchically and multi-label (Yahoo! Directory and DMOZ). They also use meta-level categories.

DMOZ is an open directory project with more than 100.000 authors and 133 languages, in many formats like postgres and prolog. It is used by AOL search, Google, Netscape, Yahoo.

It is important because this is build by human and classification of documents is of high quality. It is costly and hardly scalable. The procedure may take from several weeks to years before being reviewed by them.

Another service is MeSH (Medical Subject Headings) which provides us taxonomies for specialized subjects. Useful to train model and force them the relationship that we know - heart valves are related to heart, cardiovascular system. Employed in PubMed (the largest medical repository).

It can be done also by **hand coded rules** - the doc is given a category if the doc contains a combination of words - it needs continuous improvement of the rules, which means high maintenance costs.

3.1.5 Classification by Feedback of Users

User labels some documents as relevant/irrelevant with respect to an information need. The IR system uses these choices to build and improve the model - they reweight the terms in the vector space with a probabilistic model.

- D_r set of known relevant docs where t_k is a term
- D_{rk} subset of relevant docs containing t_k
- D_{nr} set of irrelevant known docs
- D_{nkr} subset of irrelevant docs containing t_k

$P(t_k|Relevant) = |D_{rk}|/|D_r|$ and $P(t_k|Irrelevant) = |D_{nrk}|/|D_{nr}|$

3.2 Rocchio

Model each document as a vector of TF-IDF values. For each category of documents, let's compute the **vector prototype** by summing together their vectors of the training set, divided the number of docs in the category. **Prototype:** centroid of documents belonging to the class.

Test: a new test document is classified in the category with the closest prototype according to the cosine similarity.

$$\vec{\mu}(c))\frac{1}{|D_c|}\sum_{d\in D_c}\vec{v}(d) \tag{37}$$

where D_c is the set of all documents belonging to class c and v(d) is the vector d. The centroid is not a unitary vector, even if input vector are unitary. Each Rocchio prototype is a simple generalization of documents contained in each class. It is not required to normalize in case the distance function is the cosine similarity (already normalized since the two vectors are divided by the norm)

Limits: the prototypes tend to get closer in data sets with non-contiguous categories. Rocchio is simple and often effective in test mining problems but not other mining domains.

It can just separate data using linear methods. Text mining problems are related to datasets with thousands-millions dimensions and it is easier to find a linear separation in a high-dimensional space (rather than low-dimensional) - by increasing the number of dimension, the probability of finding a separation increases.

3.3 K-Nearest Neighbor kNN

Classifies any document d into a class c as follows: first it defines the k-neighborhood N of d, namely the k documents closest to d.

It counts the number of documents i into N that belong to the class c. Then it estimates the probability that d belongs to c, namely $P(c|d) = i/k$. Finally the class c assigned to d is the one that maximizes the probability $argmax_c P(c|d)$.

It is also used **in Information Retrieval** and known as case-based learning, memory-based learning, lazy learning. Despite its simplicity, it is effective as the best classifiers. In IR, the search query which is represented as a point to be classified, is used to retrieve all docs of the category that k-NN has selected for it. The result (list of retrieved documents) can be ranked/ordered by relevance on the basis of their distance to the query or the category centroid.

Hyperparameter k: by using 1, the error increase due to scarce statistical significance since it can be just noise or an outlier. Tipical values used are odd numbers (3,5,7) to avoid parity among classes.

Usually the value of k depends from the training set, the bigger data sets the bigger k can be chosen. Used for non-linear problems. Borders can have arbitrary shapes, but usually generates polyhedra.

The efficacy depends on the similarity metric used. The simplest distance function for multi-dimensional continuous space is the **Euclidean distance**, since when vectors are normalized it is equivalent to cosine similarity. For multi-dimensional binary space is the **Hamming function** that measures the number of different features. **On textual data, the cosine similarity of TF-IDF among vectors is one of the most effective.**

3.3.1 Voting with cosine similarity

We can also **add a voting mechanism with cosine similarity**. The class c assigned to a doc dd is the one with the large number of instances among the k instances closest to d.

The vote criterion to determine the class can be based on cosine similarity between d and the k instances:

- $v(d)$ is the vector representing the doc d
- $v(d)'$ is the vector representing one of the k instances
- $I_c(d')$ is 1 iff d' belongs to c_i, else is 0
- $S_k(d)$ is the set of the k instances closest to d

$$score(c, d) = \sum_{d' \in S_k(d)} I_c(d') cos(\overrightarrow{v}(d'), \overrightarrow{v}(d)) \tag{38}$$

The doc d is assigned to the class with better score.

kNN tends to better manage non-contiguous categories wrt Rocchio. Linear classifiers such as Rocchio and Naive Bayes do not achieve in general good results. kNN behaves good by only opportunely tuning k with the training set.

3.3.2 Nearest Neighbour with Inverted Index

In naive way, finding the nearest neighbors of a doc d requires a search of linear cost in complexity in the set $|D|$ of all documents: $O(|D| * M_{ave} * M_d)$ where M_{ave} is the average number of terms in a document and M_d is the number of terms of doc d.

It is not scalable for big datasets.

Finding the k nearest neighbors is like finding the best k docs using the test document as a query against a database of training docs. The **Inverted Index** is an efficient way to find the k nearest neighbors: $O(B * M_{ave} * M_d)$ where B is the average number of training documents with at least a term of the test document in general $B << |D|$.

With this we have in the RAM only the vocabulary and check the IDs of documents that have the term in it.

3.4 Bayesian probabilistic models

Given a document d of unknown category. Let C be the hypothetical category of d, $P(C|d)$ is the **posterior probability** of the class C conditional to d. Probability that d with some terms belongs to a class C.

$P(C)$ is the a **priori probability**, that a news randomly extracted is of the class C.

$P(d|C)$ is the posteriori probability of d conditional to C, that a news of class C contain the words of the doc d.

For a document d and a class C, the probability that the doc d belong to C is (posterior probability):

$$P(C|d) = \frac{P(d|C)P(C)}{P(d)} \tag{39}$$

Document classification with Bayes: finding the most probable category $c_i \in C$ of a document d on the basis of the contained words x_i:

$$c_{MAP} = argmax_{c_j \in C} P(c_j|x_1, x_2, \ldots, x_n) = argmax_{c_j \in C} \frac{P(c_j|x_1, x_2, \ldots, x_n)P(c_j)}{P(x_1, x_2, \ldots, x_n)} \tag{40}$$

The denominator can be ignored, in fact it is constant being the same for each class. So MAP is the max posterior probability - the most probable category:

$$c_{MAP} = argmax_{c_j \in C} P(c_j|x_1, x_2, \ldots, x_n)P(c_j) \tag{41}$$

We have three different probabilistic models. All these Bayesian model are based on the **assumption of Conditional Independence**.

To classify a document we compute $\forall j P(C_j)P(x_1, x_2, \dots, x_n|C_j)$. The a priori probability $P(C_j)$ is estimated by the category frequency in the training set. While the other term is equal to $P(x_1|C_j)P(x_2|C_jx_1)\dots P(x_n|C_j, x_1, \dots, x_{n-1})$ which in the binomial model has complexity $O(2^n * |C|)$, while in multinomial one is even greater due to x_i not binary. Also the number of features in text mining is a lot bigger rather than data mining with structured data.

So we make the conditional independence assumption: the joint probability of attributes are considered equal to the multiplication of single probabilities:

$$P(x_i|C_j, x_1, \dots, x_{i-1}) = P(x_i|C_j) \tag{42}$$

This means that every term in the document is completely independent from its previous ones. In general this is wrong since they depend each other, but we still get good results.

Problem-Drawback: missing words in a topic of training set

If we want to classify a doc with a new word not contained in any doc in the training set, we will get as result 0 since the we compute for each class the probability the doc belongs to a class by using ALL the words as follows:

$$argmax_c \hat{P}(c) \prod_i \hat{P}(x_i|c) \tag{43}$$

Differently from Data Mining, we have structured data and we have the same features in training and test set. This is not guaranteed here.

A solution is **Smoothing**, namely reducing the probability of observed events and increasing the probability of not observed events.

We may add $+1$ and $+k$ to assume uniform distribution for not observed event:

$$\hat{P}(x_i|c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k} \tag{44}$$

where k is the number of values in X_i and depends on the type of model - in Bernoulli $k = 2$ since it has 2 values while in Multinomial $k = |Vocabulary|$.

There is another version which is better in case of misspelling:

$$\hat{P}(x_{i,k}|c_j) = \frac{N(X_i = x_{i,k}, C = c_j) + mp_{i,k}}{N(C = c_j) + m} \tag{45}$$

where m is a smoothing factor and $p_{i,k}$ is the frequency of $x_{i,j}$ in the corpus - which is 0 in case of misspelling.

Unigram and Ngram models

Unigrams are features made by a single word/character while Ngram are features made by N words/characters applied to consecutive words.

NB: Use text tokenization based on character when we have to deal with short texts like tweets, comments, posts, description of products. Usually, trigrams gets the best result on character level.

Probability of sequence of words $t_1, t_2, t_3 = P(t_1, t_2, t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1, t_2)$.

In models based on words' independence it becomes $P(t_1)P(t_2)P(t_3)$ which is a **unigram model.**

In **bigram models** it becomes $P(t_1)P(t_2|t_1)P(t_3|t_2)$. These are used by voice recognition systems and based on the language grammar. In Ngrams we consider a dependence of term wrt its previous one.

The drawback is that the **independence assumption is false in unigrams**. Without the independence assumption we have an exponential complexity btw.

This means that the estimated posterior probability is not correct and the values are usually extremes: that is very close to 0 or 1.

In conclusion, the correct estimations of probabilities leads to accurate correct classifications, but **correct estimation are NOT required to get accurate classification since it is sufficient a correct ordering/ranking of probabilities**.

3.4.1 Bernoulli Multivariate or Binomial Multivariate

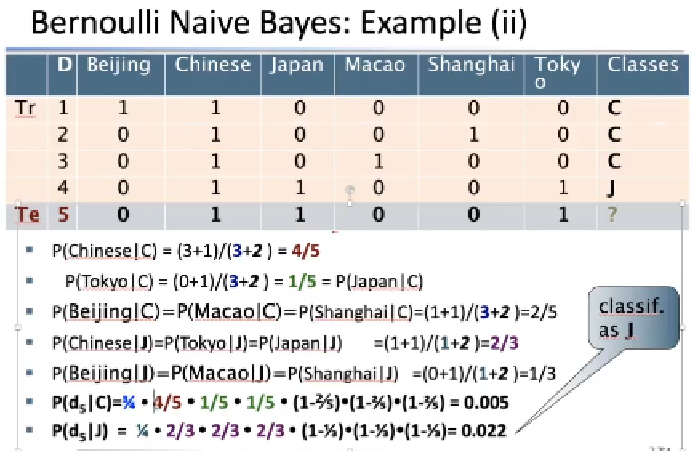
Provide to the model a binary frequency bag of words.

A feature X_w for each dictionary word is 1 if the word is in the document , else 0.

Estimates of the training set probabilities:

$\hat{P}(X_w = t|c_j)$ = fraction of the documents of category c_j containing the word w

In the following example, we calculate first the probabilities of the classes: $P(C) = 3/4, P(J) = 1/4$ and the frequencies $|N_c| = 3, |N_J| = 1$. Then the conditional probabilities (the smoothing factor is 2 since the possible values are two):



Bernoulli Naive Bayes: Example (ii)

	D	Beijing	Chinese	Japan	Macao	Shanghai	Toky o	Classes
Tr	1	1	1	0	0	0	0	C
	2	0	1	0	0	1	0	C
	3	0	1	0	1	0	0	C
	4	0	1	1	0	0	1	J
Te	5	0	1	1	0	0	1	?

- P(Chinese|C) = (3+1)/(3+2) = 4/5
- P(Tokyo|C) = (0+1)/(3+2) = 1/5 = P(Japan|C)
- P(Beijing|C)=P(Macao|C)=P(Shanghai|C)=(1+1)/(3+2)=2/5
- P(Chinese|J)=P(Tokyo|J)=P(Japan|J) =(1+1)/(1+2)=2/3
- P(Beijing|J)=P(Macao|J)=P(Shanghai|J) =(0+1)/(1+2)=1/3
- **P(d_5|C)=¾ • 4/5 • 1/5 • 1/5 • (1-⅖)•(1-⅖)•(1-⅖) = 0.005**
- **P(d_5|J) = ¼ • 2/3 • 2/3 • 2/3 • (1-⅓)•(1-⅓)•(1-⅓)= 0.022**

classif. as J

3.4.2 Multinomial Naive Bayes

We supply the document as a string of words and find the class which it belongs to.

A feature X_i for each word position of the document is X_i = the word in i-th position in the document.

Estimates of the training set probabilities:

$\hat{P}(X_i = w|c_j)$ = fraction of the frequency of the word w among all the words of documents in the category c_j

In this case all the documents of the category are concatenated forming a single document and the frequency is counted on this new one.

The doc is assigned to the class that maximizes the following multiplication:

$$c_{NB} = argmax_{c_j \in C} P(c_j) \prod_i P(x_i|c_j) \tag{46}$$

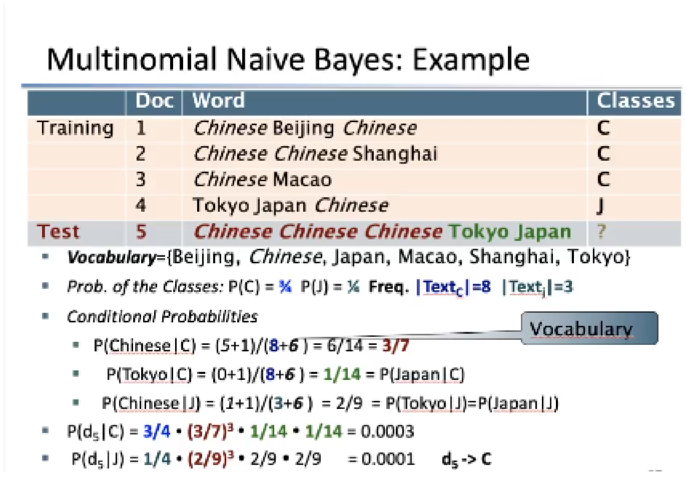
All the document words are used and features can be also URL, emails...

Training. We extract the vocabulary from corpus and compute all values $P(c_j)$ and $P(x_k|c_j)$.

For each class c_j, $P(c_j) = \frac{|docs_j|}{|totaldocuments|}$.

Given $Text_j$ the concatenation of all documents of in class c_j, and $|Text_j| = n$. For each term x_k int he vocabulary, n_k is the number of occurrences of x_k in $|Text_j|$ and α is the smoothing factor (1).

$$P(x_k|c_j) = \frac{n_k + \alpha}{n + \alpha|Vocabulary|} \tag{47}$$



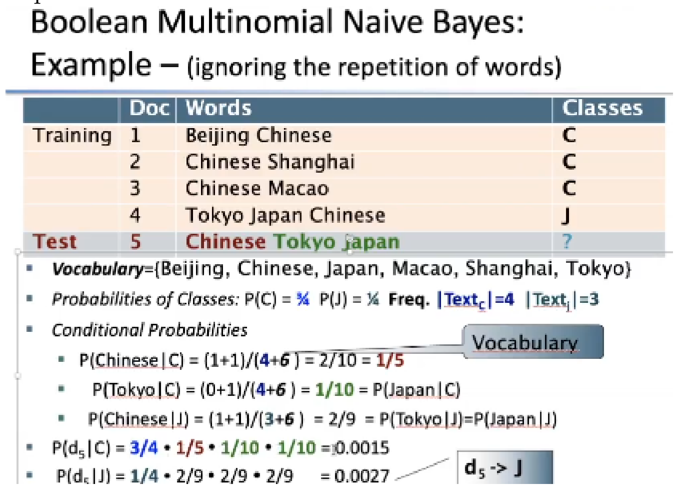
Multinomial Naive Bayes: Example

	Doc	Word	Classes
Training	1	*Chinese* Beijing *Chinese*	C
	2	*Chinese Chinese* Shanghai	C
	3	*Chinese* Macao	C
	4	Tokyo Japan *Chinese*	J
Test	5	*Chinese Chinese Chinese* Tokyo Japan	?

- ***Vocabulary***={Beijing, *Chinese*, Japan, Macao, Shanghai, Tokyo}
- *Prob. of the Classes:* P(C) = ¾ P(J) = ¼ **Freq.** $|Text_C|$=8 $|Text_J|$=3
- *Conditional Probabilities*
  - P(Chinese|C) = (5+1)/(8+6) = 6/14 = 3/7
  - P(Tokyo|C) = (0+1)/(8+6) = 1/14 = P(Japan|C)
  - P(Chinese|J) = (1+1)/(3+6) = 2/9 = P(Tokyo|J)=P(Japan|J)
- $P(d_5|C) = 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 = 0.0003$
- $P(d_5|J) = 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 = 0.0001$ d_5 -> C

The multinomial model is usually more effective than Bernoulli when the frequency helps distinguish the classes - this happens with large number of instances and features.

A variant of the multinomial model, is the **Binarized multinomial Naive Bayes**, also known as Boolean multinomial - it is like multinomial but ignores the repetition of words in documents and classes.



Boolean Multinomial Naive Bayes:

Example – (ignoring the repetition of words)

	Doc	Words	Classes
Training	1	Beijing Chinese	C
	2	Chinese Shanghai	C
	3	Chinese Macao	C
	4	Tokyo Japan Chinese	J
Test	5	Chinese Tokyo Japan	?

- *Vocabulary*={Beijing, Chinese, Japan, Macao, Shanghai, Tokyo}
- *Probabilities of Classes:* P(C) = ¾ P(J) = ¼ **Freq.** |Text$_C$|=4 |Text$_J$|=3
- *Conditional Probabilities*
  - P(Chinese|C) = (1+1)/(4+6) = 2/10 = 1/5
  - P(Tokyo|C) = (0+1)/(4+6) = 1/10 = P(Japan|C)
  - P(Chinese|J) = (1+1)/(3+6) = 2/9 = P(Tokyo|J)=P(Japan|J)
- P(d$_5$|C) = 3/4 • 1/5 • 1/10 • 1/10 = 0.0015
- P(d$_5$|J) = 1/4 • 2/9 • 2/9 • 2/9 = 0.0027

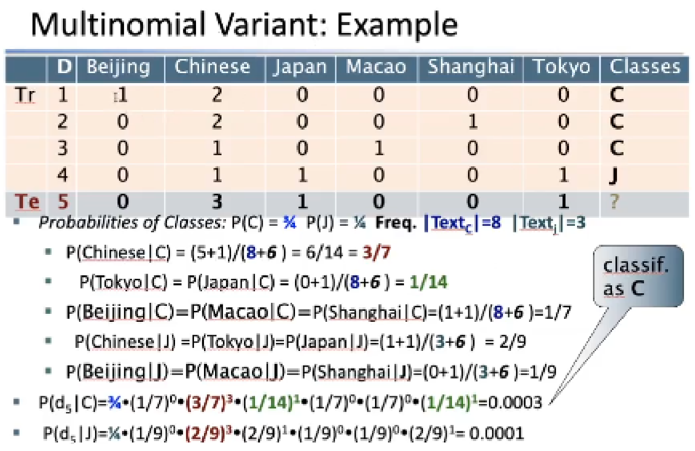
3.4.3 Multinomial Variant

Also here we have a list of words but instead of the binary value we have the frequency.

A feature T_j for each word in the dictionary is T_j = word frequency in the doc, as $tf_{t_i,d}$.

$$P(d|C) = P(< tf_{t_1,d}, \dots, tf_{t_V,d} > |C) = \frac{|d|!}{tf_{t_1,d}! * \dots * tf_{t_V,d}!} \prod_{1 \le i \le V} P(X_i = t_i|C)^{tf_{t_i,d}} \tag{48}$$

Remember to power the frequency in productorial. The first term is the **multinomial coefficient** which computes all the possible combinations of terms' ordering, but according to the independence assumption, the ordering is irrelevant so it is a constant value wrt C, therefore it has no effect to determine the class of the doc and we do not calculate it.

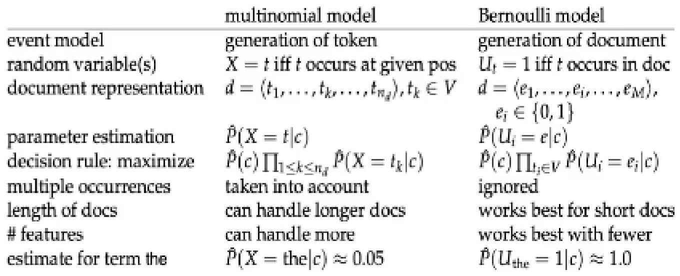


Multinomial Variant: Example

	D	Beijing	Chinese	Japan	Macao	Shanghai	Tokyo	Classes
Tr	1	1	2	0	0	0	0	C
	2	0	2	0	0	1	0	C
	3	0	1	0	1	0	0	C
	4	0	1	1	0	0	1	J
Te	5	0	3	1	0	0	1	?

- *Probabilities of Classes:* P(C) = ¾ P(J) = ¼ **Freq.** $|Text_C|$=8 $|Text_J|$=3
  - P(Chinese|C) = (5+1)/(8+6) = 6/14 = 3/7
  - P(Tokyo|C) = P(Japan|C) = (0+1)/(8+6) = 1/14
  - P(Beijing|C)=P(Macao|C)=P(Shanghai|C)=(1+1)/(8+6)=1/7
  - P(Chinese|J) =P(Tokyo|J)=P(Japan|J)=(1+1)/(3+6) = 2/9
  - P(Beijing|J)=P(Macao|J)=P(Shanghai|J)=(0+1)/(3+6)=1/9
- $P(d_5|C)=¾\cdot(1/7)^0\cdot(3/7)^3\cdot(1/14)^1\cdot(1/7)^0\cdot(1/7)^0\cdot(1/14)^1=0.0003$
- $P(d_5|J)=¼\cdot(1/9)^0\cdot(2/9)^3\cdot(2/9)^1\cdot(1/9)^0\cdot(1/9)^0\cdot(2/9)^1= 0.0001$

3.4.4 Comparison



	multinomial model	Bernoulli model
event model	generation of token	generation of document
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
document representation	$d = \langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \ldots, e_i, \ldots, e_M \rangle$, $e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t\|c)$	$\hat{P}(U_i = e\|c)$
decision rule: maximize	$\hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(X = t_k\|c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i\|c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for term the	$\hat{P}(X = \text{the}\|c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1\|c) \approx 1.0$

There are solutions that use also correction factors of the terms' frequency on the basis of the document length. The independence assumption among terms is unlikely in real documents, but practically these classifiers are among the most effective and efficient.

A successful example uses a bag of words model with term frequencies to recognize spam emails using naive bayes. It uses a multinomial model with conversion of characters to lower case, removal of punctuation and email head are used as features distinguished by email body.

The real solution uses also black list and hand-coded text patterns.

3.4.5 Computational Complexity

Given $|C|, |V|$ the number of conditional probabilities (which is computed while reading the contents only one time), $|D|$ the number of documents, L_{ave} the average length of documents.

At training time, the complexity is:

$$O(|C||V| + |D|L_{ave}) \tag{49}$$

The complexity is $O(|D|L_{ave})$ because usually $|C||V| << |D|L_{ave}$

Given L_t the average length of test documents, at test time the complexity is:

$$O(|C|L_t) \tag{50}$$

The complexity is linearly proportional to the text amount - one of the most efficient method.

3.4.6 Underflow prevention

Multiplying many probabilities, by definition between 1 and 0, may generate floating-point underflow.

It is better to perform computations by summing the log of the probabilities using the property $log(xy) = log(x) + log(y)$.

The class achieving the highest sum value of these logs corresponds anyway to the one that maximizes the correct probability:

$$C_{NB} = argmax_{c_j \in C}[logP(c_j) + \sum_{1 \le i \le |d|} logP(x_i|c_j)] \tag{51}$$

The logs of probabilities are explainable as weights for instance, how much each x_i is a good predictor for c_j

3.5 Accuracy Confidence Interval

A classification of N instances can be modelled as Bernoulli process of N independent binary events (success or error).

Given $f = success/\#instances$ the accuracy on the test set, we may predict the actual accuracy p within in interval, with a given probability, namely the confidence. When N increases, the confidence interval gets smaller.

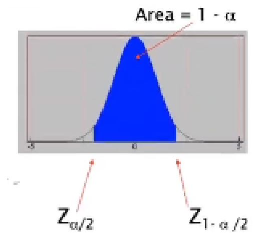
f has binomial distribution $bin(N, p)$ with average p and variance $p(p-1)/N$ with p the actual accuracy we want to estimate. For large N the distribution can be approximated with the normal standard z distribution.

$Pr[-z \le (f - p) \le z]$ = confidence (pre-computed in table for unitary standard deviation.

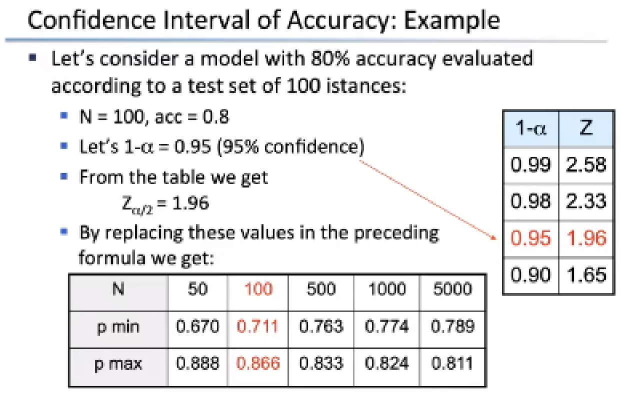
$$Pr[-z < \frac{f - p}{\sqrt{p(1-p)/N}} < z] = c \tag{52}$$

Given z, which is calculated automatically from the normal distribution when the confidence is set, we resolve for p:

$$p = (f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}})/(1 + \frac{z^2}{N}) \tag{53}$$



Using the normal distribution, after setting the confidence interval, z is equal to $Z_{\alpha/2}$, and substitute in the formula $f = 2N * acc$.



Confidence Interval of Accuracy: Example

- Let's consider a model with 80% accuracy evaluated according to a test set of 100 istances:
  - N = 100, acc = 0.8
  - Let's 1-α = 0.95 (95% confidence)
  - From the table we get
    $Z_{\alpha/2} = 1.96$
  - By replacing these values in the preceding formula we get:

1-α	Z
0.99	2.58
0.98	2.33
0.95	1.96
0.90	1.65

N	50	100	500	1000	5000
p min	0.670	0.711	0.763	0.774	0.789
p max	0.888	0.866	0.833	0.824	0.811

3.5.1 Comparing two models

We might compare two models to check if their difference is significant or due to chance. If due to chance, we choose the simpler model.

Given two models M1 with test set D1 and cardinality n1 (with error e_1) and M2 with test set D2 and cardinality n2 (with error e_2). If n1 and n2 are sufficiently large their errors can be approximated by a Normal distribution with average μ and standard deviation σ:

$$e_1 \sim N(\mu_1, \sigma_1) \qquad e_2 \sim N(\mu_2, \sigma_2) \qquad \hat{\sigma}_i^2 = \frac{e_i(1-e_i)}{n_i} \tag{54}$$

So what we have to do is calculate $d = |e_2 - e_1|$. We know that the actual difference to estimate is similar to $d_t \sim N(d_t, \sigma_t)$. So we calculate the variance as $\sigma_t^2 = \hat{\sigma}_1^2 + \hat{\sigma}_2^2 = \frac{e1(1-e1)}{n1} + \frac{e2(1-e2)}{n2}$ and finally d_t with confidence $1-\alpha$:

$$d_t = d \pm Z_{\alpha/2} * \hat{\sigma}_t \tag{55}$$

If the interval contains 0, the difference between the two models is not statistically significant.

Which confidence level makes significant the difference between models?

Given M1 with n1=30, e1=0.15 and M2 with n2=5000, e2=0.25, we calculate $d = |e2 - e1| = 0.1$ and then the variance:

$$\hat{\sigma}_d^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043 \qquad (56)$$

We should determine the value of $Z_{\alpha/2}$ such that:

$$-d < Z_{\alpha/2}\hat{\sigma}_t < d \qquad 1-\alpha = P(-\frac{d}{\hat{\sigma}_t} < Z_{\alpha/2} < \frac{d}{\hat{\sigma}_t}) \qquad (57)$$

By replacing in the example d and σ_t, which we have calculated before, we get that $Z_{\alpha/2} = \pm 1.527 \sim \pm 1.53$ that corresponds to $\alpha = 0.126, 1-\alpha = 0.874$, hence the difference becomes significant when the confidence is < 0.874.

BE CAREFUL to approximate well the Z number, since in this example if we use Z=1.53 the confidence interval contains 0 and the difference is not significant, but if we use Z=1.52 the difference will be significant as the interval no longer contains 0.

3.6 Weka

use always Nested Cross Validation to tune the hyperparameters.

kappa statistic says how much a model is better than a random classifier.

Weka - split before the dataset in train and test, perform feature extraction only on training set (NEVER WITH TEST SET INCLUDED) - find the model and test it on the test set with the same features of the test set.

4 Natural Language Understanding

Importance of natural language understanding (NLU) and text mining to automatically extract useful knowledge from the text.

Language models (LMs) are a key component of modern NLP. Neural LMs are trained as probabilistic classifiers over continuous and low-dimensional representations of words, called word embeddings. Modern continuous space embeddings encapsulate the contextualized meaning of tokens, disambiguating polysemes and quantifying semantic similarities.

Many contributions in the medical field (e.g., SciBERT, BioBERT, PubMedBERT, ClinicalBERT, Med-BERT). Before neural solutions there were Algebraic LMs (e.g., LSA on BOW + TF-IDF), Probabilistic LMs (e.g., pLSA, LDA) which achieved State-of-the-art results in many tasks related to the healthcare domain (Medical QA, Multi-document summarization, Medication recommendation, Phenomena explanation, Gene function finding)

Problems:

- **Black-box knowledge** due to high-dimensional parametric spaces (continuously growing) which make almost impossible to demonstrate what a model has really learned

- many models are not factually accurate, are fragile (can easily be fooled), biased and toxic (gender inequality)
- without explainability it can't be trust by users.

The **joint use of subsymbolic and symbolic** AI is expected to be central in many applications. **Mixed AI** aims to reconcile the stochastic nature of learning with the logical representation of extracted knowledge and reasoning: explainability, knowledge injection, common sense, small data, transfer learning.

Knowledge graphs (KGs) are constantly spreading: WordNet, Freebase, DBpedia, YAGO are among the most widely used. KGs model large collections of semantically interconnected data, they represent knowledge, integrate it and perform inference based on automatic reasoning. One of the most promising technologies of the next decade (2019 Gartner's Hype Cycle).

4.1 POIROT: Phenomena explanation from text

Unsupervised methodology of descriptive text mining (called POIROT) to learn semantic and statistically quantifiable correlations between relevant terms and documents (e.g., "citrus fruit" and "acid reflux": 87%); Construct phenomena descriptions through an original algorithm based on information retrieval and hypothesis testing, initially devised by professor Gianluca Moro.

Tested on interpretable algebraic and probabilistic LMs – Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (pLSA), and Latent Dirichlet Allocation (LDA) which has much lower embedding dimensions than neural counterparts.

Building a phenomena description.

At each step, the query is enriched with a semantically close term, and the correlation with the class (e.g., bad opinion) tends to decrease. **Fold-in** (transform a query vector in a row of the matrix V_k). Formal verification of a correlation between the query q and the class c. Chi-squared test in conjunction with R-precision (number of instances of class c).

POIROT: Generate explanations of medical concept sets from patient social posts with Linear Transformer: with fine-tuning of T5X-Base (state-of-art LM), permormer-based linear attention and pre-training on Commongen (dataset for generating sentences conditioned by concept sets of 3-5 terms).

It has some **limitations**: the general output is a flat list of correlated and quantified set of terms. Terms are not semantically tagged and there are no links between different correlation sets. If a user wants to investigate all the significant correlations between two or more types of entities, he is forced to check their instances individually.

novel unsupervised and automatic technique of knowledge graph learning (KGL) from corpora of unstructured and unlabeled texts. We employed advanced named entity recognition (NER - hierarchical taxonomies) and named entity linking (NEL - disambiguation and linking to Wikipedia, DBpedia, Freebase and PermID). solutions in tandem with semantic web technologies.

SQWRL enables powerful queries on OWL resources, using a high-level abstract syntax for the representation of First Order Logic (FOL) and Horn-like rules.

Correlations are interconnected, NER labels for greater interpretability E.g., "gemelli" is_a /medicine/hospital. Meta-level queries with unbounded terms E.g., drug $\sim$ symptom, drug $\sim$ "chest pain". Knowledge augmentation via linked open data.

4.2 Events for Natural Language Understanding

In the NLP field, information extraction is notably evolving from entities and pairwise relations to events. **Events** are an expressive method of capturing natural language statement semantics, enabling the formal representation of sophisticated processes.

Event extraction (EE) automatically creates structured symbolic representations from large unstructured text corpora. EE allows deriving exhaustive, interpretable, concise, unambiguous and quantifiable interactions otherwise buried in text spans and not readily available for further usage.

Language is highly ambiguous and events allow us to remove noise and focus only on unambiguous relational knowledge involving relevant entities.

Challenges: Directed links, N-ary relations, Multiple fine-grained types, Entity overlapping and coreference resolution, event double tagging, long-range dependencies, high-level linguistic phenomena.

- **Event trigger:** textual mention that more clearly expresses event occurrence (e.g., "stimulates", "regulates")
- **Event argument:** entity mention, value, or another event that serves as a participant or attribute, fulfilling a specific role that characterizes its contribution (e.g., "NF-kappa", "glucose")
- **Argument role:** semantic relationship between an argument and an event, like "Cause" (what is responsible for event occurring) and "Theme" (what is affected)
- **Event modifier:** property that reshapes the described interaction (e.g., negation, speculation)
- Event mention: text span describing an event all its components

EE involves the identification of a trigger, a set of multiple arguments with a semantic role, and a set of optional modifiers or properties which reshape the described interaction. EE handles complex relation with n-ary participants, nested and overlapping definitions, often including named entity recognition (NER)

Applications: Biomedical EE help scientists to do research conveniently and provide inspiration and basis for the diagnosis, prevention, treatment, and new drug research.

Much more than biomedicine . . .
– Single- and multi- document summarization
– Evaluation metrics for natural language generation
– Semantic search and information retrieval
– Predicting people's intents and reactions
– Narrative comprehension for predicting what happens next in a story
– Stock market predictions

4.3 Graphs and LMs

Knowledge graph language model (KGLM) for selecting and copying facts from a KG that are relevant to the context. KGLM outperforms even very large language models in generating facts (i.e., completing sentences requiring factual knowledge).

Structured representations allow the generation of more informative summaries with fewer unfaithful errors.

KagNet: Question answering with external, structured commonsense KGs to boost performance and make explainable inferences.

Fine-tuning LMs on KGs lead to a significant knowledge completion boost than GPT-3, with hundreds fewer parameters.

KG verbalization to generate artificial documents for pretraining dataset extension. New state-of-the-art in open question answering.

5 Deep Learning

Math of a neuron: the output of every node of every layer is computed by:

$$y_j = \sigma(\sum_i w_{j,i} * x_i + b_j) \tag{58}$$

W and b are parameters of the layer, and the activation function is continuous, monotonic and not linear (sigmoid, Relu, tanh). However the whole sum inside the activation function is ALWAYS an **Hyperplane**.

With training we progressively adjust the parameters of the NN.

5.1 LSTM

LSTM are composed by 4 layers:

- The upper line contains the **memory state**
- **forget gate:** learns how much of previous C_{t-1} should be forgotten - multiplication by 1 or close to 1 preserves information, while close to 0 deletes/reduces it.

- **input gate:** learns what new information to store in the cell state C_t - learns two hyperplanes: the first with parameters W and b and then squished with a tanh to [-1,1], the second with a sigmoid - those 2 are point-wise multiplicated and then summed to the previous cell state C_{t-1}
- **output gate:** rules what part of the cell state C_t to emit - a sigmoid learns a new hyperplane and squished with a tanh function to prevent vanishing/exploding gradient.

LSTM example. First 100 iteration of training, samples at random but separates words by spaces. After 300 iterations, starts to use quotes and put periods at end of sentence. 500, can spell short common words ”he”, ”she”. 1200, longer words are learned. Good only after 2000 iterations.

5.1.1 GRU Gated Recurring Units

Are a variation of LSTM to reduce the number of parameters. Here memory and hidden state are merged, and uses only an **update gate**(combining input and forget) and a **reset state** to control the state output.

Are more efficient and perform better than LSTM with small dataset.

5.1.2 Bidirectional LSTM

The idea is to split the neurons of a regular RNN into 2 directions, one for positive time and one for negative time direction - those 2 are not connected to each other. Input from past and future of the current time frame can be used.

5.2 Memory-based DNNs - Neural Turing Machines NTM

Neural Turing Machines use an external memory with different addressing systems, allowing to learn algorithmic tasks. They combine NNs with TMs.

These consumes input vectors to produce output vectors, interact with external addressable memory and are completely differentiable. The goal is to learn algorithms by examples.

Two types:

- **Content-based addressing:** search values in memory by similarity
- **Location-based addressing:** targets specific positions of memory - less general, but helps generalizing.

The **memory** is constituted by an NxM matrix where N is the number of memory locations and M the size of the vector stored in each location.

One or more **read heads** obtain selected information from memory at each time step.

One or more **write heads** concurrently perform write operations on memory at each time step.

Both generate at time t an N-sized vector w_t giving a distribution of locations to read/write.

The **controller** is a NN interacting with the input/output layers and with the heads. (usually an LSTM) -the number of parameters is drastically reduced (from 1M to 70k-15k).

Manages quite well longer sequences rather than LSTM and performs more repeats but fails emit the end marker.

5.3 Word Embeddings

As a mapping function to map words to high-dimensional vectors of floats.

A trained network is going to tell us the probability for every word in our vocabulary of being the nearby word that we chose. Nearby means a **window size** hyperparameter - usually 5. The output probabilities are going to relate to how likely it is find each vocab word nearby our input.

We can perform operations on Word embedding to find relationships like: man + king -woman would return *queen*, or slow-slower+shorter returns *short*.

5.3.1 Approaches for Word Representation

Word2Vec is an algorithm that transforms words into vectors, so that words with similar meaning end up laying close to each other. Is based on NN models that givne unlabelled training corpus, generates vector for each word in the corpus that encodes its semantic.

Useful to measure semantic similarity of 2 words as the cosine similarity of 2 word vectors and use them as features for supervised NLP tasks (document classification or sentiment analysis).

- **CBoW method:** given a term w_t the input are the distributed representations of context (surrounding words) - they are combined to predict the word in the middle.

$$minimize\_J = -logP(w_c|w_{c-m} \dots w_{c+m}) \tag{59}$$

- **Skipgram model**: the input is a single word w_i and the output are the words w_j's context defined by a word window of size C - all words are one-hot encoded.

$$minimize\_J = -logP(w_{c-m} \dots w_{c+m}|w_c) \tag{60}$$

GloVe - Global Vector - computes a co-occurrence matrix of terms from sentences using the concept of window size. The co-occurrence ratios between two words in a context are strongly connected to meaning.

The matrix is symmetric. The goal of training is to predict the ratios between terms.

These two approaches have the same effectiveness - Glove complexity is proportional to the number of non-zero in the matrix while word2vec is proportional to the size of corpus.

BUT they cannot deal with **polysemic terms** - with different meaning according to context.

New quadratic language models are SciBERT, FashionBERT, BioBERT, VilBERT, FLAIR, Pegasus, XLNET.

5.4 Attention

Attention mechanism proposed in [Bahdanau et al., 2014] to improve Natural Language Processing (NLP) models based on the Encoder Decoder architecture and recurrent neural networks (RNN).

It allows the decoder to focus, at each timestep , on the most important pieces of the encoder input, improving the model's ability to process longer sequences, solves the problem of vanishing gradient.

It is implemented by providing the decoder with the weighted sum of all the encoder outputs at each timestep. If the input sequence is L words long, assuming that the decoder output is just as long, it will be necessary to calculate about L^2 weights. Attention weights are learned during training from a NN called alignment model or attention layer.

The alpha weights are the result of a softmax function in the last layer:

$$a_{(t,i)} = \frac{exp(e_{(t,i)}}{\sum_j exp(e_{(t,j)}} \tag{61}$$

Called $y(i)$ the output of the encoder at timestep i, and $h(t)$ the decoder hidden state at timestep t, the way with which $e_{(t,i)}$ is computed defines different types of attention:

- **Bahdanau** 2014 or **Concatenative** Attention:

  $$e_{(t,i)} = v^T tanh(W[h_{(t)}; y(i)]) \tag{62}$$

  where v is a scaling parameter and [;] the concatenate operator.

- **Luong** 2015 or **Multiplicative** Attention:

  $$e_{(t,i)} = h_{(t)}^T W y(i) \tag{63}$$

  This is faster and more space-efficient and can be implemented efficiently with matrix multiplication. **W** can be also non-trainable and set to be an **identity matrix** (**Dot Product Attention**)

5.5 Transformer

The Transformer [Vaswani et al. 2017] is a model that allows to use the attention mechanism without the need of recurrent neural networks. It replaces the recurrent cells of the RNNs with the **Multi-Head Attention Layer** that learns the relationships of an element with each other in the input, considering different semantic meanings, also known as **self-attention**.

Applications in: Applications in: Text Classification, Machine Translation, Summarization, Question Answering.

The most significant advances were made in NLP thanks to the pre training and fine tuning strategy introduced in BERT: Pre training of Deep Bidirectional Transformers for Language Understanding.

In the first step of the decoder this layer is masked (each word is related only to the previous one.

Both the Encoder and the Decoder take as input a matrix Z obtained by transforming each input element into a relative 'input embedding'; called L the length of the input and d the size of the embeddings, Z will have dimension L x d. L must be fixed and padding is needed.

The Z matrix is then summed to a matrix P, called *positional embedding matrix* calculated in a fixed way or learned from the network itself - equivalent results:

$$p_{i,j} = sin(\frac{i}{10000 * \frac{j}{d}}) \text{ if j is even} \quad cos(\frac{i}{10000^{\frac{j-1}{d}}})\text{if j is odd}$$

The matrix X obtained gets processed by the **Multi Head Attention layer** which computed the **Scaled Dot-Product Attention** h times with different weights: hence this learns different semantic relationships between input elements.

The h results are concatenated and brought back to Lxd dimension through a linear layer.

Scaled Dot-Product Attention

Computed employing 3 matrices: keys (K), queries (Q) and values (V) obtained by applying 3 distinct transformations to the input matrix X.

- $V_i \in R^{Lxd_v} = XW_i^V, W_i^V \in R^{Lxd_v}$
- $K_i \in R^{Lxd_k} = XW_i^K, W_i^K \in R^{Lxd_k}$
- $Q_i \in R^{Lxd_k} = XW_i^Q, W_i^Q \in R^{Lxd_k}$
- those are different for each multi-head attention layer h with $h = 0, 1, \dots$
- Usually $d_k = d_v = d/h$
- Attention is calculated as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{64}$$

The numerator represents the similarity between projections of the embeddings and the denominator is a scaling factor to avoid saturation of the softmax function.

- Complexity of time is $O(L^2 d)$ and space $O(L^2 + Ld)$

5.6 BERT - Bidirectional Encoder Representations from Transformers

BERT [Devlin et al., 2018] is a network for Natural Language Processing based on the Transformer architecture. It employs a stack of N encoding blocks , removing the decoder from the model. It is called Bidirectional since , using Transformers, attention is computed in both directions. It introduces 2 self-supervised tasks for pre-training:

- **Masked Language Modeling**: the model is asked to predict randomly masked words within a sentence
- **Next Sentence Prediction**: the network is given two sentences and is asked it to predict whether the second sentence semantically follows the first one

BERT uses positional embeddings learned during training. Each token is assigned a sentence embedding which is summed to the input matrix. Sentences are separated by a special [SEP] token. A token [CLS] is added at the beginning of the input, eventually used for classification.

In the **Masked Language Modeling**, 15% of the words in each sentence are masked in different ways:

- 80% substituted with a [MASK] token
- 10% replaced with a random word
- 10% left untouched

This is required to facilitate fine-tuning. The loss used is the cross entropy computed over the masked out tokens. Called M the number of masked token, t the one-hot encoded vectors of the correct words and p the softmax output returned by the model:

$$L_{MLM} = -\frac{1}{M}\sum_{i=1}^{M} t_i log(p_i) \tag{65}$$

In **Next Sentence Prediction**, during training 50% of sentence pairs are given in the correct order while the remaining half are pairs of sentences randomly selected from the corpus. Using this leads to an improvement on question answering and NLInference problems. Loss used is the binary cross entropy - called $y \in (0, 1)$ the correct class and p the probability assigned by the model:

$$L_{NSP} = -(ylog(p) + (1-y)log(1-p)) \tag{66}$$

BERT is pretrained on BookCorpus (800M words) and English Wikipedia (2500M words). Each sentence is tokenized using WordPiece. The pair of sentences in input is so that their concatenation does not exceed 512 tokens. We have 2 BERT versions trained: BASE with 110M parameters and LARGE with 340M parameters. After pre-training, fine tuning can be done by adding a specific output layer for the specific task.

Results. On GLUE General Language Understanding Evaluation (different NLU tasks), BASE outperform GPT by 4.5% on all tasks and LARGE by a further 3%. On SQUAD (question answering), outperforms F1 by 5 points and Exact match by 5 also.

5.7 Variants of BERT

5.7.1 RoBERTa - Robustly Optimized

The main modifications proposed by the authors are the following:

- **Dynamic Masking** : Masking is performed dinamically during training for each sentence , instead of using static pre processed masking
- **Remove Next Sentence Prediction Task** : The authors showed that this task is not needed to improve downstream results on the benchmark datasets
- **Bigger batch size** : BERT was pre trained with a batch size of 256 for 1M steps; they propose to use a batch size of 2K sequences for a total of 125K steps, proving that this leads to better results with reduced training times

5.7.2 BART

Designed to combine BERT and GPT autoregressive decoder in a single architecture. Its main objective is text generation and is pre-trained in **denoising tasks** to reconstruct corrupted documents:

- **Token Masking** : As in BERT, some tokens may be replaced with a special [MASK] token
- **Token Delition** : Random tokens are deleted from the input. The model must predict the missing indices
- **Token Infilling** : Token Masking applied on sequences of tokens
- **Sentence Permutation** : A document is divided into sentences based on full stops, and these sentences are shuffled in a random order
- **Document Rotation** : A token is chosen uniformly at random, and the document is rotated so that it begins with that token. This task trains the model to identify the start of the document.

5.8 Cross-Modal Transformers

Given the success of Transformers in NLP tasks, several works in the literature have tried to extend their architecture to be able to manage inputs of various types. These models are often referred to as Cross Modal or Multi Modal Transformers. Many solutions have been proposed in particular to combine texts and images Vision and Language Transformers or V+L Transformers.

Both general purpose (e.g. ViLBERT , OSCAR, ImageBERT , Unicoder VL, ViLT and domain specific (e.g. FashionBERT , KaleidoBERT for the fashion domain).

We have 2 channels, one for text and one for images. The image is split in P squared patches which are flattened into a sequence. Then transformed into embeddings through a convolutional layer. The representations obtained are concatenated and passed to the Transformer attention layers. Masking is optional.

5.9 Optimizations for Attention

To solve the problme of limited input length,to reduce training times and allow bigger batch sizes.

Strategies:

- **Fixed/Learnable Patterns** : the idea is to compute the attention only on blocks of size B(with $B << L$)of the input, going from a complexity of $O(L^2$ to one of $O(B^2)$; the way in which the blocks are built can vary according to the algorithm and can also be learned from the network itself
- **Memory** : they use an additional memory in the form of global token that allows to aggregate information and possibly reduce the length of the input sequence
- **Low Rank Methods** : they are based on low rank approximation of the attention matrix which projects the vectors into a lower dimensional space
- **Kernels** : they use kernel functions and mathematical re writing to be able to express attention without explicitly calculating the LxL matrix
- **Recurrence** : they are based on the 'fixed patterns' mechanism but introduce recurrent connections between blocks

5.9.1 Performer

Kernel based solution proposed in 2020 to bring the temporal and spatial complexity of the Transformer from quadratic to linear.

From $O(L^2d)$ to $O(Lrd)$ in time and from $O(L^2 + Ld)$ to $O(Lr + Ld + rd)$ in space with r such that $0 < r << L$).

We rewrite attention using:

$$Attention(Q, K, V) = D^{-1}AV \qquad A = exp(\frac{QK^T}{\sqrt{d}}) \qquad D = diag(A1_L) \quad (67)$$

The entries in A can be thought as a result of the kernel K applied to the matrices Q and K. We avoid computing A using **FAVOR+ (Fast Attention Via Positive Orthogonal Features)** which allows to approximate the kernel K via a function called *positive random feature map*

5.9.2 BigBird - Sparse Graph Attention

BigBird [Zaheer et al., is a model with linear complexity that approximates attention using fixed pattern and memory mechanisms. The idea is to use a modeling based on sparse graphs in which the nodes are tokens of the input sequence and the arcs represent the connections in the attention matrix.

BigBird combines 3 types of sparsification :

- **random attention** : each node is randomly linked to other r nodes
- **window attention** : each node is connected to its w neighbours (w/2 on its left and w/2 on its right); this choice comes from the heuristic consideration for which the most important information is generally local information
- **global attention** : g nodes are completely connected

The authors proved that with this type of connections, BigBird becomes a universal approximator of sequence to sequence functions and is Turing complete.

Two models have been proposed:

- BigBird ITC Internal Transformer Construction): Global tokens are selected from within the input sequence
- BigBird ETC (Extended Transformer Construction): Global tokens are newly added tokens at the start of the input

A LARGE version was also implemented for both of them with 24 hidden layers of size 1024 and 24 'heads' of attention. State of the art in question answering and text classification. Improved results in summarization tasks since it can handle longer input sequences (i.e. up to 4096 token).

In particular BigBird Pegasus, which is specifically pre trained to generate summaries

5.9.3 Reformer

A model with space-time complexity $O(LlogL)$ which approximates attention using learnable patterns - based on a technique called **Locality Sensitive**

Hashing (LSH) which reduce the number of tokens for which attention is computed.

Uses shared parameters between W_Q and W_K matrices (Q=K). Introduces Reversible Residual Layers in the Transformer architecture: computes the input of each layer on demand rather than saving it in memory during the forward pass - can handle 1M tokens with 16GB of memory.

LSH Attention consists in using a hash function h that assigns the same bucket to similar vectors - $h(x) = argmax([xR; -xR])$ where x is the input vector and R is a matrix of random gaussian floats. Reformer computes attention weights only between vectors of Q and K belonging to the same bucket.

5.9.4 Transformer-XL

Does not directly reduce computational complexity (still $O(L^2)$) but still allows to process very long documents.

It divides long input into segments and process each one separately, losing inter-dependencies - this is kept with an additional memory.

Relative Positional Encoding must be used instead of Absolute Positional Encoding. This is necessary to ensure that the position embeddings are consistent between segments processed consecutively.

5.10 Metric Learning

Aims to find a latent embedding space that allows to separate the data according to a given metric, keeping semantically similar records close to each other while separating the dissimilar ones.

Can be used to perform various tasks like k-NN classification, clustering and information retrieval.

Two types:

- **Supervised Metric Learning:** Each data is labeled and belongs to a specific class
- **Weakly Supervised Metric Learning:** there are no labels available but the data is already split at the tuple levels.

Mathematically we want to find a Mahlanobis matrix M (positive semidefinite), such that, taking two similar elements a and b and one dissimilar n, we have $d_M(a, b) < d_M(a, n)$. d_M is the Mahlanobis distance equal to $\sqrt{(x-y)^T * M * (x-y)}$. Each Mahlanobis matrix can be written as $M = W^T W$ so we have $d_M(x, y) = ||W_x - W_y||_2$ and corresponds to the Euclidean distance between the data projected into a latent space via a transformation W.

In **Deep metric learning,** deep NN are used with trainable weights θ to learn the transformation W_θ

5.11 Loss Functions

Losses that force embeddings of similar elements to be closer than the dissimilar ones.

5.11.1 Pairwise Ranking Loss or Contrastive Loss

Given input x_1 and x_2 with labels y_1 and y_2 the loss is defined as:

$$L_{contrastive} = 1_{y_1=y_2}||W_\theta x_1 - W_\theta x_2||_2^2 + 1_{y_1 \neq y_2} max(0, a - ||W_\theta x_1 - W_\theta x_2||_2^2) \tag{68}$$

where 1_A is the indicator function which evaluates to 1 if A is True, 0 otherwise. $||.||_2^2$ is the squared Euclidean distance. This loss causes inputs with the same label to have a low distance an dinputs of different labeld to have a distance greater than a margin a chosen as hyperparameter (1 or 0.1).

5.11.2 Triplet Loss

The most popular which works with triplets of elements x_a, x_p, x_n with $y_a = y_p \neq y_n$:

$$L_{triplet} = max(0, ||W_\theta x_a - W_\theta x_p||_2^2 - ||W_\theta x_a - W_\theta x_n||_2^2 + a) \tag{69}$$

where x_a, x_p, x_n are called anchor, positive and negative. We want the anchor distance to be greater with positives than that between negatives by at least a margin a.

We can have 3 types of negatives:

- Easy: the ones which already satisfy the loss constraint and therefore do not contribute during training.
- Hard: Those negatives that have a shorter distance from the anchor than the positive
- Semihard: Those negative that have a greater distance than the positive but do not exceed the margin a.

  Using hard and semihard usually speed-up training convergence.

Both used in Image Recognition and Object Recognition in Computer Vision. There are a lot more thta can be applied also to Information Retrieval where the ranking of documents can be obtained by using the distance between the latent representations of queries and documents.

5.12 Neural Ranking Models

In IR, given query $s \in S$ from a set of documents T. A function $f(s,t)$ which assigns a similarity score, in general is: $f(s,t) = g(\psi(s), \phi(t), \eta(s,t))$.

ϕ, ψ are functions that extract feauters from queries and documents, η is the interaction function that models the relationship between queries and documents and g is the evaluation function that returns the similarity score.

In traditional IR, the first 3 are manually defined while g can be instead a ML model. Recently, all the functions are defined through deep neurale networks in Neural Ranking Models.

Two architectures which can also be applied in Cross-Model Retrieval (mainly in Text-to-Image and Image-to-Text)

- **Representation-Focused:** Explicitly define ϕ, ψ and compute the similarity using g through metric learning - efficient but less accurate since interaction between query and docs is not modeled.

  Tipically use LSTMs as text encoders and pre-trained CNNs as image encoders (VSe, SCAN, PFAN).

- **Interaction-Focused:** Not defined directly ϕ, ψ but uses the function η to model the interaction - allow to learn deep relationships between queries and docuemnts with higher accuracies - inefficient for online retrieval.

  Based on Cross-modal transformers (ImageBERT, OSCAR, Vilt, Fashion-Bert) and known also as **SOTA solutions.** Those, btw, are inefficient due to quadratic complexity in attention mechanism and slow online search.

In SOTA we have two stage training:

- Pre-training with text-image interaction where we have: *1-Masked Language Modeling* to predict masked tokens in the sentence, *2-Masked Path Modeling* to predict masked patches in the image and *3-Text Image Alignment* to predict whether the input text describes the image.
- Training with Triplet Loss, decoupling text and images - anchor and positive are an image and its caption, negative is a random image in the dataset.

One of the advantages of having a latent embedding space in which queries and documents are decoupled, is that it is also possible to define **multidimensional indices** on document embeddings to drastically reduce retrieval times.

To speed up more we can use approximate search techniques like **Inverted File Index** or **PCA**.