UNIVERSITY OF STAVANGER

Information Retrieval using applied Supervised Learning for Personalized E-Commerce

by

Hellum, Kjell Arne

A thesis submitted in fulfillment for the
degree of Master of Science

in the
Faculty of Electrical Engineering and Computer Science

June 2017

UNIVERSITY OF STAVANGER

Abstract

Faculty of Electrical Engineering and Computer Science

Master of Science

by Hellum, Kjell Arne

This paper describes our efforts on a learning to rank problem applied on the topic of personalized e-commerce. The core of the project comes from the *Personalized E-Commerce Search Challenge* issued by the International Conference on Information and Knowledge Management. By analyzing historical data containing browsing logs, queries, user interactions, and static data in the domain of an online retail service, we attempt to extract patterns and derive features from the data collection that will subsequently improve prediction of relevant products. A selection of supervised learning models will utilize an assembly of these features to be trained for prediction of test data. Prediction is performed on the queries given by the data collection, paired with each product item originally appearing in the query. We experiment with the possible assemblies of features along with the models and compare the results to achieve maximum prediction power. Lastly, the quality of the predictions are evaluated towards a ground truth to yield scores.

Keywords - learning to rank, product search, e-commerce, personalization, machine learning, supervised learning, regression, cikm cup

Chapter 1

Introduction

This document serves as the author's Master thesis in Computer Science. The subject of this thesis builds upon the author's previous project *Information Retrieval using applied Supervised Learning for Personalized E-Commerce* [20]. During the course of the project, the author has received assistance and direction from Krisztian Balog, Ph. D., from the Department of Electrical Engineering and Computer Science located at the University of Stavanger.

The Personalization of e-commerce is a hot topic for researchers as applying it on a service yields benefits for many businesses and their consumers. Learning insights on action patterns could prove invaluable for increasing revenue and customer satisfaction, as tailoring the overall experience for customers leads to higher conversion rates. With the progression of the internet and the increasing amount of information available, the possibility of taking advantage of the information becomes significantly better. Through machine learning methods and analytical decision making, we can extract patterns from the consumers' behaviours and discover new and better ways to perform the presentation of products, marketing strategies, personalization and generally improve the end user's experience.

The particular problem of this thesis is classified as a learning to rank problem[34]. The International Conference on Information and Knowledge Management (CIKM) issued a competition, the CIKM Cup 2016 [1] where entrants, be it academics or researchers or students alike, could take on the problem of learning to rank. CIKM provides a dataset for the challenge which contains realistic data on products, users and their respective relations as purchases, views, clicks and other meta-data. The goal is to find new and possibly innovative ways to rank items retrieved by a query issued by a user. By utilizing the information given in the queries, the meta-data from the users and their respective actions on the products, we can learn features that may reveal correlations in terms of relevance. The heart of the challenge is to improve the existing rankings through

[1] http://cikm2016.cs.iupui.edu/cikm-cup/

supervised machine learning methods and custom features. In this paper we will discuss how we approached this problem and present the features used, the results of evaluating these, and compare these results with other participants of the competition.

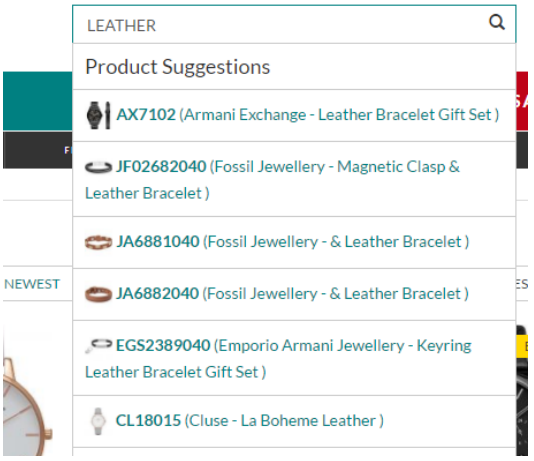

FIGURE 1.1: Keyword search [2]

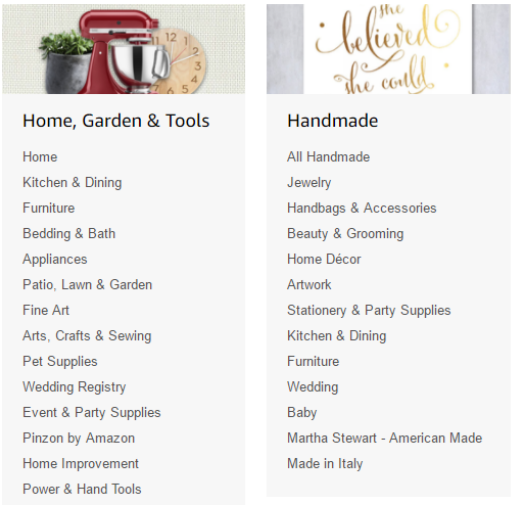

FIGURE 1.2: Category browsing [3]

1.1 Main Contributions

This thesis contributes to the area of Information Retrieval with emphasis on supervised learning, and learning to rank in the domain of an online retail business. Specifically, this thesis introduces custom features that connect products together with meta-data and history, with the primary objective of improving the ranking function of items retrieved by given queries.

1.2 Outline

- Chapter 2 introduces parts of relevant theory used as basis for the development and research of this thesis. This chapter provides basic understanding for the

[2] http://www.watchshop.com/All-Watches.htmltab5=leather?st=leather
[3] https://www.amazon.com/gp/site-directory/ref=nav_shopall_btn

Information Retrieval and Machine Learning fields of Computer Science, and is formally presented with the essential aspects relevant to the problem.

- Chapter 3 contains a detailed overview of the problem itself, including definitions of the primary objective. Additionally this chapter describes the details and statistics of the dataset associated with the problem. Lastly we introduce evaluation of our experiments, alongside its revolving methodology.
- The fourth chapter provides in-depth details of our approach to reach the goal of this thesis. We present key points as:
  - Formalizations of custom features connecting products to meta-data and associated history
  - Assembling features into feature vectors
  - Statistical models chosen to train on the feature vectors
  - In-depth details of our implementation and setup
- In Chapter 5 the results of our experiments are presented, in comparison with the baseline score.
- In Chapters 6 and 7 we interpret and discuss the results, and observations made on them. Lastly, we conclude with opinions and remarks for the contributions of this thesis.

Chapter 2

Background

2.1 Information Retrieval

The general activity of collecting useful information or resources from a source to satisfy a specific need is known as Information Retrieval (IR). Information can appear in any form, such as text, documents, images, audio or video. With the rapid growth of data appearing on the Internet, an equal growth of possibilities occurs. Search engines, online retailers, database systems, social media, surveys and many more applications may contain vast amount of resources that can be explored and utilized for their mathematical properties, statistical correlations and insights.

Retrieval of information is commonly associated with a user providing queries as input to a particular system. A query may be expressed in free text (e.g., Google Search) or structured text (e.g., Structured Query Language). These queries produce outputs which are expected to satisfy the demands of the query, either as a singular result (e.g., searching for a particular book on Amazon) or as a ranked list with respect to relevance (e.g., search engines).

Determining the relevance of retrieved information is a challenge in itself, as it requires some understanding of the basic needs of the user. In structured queries, the output expected is already formulated in the query itself, but in free text queries, the case may not be trivial. If user Alice performs a search on Google for "John Kennedy", is she looking for the biography of the former US president or the contact information of a coworker similarly named? If user Bob performs a search on eBay for "used car", how does the search engine know which cars are interesting and/or affordable to him?

2.1.1 Retrieval Evaluation

Most IR applications can determine the relevance of a resource with respect to the query by assigning a numeric score, which provides basis for the ranking, meaning the top ranked resources are most relevant. How these scores are derived strongly depends on the context of the query and the service running, as they are determined by internal and/or external associative features. It is hard to be determine whether the order of the output satisfies a given query optimally, let alone all given queries. However, means of evaluating the results prove themselves valuable as basis for improvement [17], with the potential of pinpointing specific queries that perform particularly well or poorly. In order to evaluate the precision of a retrieval system, rank-based evaluation measures can be used. Let's consider resources retrieved from an IR system as documents.

TABLE 2.1: Retrieved list of documents, ranked on relevance

Documents	Relevance
d_1	Relevant
d_2	Relevant
...	...
d_n	Relevant
d_{n+1}	Non-relevant
...	...
d_{m-n+1}	Non-relevant

List of n relevant documents, with trailing m non-relevant documents

Ideally a ranked list would be in the form as seen in Table 2.1. In this section, we portray relevance as binary for the sake of simplicity; although, later on we will see examples where relevance might be more nuanced.

TABLE 2.2: Possible retrieval outcomes [56]

		Action	
		Retrieved	Not retrieved
Doc	Relevant	a	b
	Not relevant	c	d

Before introducing some useful measures to evaluate the precision of a retrieval, we need to understand the outcomes of a retrieval. As seen in Table 2.2, there are four distinct combinations of relevance and retrieval, which are taken into account. A document may or may not be retrieved, and the same document may or may not be relevant to the user. Two measures can be extracted from this table, namely *precision* and *recall*. These single-value metrics have proved to be important and are used widely in many other evaluation problems, e.g., machine translation [38], monitoring of distributed systems [54] and medicinal image search [16].

Precision can be expressed using Table 2.2 as

$$Precision = \frac{a}{a+c} \tag{2.1}$$

Similarly, recall is expressed as

$$Recall = \frac{a}{a+b} \tag{2.2}$$

For ideal results, $precision = recall = 1.0$. Precision can be thought of intuitively as the ratio of relevance among retrieved documents. If four documents were retrieved, and three of them are considered relevant, the resulting precision would be equal to $\frac{3}{4} = 0.75$. Recall tells us about the ratio of retrieved documents out of all relevant documents; if three documents are relevant, and only two of these are retrieved, the resulting recall would be equal to $\frac{2}{3} = 0.67$. Maximizing recall is trivial, as all documents could be retrieved, so this is not a good measure alone for measuring the ratio of non-relevant documents.

Having the ideal outcome of $precision = recall = 1.0$ tells us that *all relevant documents are retrieved* and equally *all retrieved documents are relevant*. In reality, having a high value for recall tends to mean a low value for precision. This is a consequence of attempting to retrieve as many relevant documents as possible, as going further down in a list searching for relevant documents, we find even more non-relevant documents. These metrics may be applied for the entire list of retrieved documents, or on a limited portion of the list, known as *precision at n* or *P@n*. Precision applied to a cutoff portion of the list only considers the topmost items, preserving the original ranking.

Precision in its simplest form is not a very yielding measure when it comes to evaluating multiple items retrieved, therefore we introduce a new measure, *average precision* (AP). Considering a set (list) consisting of n documents $L = \{l_1, l_2, ...l_n\}$, we wish to calculate the average precision of each l_i.

AP [57] is defined mathematically as

$$AP(L) = \frac{1}{|Rel|} \sum_{i=1}^{n} p(i) \tag{2.3}$$

where L is the ranked list of documents retrieved, $p(i)$ is the precision of the document at position i and $|Rel|$ denotes the cardinality of the set of relevant documents associated with the entire collection of documents. Using Eq. (2.3) we can assign a numerical score being the sum of precision for each document divided by the number of relevant documents. This gives us the precision of a retrieval performed by a system.

Average precision as a measure is useful for evaluating single retrievals. However, when there are multiple retrievals performed by a singular system, how can we evaluate the precision of the system as a whole?

Let's continue having the mindset of averaging the numbers, and introduce *Mean average precision* (MAP). Considering a set of m ranked lists, i.e., retrievals performed by a system, $\mathcal{L} = \{L_1, L_2, ...L_m\}$, we wish to calculate the mean average precision of each retrieval.

Mean average precision (MAP)[57] is defined mathematically as

$$MAP(\mathcal{L}) = \frac{1}{m}\sum_{i=1}^{m} AP(\mathcal{L}_i) \tag{2.4}$$

MAP gives a fair representation of the precision of retrievals performed by a single system. This measure allows us to compare multiple systems and cross-reference how the overall precision differs in each system.

TABLE 2.3: Example of multiple retrievals, with relevance labels, precision and recall

		Documents	Relevance	AP	Recall
Retrievals	Ideal	d_1,d_2,d_3	r,r,r	1.0	1.0
	System A	d_5,d_4,d_1,d_3,d_2	n,n,r,r,r	0.58	1.0
	System B	d_3,d_2	r,r	1.0	0.67

It should be noted that shuffling around the order of documents in a retrieval does not effect the measures for precision and recall, as their (changes in) positions are not reflected in the equations. In Table 2.3, the hypothetical System A has retrieved all relevant documents with two non-relevant documents in position 1 and 2. This gives a good recall score, but having non-relevant documents retrieved decreases the average precision. System B has only retrieved relevant documents, maximizing precision, but it does not manage to capture every relevant document available. Which system performed the retrieval better?

In order to take the position (i.e., the rank) of the document into account, we can derive a formula to emphasize reward on higher rankings (i.e., lower positions) and oppositely penalize lower rankings. Figure 2.1 displays one particular function fitting to our specifications, i.e., $f(x) = \frac{1}{x}$.

Reciprocal rank (RR) [6] is mathematically defined as

$$RR(L) = \frac{1}{rank_i} \tag{2.5}$$

where $rank_i$ denotes the rank of the first appearing relevant document.

Reciprocal rank is a weighting coefficient to penalize documents with lower rankings, specifically used with the rank of the first appearing relevant document retrieved. A

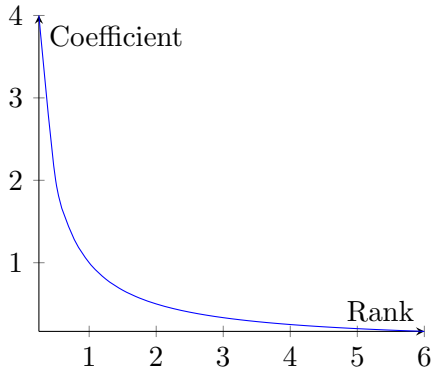

FIGURE 2.1: Continuous representation of reciprocal rank $RR = 1/rank_i$

set of retrieved documents $L = \{d_1, d_2, d_3\}$ with d_3 being the first relevant document evaluates as $RR = \frac{1}{3}$.

Mean reciprocal rank (MRR)[6] is mathematically defined as

$$MRR(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{i=1}^{|\mathcal{L}|} \frac{1}{rank_i} \tag{2.6}$$

where $rank_i$ is the rank of the first appearing relevant document of query i.

Mean reciprocal rank produces an average of the respective reciprocal ranks of a retrieval set. This measure tells us the average distance to the first relevant document in the retrievals made by a system. It is a useful measure, as it is proportional to the effort a user spends on finding a result.

Now we have covered some of the basic measures that come in handy when considering relevance of documents as either a) relevant or b) non-relevant. However, the way relevance is interpreted may allow for multiple degrees of relevance to exist [21]. Average precision only operates on binary relevance values, and therefore compressing the relevance degree into a binary one causes information loss. For example, if relevance was in the range of $[1, 3]$, a compromise of letting 1 be non-relevant and $[2, 3]$ be relevant would not be beneficial as we lose the information between relevance degrees 2 and 3. The same problem applies to MAP and MRR. How can we accommodate evaluation of retrievals when documents have multiple degrees of relevance?

Imagining a user browsing through a list of documents retrieved, he would be mostly interested in the top results. From the user's perspective, the relevance comes from the utility of the document, and is therefore effectively the *information gain*. Summing the gain for each document in the top results, we end up with a new measure, Cumulative Gain (CG) [33] defined as

$$CG(L) = \sum_{i=1}^{n} r_i \tag{2.7}$$

where r_i is the gain of document at position i, and n is the cutoff point in the list of documents the user is looking at. Depending on the number of documents looked at (i.e., where we put the cutoff point n), the cumulative gain will correspond to the amount of effort put in by the user in viewing more documents.

TABLE 2.4: Example of cumulative gain of top 10 results

	Gain (Relevance)	Cumulative gain
d_1	3	3
d_2	1	3 + 1
d_3	2	3 + 1 + 2
d_4	1	3 + 1 + 2 + 1
d_5	1	...
d_6	3	
d_7	2	
d_8	1	
d_9	1	
d_{10}	2	
...	...	

Illustrated in Table 2.4 is an example of applied cumulative gain on a document collection, with cutoff point at $n = 10$. Note that different retrievals may yield differing values of CG depending on the quality of the retrieval. However, shuffling the order of documents retrieved within $[0, n]$ yields the same cumulative gain, similar to average precision, it does not take rank or position into account. Assuming a retrieval is at best effort, we can also assume that the highest ranked documents are the most relevant, even though this is not always the case as seen in Table 2.4. With these assumptions in mind, we can apply a slight discounting in relation to the positions of the documents. The first document does not require any discounting as the user always sees this first. With this, we can build upon CG and introduce discounted cumulative gain (DCG) as

$$DCG(L) = r_1 + \sum_{i=2}^{n} \frac{r_i}{\log_2 i} \tag{2.8}$$

As we divide every document's gain by a logarithm of its position in a list, we effectively get a lower score if a highly relevant document appears lower in the list. Similarly, the relevant documents higher ranked receive higher score. This measure is maximized when the order of the documents is optimal, i.e., sorted on relevance. In this manner, we have improved the previous deficiency and furthermore take the position into account.

Cross-comparison of queries with this measure is at this point not possible as the value of DCG will differ if the documents retrieved are different. If we normalize the DCG we solve this problem. The normalized discounted cumulative gain is expressed as:

Normalized discounted cumulative gain (NDCG) [33].

$$NDCG(L) = \frac{DCG(L)}{IDCG} \tag{2.9}$$

where IDCG is the ideal DCG for a given query, signifying the retrieval is of an optimal order in terms of relevance. NDCG is widely used and proves to be an accurate evaluation measure of the quality of a retrieval. It is easily comparable across different queries, as we expect a value between 0 and 1, with 1 meaning $DCG(L) = IDCG$.

In this paper, we have chosen NDCG as the method of evaluation.

2.1.2 Vector space models

A vector space is a set of V basis vectors for V dimensions, where each vector is linearly independent. In Information Retrieval, any text in general can be represented in vectors describing its presence. Term Vector Models, also known as Vector Space Models [46], are algebraic models that represent documents by the use of vectors, by having each term appearing in a vocabulary as its own dimension. This allows for transforming text as query keywords and terms in a document into term vectors.

TABLE 2.5: Example of multiple documents with similar terms

Docs	Terms	Terms (w/o stopwords)
d_1	The sky is blue	sky blue
d_2	The sun is bright today	sun bright
d_3	The sun in the sky is bright	sun sky bright
d_4	We can see the shining sun, the bright sun	shining sun, bright sun

Consider the documents available in Table 2.5. By counting every term for each document, a term vector can be constructed. As each document is different, the corresponding vectors will point in different directions as a result. Similarly if two documents were equal, they would be point in the exact same direction. This is a result of each basis vector being dependent on the presence of its designated term in the documents.

Note that the majority of terms in the above documents do not give any information. Terms as “the”, “is”, “in” etc. are some of the most common words in the english language, and are known as *stopwords*. Stopwords have been around for a while, and were defined as words to be filtered out before processing natural language text [44] to improve performance of many search engines, depending on the context. In some cases, these words are important and should not be filtered, e.g., when a user performs a search for “The Who”. For the sake of emphasizing the important terms in the documents in

Table 2.5, stopword removal is performed. Another procedure applicable to reduce the number of possible variations of a word include stemming. This is useful for particularly verbose queries that may occur when a user writes her query in natural text [15].

Displayed in Figure 2.2 is what a term vector would look like for document d_3 from Table 2.5.

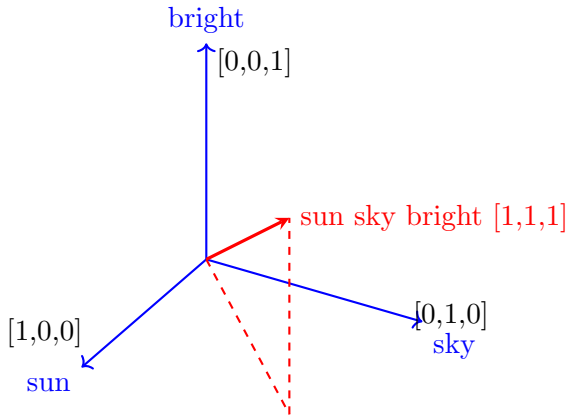

FIGURE 2.2: Three-dimensional term vector for "sun sky bright"

These vectors can be measured for similarity between documents, and between document and query. Cosine similarity [45] is a measure that tells how similar two term vectors are, based on the presence of terms in each vector. If a user performs a keyword search on "bright sky", a similar vector could be constructed based on the terms appearing in the query. Subsequently, these two vectors can be compared using the following function:

$$cos(\theta_{q,d}) = \frac{\sum_{j=1}^{n} term_{q,j} \cdot term_{d,j}}{\sqrt{\sum_{j=1}^{n} term_{q,j}^2} \cdot \sqrt{\sum_{j=1}^{n} term_{d,j}^2}} \tag{2.10}$$

where $\theta_{q,d}$ denotes the angle between query q and document d, and $term_{q,j}$ and $term_{d,j}$ represent the components of the term vectors for q and d respectively. This measure yields a value between 0 and 1, where a higher value indicates higher similarity, with 1 meaning the two vectors are identical.

In the above example, the case is trivial as there are few distinct terms and we can simply perform raw comparison on queries using only cosine similarity. However, this performs poorly when the document collection is of a realistic size with an even larger vocabulary (e.g., Wikipedia entries). Topical relevance can be estimated by determining which terms are frequent in a document, and which terms appear more in this document than other documents. This may be done by weighting the documents using *Term Frequency* (TF) and *Inverse Document Frequency* (IDF) [47].

The term frequency can be written as:

$$TF(t,d) = \frac{f_{t,d}}{|d|} \tag{2.11}$$

where $f_{t,d}$ is the number of times term t appears in document d and $|d|$ denotes the number of terms in d. The inverse document frequency is defined as:

$$idf_{t,D} = log(\frac{|D|}{|d \in D : t \in d|}) \tag{2.12}$$

where D is the document collection, and $|d \in D : t \in d|$ denotes the number of times the term t appears in the entire document collection. Incrementing the denominator by 1 will prevent a division-by-zero.

TF-IDF is a widely used measure as one of the most popular weighting schemes. This measure is used in approx. 83% of text-based recommendation systems of digital libraries [2].

Instead of counting each term in a document and use the raw count in the term vector, TF-IDF weighting is performed to distinguish documents containing document-unique terms apart from documents with common terms. The higher value for TF indicates that the term is more frequent, while the higher value for IDF indicates the term is rare in the document collection. A high value for combined TF-IDF indicates high term specificity.

Calculating the Term Frequency for each term in the documents in Table 2.5 yields:

TABLE 2.6: Example of calculating Term Frequencies

	Term Frequency			
Terms	d_1	d_2	d_3	d_4
sky	0.5	0	0.33	0
blue	0.5	0	0	0
bright	0	0.5	0.33	0.25
sun	0	0.5	0.33	0.5
shining	0	0	0	0.25

Furthermore, we can calculate the IDF of each term:

TABLE 2.7: Example of calculating Inverse Document Frequencies

Terms	sky	blue	bright	sun	shining
IDF	0.30	0.60	0.125	0.125	0.60

Combining the term frequencies from Table 2.6 and multiplying them with inverse document frequencies from Table 2.7 yields:

TABLE 2.8: Example of calculating TF-IDF

	TF-IDF			
Terms	d_1	d_2	d_3	d_4
sky	0.150	0	0.100	0
blue	0.300	0	0	0
bright	0	0.063	0.043	0.031
sun	0	0.063	0.043	0.063
shining	0	0	0	0.150

By applying the weighting scheme, the term vector of d_3 becomes $(0.043, 0.1, 0.043)$. Also introducing a query q = "bright sun", the vector for this query is $(0.063, 0, 0.063)$. Vectors for q and d_3 are visually represented in Figure 2.3.

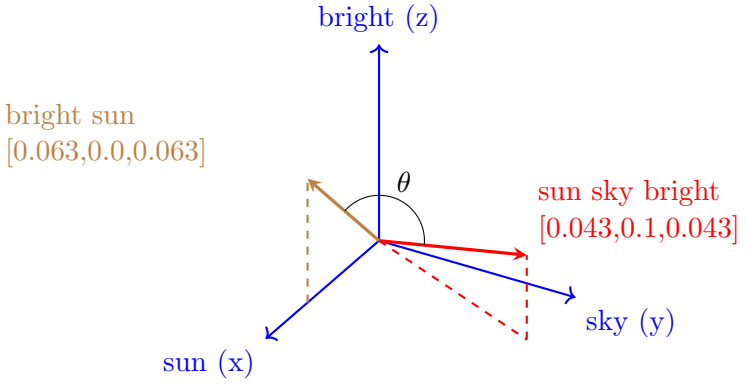

FIGURE 2.3: TF-IDF weighted term vector for "sun sky bright"

At this point, it is possible to measure how similar the query is to all the documents by calculating the cosine similarity seen in Eq. (2.10):

TABLE 2.9: Example of calculating cosine similarity on TF-IDF weighted term vectors

	Cosine similarity			
Documents	d_1	d_2	d_3	d_4
Query q	0	1	0.864	0.401

By inspecting the similarities calculated for each document d_i against the query q, it is clear that:

- d_1 is not similar to q as they do not have any overlapping terms.
- d_2 is identical to q as they have equal terms.
- d_3 is highly similar to q as they have two overlapping terms.
- d_4 is less similar to q as the TF-IDF values for the term "shining" in d_4 is over-shadowing the terms "bright" and "sun".

Cosine similarity with TF-IDF weighting is a well suited measurement for comparing segments of text, which we will utilize in this paper as a custom feature for comparing queries towards product names.

2.2 Machine Learning

Explicitly programming algorithms to extract information or perform tasks has its limitations in certain applications. When the application domain is unknown or volatile, machine learning can be applied. Machine learning is a sub-field of Computer Science "that allows computers the ability to learn without being explicitly programmed"[51]. Machine learning has overlapping properties with artificial intelligence, in which it bestows principles of intelligent behavior upon machines.

Machine learning covers the development and research of programs and algorithms that can predict, classify, unravel patterns, and adapt to application domains in ways strictly static programming would prove to be infeasible. The main family of problems Machine learning proves to be useful on is of the kind where future (i.e., previously unseen) data can be classified using similar historic data. Spam filtering [30], optical character recognition (OCR) [48], and search engines are just a few examples where learning is important.

Sub-fields of machine learning include *supervised learning*, *unsupervised learning* and *reinforcement learning*. Supervised learning can be described as providing a machine with pairs of example input (predictors) and desired output, in order to learn rules that fit the end goal [18]. This relies on a human interacting with the system, with a predefined goal. On the other hand, unsupervised learning does not require a human user and is single-handedly responsible for establishing relational rules between the data points with a bigger potential of discovering non-obvious patterns [19]. Lastly, reinforcement learning is a learning method focused on the intermediate steps of a problem, often with trial and error [13]. Actions that increase the probability of achieving a specified goal receive rewards, and oppositely actions that decrease the same probability receive penalties. The rewards and penalties ultimately makes up a cumulative score (i.e., the measure of fitness), which should be maximized for best results.

The underlying models and methodologies in machine learning algorithms varies between the different applications, and should be decided with respect to the end goal. Methodologies include neural networks[37], linear and logistic regression [23, 28], decision trees and forests [39, 7], support vector machines (SVMs) [25] and combinations of these can be incorporated into ensemble models [43].

In Figure 2.4 seen above, the basic notions of Machine Learning is given. It can be summarized as follows:

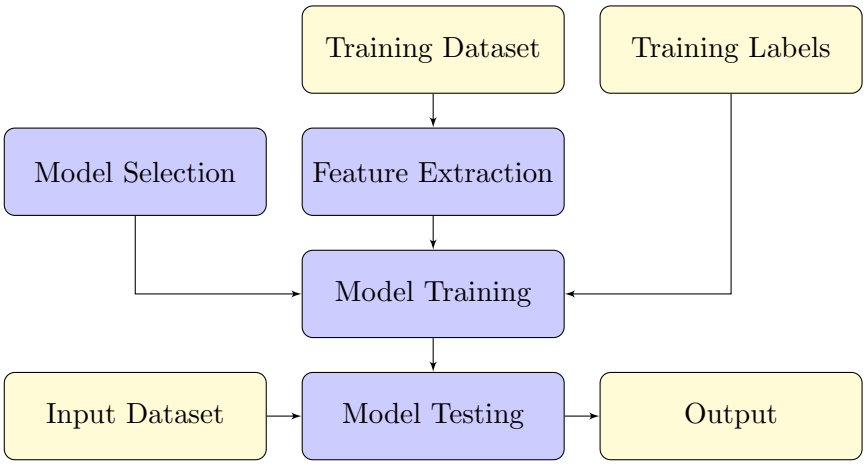

FIGURE 2.4: Basic overview of Machine Learning

1. **Model selection** - Selecting a statistical model out of a set of qualifying models. This involves considering the structure of the data set provided, and how well the model suits the data.

2. **Learning** - Observing the data and designing features that connect (portions of) the data together to facilitate the training of a model. Feature extraction and the model combined results in a learning algorithm that uses the training data as input for the model, allowing the model to fit the data. In addition to the input data, training labels should be provided to indicate how the input should be classified. This classification procedure plays a vital part in improving the overall performance of the model.

3. **Prediction** - With a model trained on the training data, we can apply the model on new input data (referred to as test data when used for evaluating the model) to predict or classify the corresponding output.

A problem of the kind where new observations have to be categorized (e.g., small, medium, large) or labelled (e.g., "income <50k", "income >50k"), is known as a *classification* problem. In Machine Learning, classification is considered supervised learning and revolves around identifying previously unseen information and giving them correct labels (i.e., discrete values) based on training. The unsupervised counterpart of classification is known as clustering. Typical examples of modern classification problems are speech recognition [49], OCR [48], DNA Sequence classification [3], credit scoring [35] and more.

One predictive model often used in classification problems is the *Decision Tree (DT)*. A decision tree is characterized by having a set of choices (i.e., branches) that are based on conjunctions of features that lead to target classes known as leaf nodes. Decision trees can be visually drawn to represent explicit decisions in respect to the attributes of the input data. They are simple by nature, as each decision branches into two or more

decisions or leaf nodes forming a finite set of paths in order to classify data. The input data can consist of both continuous (numeral) and discrete (categorical) values, without performing normalization. However, DTs have limitations as they may become overly complex and perform poorly at generalizing the training data very well. The problem of constructing an optimal DT is classified as an NP-Complete problem [22]. In general, they do not perform as accurate as other models [24].

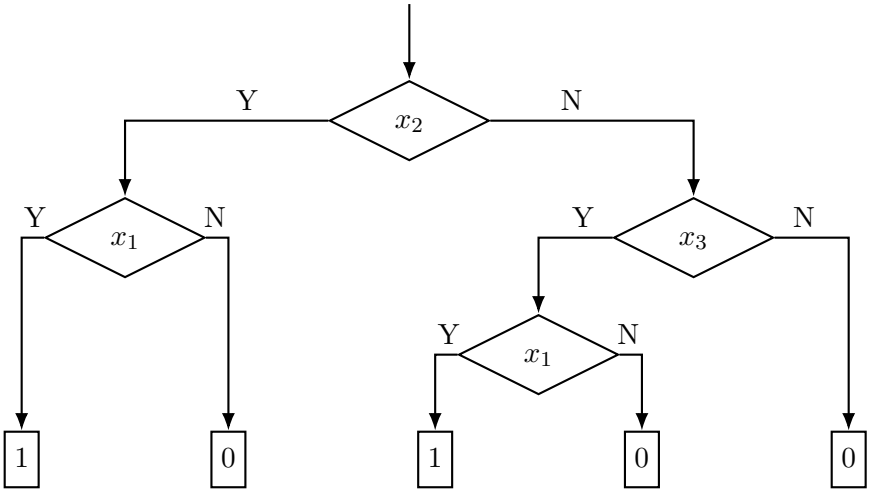

FIGURE 2.5: Example of a decision tree [1]

Improving on DTs' limitations, a *Random Forest(RF)* [7] is an ensemble learning method that effectively consists of multiple DTs which learn using different features from the same training data. The forest specifically corrects the overfitting of a singular DT, by selecting the mode of each individual tree's output it averages these to reduce the variance. When a decision tree is used as a stand-alone model, it will use all n features, while a forest typically uses $\sqrt{n}$ features randomly selected for each tree.

Another statistical learning method commonly used is *Regression*. Regression revolves around learning the relationship of independent continuous data given as input (i.e., predictors or regressors), and its dependent output. The predictors passed in are typically features derived from the input data, used to map their variance in the relation to the dependent variable. A common pitfall known to occur while using regression is that correlation is often mistaken for causation [1]. There are multiple types of regression, e.g., linear regression [23], polynomial regression [9], logistic regression [28] and more.

Linear regression is the particular case of regression where the mathematical functions used in predicting the value of dependent variables are linear. It uses the principles of Ordinary Least Squares (OLS) [42] in terms of minimizing the sum of squared differences between the learned function and the observed data. This means finding a function $f(x)$ defined as:

[1] https://tex.stackexchange.com/questions/289642/how-to-draw-a-proper-decision-tree

$$f(x) = \sum_{i=1}^{n} a_i x^i = a_0 + a_1 x + ...a_n x^n \tag{2.13}$$

where every coefficient a_i is learned by the model. If this function is a straight line, all coefficients for $a_i, i > 1$ are 0.

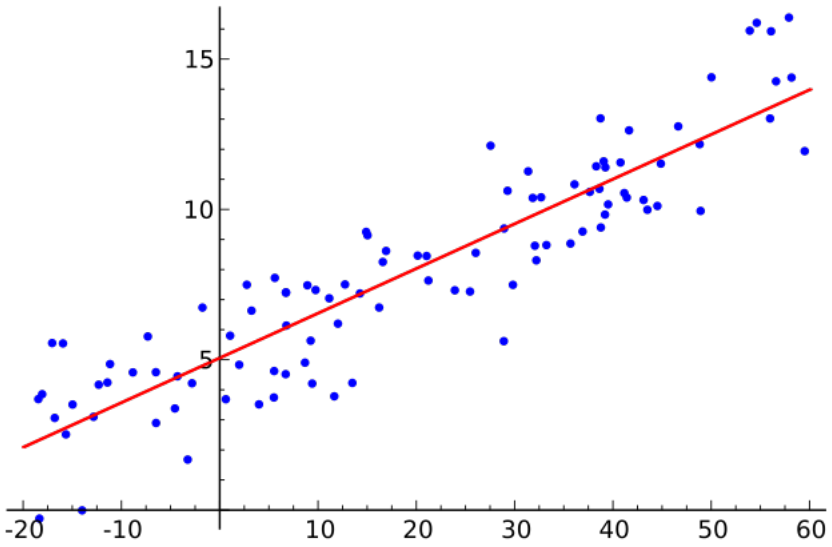

FIGURE 2.6: Illustration of linear regression [2]

In this paper, we run our experiments using linear regression.

2.3 Learning to rank

Learning to rank is becoming an increasingly popular research topic in information retrieval. By applying machine learning techniques to construct ranking models, we can solve ranking problems. Meaning, we can rank previously unseen lists based on familiar lists of the same type. Typically we use training data with labels signifying the importance of the data (see Fig. 2.4). The main differences between Learning to Rank and supervised machine learning is; machine learning can solve problems with predictions (e.g., classification), on a single instance at a time, while Learning to Rank solves ranking problems on multiple items at a time. It is not of significance what each value (e.g., relevance estimate) is, as long as the ordering in respect to items in a list is achieved.

The domain where learning to rank algorithms are applicable is large, including natural language processing [41], recommender systems [36], machine translation [12], computational biology [12], and e-commerce to name a few.

[2] Public Domain licensed, https://commons.wikimedia.org/wiki/File:Linear_regression.svg

Consider a system maintaining a collection of documents. For any query q, the system processes this query to locate documents containing all keywords from query q, ranks them and retrieves a list of the top ranked documents d_i. Ordinarily, the system performs this ranking with a function $f(q, d)$ which requires no training. This function is based on the probability of any particular document being relevant to the query, where the relevance is derived from words appearing in both q and d.

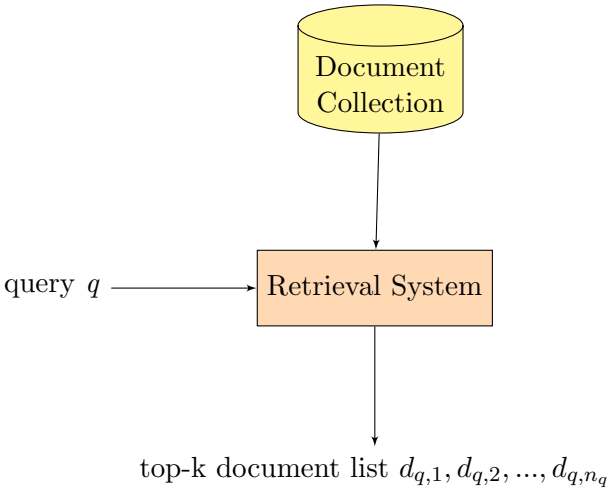

FIGURE 2.7: Overview of a top-k retrieval

In newer applications of learning to rank, especially in web search (e.g., PageRank [31]), machine learning methods have been applied to compute the ranking function automatically. Learning to rank goes under the category of supervised learning and involves training and testing of a model. For training models, feature vectors are created that indicate what makes a document relevant to the query. Each feature is a numerical score based upon properties and relationships between the document and the query, as semantic similarity, topical relevance and more. Extracting these features and constructing numerical patterns is known as relevancy engineering [52].

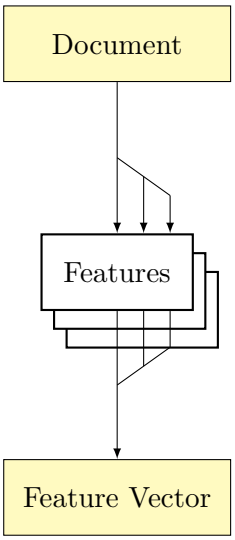

FIGURE 2.8: Passing features from a document into feature vectors

Machine-learned ranking constructs ranking models for a retrieval system, and is commonly implemented for re-ranking. Let's say a search engine with indexing and top-k (for some k, commonly top 10) retrieval already exists, and a user inputs a query, which prepares a list of all documents from the document collection matching the query. The list is not directly sent back to the user, but instead the top 10 elements are ordered by a ranking model before becoming the result page visible to the user.

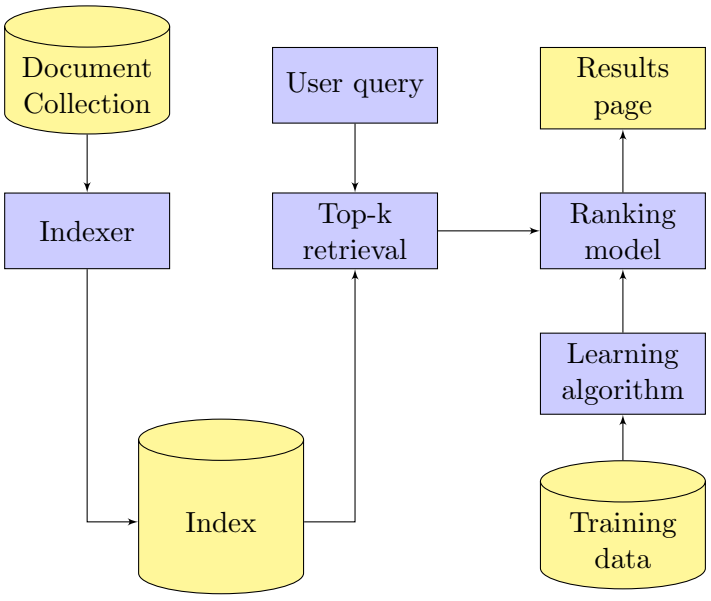

FIGURE 2.9: Conceptual overview of machine-learned ranking

Traditional search engines from the past have only been returning with the top-k retrieval, as depicted in the leftmost part of Figure 2.9. The machine-learned ranking is an additional step to improve on the order of documents retrieved, with more relevant documents first. It is computationally cheaper to perform the ranking on the list after

the top-k retrieval rather than before, due to the possible amount of matches in a large document collection.

In this paper, we utilize supervised learning with a query-product pair as input to a learning function, and build features with a calculated relevance score as output, to learn a ranking model.

2.4 Product Search

The Internet today has grown to have more online retailers, marketing services, libraries and e-commerce in general than ever before. The online search for resources is equally growing, and in particular products in retailer databases. Most online retailers incorporate search engines into their websites, engines that are based on adaptations of theoretical models developed for information retrieval.

Google is considered the number one search engine in terms of popularity [32], yet it was surpassed by Amazon [8]. Amazon attracted a sizable amount of corporate and indivitual listers (i.e., sellers) to its market place, due to its unique search algorithm. The difference between the two is; everything ranking-related is contained internally on Amazon, while Google uses external factors as history, link signals from external websites and social indicators.

Product search is tightly connected to the principles of information retrieval, with the key aspect of retrieving relevant and desired products based on what the customer is looking for. To benefit both the merchant and the customer, the merchant can come up with ways to enhance this challenge with personalization by utilizing user history as click-through rates (CTR), buy-through rates (BTR) and other resources of the customer base.

The data entailed in these collections tend to have the properties of being largely heterogeneous, and therefore serve as viable basis for research. However, while such data and their characteristics are valuable to study, they are in most cases considered sensitive and legally protected due to privacy laws [50].

Users shopping online tend to enjoy the convenience of lower prices compared to visiting a local physical retailer. Although this is a benefit to the user, they lack the ability to physically interact with or inspect products they are interested in. To compensate for this, resources as images, textual descriptions, retailer/lister reputation and product reviews play key roles. For applications where users may act as sellers as well as buyers, trust is essential[26]. Having a good track-record of past transactions with users in terms of reviews and buyer satisfaction increases the grade of trust.

Studies show that the visual attractiveness of products have high impact on user interest [10, 5]. In fact, the interest grows for higher quality images. Furthermore, evidence shows that users feel safer when purchasing products online when the images are sufficient in numbers, and have reasonably high quality.

Tuning which results are relevant in a retrieval is essential in personalizing web search. Papers written in this field describe probabilistic field mapping [14], probabilistic mixture models [11], dirichlet allocation based diversified retrievals [55] to name a few.

Chapter 3

Personalized E-Commerce Search Challenge

3.1 Task Definition

The problem at hand revolves around personalizing e-commerce in terms of fetching relevant results for user-issued queries based on the users' history and by association their inferred preferences. Given is a collection of data containing browsing logs, a product catalog and a set of queries. All of the data originates in a real e-commerce environment and therefore revolves around real users, real products and history. This allows for the design of custom features from the data, applying machine learning methods to predict relevance labels and rank products accordingly. Originally the queries contain a ranked list of products returned by a non-personalized ranker, upon which our goal is to re-rank. This challenge presents a unique opportunity for academic as well as industrial researchers to design and utilize methods and techniques to improve the prediction of relevant search results.

There are two main types of queries in the dataset, **queryfull** and **queryless**. Queryfull queries are recognized as they contain search string tokens (e.g., *"Used Car"*, or in our case something along the lines of *"1528, 230"*) while queryless queries only contain a category id (e.g., "Books", or rather *"232"*). The distinction is important as these are evaluated separately in accordance to the challenge, in addition to a final (average) score based on the two.

3.2 Dataset

The collection of data provided consists of 900MB of browsing logs, purchase history, products and meta data. Although this data is real, it has been obfuscated and

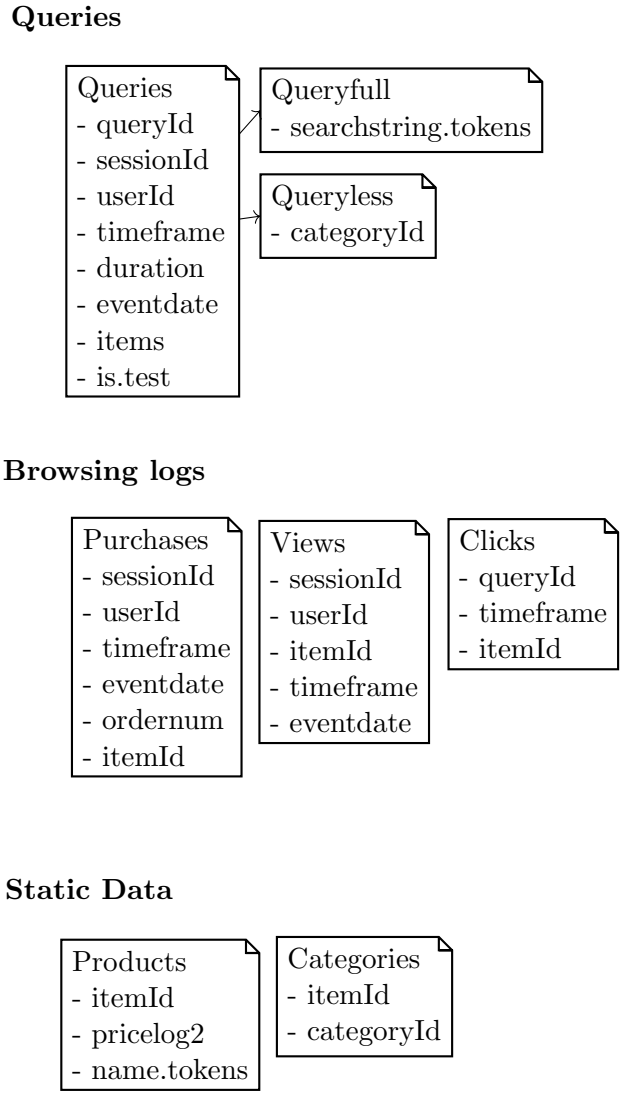

FIGURE 3.1: Illustration of the entities contained in the dataset

anonymized with the intention of making it impossible to trace the origin, preserving privacy of the connected retailer and users. Product names, user ids, query search string tokens, category names, prices, and more attributes have been transformed (hashed) in a way that preserves the integrity of the data, allowing the same attributes to be identified across different sources. e.g., a product with name tokens *"Macbook Air"* might be transformed to *"142,423"*, and similarly *"Macbook Pro"* might be transformed to *"142,512"*.

Illustrated in Figure 3.1 the different tables are presented with their associated attributes. The dataset comes as six different comma-separated-values (csv) files and contains the following:

- **Products** - Each row represents products in the catalog which are sold by the retailer with item id, price (transformed logarithmically) and the hashed product

name tokens in a list.

TABLE 3.1: Excerpt from products.csv

itemId	pricelog2	product.name.tokens
1	10	4875,776,56689,18212,18212,4896
69585	6	7583,18117,41805,41805,2371
90939	6	604,18117,41805,41805,2371

- **Product-categories** - Each row contains a mapping between a product and its respective category id.

TABLE 3.2: Excerpt from product-categories.csv

itemId	categoryId
139578	1096
417975	1096
291805	1096

- **Train-clicks** - Each row represents a click on a product and contains information about which query it originated in, the timeframe (milliseconds since the first query in a session) and the id of the product.

TABLE 3.3: Excerpt from train-clicks.csv

queryId	timeframe	itemId
1	16338861	24857
46255	1604912	30792
46689	3831948	8252

- **Train-item-views** - Each row represents a product being viewed in a session by a user containing the session id, user id (may be anonymous), item id, timeframe and the date (i.e., the timestamp) of the viewing.

TABLE 3.4: Excerpt from train-item-views.csv

sessionId	userId	itemId	timeframe	eventdate
1	NA	81766	526309	2016-05-09
1	NA	31331	1031018	2016-05-09
1	NA	32118	243569	2016-05-09

- **Train-purchases** - Each row represents a purchase a user has made on a product with information about the session, the user id, timeframe, eventdate, item id and the order number (one or more items purchased simultaneously are group by this number).

TABLE 3.5: Excerpt from train-purchases.csv

sessionId	userId	timeframe	eventdate	ordernumber	itemId
150	18278	17100868	2016-05-06	16421	25911
151	NA	6454547	2016-05-06	16290	175874
156	7	1721689387	2016-05-27	21173	35324

- **Train-queries** - Each row represents a query issued by a user, be it in the form of clicking a category or typing in free text. The rows contain data about the session, the user, timeframe, duration, eventdate, the items originally retrieved by the non-personalized ranker, and a flag denoting if this query is usable for training or for test. This flag being assigned the 'test' value indicates it is part of the test data, and similarly for training. In addition, depending on what type of query it is, it contains either search string tokens if it is queryfull or category id if it is queryless.

TABLE 3.6: Excerpt from train-queries.csv

queryId	sessionId	userId	timeframe	duration	eventdate	searchstring.tokens	categoryId	items	is.test
1	1	NA	16327074	311	2016-05-09	16655,244087,...	0	*7518,71,30311,7837,30792,...*	FALSE
2	2	NA	705527	314	2016-05-09	528941,529116	0	*70095,15964,,8627,134850,32754,...*	FALSE
3	3	NA	0	502	2016-05-09	133713,16655,...	0	*59081,51125,9338,9550,32087,...*	TRUE

TABLE 3.7: General data and behaviour statistics

#real users	232 816
#anonymous users	333 097
#sessions	573 935
#products	134 319 529
#product-terms	164 774
#views	2 451 565
#clicks	1 877 542
#purchases	68 818
#purchasing real users (distinct)	4 426

TABLE 3.8: Mean behaviour statistics

Avg. #sessions per user	1.608
Avg. #clicks per session	3.271
Avg. #views per session	4.271
Avg. #purchases per session	0.119
Avg. #purchases per real user	1.514

TABLE 3.9: Query Statistics

#queries	923 127
#query-terms	130 987
#query-item term overlap	37869
#query-item term overlap %	14.68
#query-less queries	869 699
#query-full queries	53 428
#queries by real users	348 240
#queries by anonymous users	574 887
Avg. #queries by real user	1.495

3.3 Evaluation Methodology

The predictions we make and the re-ranking of items is evaluated by a script provided with the dataset. We must format the predictions as seen in Table 3.10 where each row contains a query id, followed by the ranked items in order..

TABLE 3.10: Example of expected prediction result

queryid_1	$\text{itemid}_{11}, itemid_{12}, ...$
queryid_2	$\text{itemid}_{21}, itemid_{22}, ...$
...	...
queryid_n	$\text{itemid}_{n1}, itemid_{n2}, ...$

Running the script with our predictions along with the ground truth, results in an output containing three NDCG scores (see Eq. (2.9)): the query-less NDCG, the query-full NDCG and a final weighted average. These scores are used as basis for improvement over the baseline given by CIKM. Meaning, our predictions should produce an NDCG score higher than those of CIKM, preferably on all three points.

The baseline presented by CIKM is a simple one, but gives a strong starting point for improvement. It is defined as:

$$\hat{y} = 3 \cdot \#purchases + 2 \cdot \#clicks + \#views \tag{3.1}$$

where $\hat{y}$ is the estimator to be used in the learning procedure. It is, in itself, a feature combining the three types of user behaviour upon an arbitrary item with different weights. As purchases is the strongest indicator of relevance, it has the highest weight of 3, while clicks and views are respectively 2 and 1, being weaker indicators. Ranking the items based on Eq. (3.1) provides the scores seen on the last row in Table 3.11. Note that the weighted NDCG score tends to be closer to the NDCG value of query-less queries, as they take up 94.2% of all queries. According to the competition rules, the equation for computing the weighted NDCG score is defined as:

$$score = 0.8 \cdot NDCG_{ql} + 0.2 \cdot NDCG_{qf} \tag{3.2}$$

3.4 CIKM Cup and Participants

The international Conference on Information and Knowledge Management (CIKM) provides a forum for discussion and presentation on research within information management. Every year they hold competitions on problems in the domain targeted towards both industrial researchers and academia, attracting bright minds to participate and contribute with their experience. In 2016 they issued a data science competition cup, with multiple tracks, where one in particular was named *Personalized E-Commerce Search Challenge*. The challenge had 88 registered participants, with over 600 submissions in 2 months by top 5 teams.

TABLE 3.11: The top five winners and their results in terms of NDCG score, in comparison with the official baseline approach.

	final	**queryfull**	**queryless**
1. minerva	0.4262	0.5574	0.3935
2. Dmitrii_Nikitko	0.4149	0.5301	0.3861
3. tjy	0.4056	0.4570	0.3928
4. wistuba	0.3769	0.4495	0.3588
5. joaopalotti	0.3712	0.4860	0.3425
...	...	...	...
cikmcup2016_baseline	0.3514	0.3840	0.3433

Following is a brief summary for each of the top ranked participants from the testing leaderboards, as seen in Table 3.11 (Dmitrii_Nikitko's report was unfortunately never officially released):

1. **minerva (champion)** - The winning team of the challenge improved the results with 14.2% over baseline. Their approach includes use of different models, including gradient boosted decision trees, rank svm, logistic regression and lastly a novel

deep match model [4]. These models were incorporated into an ensemble that allowed for an intelligent prediction framework.

Features extracted include statistical features (i.e., static popularity), time-based features (dynamic popularity over time), query-item features (category-based token features, cross token features) and session features (emphasis on repetitive user behaviour patterns).

The proposed factor of success in this team's approach is the use of a stacking model ensemble. Layers of models within the ensemble are responsible for different parts of the prediction, and subsequently are subject to selection from a higher layer consisting of a model selector. This allows customizing singular models to predict on isolated product categories, as well as individual features.

2. **tjy** - Participants in third place of the challenge improved the baseline with an increase of 0.0542 in the weighted NDCG score. They put emphasis on deep feature engineering [29]. Focusing on designing clever features, they categorized into three groups: user features (category period and price preference), item features (local and global popularity, and category count), and lastly user-item features also known as personalized features (query-product token comparison, user-item scores).

   The experimental part of team tjy's approach involved testing the best combination and order of mentioned feature groups. First they evaluated each feature as an individual, discovering which feature had most predictive potential. Secondly, they grouped features together and tested for higher evaluation scores. Lastly they probed different permutations and combinations of both features internally and groups of features.

   Details surrounding the statistical model that was used for learning are not specified.

3. **wistuba** - Taking the fourth place, team wistuba improved with an increase 0.0255 in the weighted NDCG score. Team wistuba's approach built upon gradient boosted decision trees with hyperparameter optimization, with parameter configurations originating from other experiments on similar data sets [53].

   They employ a vast amount of features, some of which are directly derived from the dataset, and some which are preprocessed and aggregated into new combinations of data. These features include TF-IDF of search tokens and product name, click-through rate, buy-through rate, category-wise normalized clicks and purchases, and more. In an effort to simplify relevance prediction, they combine purchased and clicked items into one category marked relevant, and all others into another category marked non-relevant. They later stated that this simplification was unwise, as they lose information in the difference between the clicked and purchased products.

4. **jaopalotti** - Team joaopalotti made fifth place in the challenge with an increase in NDCG score of 0.0198. By utilizing a framework named XGboost with gradient boosted decision trees, he designed a large series of features categorized as follows: item-dependent, query-dependent, session-dependent, and item-query-dependent features [40]. Item-dependent features include popularity signs (views, clicks, purchases), ranking signs (original item rank), textual signs, and price signs. Query-dependent features include ranking signs, textual signs, price signs, and IR measure signs (NDCG, MRR). Session-dependent features include IR measure signs (NDCG per session) and length signs (queries per session, average session duration). Finally, item-query dependent features include popularity signs, ranking signs, price signs, textual signs, and group signs (category percentage).

   He emphasizes his biggest highlights as the large set of features, and his distinction of query-less and query-full queries into different sets.

3.5 Challenges

This problem comes with some challenging aspects which requires a certain amount of effort to work around. One of the key aspects is that the dataset is anonymized, which effectively removes the possibility of applying intuition based on actual product or category names. It is impossible to make any sense of the product names or the query string tokens as to what they actually signify, as intended. Though this is to protect the integrity of the retailer who has donated these data, this urges the challenge participants to apply generalized features upon user-item interactions instead of taking shortcuts saying e.g., if a product name contains the word "iPhone" it should immediately be marked as relevant.

Another challenge is to attempt to personalize the experience of *cold-start users*, i.e., users with no history. If a user just recently registered, and she has not yet purchased, clicked or even viewed any products, other than token matching on her search criteria and ranking based on general popularity, how could we know what to present as relevant for her?

Working with a dataset on this size requires certain memory and processing capabilities, as the experiments required to get results can be extremely memory-intensive and time-consuming. Stitching together different segments of data for calculations, depending on the complexity of the features in question, will easily exceed the limitations of what a personal home computer can perform.

Chapter 4

Approach

In this chapter, we describe the methods used including the features extracted, the statistical models selected and the details of how we evaluated our efforts to improve the relevance prediction.

4.1 Overview

The experiments we perform in this approach can be divided into four main phases:

1. Feature Extraction
2. Model Training
3. Model Testing
4. Prediction Evaluation

The first phase of our involved observing the dataset and experimenting with the possibilities of features to extract. By inspecting the dataset, we wanted to select a group of features that intuitively could yield information about relevance. Before doing so, we need to establish a numeric measure of relevance. From the perspective of e-commerce, the most important thing is the conversion rates, i.e., users purchasing products. Next, we define the clicks as second most valuable, as it is in all cases the precursor to a purchase. Following in this order would be views, but this resource was on dropped as an indicator of relevance due to its poor performance compared to purchases and clicks. This leaves us with three relevance indicators, the third being the case where an item was not either clicked or purchased, which we can use alongside the training data as *training labels*. The values for these indicators are matched against query-item pairs and are defined as:

- $r = 2$: The item was purchased as a result of the query
- $r = 1$: The item was clicked as a result of the query
- $r = 0$: The item was neither purchased or clicked as a result of the query

Using these values, we effectively specify the relevance grade of known results of queries from the training data. This allows us to train a model to recognize which of the provided features have significant impact on the relevance.

The features derived from the dataset were composed with a sense of intuition, observation, and mostly trial and error. For this purpose, an extensive experimental framework was implemented to facilitate all of the different tasks involved from designing a feature, and to evaluate its performance. This framework lets us perform feature extraction, model training and testing, and predictions by entering few command line inputs, allowing us to only focus on implementation of features and their respective evaluations.

In the later phases of the experiments, we provided input in the form of a query-item pair, and the corresponding output was represented by a relevance label. Intermediately, depending on what features are used in the models we trained, the feature values are calculated using the data connected to the query-item pair. Subsequently, these feature values are incorporated into *feature vectors*, which consist of the features selected for each experiment. For all query-item pairs, we build a set of feature vectors and simultaneously build a set of relevance labels corresponding to each feature vector. A model is finally fitted requiring the feature vectors and relevance labels as input, and we serialize the model to disk so we can retrieve it for further operations (e.g., testing).

A list of the final feature groups is displayed in Table 4.1 with a brief explanation of each feature. In the next section we explain our thought process behind designing these features, and describe how they are calculated.

4.2 Feature selection

The challenge put strong emphasis on distinguishing between queryless and queryfull features, so we attempted to design the features accordingly. Considering both types of queries, we extracted a set of features that allowed improvement on both query families. We have divided our features into three groups, *Query-item-dependent features*, *Item-dependent features*, and *User-item-dependent features*. Several of our features were designed exclusively to a designated type of query, yet others yielded value for either type. When features are selected and subsequently computed, we build a vector containing these features, and we divide the type of vectors into queryfull and queryless so that we can train a model for each type of query. This allows for more precise learning in each model of its query type.

TABLE 4.1: Table displaying overview of features

Query-item-dependent features		**Value**
category_clicks (c, i)	Frequency of clicks performed on an item normalized on total item-clicks in respective query-category	[0, 1]
category_purchases (c, i)	Frequency of purchases made on an item normalized on total item-purchases in respective query-category	[0, 1]
mean_prior_rank (q, i)	Average ranking of an item in respect to a category in the original list of retrieved items	[0, 1]
cosine_similarity (q, i)	Cosine similarity between query search-string and product name (TFIDF-weighted) token vectors	[0, 1]
Item-dependent features		
clicks_normalized (i)	Frequency of clicks performed on a given item, normalized on maximum of all clicks	[0, 1]
purchases_normalized (i)	Frequency of purchases made on a given item, normalized on maximum of all purchases	[0, 1]
associated_purchases (i)	Number of items purchased together with a given item, normalized on largest order size	[0, 1]
User-item-dependent features		
user_revisit (u, i)	Total number of past clicks a given user has performed on the given item	[0, inf]

4.2.1 Query-item-dependent features

Query-item-dependent features are, as the description suggests, features that rely on data connected to the relationship between a query and an item retrieved by the respective query. Following are the four features we selected as part of the query-item-dependent group.

Our first feature, *category_clicks*, is derived from item clicks in respect to its category. As queryless queries only contain a category id and no search string, this feature proved to be of value. Considering a query-item pair (q, i) we can compute this feature as follows:

$$category\_clicks(c, i) = \frac{\#clicks(i)}{\sum_{k \in c} \#clicks(k)} \tag{4.1}$$

where c denotes the category provided by the query q, $\#clicks(i)$ denotes the number of clicks for item i, and k denotes an item existing in category c. The ratio given by this feature is the number of clicks made on an item, divided by the total number of clicks on all items in the same category. If a category contains a large number of items, the most clicked items will take up a larger portion of the respective category's clicks and Eq. (4.1) reflects this accordingly. Oppositely if the category has a smaller selection

query, **category**, **items**
53454, 480, *205906,88040,156599,268361,193340,18373,149012...*

60083, 480, *156599,268361,132892,55772,44931,149016,121979...*

61001, 480, *156599,395381,268361,193340,18373,132892,55772...*

61123, 480, *156599,268361,18373,132892,55772,44931,141365,...*

FIGURE 4.1: Excerpt from the set of queries, displaying varying permutations of items

of items, the denominator is smaller and thus the feature score will be higher. This feature is by nature best described as a queryless feature, but c may be substituted with a category the item is contained in, in order to be applied on queryfull queries as well.

A similar feature is named *category_purchases*, which is expressed as:

$$category\_purchases(c, i) = \frac{\#purchases(i)}{\sum_{k \in c} \#purchases(k)} \tag{4.2}$$

This feature applies the same calculations on the purchase resource instead of clicks, and could therefore also be used as a query type-independent feature.

Upon close inspection of the dataset, an assumption was made that the original rankings were in some way reflecting the relevance of the items in the retrievals. This possible reflection was worthy of further investigation, as the rankings themselves could serve as a feature. However, as all queries have distinct identifiers, there was no way to reuse a query's original ranking in its given shape as queries in the training set did not overlap with any queries in the test set. A workaround was then implemented, by using the category from each queryfull query instead of the raw query id. Upon further inspection, it turned out that for all queries in the same category, the rankings had slight variations.

Figure 4.1 displays an excerpt from the dataset containing multiple queries on the same category. In most cases the highest ranked items remained in the top spots, although shuffled. One assumption is that the rankings' variation could be explained by factors as decisions made by the retailer over time. Another assumption is that the variations could be explained as an already existing form of personalization. No conclusions were to be made, but we managed to construct a feature on this behaviour, named *mean_prior_rank*:

$$mean\_prior\_rank(q, i) = \begin{cases} 1 - \frac{\sum_{c \in q, i \in q} rank(i)}{n_{q,i}}, & \text{if } i \in q \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

where $rank(i)$ denotes the normalized rank (i.e., index) of item i with respect to all items in query q, and $n_{q,i}$ denotes the number of queries item i appear in. We invert the result of the fraction to get a feature score in the range $[0, 1]$, where a value closer to 1 indicates it is on average ranked among the top items.

Queryfull features include searchstring terms, which in some cases directly overlap with product name terms. This is an example of a direct textual match of the search, but as there is a mere 14.7% overlap between the searchstring terms and product name terms in total, this does not occur frequently. The next feature, *cosine_similarity* tries to find items with product name terms similar to the search string terms through the use of TFIDF weighted values. By using the TFIDF values from the product name tokens and the query searchstring tokens we can perform a cosine similarity function upon the two sets. Given vectors containing the presence of terms in a query and a product name, the cosine similarity is represented on a scale from 0 to 1. The equation is formally described as:

$$cosine\_similarity(q, i) = cos(\theta_{q,i}) = \frac{\sum_{j=1}^{n} term_{q,j} \cdot term_{i,j}}{\sqrt{\sum_{j=1}^{n} term_{q,j}^2} \cdot \sqrt{\sum_{j=1}^{n} term_{i,j}^2}} \tag{4.4}$$

where $\theta_{q,i}$ represents the angle between the TFIDF values of the query terms and the item terms, $term_{q,j}$ and $term_{i,j}$ denotes the TFIDF values of the terms in the query and the item respectively. This feature effectively represents how well the query matches the item in terms of tokens.

4.2.2 Item-dependent features

In addition to the query-item dependent features, we also include a set of item-dependent features. These are features that depend on the item itself and its related behavioural statistics. Following are the three features selected for the item-dependent group.

By assuming purchases and clicks being the most valuable indicators for relevance, we focus on features that utilize these resources. The two next features are based on general popularity of the items in the product catalog, namely *clicks_normalized* and *purchases_normalized*. They make up two equations as follows:

$$clicks\_normalized(i) = \frac{\#clicks(i)}{max(\#clicks(P))} \tag{4.5}$$

where $\#clicks(i)$ denotes the number of clicks on item i, and $\#clicks(P)$ denotes the clicks of products in the product catalog P. The score of this feature is normalized

as the denominator of Eq. (4.5) selects the number of clicks with respect to the most frequently clicked item.

$$purchases\_normalized(i) = \frac{\#purchases(i)}{max(\#purchases(P))} \tag{4.6}$$

where the same definitions apply as seen in Eq. (4.5). Both of these features are considered query type-independent, and relies only on the respective user behaviours. Having this property, we can include these features in both sets of feature vectors.

Lastly, we try to extract information that could be described as purchases associated with an item. Items purchased are grouped by an attribute named *ordernum* which tells us which items were bought together. By looking at items that are frequently bought together, an assumption is developed claiming the items bought in the same order as a given item may be of relevance. Also, this relevance would be proportional to the frequency of the items coexisting in orders. We perform preprocessing on this feature in order to count all pairs of items bought together, and group them in a dictionary of bins. Each key of the dictionary corresponds to an item id, and the value is a bin of items that were bought with the respective item id. The sizes of these bins tells us about the range of items that are purchased alongside each item. The feature *associated_purchases* is defined as:

$$associated\_purchases(i) = \frac{n_{i,o} - 1}{max(n_o)}, i \in o \tag{4.7}$$

where $n_{i,o}$ is the number of items in an order o containing item i, and the normalizing denominator $max(n_o)$ denotes the order with highest amount of items. We subtract 1 from the size of the order to effectively skip the counting of item i. As a result, this feature provides a zero value when an item has never been purchased alongside other items. However, when items have been purchased with other items, we raise the value proportionately to the number of associated items.

4.2.3 User-item-dependent features

The last category of features contains one feature, which is named *user_revisit*. The idea of this feature is that when a user makes a purchase, it is likely that the user has visited (i.e., clicked) the item one or more times beforehand. Items that have been revisited by the same user multiple times indicate complex signals [27]; interest and hesitation, which in turn may serve as basis for a feature. By counting and grouping all clicks made by a user, we establish a dictionary where the key is a (user id, item id)-tuple, and the value is the amount of times the user has clicked this item. It is formally defined as:

$$user\_revisit(u,i) = \begin{cases} \#clicks(u,i) & \text{if } u \in U \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where u is the user id, and i is the item id. This equation is simple where the user id exists, as we make the assumption that all test queries appear after the train queries. Meaning, we intuitively think of the train queries as "past" or "history", while the test queries represent "present" queries. There is no normalization performed, so this feature will give raw values for the amount of times an item has been clicked by the same user. It is often the case that the user is anonymous, in that case the feature provides a zero value.

4.3 Supervised Learning

To make use of the features extracted, we need to put them into *feature vectors*. An example of a feature vector looks as follows:

$$F = (f_1, f_2, ..., f_n) \tag{4.9}$$

where F is the feature vector containing n features f_i. Note that as we distinguish between the different query types, we subsequently need to distinguish between two kinds of feature vectors. This gives us (1) the queryless feature vectors including features that are queryless or compatible with both and (2) the queryfull feature vectors including features that are queryfull or compatible with both. This is important as we train two separate models for each query type.

The performance of these features are evaluated individually and in groups as seen in Table 4.2.

Having extracted the features and vectorized them, we also need to compute the actual relevance of our training set. For convenience we performed preprocessing on all the training queries, and the items contained, in order to label them with a defined measure

TABLE 4.2: Features and feature groups subject to performance measuring

Label	**Feature vector configurations**	**Query target**
1	category_clicks	both (substitute cat. for q.f.)
2	category_purchases	both (substitute cat. for q.f.)
3	mean_prior_rank	queryless
4	cosine_similarity	queryfull
5	clicks_normalized	both
6	purchases_normalized	both
7	associated_purchases	both
8	user_revisit	both
	Queryfull + Queryless f. v. config.	
Group A	5,6,7,8 + 3,5,6,7,8	mixed
Group B	4,5,6,7,8 + 3,5,6,7,8	mixed
Group C	1,2,4,5,6,7,8 + 1,2,3,5,6,7,8	mixed

of relevance as stated in Section 4.1. These labels are referred to as training labels, and specify relevance values for the query-item pairs. By providing a vector of feature vectors as *train_x* and a vector of relevance labels *train_y* where each vector in train_x corresponds to a relevance label in train_y as input to the model, we perform the fitting. The training procedure is explained in Algorithm 1:

Algorithm 1 Training models with feature vectors and relevance labels

```
1:  procedure TrainModel(train_queries, train_labels, feature_dict)
2:      model_qf ← init LinearRegression
3:      model_ql ← init LinearRegression
4:      f_qf ← []                          ▷ Vector of queryfull f. vectors
5:      f_ql ← []                          ▷ Vector of queryless f. vectors
6:      rel_qf ← []                        ▷ Vector of queryfull rel. labels
7:      rel_ql ← []                        ▷ Vector of queryless rel. labels
8:      for all train_query in train_queries do
9:          for all item in train_query do
10:             rel_val ← relevance label from train_labels
11:             if train_query is queryless then
12:                 f_vector ← queryless feature values from feature_dict
13:                 f_ql ← f_ql + f_vector
14:                 rel_ql ← rel_ql + rel_val
15:             else if train_query is queryfull then
16:                 f_vector ← queryfull feature values from feature_dict
17:                 f_qf ← f_qf + f_vector
18:                 rel_qf ← rel_qf + rel_val
19:     trained_model_qf ← model_qf.fit (f_qf, rel_qf)
20:     trained_model_ql ← model_ql.fit (f_ql, rel_ql)
21:     return trained_model_qf, trained_model_ql
```

where *train_queries* represents all the queries from the train set, *train_labels* is the list of relevance labels that correspond to each query in train_queries, and *feature_dict* is a preprocessed dictionary with a (feature_name, query_id, item_id) triple as key, and the feature score as value.

After the models have been successfully fit with training data and training labels, they are applicable for predicting results of previously unseen queries. The prediction phase is similar to the training phase, although as we predict scores query by query, we do not require a vector of feature vectors. Instead, we produce a similar feature vector and feed it into a model we effectively estimate the relevance as output. The procedure of predicting is described in Algorithm 2:

Algorithm 2 Predicting and writing results out to file

```
1:  procedure PerformPrediction(model_ql, model_qf, test_queries, feature_dict)
2:      for all test_query in test_queries do
3:          item_score_map ← {}
4:          for all item in test_query do
5:              if test_query is queryless then
6:                  f_vector ← queryless feature values from feature_dict
7:                  score ← model_ql.predict (f_vector)
8:              else if test_query is queryfull then
9:                  f_vector ← queryfull feature values from feature_dict
10:                 score ← model_qf.predict (f_vector)
11:             item_score_map[item] ← score
12:         scorestr ← item_score_map keys ranked by score sep. by comma
13:         writeln (test_query + scorestr)
```

where *model_ql* is a trained queryless model, *model_qf* is a trained queryfull model, and *test_queries* represents all the queries from the test set.

The fourth and final phase of the experiment revolves around evaluating the predictions retrieved in phase three. CIKM published their scoring script used in the competition which we use for our predictions. Alongside the scoring script, a ground truth reference document is also provided, describing the optimal rankings of the test queries. Iterating through all of the experimentally predicted queries, each query gets a respective NDCG scores using an item-relevance look-up from the ground truth reference. The final output is a set of three NDCG scores; one for queryfull queries, one for queryless queries, and the weighted average of the two.

4.4 Implementation

The actual implementation is a framework written in Python 3 with Anaconda version 4, and uses modules as Scikit-Learn for the learning of statistical models, NumPy as basis for scientific computation, and Pandas for data structures and transformation of these.

The resulting framework performs single or batch operations for feature extraction, model training, model testing and prediction evaluation. In other words, every phase of the main experiments can be executed as a task in the framework.

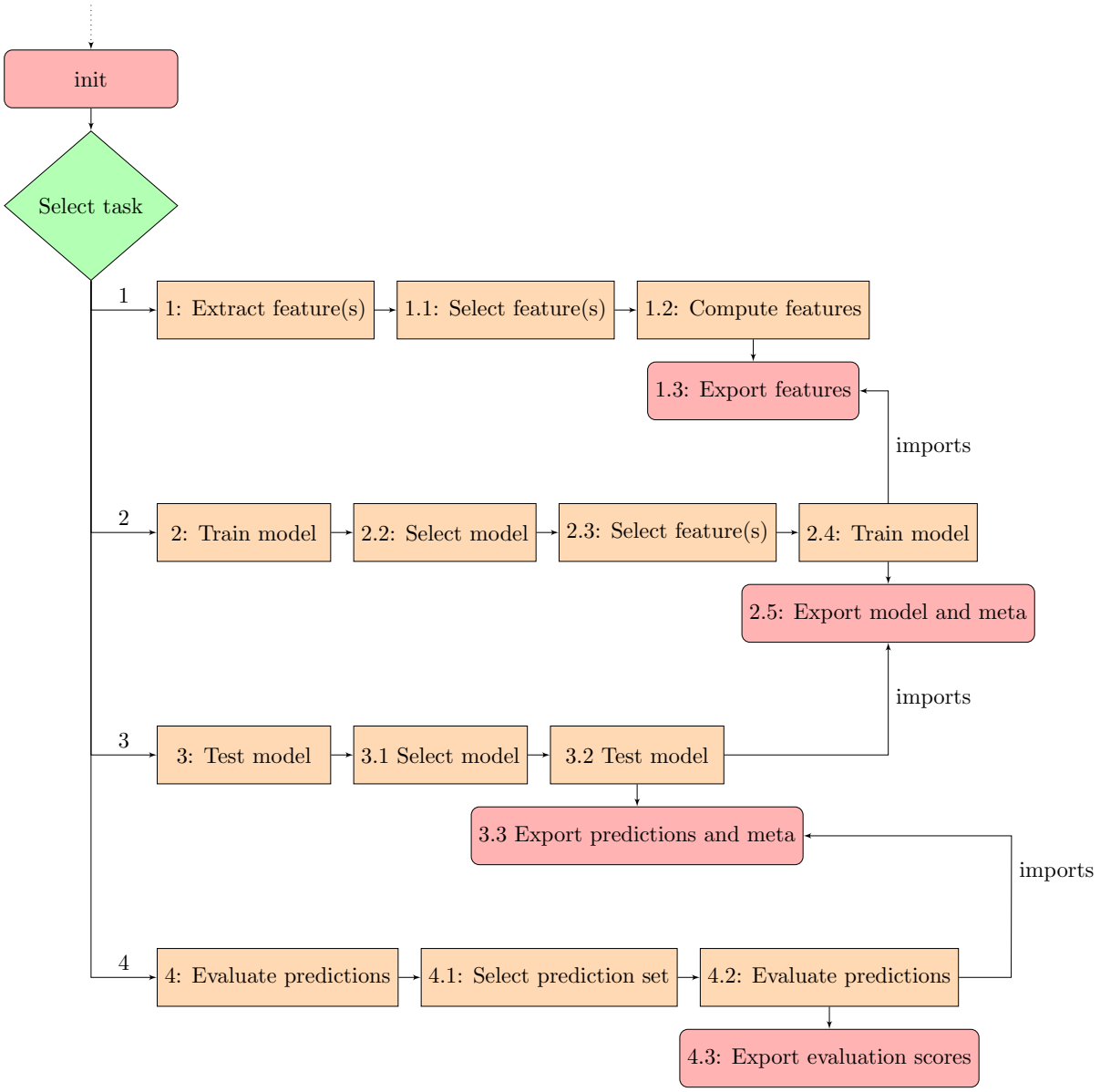

FIGURE 4.2: Architectural overview of the experimental framework

As displayed in Figure 4.2, each task produces an export of data. Task 1 produces features extracted, Task 2 produces a trained model and meta-data about features used, Task 3 exports predictions and similar meta-data (for human readability), and Task 4 exports the scores of prediction evaluation. The strength in dividing the experiments into these 4 steps is minimizing the amount of work to redo if an exception occurs because of invalid data, erroneous coding or by running out of memory. As each task exports the data it has computed, the next task can read this data from disk on demand.

In addition, we can rerun subsequent tasks as many times as we like as the exported data from previous tasks is not modified.

This framework has undergone multiple revisions to perform the computational tasks as efficient as possible with respect to running time and memory usage. Loading one or several large dataframes into memory and merging them into features has to be done with caution in order to stay below the memory limitations of the computer running the experiments. Processing the data loaded from large files in chunks at a time helped by significantly reducing the memory usage.

In detail, Task 1-4 from Figure 4.2 are summarized as follows:

- **Task 1: Extract features** - Load and transform data from training files into features to be used in the model training. The feature-options available in Step 1.1 correspond to scripts that have been written with two key concerns; preprocessing of data and computation of a specific feature value for a given query-item pair. All feature-scripts share these two functions, and are responsible for loading and preprocessing the data required for computation of its feature values, so that we can efficiently get them on demand when training the model. To achieve this, we produce intermediate files containing triples of $(queryid, itemid, featureval)$ to be used in Task 2.

```
orders = pd.read_csv(os.path.join(feature.options["data_dir"],"train-purchases.csv"),
sep=";").groupby(["ordernumber","userId"]).agg({'itemId': lambda x: ",".join(str(i)
for i in list(x))}).reset_index()
```

FIGURE 4.3: Excerpt from a feature extraction script, mapping purchase orders of users to lists of items purchased

- **Task 2: Train model** - Decide upon a model (e.g., linear regression, random forest regression) and load the intermediate files exported in Task 1. When training the model, a choice is presented as to which features should be included. This allows for dynamic selection of features, for either evaluation of individual features or groups of features. Once the model is trained, it is serialized and stored on disk alongside a meta-data file containing the names of the features used in the training. These outputs are used in Task 3.

```
f_qless.append([[[c for c in file_row.strip().split(",")]
for file_row in joint_row][i][-1] for i in qless_indices])
```

FIGURE 4.4: Excerpt from model training, appending queryless features to a vector of feature vectors

- **Task 3: Test model** - Predict the ranking of the items in a query for all queries in the test set, using a selected pre-trained model and its related meta-data. The

meta-data comes into play when we want to recreate feature vectors, as we need to know which features to include. In a similar fashion to that of Task 1, we extract features for said vectors, to provide new observations in the same format familiar to the model. As each query-item pair receives a predicted relevance score, we can rank all the items within a query. Lastly, all queries and their ranked items are exported to disk for evaluation in Task 4. The meta-data exported from this is mainly used by a human to know which combination of model and feature set was used.

```
f.write(" ".join([str(query_params["query_id"]), ",".join([str(x[0]) for x in
sorted(item_scores.items(), key=operator.itemgetter(1), reverse=True)])]+"\n")
```

FIGURE 4.5: Excerpt from model testing, writing predictions with leading query id and a comma separated ranked list of item ids

- **Task 4: Evaluate predictions** - Run the scoring script with the exported predictions to receive NDCG scores. This script is provided by CIKM and is slightly modified to be contained within the framework as a cradle-to-grave solution. The output of task 4 is written to disk, and is the result of our experiment as a whole, being the scores of how well the model performed with the extracted features.

Chapter 5

Results

Recall, the goal of this challenge is to re-rank the items retrieved by all queries in the dataset provided by CIKM. Doing so, we wish to improve the NDCG scores by using our custom features in combination with a model trained through linear regression. This chapter provides results for our attempts. Recall that the NDCG measure is calculated on our predictions with respect to the ground truth, and for this task the score is divided into three parts; queryfull NDCG, queryless NDCG and a weighted average between the two.

5.1 Individual Features

The results presented in Figure 5.1 and 5.2 are NDCG scores for evaluating models trained on individual features, ranked in descending order with respect to the average NDCG score. The labels of each feature on the horizontal axis correspond to those of Table 4.2.

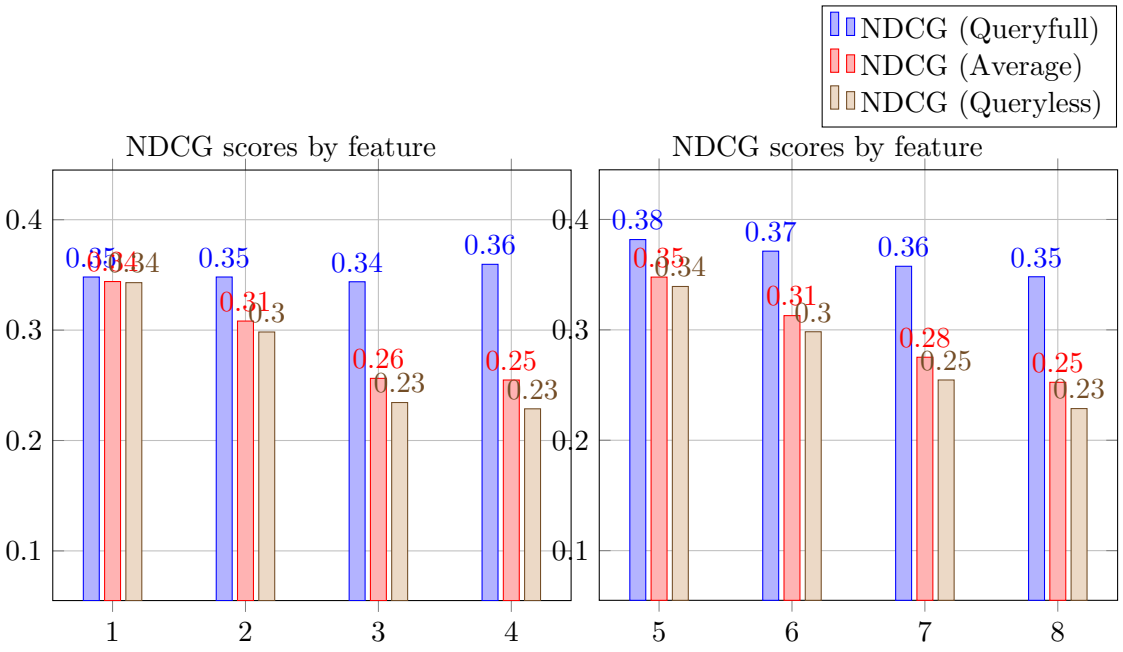

FIGURE 5.1: Q.-I.-dep. features

FIGURE 5.2: I.-dep. and U.-I.-dep. features

Exact values for the individual feature evaluations are displayed in Table 5.1.

TABLE 5.1: Evaluation scores for models trained on individual features, ranked on average NDCG

Label	NDCG (Final)	NDCG (Q.f.)	NDCG (Q.l.)	Feature
5	.3478	.3818	.3393	clicks_normalized
1	.3436	.3481	.3425	category_clicks
6	.3129	.3713	.2983	purchases_normalized
2	.3082	.3481	.2983	category_purchases
7	.2752	.3576	.2546	associated_purchases
3	.2563	.3438	.2344	mean_prior_rank
4	.2548	.3596	.2287	cosine_similarity
8	.2525	.3481	.2287	user_revisit

Looking at the results of the individual features, clicks_normalized performed best on queryfull with a score of .3818 and on the average score with a score of .3478. The best performing feature on queryless queries is category_clicks with a score of .3425.

5.2 Feature Combinations

The results presented in Figure 5.3 are NDCG scores for evaluating models trained on feature groups, ranked in descending order with respect to the average NDCG score. The labels of each feature on the horizontal axis correspond to those of Table 4.2.

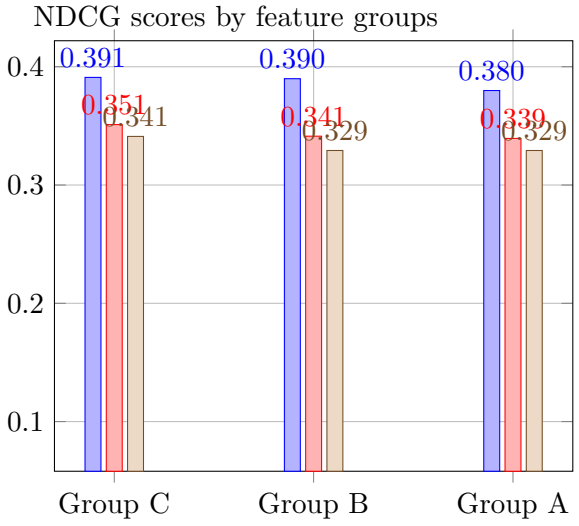

FIGURE 5.3: Feature Combinations

Exact values for the feature combination evaluations are displayed in Table 5.2

TABLE 5.2: Evaluation scores for models trained on groups of features, ranked on average NDCG

Label	NDCG (Final)	NDCG (Q.f.)	NDCG (Q.l.)	Feature conf. (Q.f. + Q.l.)
Group C	.3511	.3910	.3411	1,2,4,5,6,7,8 + 1,2,3,5,6,7,8
Group B	.3413	.3899	.3292	4,5,6,7,8 + 3,5,6,7,8
Group A	.3394	.3799	.3292	5,6,7,8 + 3,5,6,7,8

Group C performed best out of the three groups of features.

5.3 Overall Results

The overall results for the top performing features and combinations are displayed in Table 5.3.

TABLE 5.3: NDCG scores for best performing feature configurations for queryless and queryfull features

Label	NDCG (Final)	NDCG (Q.f.)	NDCG (Q.l.)	Feature(s)
Best feature (QF)	0.3478	0.3818	0.3393	clicks_normalized
Best feature (QL)	0.3436	0.3481	0.3425	category_clicks
Best feature group	0.3511	0.3910	0.3411	Group C
CIKM Baseline	0.3514	0.3840	0.3433	

Chapter 6

Discussion

The features chosen for this task were developed with a single concern, to act as indicators for whether or not a feature would increase the relevance for an item in relation to a query it is contained in. Effectively, every feature can be thought of as its own hypothesis stating e.g., "The similarity of the keyword search entered by a user and the product names in the results will impact the relevance of the products". The scores calculated for each feature then proves the correctness of the hypotheses in a scale from zero to one, where higher values mean higher correctness.

It is clear that the click-oriented features in Table 5.1 perform best overall. The clicks_normalized feature works well on queryfull features as well as queryless features, as it is solely derived from the items' interaction statistics. On the other hand, category_clicks improves on the queryless queries as intended, but performs worse on the queryfull queries, which is sensible as this feature normalizes item clicks on the total amount of clicks in a category specified by the queryless query. Substituting the category with an individual item's respective category may have distorted this feature in the case where the query returns items within multiple categories, each with significant differences in the cumulative amounts of clicks.

The purchase-oriented features were expected to have more impact than the click-oriented features, due to their assumed strong indication of relevance. However, even though the scores these features received were adequate, they did not surpass the click-oriented ones. This is most likely a result of having a significantly higher amount of click events occurring than purchase events, leading to an inadequate amount of basis data for the model training. Interestingly, the purchase-features received identical scores on queryless queries whereas the click-features showed improvement when categorizing. Again, this is probably explained by the insufficient amount of purchase data as shown in the statistics in Table 3.7.

Looking at the results for the features combined in groups from Table 5.2, the best results are better than the results of each individual feature. This is not surprising as

the scoring of individual features acted only as precursors for our main attempts with the combinations of features. Group A was the first group to be composed with a set of features that could fit in both query types, along with mean_prior_rank for queryless queries. This configuration scored .3394 on the average score, which is just below the two top performing individual feature-scores.

The key difference from Group A to Group B was to emphasize the queryfull queries, so the feature cosine_similarity was added to the feature vectors. Even though the cosine_similarity scored relatively low compared to the other features individually, this feature still improved the queryfull score to a value of .3899. The baseline scored .3840 for queryfull queries, so we succeeded in improving the NDCG for this type of queries.

Group C is a continuation of Group B with two added features to further improve the scores of the queryless queries; category_clicks and category_purchases. These two features improved all of the scores, making the average score our best at .3511. Overall the features ended up being effective towards making the goal of this thesis.

Chapter 7

Conclusion

In this thesis we have addressed the task of learning to rank on an e-commerce domain acting as our submission to CIKM Cup 2016 Personalized Search Challenge. We have applied our best efforts in analyzing historical data, browsing logs and user interactions from the data set and accordingly, we designed custom features. The features extracted have been applied to train statistical models to predict and maximize relevance of items in respect to the queries they appear as results.

We have developed an experimental framework that performs each segment of the experiments in tasks for convenience, allowing efficient processing as opposed to taking greedy approaches prone to memory issues. We have utilized this framework to run tasks that yielded all the necessary data as basis for insights for this thesis.

We have designed custom features by thorough inspection and observation of the data set provided. We found that for our selection of features, the click-oriented ones performed the strongest. All features we have developed proved to be effective in improving the re-ranking of queries. The scores for the feature groups have successfully shown to be increasing for added complexity in the experiments. As a result, the baseline has been surpassed on the queryfull scores.

In future work, more features can be derived with more focus on improving the query-less features. More advanced machine learning algorithms should be explored to cross-compare performance. With limited resources available at the time of writing, we have not performed the experiments on more complex models as the hardware specifications required for running reasonably large experiments exceeded the capabilities of the hardware we disposed. The experiments we have performed were run on a virtual machine in Azure Cloud Services, which if we were to continue using, would become too expensive.

Further improvements may also be done on personalizing features as there have been a list of attributes we have not pursued, as well as an abundance of anonymous users.

Time-specific popularity windows, trends and more sophisticated means of finding indications of relevant aspects of the data could be explored. Further exploration could also be done on sessions, connecting associated user-interactions to the corresponding sessions to gain new insights. In the case of textual comparison between query and product, there are methods to deduct semantic relevance between (groups of) tokens that possibly could overcome the problem of having insufficient overlapping tokens.

Acknowledgement

I would like to thank my mentor and supervisor, Krisztian Balog from the University of Stavanger, for sharing his expertise and thus providing valuable insights and pointing me in the right direction in times of need.

I would also like to thank Glenn F. Henriksen from Capgemini for providing access to state-of-the-art resources through Azure. Running highly memory-intensive experiments was one of the main challenges over the entire run, however with access to powerful virtual machines in the cloud, this was no longer a problem. I could not have gotten nearly as many experiments done without his contribution.

Bibliography

[1] J. Scott Armstrong. "Illusions in regression analysis". In: *International Journal of Forecasting* 28.3 (2012), pp. 689–694.

[2] Joeran Beel et al. "Research-paper recommender systems: a literature survey". In: *International Journal on Digital Libraries* 17.4 (2016), pp. 305–338.

[3] Giosué Lo Bosco and Mattia Antonino Di Gangi. "Deep learning architectures for DNA sequence classification". In: *International Workshop on Fuzzy Logic and Applications*. Springer. 2016, pp. 162–171.

[4] Ming Yan Chen Wu and Luo Si. *Ensemble Methods for Personalized E-Commerce SearchChallenge at CIKM Cup 2016*. Tech. rep. 2016.

[5] Sung H. Chung et al. "The Impact of Images on User Clicks in Product Search". In: *Proceedings of the Twelfth International Workshop on Multimedia Data Mining*. MDMKDD '12. Beijing, China: ACM, 2012, pp. 25–33.

[6] Nick Craswell. "Mean Reciprocal Rank". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Boston, MA: Springer US, 2009, pp. 1703–1703.

[7] Adele Cutler, D. Richard Cutler, and John R. Stevens. "Random Forests". In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 157–175.

[8] Andrew Davis. *Amazon Passes Google as Top Destination for Shopping Research [Report]*. https://searchenginewatch.com/sew/study/2196747/amazon-passes-google-as-top-destination-for-shopping-research-report. Accessed: 2016-12-23.

[9] Angela Dean, Daniel Voss, and Danel Draguljić. "Polynomial Regression". In: *Design and Analysis of Experiments*. Springer, 2017, pp. 249–284.

[10] Wei Di et al. "Is a Picture Really Worth a Thousand Words?: - on the Role of Images in e-Commerce". In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: ACM, 2014, pp. 633–642.

[11] Huizhong Duan et al. "A probabilistic mixture model for mining and analyzing product search log". In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. CIKM '13. San Francisco, California, USA: ACM, 2013, pp. 2179–2188.

[12] Kevin Duh and Katrin Kirchhoff. "Learning to rank with partially-labeled data". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '08. Singapore, Singapore: ACM, 2008, pp. 251–258.

[13] Amir-massoud Farahmand and Csaba Szepesvári. "Model selection in reinforcement learning". In: *Machine Learning* 85.3 (2011), pp. 299–332.

[14] Aman Berhane Ghirmatsion and Krisztian Balog. "Probabilistic Field Mapping for Product Search". In: *CLEF*. 2015.

[15] Manish Gupta, Michael Bendersky, et al. "Information retrieval with verbose queries". In: *Foundations and Trends in Information Retrieval* 9.3-4 (2015), pp. 209–354.

[16] Nadjla Hariri, Zahra Emami, and Mojtaba Malek. "The Precision and Recall of General Search Engines in Retreival of Images Related to Endocrine Diseases". In: *Iranian Journal of Endocrinology and Metabolism* 17.2 (2015), pp. 97–104.

[17] Eric Harth and Philippe Dugerdil. "Document Retrieval Metrics for Program Understanding". In: *Proceedings of the 7th Forum for Information Retrieval Evaluation*. FIRE '15. Gandhinagar, India: ACM, 2015, pp. 8–15.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Overview of Supervised Learning". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, pp. 9–41.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Unsupervised Learning". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, pp. 485–585.

[20] Kjell Arne Hellum. *Information Retrieval using applied Supervised Learning for Personalized E-Commerce Search at CIKM Cup 2016*. Tech. rep. 2016.

[21] Sujuan Hou et al. "Multi-label learning with label relevance in advertising video". In: *Neurocomputing* 171 (2016), pp. 932–948.

[22] Laurent Hyafil and Ronald L. Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1 (1976), pp. 15–17.

[23] Gareth James et al. "Linear Regression". In: *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer New York, 2013, pp. 59–126.

[24] Gareth James et al. "Tree-Based Methods". In: *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer New York, 2013, pp. 303–335.

[25] Thorsten Joachims. "Support Vector Machines". In: *Learning to Classify Text Using Support Vector Machines*. Boston, MA: Springer US, 2002, pp. 35–44.

[26] Dan J. Kim, Donald L. Ferrin, and H. Raghav Rao. "A Trust-based Consumer Decision-making Model in Electronic Commerce: The Role of Trust, Perceived Risk, and Their Antecedents". In: *Decis. Support Syst.* 44.2 (Jan. 2008), pp. 544–564.

[27] Khamsum Kinley et al. "Modeling users' web search behavior and their cognitive styles". In: *Journal of the Association for Information Science and Technology* 65.6 (2014), pp. 1107–1123.

[28] David G. Kleinbaum and Mitchel Klein. "Introduction to Logistic Regression". In: *Logistic Regression: A Self-Learning Text*. New York, NY: Springer New York, 2010, pp. 1–39.

[29] Deqjang Kong et al. *Personalized Feature based Re-Ranking Method for E-commerce Search at CIKM Cup 2016*. Tech. rep. 2016.

[30] Santosh Kumar et al. "A Machine Learning Based Web Spam Filtering Approach". In: *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*. IEEE. 2016, pp. 973–980.

[31] Amy N Langville and Carl D Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011, pp. 25–30.

[32] Adam Lella. *comScore Releases September 2014 U.S. Search Engine Rankings*. http://www.comscore.com/Insights/Rankings/comScore-Releases-September-2014-US-Search-Engine-Rankings. Accessed: 2016-12-23.

[33] "Normalized Discounted Cumulated Gain (nDCG)". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Boston, MA: Springer US, 2009, pp. 1920–1920.

[34] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '10. Geneva, Switzerland: ACM, 2010, 225–331.

[35] Cuicui Luo, Desheng Wu, and Dexiang Wu. "A deep learning approach for credit scoring using credit default swaps". In: *Engineering Applications of Artificial Intelligence* (2016).

[36] Yuanhua Lv et al. "Learning to Model Relatedness for News Recommendation". In: *Proceedings of the 20th International Conference on World Wide Web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 57–66.

[37] Hanspeter A. Mallot. "Artificial Neural Networks". In: *Computational Neuroscience: A First Course*. Heidelberg: Springer International Publishing, 2013, pp. 83–112.

[38] I Dan Melamed, Ryan Green, and Joseph P Turian. "Precision and recall of machine translation". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers-Volume 2*. Association for Computational Linguistics. 2003, pp. 61–63.

[39] M. Narasimha Murty and V. Susheela Devi. "Decision Trees". In: *Pattern Recognition: An Algorithmic Approach*. London: Springer London, 2011, pp. 123–146.

[40] Joao Palotti. *Learning to Rank for Personalized E-Commerce Search at CIKM Cup 2016*. Tech. rep. 2016.

[41] Generalized Transition-based Dependency Parsing. "Natural language processing". In: *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata (StatFSM)*. 2016, pp. 32–41.

[42] Michael Patrick Allen. "Assumptions of ordinary least-squares estimation". In: *Understanding Regression Analysis*. Boston, MA: Springer US, 1997, pp. 181–185.

[43] Robi Polikar. "Ensemble Learning". In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 1–34.

[44] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011, pp. 1–17.

[45] Rajnish M Rakholia and Jatinderkumar R Saini. "Information Retrieval for Gujarati Language Using Cosine Similarity Based Vector Space Model". In: *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*. Springer. 2017, pp. 1–9.

[46] G. Salton, A. Wong, and C. S. Yang. "A Vector Space Model for Automatic Indexing". In: *Commun. ACM* 18.11 (Nov. 1975), pp. 613–620. ISSN: 0001-0782.

[47] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5 (1988), pp. 513–523.

[48] Zuhaib Ahmed Shaikh et al. "Machine Learning based Number Plate Detection and Recognition". In: *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods*. SCITEPRESS-Science and Technology Publications, Lda. 2016, pp. 327–333.

[49] Zuhaib Ahmed Shaikh et al. "Machine Learning based Number Plate Detection and Recognition". In: *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods*. SCITEPRESS-Science and Technology Publications, Lda. 2016, pp. 327–333.

[50] Lidan Shou et al. "Supporting privacy protection in personalized web search". In: *IEEE transactions on knowledge and data engineering* 26.2 (2014), pp. 453–467.

[51] Phil Simon. "The Elements of Persuasion: Big Data Techniques". In: *Too Big to Ignore*. John Wiley Sons, Inc., 2012, pp. 77–109.

[52] Li Sujian. "Research of Relevancy between Sentences Based on Semantic Computa tion [J]". In: *Computer Engineering and Applications* 7 (2002), pp. 75–76.

[53] Martin Wistuba and Lars Schmidt-Thieme. *Gradient Boosted Decision Trees for Personalized E-Commerce Search at CIKM Cup 2016*. Tech. rep. 2016.

[54] Sorrachai Yingchareonthawornchai et al. "Precision, recall, and sensitivity of monitoring partially synchronous distributed systems". In: *International Conference on Runtime Verification*. Springer. 2016, pp. 420–435.

[55] Jun Yu et al. "Latent Dirichlet Allocation Based Diversified Retrieval for e-Commerce Search". In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: ACM, 2014, pp. 463–472.

[56] ChengXiang Zhai and Sean Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2016, 172–172.

[57] Ethan Zhang and Yi Zhang. "Average Precision". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Boston, MA: Springer US, 2009, pp. 192–193.

Appendices

Appendix A

Attachments

Attached in this document is all relevant source code in a zipped (.zip format) file, and a readme.txt describing details on how to run the code.