

An Experiment System for Text Classification

Mathias Niepert

B652: Computer Models of Symbolic Learning
Final Write-Up, Spring 2005

Abstract

There are many different machine learning approaches and empirical experiments addressing the problem of text classification. The goal of this project is the implementation of all the steps of a text categorization system in order to simplify experiments and further investigations in the area. The main purpose of the program is to make it easy to add new modules (e.g. neural network classification, other statistical methods, case- and rule-based systems) and give opportunities to compare and evaluate different approaches for different steps with each other. In research areas where most of the evaluation is based on statistical methods and many different experiment setups, it seems to be important to standardize as many steps as possible. The final program will also be useful for educational purposes.

Introduction

This paper is roughly divided into two main parts. The first part deals with the actual experiment system (TCEJ, Text Categorization Environment implemented in Java) in a tutorial style, mixed with a short overview of the theoretical background. In this first section I will examine the capabilities and advantages of the system and explain how it should be used. The second part of the paper contains an example and intends to show what the system can do when it is applied to a concrete problem. Additionally, I will explain some interesting findings I made during the testing of the system. Most of my evaluations and examples will cover the Reuters 21578 corpus, which is the most used corpus in text classification literature. Even though it is very likely that it will be replaced by the new Reuters Corpus RCV1, it is still important to compare the outcomes to already existing and approved results.

Text Classification

Text Classification or Categorization, the problem of automatically assigning semantic categories to natural language text, has become one of the most important methods for organizing textual information. Since the classification by hand is costly and in most cases highly unpractical due to the increasing number of documents and categories in many corpora, most state of the art approaches employ machine learning techniques to automatically learn text classifiers

from training examples. Unlike many other classification tasks, text classification involves also preprocessing steps, e.g., stemming and dimensionality reduction, which have an important influence on the effectiveness of the actual classification outcome. For an excellent introduction to text categorization see (Sebastiani 2002).

The System

The Main Interface

The main interface is the heart of the program. It allows the user to change the setting for the four main steps in Text Categorization: Loading, Indexing, Classification, and Evaluation. Besides the main interface, the menu offers many functions which will be introduced later in this paper.

Loading the Corpus

The first step towards the final classification and evaluation results, is to load the training corpus, that is, to read all the corpus documents, and count up the term and document frequencies for every term. The term and document frequencies are saved for every document as well as for every category and the entire corpus. This is necessary since these values will be of need in later calculations. The term frequency of a term in a document/category/corpus is the number of times the term occurs in the document/category/corpus. The document frequency of a term in a category/corpus is the number of documents which both, contain the term at least once and belong to the category/corpus, whereas the document frequency of a term in a document is a binary value which indicates if the term occurs in the document or not. For the actual implementation in Java, I used one class for terms, documents, categories, and corpora, respectively.

The System is already able to load two corpora: The "self-made" corpus and the Reuters 21578 corpus. In order to chose one of the already implemented corpora for an experiment, the user has to set the drop down box "Select Corpus" to the desired value. For more information about these freely available corpora, see Appendix A "Corpora". To make the system capable of loading other corpora, see Appendix B "Loading a new corpus".

Removal of stop words

In most of the applications, it is practical to remove words which appear too often (in every or almost every document) and thus support no information for the task. Good examples for this kind of words are prepositions, articles and verbs like "be" and "go". If the box "Apply stop word removal" is checked, all the words in the file "swl.txt" are considered as stop words and will not be loaded. This file contains currently the 100 most used words in the English language [National Literacy Trust, 2005] which on average account for a half of all reading in English. If the box "Apply stop word removal" is unchecked, the stop word removal algorithm will be disabled when the corpus is loaded.

The system only considers words which have length greater than one as valid tokens. This is due to the fact that all words with length one seem to be unimportant and would have been sorted out by the later dimensionality reduction step anyway. However, it is possible to change that easily in the code.

Stemming

Stemming is the process of grouping words that share the same morphological root. E.g. "game" and "games" are stemmed to the root "game". The suitability of stemming to Text Classification is controversial. In some examinations, Stemming has been reported to hurt accuracy. However, the recent tendency is to apply it, since it reduces both the dimensionality of the term space and the stochastic dependence between terms. The user has the possibility to chose either "No Stemming" or one of the well known Stemmers from Porter (Porter 1980) and Paice (the Lancaster Stemmer) (O'Neill & Paice 2001). A comparison of these stemmers can be found at <http://chrisonellfyp.50megs.com/>.

After the corpus has been loaded with the given settings, the distribution of term and document frequencies can be plotted (Menu: Corpus → Distribution Graphs → Document Frequency/Term Frequency") and the category properties can be examined in a table (Menu: Corpus → Category Table). The Distribution Graph is a plot of the gaussian distribution of the document or term frequency in the corpus. The table containing the category properties shows all the labels of the different categories, how many documents belong to these categories, the average amount of words per document, and the number of processed words, that is, the cumulative sum of words in every category, which were considered as valid tokens.

Term Weighting & Dimensionality Reduction

For many machine learning algorithms it is necessary to reduce the dimensionality of the feature space, if the original dimensionality of the space is very high. In most of the cases this improves not only the performance but also the accuracy of the classification itself. Term Weighting is the process of assigning values to all the terms in the corpus

according to their importance for the actual classification part. Here, importance is defined as the ability of the term to distinguish between different categories in the corpus. Usually, the more important a term is the higher is the assigned weight value. There are already nine different weighting functions implemented in TCEJ:

chi square

$$w(t_k, C_i) = \frac{|T| [P(t_k, C_i)P(\bar{t}_k, \bar{C}_i) - P(t_k, \bar{C}_i)P(\bar{t}_k, C_i)]^2}{P(t_k)P(\bar{t}_k)P(C_i)P(\bar{C}_i)}$$

mutual information

$$w(t_k, C_i) = \log \frac{P(t_k, C_i)}{P(t_k)P(C_i)}$$

odds ratio

$$w(t_k, C_i) = \frac{P(t_k|C_i)(1 - P(t_k|\bar{C}_i))}{(1 - P(t_k|C_i))P(t_k|\bar{C}_i)}$$

NGL

$$w(t_k, C_i) = \frac{\sqrt{|T|} [P(t_k, C_i)P(\bar{t}_k, \bar{C}_i) - P(t_k, \bar{C}_i)P(\bar{t}_k, C_i)]}{\sqrt{P(t_k)P(\bar{t}_k)P(C_i)P(\bar{C}_i)}}$$

GSS

$$w(t_k, C_i) = P(t_k, C_i)P(\bar{t}_k, \bar{C}_i) - P(t_k, \bar{C}_i)P(\bar{t}_k, C_i)$$

relevancy score

$$w(t_k, C_i) = \log \frac{P(t_k|C_i) + d}{P(\bar{t}_k|\bar{C}_i) + d}$$

information gain

$$w(t_k) = - \sum_C P(C) \log P(C) \\ + P(t_k) \sum_C P(C|t_k) \log P(C|t_k) \\ + P(\bar{t}_k) \sum_C P(C|\bar{t}_k) \log P(C|\bar{t}_k)$$

document frequency

$w(t_k)$ = the document frequency of the term in the corpus

"MSF"

$$m_{tk} = \frac{1}{|C|} \sum_C P(t_k|C) \\ w(t_k) = \sqrt{\frac{1}{(|C| - 1)m_{tk}} \sum_C (P(t_k|C) - m_{tk})^2}$$

Where $|\mathcal{T}|$ is the number of unique terms in the corpus and the probabilities are interpreted on an event space of documents (e.g., $P(t_k, C_i)$ is the probability that, for a random document d , term t_k occurs in d and d belongs to category C_i), and are estimated by counting occurrences in the training set (i.e., by using the document and term frequencies). All functions (here, "document frequency" and "MSF" are exceptions), are computed locally to a specific category C_i . In order to calculate the weight value of a term t_k in a global, category independent sense, either the sum, the weighted sum, or the maximum of their category-specific values are calculated. The user can choose between all these options in the "Indexing" window of the program. Additionally, it is possible to ignore – to assign a zero weight to – terms which have both their document frequency and term frequency smaller than a given integer threshold. I found that this improves the performance of some classifiers as it is a suitable method to reduce noise in the corpus. For more details and a comparison of the weighting functions "chi square", "information gain", and "MSF", see chapter "A comparison of three weighting functions" later in this document.

The next step after the term weighting is the actual reduction of the feature space dimensionality. To achieve that, the user has to fill in the desired dimensionality $|\mathcal{T}_r| < |\mathcal{T}|$ in the textfield "Number of Features" and hit the "Reduce Dimensionality" button. This can be done at any time, even if the terms have not been weighted before. In this case, the first $|\mathcal{T}_r|$ terms are chosen to be the features, independently of their weight values. However, if a weighting function has been applied, the terms are always chosen according to their weight values.

After the dimensionality has been reduced to a certain size $|\mathcal{T}_r|$, all the TF-IDF values and the mean and standard deviation of the terms in the feature space are calculated. This makes new graphical gimmicks available:

- The Distribution Graphs (Menu: Corpus → Distribution Graphs → Document Frequency/Term Frequency) also plot the gaussian distributions of the term and document frequency of the terms in the new feature space.
- The Document Table (Menu: Corpus → Document Table → Document Frequency/Term Frequency/TFIDF) gives all the document vectors filled with either the document frequency, the term frequency, or the TF-IDF values.
- The Term Table (Menu: Corpus → Term Table) lists all the terms (which were chosen as dimensions in the feature space) with the following attributes for every term: The rank, the actual token, the document frequency, the inverse document frequency, the term frequency, the weight value (as set by the weighting function), and the probability $P(t_k)$.

Classification

Classification is the process which assigns one or more labels – or no label at all – to a new (unseen) document. There are many machine learning algorithms which have been applied to the problem of text categorization, ranging from statistical methods (e.g. Naïve Bayes) to black-box algorithms (e.g. Neural Networks). There are already two classifiers implemented in TCEJ. The k-nearest neighbor and the Naïve Bayes (Bernoulli) classifier.

k-nearest neighbor The k-NN algorithm is a so called non-linear, lazy learner, and belongs to the class of example (instance) based classifiers. For more details and the theoretical background, see (Mitchell 1997). Here, the k-NN algorithm uses the cosine similarity as similarity function:

$$sim(d_i, d_j) = \frac{\vec{d}_i \cdot \vec{d}_j}{|\vec{d}_i| |\vec{d}_j|} = \frac{\sum_{k=1}^n w_{k,i} w_{k,j}}{\sum_{k=1}^n w_{k,i} \sum_{k=1}^n w_{k,j}}$$

where $w_{x,y}$ is the value (e.g., the TF-IDF value) of term x in document y .

When the document vectors are normalized, only the vector product has to be calculated, which can be done faster than the computation of the euclidian distance, especially for sparse vectors. The cosine similarity measures the cosine of the angle between two vectors. The bigger the value, the smaller is the actual angle and the more similar are the two vectors. This makes the distance metric also independent of the length of the documents, as document vectors of different length, but with the same angle to each other, will have zero distance. The k-NN algorithm is especially suitable to compare the accuracy of the weighting functions in the dimensionality reduction step. (Yang & Pedersen 1997)

Since the test documents are loaded together with the training documents, the test document can now be "scored" by clicking on the "Score test documents" button in the "Classifiers" interface. For every test document, the 50 most similar training documents are then saved. This makes it possible to test different values for k (up to 50) very fast in the actual evaluation step. Additionally, it is possible to save this scoring to an external file. (Menu: Classifiers → k nearest neighbors → Save Scoring). Since the scoring of the test documents can take a very long time (e.g., for the Reuters 21578 ModApté split the scoring takes ~ 20 hours on a Pentium M 1.5 GHz processor) the scoring is "valuable" information and should be save and loadable. To load a scoring from an external file (Menu: Classifiers → k nearest neighbor → Load Scoring), the corpus belonging to the scoring has to be loaded before.

Independently from the scoring of the test documents, it is also possible to classify documents with the button "Classify file". The user can then chose an external file, containing an arbitrary document, which is immediately classified by the k-nearest neighbor algorithm, giving a report of the results in the console window. However, the

classification of an external file does only work, if the feature space has been reduced before (that is, the training documents are holding the normalized TF-IDF values, according to the current feature space).

Naïve Bayes Bernoulli The Naïve Bayes Bernoulli classifier calculates for every category C and every test document d_t the normalized likelihood, that the document belongs to the category, given the conditional class probabilities $P(t_k|C)$, the prior class probabilities $P(C)$, and the binary values indicating if a feature term t_k occurs in the document d_t or not. For a discussion of this classifier and a comparison with the slightly advanced multinomial Naïve Bayes classifier, see (McCallum & Nigamy 1998).

The scoring of the test documents works analogous to the scoring for the case of the k-nearest neighbor classifier, with the exception, that the scoring is a ranking of category probabilities for every test document rather than a training document ranking for every test document. Scores can be saved and loaded over the menu (Classifier → Naïve Bayes Bernoulli → Load/Save Scoring) and a document in an external file can be classified in the same way as for the k-nearest neighbor classifier.

Evaluation

As already mentioned earlier, the evaluation of document classifiers is mostly conducted experimentally rather than analytically. The reason is that, in order to evaluate a system analytically (e.g., proving that the system is correct and complete), we would need a formal specification of the problem that the system is trying to solve (e.g., with respect to what correctness and completeness are defined), and the central notion of Text Categorization is, due to its subjective character, inherently nonformalizable. The experimental evaluation of a classifier usually measures its effectiveness (rather than its efficiency), that is, its ability to take the right classification decisions.

Since every classifier in TCEJ provides a category ranking for every test document (the k-nearest neighbor classifier after specifying a particular value for k), a threshold t has to be provided by the user. If, for a test document, the score for a certain category is greater than the threshold, the document is classified into the category. Thresholding is indispensable for multi-label categorization, where one test document can belong to more than one category or no category at all. A threshold can be derived either globally or locally. Global thresholding provides only one threshold which is applied to every category score, whereas local thresholding derives a threshold for every category. It seems to be clear, that local thresholding is the better choice if the effectiveness of the classifier for new, unseen documents is supposed to be as high as possible. However, global thresholding is sufficient to compare different weighting functions and dimensionality reductions, and thus, I have only implemented global thresholding so far. Local thresholding will be implemented in future versions of TCEJ.

When a particular threshold (and a particular value for k for the case of the k-nearest neighbor classifier) is set by the user, the test documents of the loaded corpus can be classified given the scoring (which has to be either computed or loaded before) and the classification is evaluated. This is done by clicking on the button "Evaluation for Threshold". The global results (macro precision, macro recall, micro precision, and micro recall) are written to the console window. The local evaluation values for every category are already available "in the system" but are not displayed by the GUI yet.

Precision and recall alone do not say much about the effectiveness of the classifier. Hence, it is necessary to compute different standard values which combine precision and recall, to derive a robust measure of the effectiveness of the classifier. TCEJ is able to calculate the break even point, the 11-point precision and "average precision" a value which is easy to compute "on the fly", when the first two values are calculated. What the program internally does, is to evaluate the classification for a threshold ranging from 0 (recall = 1) up to a value where the precision value equals 1 and the recall value equals 0, incrementing the threshold with a given threshold step size. The break even point is the point where recall meets precision and the eleven point precision is the averaged value for the precision at the points where recall equals the eleven values 0.0, 0.1, 0.2, ..., 0.9, 1.0. "Average precision" refines the eleven point precision, as it approximates the area "below" the precision/recall curve:

average precision =

$$\sum_{i=1}^m \frac{pr(t_i) + pr(t_{i-1})}{2} |re(t_i) - re(t_{i-1})|$$

with m is the first value n for which $precision(n) = 1$ and $recall(n) = 0$, $t_{i+1} = t_i + step_size$, $t_0 = 0$, $re(t_0) = 1$, and $pr(t_i)$ and $re(t_i)$ are the precision and recall values for threshold t_i .

The following integral is the size of the area under the precision/recall curve, which is approximated by the "average precision" sum:

$$\int_{recall=0}^1 pr(recall) d recall$$

where $pr(recall)$ is the precision value corresponding to the recall value $recall$.

But this is just another measure I thought could be interesting to have. In the program, the user can start the computation of these values by giving the threshold step size (the smaller the more accurate are the results, but the longer takes the calculation) and by clicking on the "Precision/Recall" button. When the computation is finished, the results are written to the console. Additionally, it is possible to open the precision-recall graph (see Figure 1), commonly used in

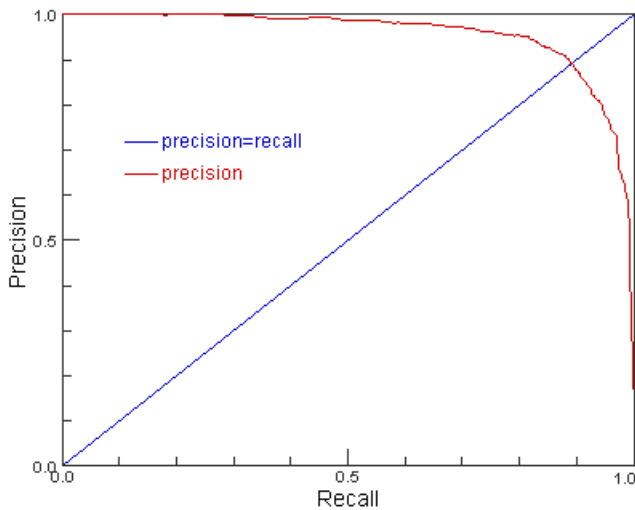


Figure 1: TCEJ, precision/recall plot for the ModApté [10] split, dimensionality 200, weighting function MSF, 30-nearest neighbor classifier.

text classification literature, in a new window (Menu: Classifiers → k-nearest neighbor or Naïve Bayes (Bernoulli) → Precision/Recall Graph).

Batch Evaluation

Sometimes it is interesting to compare different settings of the different parts of the classification with each other. Yang (Yang & Pedersen 1997), for example, compared the 11 point precision of several weighting functions (chi square, information gain, mutual information, term strength, and document frequency) for the Reuters 22173 corpus (the predecessor of the Reuters 21578 corpus) for different sizes of the feature space and, in another article (Yang 2001), several thresholding methods (Rcut, Pcut, etc.) with each other.

TCEJ can run evaluations for any desired settings (e.g., dimensionality, k for the k-nearest neighbor classifier, different weighting functions), plot the results (multiple curves in one graph) and save the results in an external file. However, the actual changes on the settings ("what" should be tested) can only be made in the code so far. To see an example of a batch evaluation, open a file (Menu: Batch Evaluation → Load Evaluation) out of the folder "evaluations", plot the results (Menu: Batch Evaluation → Evaluation Graph) and show them in a table (Menu: Batch Evaluation → Evaluation Table).

A comparison of three weighting functions

For the evaluation of the three different weighting functions, I used a subset of the Reuters 21578 ModApté split. (All ModApté training documents out of the files reut2-000.xml, reut2-001.xml, reut2-002.xml, and reut2-003.xml, and all ModApté test documents out of the file reut2-018.xml). This resulted in 2561 training documents. 2008 of them were

labeled with at least one category. However, I used all of the training documents! After stop word removal and stemming the number of unique terms was 23706. The 30 nearest neighbors classifier was used to classify the 338 test documents. Break even points, eleven point precisions and average precisions for the reduced feature space dimensionalities 500 (aggressiveness: 47.4), 1000 (23.7), and 2000 (11.8) were computed.

Results

dimensionality of feature space = 500

	chi square	information gain	MSF
break even	0.511	0.686	0.519
11 point	0.561	0.716	0.548
avg precision	0.563	0.741	0.550

dimensionality of feature space = 1000

	chi square	information gain	MSF
break even	0.664	0.696	0.678
11 point	0.690	0.731	0.703
avg precision	0.708	0.754	0.729

dimensionality of feature space = 2000

	chi square	information gain	MSF
break even	0.674	0.683	0.679
11 point	0.712	0.721	0.712
avg precision	0.740	0.748	0.735

Even though these numbers are not comparable to other results since a subset and not the complete Reuters 21578 ModApté split was used, they provide still interesting insights. Especially the fact, that for the same weighting function and the same dimensionality, it happens that, e.g., the break even value is higher compared to another function but the eleven point precision is lower, compared to the same function. It also shows that "MSF" could be an interesting alternative to chi-square and information gain, not only for feature selection in text classification, but also to weight the importance of features in other classification tasks.

More results and images of several recall-precision plots can be found at the project homepage. The scorings (see chapter Classification in this document) used for this evaluation can be found in the folder "scores", and can be loaded into the system.

Source Code & Documentation

The Java Documentation and the source code can be found at the project homepage <http://tc.matlog.net>. Additionally, I put some typical plots and tables on the homepage.

Future Work

Fortunately, there is still a lot of work to do. All of the involved parts of the program can be improved and extended. I put emphasis on making the program as scalable

as possible. The actual text classification package is not entangled with the graphical user interface. New corpora and weighting functions can be implemented. It is also possible to export the document vectors after the dimensionality reduction to other data formats, which can then be applied to several other classifiers, provided by Toolkits like Matlab or SVM^{light}. Other thresholding methods (e.g., PCut, that is, a document ranking for every category, or individual thresholds for every category) can easily be implemented in the future. Many interesting things are already "coded" but have not been made accessible yet through the graphical user interface due to the obvious time constraints. For example, the precision and recall results for single categories, or the distribution of the number of documents over the categories. I am also planning on implementing several new classifiers. Especially boosting, clustering and regression methods seem to provide interesting results and insights.

Obviously, developing this program let me understand the task of text classification very deeply. It is a great basis for further work in other projects like information extraction.

Appendix A: Implemented Corpora

The self-made corpus

For the development phase I used a small self-made corpus since the running time needed to be as short as possible. I collected articles online from the *New York Times*, *Washington Post* and *CNN.com* out of the standard categories "Science", "Business", "Sports", "Health", "Education", "Travel", and "Movies". This includes easy (e.g. Sports ↔ Business) and more difficult (Education ↔ Science ↔ Health) classification tasks. I collected 150 documents with the following categories: Sports {30 Training Documents}, Health {30}, Science {27}, Business {23}, Education {24}, Travel {6}, Movies {10}, with in average 702 words per document.

The corpus is sufficient for exploring, testing and demonstrating the program, since all the algorithms, especially the final classification of the test documents, can be computed in a small amount of time. However, the corpus is much too small to reliably evaluate the effectiveness of different parts of the classification process. The framework of the program allows to scale up to arbitrary large corpora as long as the machine on which the program is running can handle the amount of data. The self-made corpus is also downloadable from the project homepage <http://tc.matlog.net>.

The Reuters 21578 corpus

The second corpus already included in the system is the frequently used Reuters 21578 corpus. The corpus is freely available on the internet (Lewis 1997). Since TCEJ uses an XML parser, it was necessary to convert the 22 SGML documents to XML, using the freely available tool SX (Clark 2001). After the conversion I deleted some single characters which were rejected by the validating XML parser as they had decimal values below 30. This does

not effect the results since the characters would have been considered as whitespaces anyway.

Appendix B: Loading a new corpus

It is not very difficult to add a new corpus to the system. The abstract class TCCorpusLoader with its two main methods "loadCorpus" and "getTestSet" has to be extended. The two classes TCCorpusLoaderReuters and TCCorpusLoaderSelfmade contain more information regarding this task. TCCorpusLoaderReuters loads a corpus using an XML parser (Xerces) and TCCorpusLoaderSelfmade loads a corpus using files (one document per file) in a folder. When the class, which reads the training and test documents, has been implemented, both, the Array "corpusNameStrings" (which contains all the names of the corpora) in the class TCEJ, and the method "loadCorpus" in the class TCEExperiment have to be updated accordingly.

References

- Clark, J. 2001. Sx, sgml to xml converter. <http://xml.coverpages.org/sx-doc.html>.
- Eui-Hong (Sam) Han, G. K., and Kumar, V. 1999. Text categorization using weight adjusted k-nearest neighbor classification. *Lecture Notes in Computer Science*.
- Lewis, D. D. 1997. Reuters-21578 text categorization test collection, distribution 1.0. davidlewis.com/resources/testcollections/reuters21578.
- McCallum, A., and Nigamy, K. 1998. A comparison of event models for naive bayes text classification. *AAAI-98 Workshop on Learning for Text Categorization*.
- Mitchell, T. 1997. *Machine Learning, McGraw Hill*.
- O'Neill, C., and Paice, C. D. 2001. The lancaster stemming algorithm.
- Porter. 1980. An algorithm for suffix stripping. *Program, Vol. 14, no. 3, pp 130-137*.
- Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM Computing Surveys Volume 34, Issue 1*.
- Yang, Y., and Pedersen, J. O. 1997. A comparative study on feature selection in text categorization. *Proceedings of ICML-97, 14th International Conference on Machine Learning*.
- Yang, Y. 2001. A study on thresholding strategies for text categorization. *Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval*.