Int. J. Advance Soft Compu. Appl, Vol. 7, No. 3, November 2015
ISSN 2074-8523

Predicting the Relevance of Search Results for E-Commerce Systems

Mohammed Zuhair Al-Taie[1], Siti Mariyam Shamsuddin[1], and Joel Pinho Lucas[2]

[1]UTM Big Data Centre
Universiti Teknologi Malaysia (UTM),
Skudai, 81310 Johor, Malaysia
e-mail: mza004@live.aul.edu.lb, mariyam@utm.my
[2]Tail Target
Av. Pedroso de Morais, 1619 - Pinheiros,
São Paulo - SP, 05419-001, Brazil
e-mail: joelpl@gmail.com

Abstract

Search engines (e.g. Google.com, Yahoo.com, and Bing.com) have become the dominant model of online search. Large and small e-commerce provide built-in search capability to their visitors to examine the products they have. While most large business are able to hire the necessary skills to build advanced search engines, small online business still lack the ability to evaluate the results of their search engines, which means losing the opportunity to compete with larger business. The purpose of this paper is to build an open-source model that can measure the relevance of search results for online businesses as well as the accuracy of their underlined algorithms. We used data from a Kaggle.com competition in order to show our model running on real data.

Keywords: *CrowdFlower, E-Commerce, Kaggle Competition, Random Forest, Relevance Prediction, Scikit-learn, Support Vector Machines.*

1 Introduction

In this work, we show the use of machine-learning algorithms, as well as the use and implementation of preprocessing methods, applied to the prediction of text search relevance. For this purpose, the paper describes first some basic concepts about data preprocessing and text retrieval. In this context, some properties of *tf-idf* are described next. Then, we describe the dataset used for this study (from

CrowdFlower) as well as some proposal and hypothesis for data preprocessing towards feature extraction. Afterwards, we describe the application of two machine-learning algorithms on the processed data. Finally, on the last section, we show benchmark results we obtained on running four combinations of feature extraction methods and machine-learning algorithms.

2 Data Preprocessing

Data preprocessing usually consists of four steps: data cleaning (a.k.a. data cleansing or scrubbing) which aims at removing noise, filling missing values and correcting inconsistencies in data. This step also involves the identification and removal of outliers [1]. Data integration seeks to combine data from varied and different sources into coherent data storage. This task is not simple and requires matching the schema from different sources. However, data from CrowdFlower is already integrated in a single dataset. Thus, in this study we will not perform data integration. Data transformation aims at converting data to a more appropriate form. The goal is to have more efficient data mining operations by making the data more understandable [1]. Finally, data reduction aims at reducing the size of data while minimizing any possible loss of information.

3 Text Retrieval Systems

Text retrieval involves retrieving the documents that contain particular keywords for a given query [3]. Strings are traditionally known as consisting of a series of characters. Given a string $T=t_1\ t_2 \ldots t_n$, and a particular keyword pattern $P=p_1\ p_2 \ldots p_m$, the goal is to verify whether P is a substring of T.

Pattern matching can be applied in two ways: forward pattern matching, where the text and pattern are matched in the forward direction; and backward pattern matching, where the matching process is done from right to left.

3.1 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) [4] is basically a combination of two earlier methods: Term Frequency (TF) and Inverse Document Frequency (IDF). $tf - idf$ produces a weight for each term f_i in each document d_j, as defined in Eq. 1:

$$tf - idf(f_i, d_j) = tf_{ij} * idf\ (f_i) \tag{1}$$

The tf measure is one of the earliest term selection methods. It assumes that documents using similar words should belong to the same topic. While it considers the very frequent terms that are normally distributed across different topics, it looks at the very rare terms, stop words, and uninformative words as should be ignored. tf is defined as in Eq. 2:

$$tf(f_i) = \sum_{j \in D_{f_i}} tf_{ij} \quad (2)$$

where D_{f_i} is the documents that contain f_i, and tf_{ij} is the frequency of f_i in document j.

On the other hand, idf measures the weight of terms by assigning high values to the rare terms and low values to the highly frequent terms across all documents. It was designed to deal with the shortcomings associated with the implementation of tf, which assigns the same weight to all selected terms. This means that distinguishing between frequent words appearing in a small set of documents is not possible, even if the terms have discriminative power. idf is defined as in Eq. 3:

$$idf(f_i) = log \frac{|D|}{|D_{f_i}|} \quad (3)$$

where $|\mathrm{D}|$ is the total number of documents and $|\mathrm{D_{f_i}}|$ is the number of documents that have the term f_i.

In this way, $tf - idf$ gives greater values to the terms that occur frequently only in a small set of documents, and gives lower values to the terms that occur in more documents. In this way, the higher the term frequency of a document is, the lower its $tf - idf$ is.

4 The CrowdFlower Dataset

Having the right data to apply analysis has been always an issue for researchers who want to test their algorithms.

Kaggle.com provides avenue for researchers and companies to present their data. Statisticians and data miners can experiment their approaches and compete against each other in order produce the best machine learning models. Submissions from participants are scored immediately for most competitions based on a hidden test dataset. Finally, the competition hosts give the winners compensation, which can be money, job vacancies or knowledge.

The dataset used in this study has been hosted by CrowdFlower, which is a data mining, data enriching and crowdsourcing company. CrowdFlower created its dataset with the help of its crowd who rated a list of products from a handle of ecommerce websites on a scale from 1 to 4, where 4 indicates the product completely satisfied the search query (it was the product the user was looking for) or 1 to indicate that the result did not match the query search (irrelevant content for the user). CrowdFlower generated 261 search terms for this goal.

The dataset originally encompassed six attributes: id, query text, product title, product description, median relevance and relevance variance of the query. The train file that was made available by CrowdFlower contains 10,158 rows in total.

For this challenge, CrowdFlower also provided a test file containing another portion of data with the same attributes, except the attributes related to the query relevance, which is the label attribute to be predicted.

Since the provided dataset is raw and has some noise, it was necessary to apply data preprocessing, which included removing undesired content to improve data quality and make it ready for mining and analysis. If any of the data preprocessing phases is not properly handled, machine-learning algorithms will result in inaccurate predictions or even will not run.

5 Data Cleaning and Transformation

At this point, the CrowdFlower dataset described in last section is just an amount of raw tuples related to text queries and product descriptions. In this way, we need to pre-process the data in order to extract some value and then transform it to provide an input for some machine learning algorithms.

To this end, the first task to be accomplished is feature extraction. We need to transform raw text search attributes in valuable features for running in machine learning algorithms.

Subsequently, we describe, with the use of two methods, how we extracted some features from the dataset.

5.1 Word Match Counting

As the dataset encompasses text attributes related to textual searches, the first and most straightforward method to extract feature is counting how many words, in each text search, match the product title and product description. In this way, we are able to acquire two numerical features for this study.

In order to implement word match counting, we took in advantage facilities and expressiveness of the Python language, along with some widely employed built-in packages for data manipulation, such as NumPy [7] and SciPy [8]. In this way, we firstly extracted word tokens from text (i.e.: splitting text by spaces). Secondly, we removed non-alphanumeric characters from text. Words with just one or two characters (mostly articles and prepositions) were also removed. Afterwards, we transformed each list of words (from search text attributes) in an array and, then, Numpy methods were applied for intersecting the search word vector with product title and product description vectors, respectively. At this point we simply counted how many words remained in each intersection. As described below, $S=\{t_1,t_2,\ldots,t_n\}$ is the search words (terms) vector, $D=\{t_1,t_2,\ldots,t_n\}$ the product description words vector and $T=\{t_1,t_2,\ldots,t_n\}$ the product title words vector, respectively:

$$\begin{aligned} Feature_1 &= \text{count}\ (T \subset S) \\ Feature_2 &= \text{count}\ (D \subset S) \end{aligned} \tag{4}$$

At this point, we have got $Feature_1$ and $Feature_2$, which will be the features we are going to use as input for machine learning algorithms.

5.2 Improving Word Match counting with tf-idf

Since $tf - idf$ weight-based algorithms are broadly employed by e-commerce and publishing companies (such as Google and Yahoo) for text retrieval and searching, we decided to make use of it for enhancing the expressiveness of two features we acquired so far. In this way, instead of simply expressing numeric values related to the number of word intersections, now we can express such intersections taking into account word weights.

In order to acquire word weights we used the TfidfVectorizer class available in scikit-learn Python package. scikit-learn [5] is a machine learning library for Python to serve different purposes such as classification, regression and clustering. Examples of incorporated algorithms include Naïve Bayes, Support Vector Machines, Logistic regression, *k*-means and DBSCAN. The project started in 2007 as a Google Summer of Code project, and was maintained later by a number of volunteers. It is an easy to use bundle of tools, and has a rich and well-organized documentation [6]. The library is largely written in Python and is designed particularly to work with NumPy and SciPy libraries.

Thus, we only need to provide the text as input and then we can make use of the attributes from this class, including *idf* weights. For calculating such weights from words, stop words are ignored, which means that irrelevant words for searching, such as articles, prepositions and pronouns, are ignored.

Now features we extracted are much more expressive, since they take into account weights of word terms, where higher weights are related to rare terms and lower weights are related to high frequent terms across all documents.

Feature values are now the sum of all intersection word weights. Instead of merely using a string to represent words in vectors, now we use Python dictionary to represent words along with their weights, where words are the keys and weights are the values in dictionaries. In this way, we have a search dictionary $S_{dict}=\{t_1 \rightarrow w_1,\ t_2 \rightarrow w_2,\ \ldots,\ t_n \rightarrow w_n\}$, a product description words dictionary $D_{dict}=\{t_1 \rightarrow w_1,\ t_2 \rightarrow w_2,\ \ldots,\ t_n \rightarrow w_n\}$, and a product title words dictionary $T_{dict}=\{t_1 \rightarrow w_1, t_2 \rightarrow w_2, \ldots, t_n \rightarrow w_n\}$. We may also represent, as vectors, the keys from dictonary: $S_{keys}=\{t_{k1}, t_{k2}, \ldots, t_{kn}\}$. Thus, features are calculated as follows:

$$\begin{aligned} Feature_1 &= \text{sum}(\text{values}(T_{keys} \subset S_{keys})) \\ Feature_2 &= \text{sum}(\text{values}(D_{keys} \subset S_{keys})) \end{aligned} \tag{5}$$

6 Data Analysis and Prediction using Machine Learning

After acquiring the two new features described previously, we are now able to apply machine-learning algorithms. The main goal with CrowdFlower data is to predict the relevance of search queries. In this way, the label attribute for this study will be the median relevance of the search.

In this context, we first tried to predict the values of median relevance from CrowdFlower test set rows employing the (SVM) Support Vector Machine [9] implementation from scikit-learn. SVMs are also called Support Vector Networks and consist of a supervised learning algorithm, which is based on the concept of decision hyper plane defining decision boundaries. Thus, SVMs take labeled training data and output an optimal hyper plane categorizing new (text) examples.

As SVM is a simple and straightforward algorithm, we also tried CrowdFlower data with a more sophisticated algorithm. In this way, we chose an ensemble learning method that is largely employed in machine learning and scikit-learn user communities: the Random Forest, which was developed by Breiman [10] and Cutler [11]. Random Forests combine bootstrap aggregating and random selection of features in order to build a set of decision trees with controlled variance. In other words, each decision tree is constructed by using a random subset of the training data. In this way, a random forest fits a set of decision tree classifiers on multiple sub-samples of the data and then uses averaging to control over-fitting and enhance accuracy.

Applying both SVM and Random Forests is a straightforward task in Python using scikit-learn as both algorithms share a common named functions for training. Thus, after acquiring the training data, and storing it properly in data frame, learning in scikit-learn should be taken in three simple steps: initializing the model, fitting it to the training data, and predicting new values. At this moment, we store the train set provided by CrowdFlower, but using our extracted features, in a data frame and provide it as input for both algorithms. After initializing and fitting train data, we can now predict the label attribute (search terms median relevance). It is important to highlight that both algorithms only work with numeric feature. Such scenarios satisfy completely the features we extracted in this study.

Both implementations (feature extraction and machine learning algorithms use in the scope of CrowdFlower data) are public and freely available in a Github repository[1].

6.1 Learning Models Benchmark

In this section we describe the resulted scores we obtained running the four combination methods we applied: SVM with word match counting based features, SVM with *tf-idf* based features, Random Forest with word match counting based

features and, Random Forest with *tf-idf* based features. Bellow, on table 1, we show the scores obtained when submitting, for the Kaggle platform, the CrowdFlower test set filled with their respective predicted median relevance values (label attribute).

Table 1: Achieved Scores

	Match Counting	tf-idf
SVM	0.51241	0.57654
Random Forest	0.53834	0.59211

As shown on table 1, the best score was obtained with Random Forest with *tf-idf* based features. We can also notice that Random Forest obtained better score than SVM and, conversely, that *tf-idf* obtained better results than word match counting based features. However, the impact of applying *tf-idf* on preprocessing was substantially higher than Random Forest over SVM.

7 Big Data Implementation Concerns

In spite of the CrowdFlower dataset being small, we could so far demonstrate and reproduce learning approaches that can be applied similarly in much bigger datasets. Moreover, numerous big data approaches are tested and demonstrated on small data before scale, since it is much cheaper, easy to retest and feasible to run algorithms several times. Nevertheless, some considerations need to be stated relatively to scalability.

Firstly, in the preprocessing point of view, we need to scale out (i.e.: adding more hardware as needed) the proposed methods. For merely counting word matches, the strategy is simple and straightforward, since we only need to equally distribute rows data to be processed across multiple machines. This positive scenario exists because word matches are identified and counted locally, separately in each row.

On the other hand, *tf-idf* cannot be run locally in each row, since it needs to calculate, for each word, the total frequency among all rows. In this context, we would probably need to develop a *tf-idf* implementation out of the scikit-learn package, but still making use of NumPy and SciPy packages facilities. Moreover, in order to scale out, counting word frequencies on the top of a map-reduce paradigm is straightforward. In this context, mappers emit, for each word, a pair of words and the number 1 (i.e.: <word,1>). Subsequently, reducers take as input mappers output with the same key (word) and sum up their values (in this case, number one) in order to output a pair <word, count> representing a word frequency pair among all rows.

From the point of view of the applied machine learning algorithms, Random Forests are more sensible for big data as two common problems are associated with decision tree implementation with large databases: first, the very large

number of candidate splitting conditions, and second the recursive nature of the algorithm [12]. However, for both SVM and Random Forests we are able to train an effective model using only a representative part of the dataset. In this way, the challenge here is to find a representative portion of data. It may be chosen randomly, or filtering rows according to given criteria over row attributes or even combining both strategies.

8 Conclusion

In this paper, we were able to show the whole cycle and steps for performing a data analysis on real world data, from data preprocessing until feature prediction. Moreover, as described on the last section, such approaches can be effectively applied on large datasets conversely.

Throughout this study, we could testify scikit-learn package, as well as other built-in Python packages, saves substantial development time. With our benchmark on CrowdFlower test set, we could attest that Random Forest is an efficient machine-learning algorithm. It shown better accuracy compared to a simple SVM implementation. This, in part, is due to the ensemble nature of Random Forest, which allows multiple learning algorithms to be run. Beside this fact, the nature of the dataset is also another reason for SVM disadvantage, since such algorithm is likely to provider poorer performances when the number of features is much greater than the number of samples. This was exactly the configuration of the CrowdFlower dataset.

Finally, we could also testify that the preprocessing step is crucial for the whole data analysis. It also consumes most of the time and implementation efforts needed on the whole analysis. Moreover, our study also shown that preprocessing may be more critical for precision than the machine-learning algorithm itself, since SVM combined with *tf-idf* have shown higher score than Radom Forest combined with word matches counting.

In this way, we can argue that employing *tf-idf* for preprocessing features and, subsequently, employing Random Forest is a powerful and effective approach for predicting, and measuring, the relevance of text search in e-commerce scenarios. Such approach suits entirely this kind of needs even for small e-commerce businesses emerged in big data necessities.

ACKNOWLEDGEMENTS
This work is supported by Universiti Teknologi Malaysia under the Flagship Project: *UTM E-Learning Big Data Analytics*. The authors would like to thanks Research Management Centre (RMC), Universiti Teknologi Malaysia (UTM) for the support in R & D, and *Soft Computing Research Group* (SCRG) for the inspiration in making the study a success. The authors would also like to thank the anonymous reviewers who have contributed enormously to this work.

References

[1] Han, J., et al. (2006). Data mining, southeast Asia edition: Concepts and techniques, Morgan Kaufmann.

[2] García, S., et al. (2015). Data Preprocessing in Data Mining, Springer.

[3] Melichar, B., et al. (2005). "Text searching algorithms." Available on:http://stringology.org/athens.

[4] Luhn, H. P. (1957). "A statistical approach to mechanized encoding and searching of literary information." IBM Journal of research and development **1**(4): 309-317.

[5] Pedregosa, F., et al. (2011). "Scikit-learn: Machine learning in Python." The Journal of Machine Learning Research **12**: 2825-2830.

[6] Bressert, E. (2012). SciPy and NumPy: An Overview for Developers," O'Reilly Media, Inc.".

[7] Walt, S.; Colbert, S. C.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[8] Jones, E.; Oliphant, E.; Peterson, P., et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/

[9] Cortes, C.; Vapnik, V. (1995). Support-vector networks. Machine Learning 20 (3): 273

[10] Breiman, Leo (2001). Random Forests. Machine Learning 45 (1): 5–32. DOI:10.1023/A:1010933404324

[11] Cutler, A.; Cutler, D. R; Stevens, J. R. (2012). Random Forests. Ensemble Machine Learning. 157-175. Springer US

[12] Aggarwal, C. C. (2014). Data classification: algorithms and applications, CRC Press.