Detecting Textual Analogies Using Semi-Supervised Learning

Rei Rembeci
Department of Computer Science,
Earlham College,
Richmond, IN 47374
rrembe15@earlham.edu

ABSTRACT

Semi-supervised learning is a family of techniques in machine learning, which has seen a growth in interest due to the fact it requires less human effort and can provide better accuracy compared to other machine learning techniques. Analogy is an essential aspect of human communication, understanding, and knowledge sharing. However, strategies for detecting textual analogies using machines are largely unexplored. There is also no standard corpus of textual analogies. This paper presents a system for detecting analogies in a given text using two semi supervised learning techniques; transductive support vector machines (TSVMs) and label propagation. Count vectorization, tf-idf, and hash vectorization are the explored feature extraction tools. The Brown corpus is divided into three parts: unlabeled data used for training the classifier, manually labeled data used for training the classifier, and manually labeled data used for testing the classifier. The accuracy of the classifications, confusion matrices, as well as the f-scores, are used to evaluate the performance of the system.

KEYWORDS

analogy, natural language processing, semi-supervised learning, transductive support vector machines, label propagation

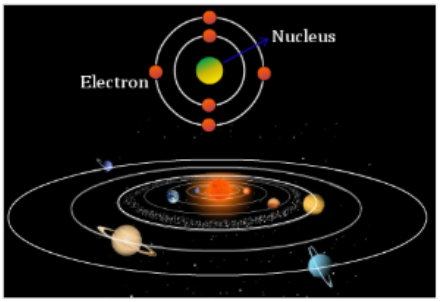
1 INTRODUCTION

Natural language processing (NLP) is an intersectional field concerning interactions and relationships between computers and natural languages. Machine learning can be used to recognize patterns in natural languages and use such patterns to make predictions or classifications. The machine learning process can be unsupervised, supervised or semi-supervised. Semi-supervised learning considers the problem of classification when only a subset of the training data have corresponding class labels. Such problems are of immense practical interest in a wide range of applications, including image search, natural language parsing, and speech analysis, where unlabeled data is abundant, but obtaining class labels is expensive or impossible to obtain for the entire data set [9].

Analogy involves the comparison of two structured representations. The representations being compared typically include labeled relationships between entities and between other relations [6]. For example, when reading "the Rutherford model of the atom is analogous to the solar system," we can draw similarities between the two structures. Specifically, we can learn that the nucleus of the atom has the same role as the sun in the solar system, and electrons orbit the nucleus just like the planets orbit the sun. These similarities are illustrated in figure 1. Other cognitive processes such as solving a problem in a similar way to another problem previously solved, involve analogy-making. Cognitive systems that can process through textual analogies can learn more from what they read [6]. Thus, being able to detect analogies is an essential aspect of human communication, understanding, and knowledge sharing, as it serves as a foundation for other types of thinking. To the best of my knowledge, strategies for detecting analogy with a machine are largely unexplored. So far as I have been able to determine, there is no standard corpus of analogy text, an important tool for developing and evaluating algorithms for analogy text classification.

Rutherford Atom and Solar System

Atom	Solar System
Nucleus	Sun
Electron	Planet
Electromagnetism	Gravity



http://images.tutorcircle.com/cms/images/44/rutherfords-atomic-model1.png

Figure 1

This paper offers a system which uses semi-supervised learning to detect textual analogies. In particular it uses two semi-supervised learning techniques; transductive support vector machines and label propagation. Count vectorization, tf-idf, and hash vectorization are the explored feature extraction tools. A comparison between the performance of the classifiers and between the feature extraction tools is offered. Section 2 offers a survey report of the related work that influenced this system. The framework of the system is described in section 3, and the implementation is described in section 4. Section 5 offers the evaluation of the system, followed by a conclusion and future goals in section 6.

2 RELATED WORK

In this section, an overview of some of the most important parts of the system are described. In the first subsection, an explanation of count vectorization, tf-idf, and hashing vectorization is offered to display their differences. Next, an overview of the transductive

support vector machines and label propagation is presented. In the last subsection, the importance of analogy is further described.

2.1 Feature extraction

Text analysis is a major application field of machine learning algorithms. However the raw data cannot be fed directly to the algorithms themselves as most of them operate over numerical feature vectors with a fixed size rather than raw text documents with variable length [12]. To address this, feature extraction methods can be used to transform other types of data, such as text or images, into numerical features usable for machine learning. There are different methods which allow such transformation, such as: count vectorization, tf-idf vectorization, hash vectorization and many more.
Countvectorizer implements both occurrence counting and tokenization. Tokenization is the process of chopping up strings into pieces called tokens [12].
In a large text corpus, some words such as "the", "a", and "is" will be very frequent. Such words carry very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. Thus, in order to re-weight the count features into floating point values suitable for usage by a classifier, it is very common to use the tf-idf transformation. Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. Term-frequency represents the number of times a term occurs in the text corpus. Inverse document-frequency diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely [12].
Hashing Vectorizer implements the hashing trick, the application of a hash function to the features and using their hash values as indices directly, rather than looking the indices up in an associative array [14]. This methods offers several advantages. It requires a very low memory for large datasets, it is fast to pickle and un-pickle as it holds no state besides the constructor parameters, and it can be used in a streaming (partial fit) or parallel pipeline as there is no state computed during fit [12]. However, some of the disadvantages of HashingVectorizer are the following: there is no way to compute the inverse transform (from feature indices to string feature names) which can be a problem when trying to introspect which features are most important to a model, there can be collisions, and IDF weighting cannot be used since it would would render the transformer stateful [12].

2.2 Semi-supervised learning

Because semi-supervised learning requires less human effort and gives higher accuracy compared to other machine learning techniques, it is of great interest both in theory and in practice [15]. Commonly used methods for semi-supervised learning, include: transductive support vector machines, generative models, and graph-based models.
Transductive support vector machines are an extension of standard support vector machines. They incorporate the usage of unlabeled data, to find a labeling of the unlabeled data so that a linear boundary has the maximum margin on both the original labeled data and the now labeled, unlabeled data [15].
Bennett and Demiriz present a semi-supervised SVM model using the entire data available from both the training set and the testing/working set, namely S^3VM [2]. Next, they show that this model can be converted from a 1-norm linear SVM to a mixed-integer program. The integer S^3VM, when tested on transduction using overall risk minimization, either improved or showed no significant difference in generalization compared to the usual structural risk minimization approach. Moreover, Joachims presents transductive support vector machines as a method used in semi-supervised learning for text classification [8]. It begins by providing an explanation of the transductive approach taken with the SVMs. Joachims also explains why TSVMs are well suited for text classification. Next, Joachim conducts experiments which provide a significant improvement in the accuracy of text classification when compared to SVMs or multinomial Bayes [8].
Transductive support vector machines might seem to be the perfect semi-supervised algorithm, since it combines the powerful regularization of SVMs with a direct implementation of the cluster assumption. However, its main drawback is that the objective function, which places the decision boundary in low density regions using gradient descent, is non convex and thus difficult to minimize [4].
Graph-based semi-supervised methods define a graph where the nodes are labeled and unlabeled examples in the dataset, and edges reflect the similarity of examples. These methods usually assume label smoothness over the graph. Graph methods are discriminative, and transductive [15].
Szummer and Jaakkola combine a limited number of labeled examples with a Markov random walk representation over the unlabeled examples. Next, the authors develop and present the theory behind the estimation criteria/algorithms developed. These algorithms are suited to the Markov representation and the tests are done for text classification. The results show that the Markov random walk representation outperformed SVM [13]. Moreover, Zhu and Ghahramani present label propagation, which is closely related to the Markov random walks algorithm but approaching the problem from a different perspective [16].

2.3 Computational models of analogy

Analogy involves the comparison of two structured representations. The representations being compared usually include labeled relationships between entities and between other relations.[6]
Analogy-making is a basic cognitive ability. It is present in humans from a young age, and develops over time [10]. For example, it starts with a baby imitating an adult, and progresses with developing the ability to recognize an analogy between a picture and a corresponding real object. Moreover, other cognitive processes such as perceiving a stone as a human face, solving a problem in a way similar to another problem previously solved, arguing in court for a case based on its common structure with another case, and many more, involve analogy-making [10]. Hence, analogy-making seems to be crucial for human cognition as it serves as a foundation for other types of thinking. As a result, it is important to develop computational models of analogy-making.
The computational modeling of analogy has progressed rapidly in

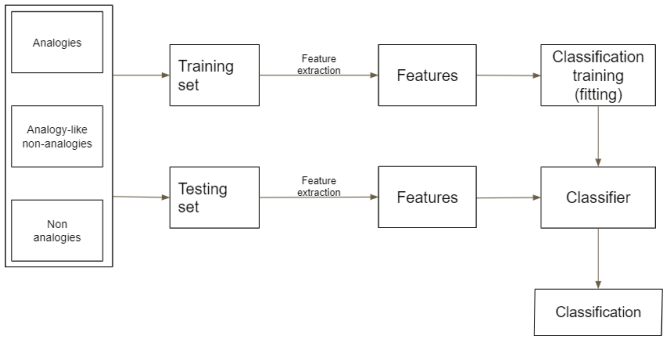


Figure 2: System design

the past 25 years. It has been fueled by a collaboration between psychologists, AI scientist, linguists and philosophers. Gentner's model of analogy is broken down into: retrieval, mapping, abstraction, and re-representation. Out of the four categories aforementioned, mapping is the most important one. Mapping consists of aligning two given situations to produce a set of correspondences that indicate "what goes with what," candidate inferences that follow from the analogy, and a structural evaluation score which provides a numerical measure of how well the base and the target align [6]. Barbella and Forbus, focusing on instructional textual analogies, explore how analogy can be integrated into dialogue act theories based on an analysis of how the functional constraints of analogical mapping and case construction interact with the properties of discourse [1].

3 SYSTEM DESIGN

This section describes the different aspects of the system design shown in figure 2, specifically: extracting and cleaning the corpora; building the training and testing sets; feature extraction; exhaustive search on the parameters; and finally training and testing the classifier.

3.1 Extracting and cleaning the corpora

First, the corpus is extracted into a CSV format. Then, a word hunting script, which looks for phrases that might indicate the presence of an analogy, shown below by figure 3, splits the data into two datasets.

is like	similarly
are like	the same as
like a	likewise
like an	equivalent to
analogous	identical to
analogy	comparable to
similar to	as __ as

Figure 3: List of phrases

The first dataset contains all the sentences which, according to the word hunt are more likely to contain analogies. Similarly, the second dataset contains all the sentences which according to the word hunt are less likely to contain analogies. Now, all the sentences from the corpus need to be preprocessed to remove metadata in order to avoid the risk of debasing the experiments. The metadata is initially added when extracting the corpus in order to give each sentence a unique identifier. For instance, a sentence will look like: "[sourcename.txt, PARA#1, SENT#4], That just burns my toast, you know what I mean?." In this case, we are only interested in the sentence itself, and the source name, paragraph number and sentence number are not given to the system for training or testing.

3.2 Building the training set and the testing set

The dataset with potential analogies is manually labeled. This results in two subsets: analogies, and analogy-like nonanalogies. Analogy-like nonanalogies consist of those sentences which once labeled, turn out to not contain analogies. Analogies and analogy-like nonanalogies, combined with the non analogies dataset, the dataset that word hunt classified as non potential analogies, form the complete set. The complete set is randomly shuffled and 85% of it form the training set, while the remaining form the testing set. Since semi supervised learning will be later used, only 2% of the training set is unlabeled, while the testing set is entirely labeled.

3.3 Feature extraction

Once the training set and the testing set are created, the textual data from the sets needs to be converted into a numerical matrix fit for the classifiers. The system provided in this paper utilizes all the methods described in the survey section, namely count vectorization, tf-idf vectorization, and hash vectorization.

3.4 Exhaustive search on the parameters

To boost the performance of the classifiers, an exhaustive search over specified parameters values for each combination of feature extraction tools and classifier is computed. The exhaustive search implements the usual estimator API: when fitting it on a dataset, all the possible combinations of parameter values are evaluated and the best combination is retained [3]. The list of the parameters search over are shown in the figure 4 below.

C	penalty parameter C of the error term
kernel	specifies the kernel type to be used in the algorithm (linear or rbf)
gamma	kernel coefficient for 'rbf'
lamU	cost parameter that determines influence of unlabeled patterns
Label Propagation	
kernel	specifies the kernel type to be used in the algorithm (knn or rbf)
tol	convergence tolerance: threshold to consider the system at steady state
n_neighbors	parameter for the number of k when the kernel is set to be the k-nearest neighbor

Figure 4: List of parameters

3.5 Training and testing the classifier

Once the textual data is converted into a numerical sparse matrix, the classifier is ready to be trained using the training set. Transductive support vector machines and label propagation are the semi supervised learning classifiers used by the system described in this paper.

4 IMPLEMENTATION

The implementation on the system and experiments are accomplished in Python. The written scripts are a combination of self-written scripts, Scikit Learn modules, and adaptation of the transductive support vector as offered by Gieseke et al. [7].

4.1 Python

4.1.1 *Scikit Learn.*

The Scikit Learn module[3] provides implementation for the following:

a) Feature extraction
b) Exhaustive searches
c) Label Propagation
d) Confusion matrix

4.1.2 *Self-written scripts.*

The following scripts are used to extract the corpora, build the training set and the testing set, and analyze the results.

a) compile_folder
compile_folder is used to extract and convert all the text files of the corpus to a CSV file.
b) wordhunt
wordhunt goes through each sentence in the CSV file generated by compile_folder and looks for phrases shown in Figure 3. It then creates two CSV files, one with sentences that include the aforementioned phrases, and one with sentences that don't.
c) functions
functions build the training and the testing set, as well as extract the features from these sets
d) main_grid
main_grid implements the exhaustive search on the parameters. It takes as input the name of the classifier and the set of parameter values to be searched over. It returns the set of parameters which produced the highest score when training the classifier, along with the score.
e) main_interface
main_interface is the central script. It takes as input the positive set, the negative set, and name of classifier. Its output is the overall accuracy, precision, recall, f1-score, and the confusion matrix.
f) overlap-test
overlap-test runs an overlapping test in error between two sets of (classifier - feature extraction tool) pairs.

4.1.3 *Tranductive support vector machines.*

The transductive support vector machines script used is the Scikit wrapper provided by Madl [11]. The initial Scikit wrapper was designed under Python 2, but the system provided in this paper uses Python 3. As a result, necessary changes were made in the code.

5 EXPERIMENTS

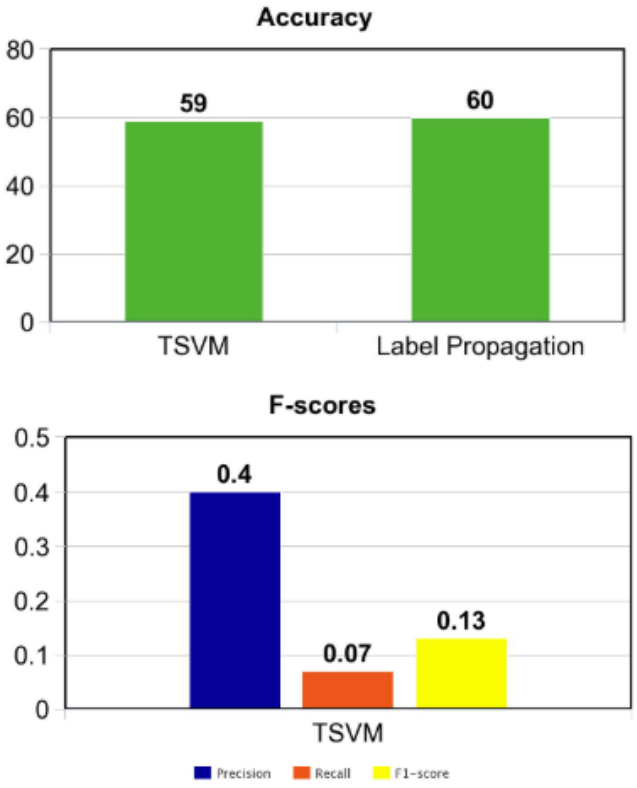
5.1 The Brown Corpus

The Brown Corpus contains of over 1 million words, consisting of 500 samples, distributed across 15 genres including political reportage, book reviews, government documents and many more [5]. For the experiments, 20456 random sentences were used in total. Out of these 20456 sentences, 69 were part of the testing set while the remaining 20387 were part of the training set. Out of the 20387 sentences in the training set, 20001 were unlabeled and the remaining 386 were labeled.

5.2 Analysis

The performance of the system is evaluated based on the overall accuracy, confusion matrix, and f-scores. To further analyze such results, an overlapping test between the two classifiers is run, which determines the overlap in error between the performances of different classifiers. The results of the tests can provide important information about the features and the classifiers. Provided that the classifiers produce different set of right and wrong classifications, a combination of the classifiers can potentially be used to increase the overall accuracy of the system. On the other hand, if the classifiers produce similar set of right and wrong classifications, a new strategy needs to be formulated for labeling the "hard to classify" examples correctly.

6 CONCLUSION AND FUTURE WORK



Transductive support vector machines and label propagation showed similar results when considering the accuracy of the model. However, the f-scores were significantly different. For the transductive support vector machines, the precision was 0.4, the recall was 0.07, and f1-score was 0.13. These results are displayed in the figure

above. On the other hand, label propagation classified everything as non-analogies, leading to a precision, recall, and f1-score equal to 0. This meant that the overlapping test between the classifiers produced similar sets of wrong classification, which suggest a better strategy is needed for detecting textual analogies. As a result, for future work I would like to use a parser to extract the base and target from each sentence instead of considering the full sentence. While three different feature extraction tools were used (count, tf-idf, hash) they did not have any substantial effect on the performance of the classifiers.

7 ACKNOWLEDGMENTS

This project was supported by the Earlham College Computer Science Department as part of the Senior Capstone. I would like to thank my capstone adviser, David Barbella, who provided his constant assistance and knowledge throughout the process. I would also like to thank Xunfei Jiang, Charles Peck, and Ajit Chavan for their feedback and suggestions during Methods for Research and Senior Capstone. Lastly, thanks to Phi Nguyen, for helping with debugging.

REFERENCES

[1] David Michael Barbella and Kenneth D Forbus. 2011. Analogical Dialogue Acts: Supporting Learning by Reading Analogies in Instructional Texts.. In *AAAI*.
[2] Kristin P. Bennett and Ayhan Demiriz. 1999. Semi-supervised support vector machines. In *Advances in Neural Information processing systems*. 368–374.
[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.
[4] Olivier Chapelle and Alexander Zien. 2005. Semi-Supervised Classification by Low Density Separation.. In *AISTATS*. Citeseer, 57–64.
[5] W. N. Francis and H. Kucera. 1979. *Brown Corpus Manual*. Technical Report. Department of Linguistics, Brown University, Providence, Rhode Island, US. http://icame.uib.no/brown/bcm.html
[6] Dedre Gentner and Kenneth D Forbus. 2011. Computational models of analogy. *Wiley interdisciplinary reviews: cognitive science* 2, 3 (2011), 266–276.
[7] Fabian Gieseke, Antti Airola, Tapio Pahikkala, and Oliver Kramer. 2014. Fast and simple gradient-based optimization for semi-supervised support vector machines. *Neurocomputing* 123 (2014), 23–32.
[8] Thorsten Joachims. 1999. Transductive inference for text classification using support vector machines. In *ICML*, Vol. 99. 200–209.
[9] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*. 3581–3589.
[10] Boicho Kokinov and Robert M French. 2003. Computational models of analogy-making. *Encyclopedia of cognitive science* 1 (2003), 113–118.
[11] Tamas Madl. 2016. Semisupervised learning frameworks in Python. https://github.com/tmadl/semisup-learn
[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[13] Martin Szummer and Tommi Jaakkola. 2002. Partially labeled classification with Markov random walks. In *Advances in neural information processing systems*. 945–952.
[14] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2009. Feature hashing for large scale multitask learning. *arXiv preprint arXiv:0902.2206* (2009).
[15] Xiaojin Zhu. 2005. Semi-supervised learning literature survey. (2005).
[16] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. (2002).