Classification of Breach of Contract Court Decision Sentences

Wai Yin Mok
The University of Alabama in Huntsville
Huntsville, AL 35899, USA
mokw@uah.edu

Jonathan R. Mok
Legal Services Alabama
Birmingham, AL 35203, USA
jmok@alsp.org

CEUR-WS.org/Vol-2385/paper7.pdf

ABSTRACT

This research project develops a methodology to utilize machine-learning analysis of live disputes to assist legal professionals in narrowing issues and comparing relevant precedents. An important part of this research project is an ontology of target court decisions, which is first constructed from relevant legal statutes, probably assisted by human legal experts. Utilizing the ontology, we extract and classify sentences in court decisions according to type. Since court decisions are typically written in natural languages, such as English and Chinese, chosen court decisions are pre-processed with the Natural Language Processing library spaCy, and then fed into the machine learning component of the system to extract and classify the sentences of the input court decisions.

KEYWORDS

Ontology; Machine-Learning; spaCy; Natural Language Processing

1 INTRODUCTION

This research project develops a methodology to utilize machine-learning analysis of live disputes to assist legal professionals in narrowing issues and comparing relevant precedents. At this early stage of the research, we limit the scope to a specific methodology for legal research of breach of contract issues that also creates a general template for all other legal issues with common elements and/or factors. Our first step is to extract and classify sentences in breach of contract court decisions according to type. Such court decisions have five basic sentence types: sentences on contract law, sentences on contract holding, sentences on contract issues, sentences on contract reasoning, and sentences on contract facts. Classifying sentences is an important first step of our long-term research goal: developing a machine learning system that analyzes hundreds or thousands of past breach of contract court decisions similar to the case at hand, thus providing a powerful tool to legal professionals when faced with new cases and issues. Further downstream processing can be made possible by this project, such as constructing decision trees that predict the likely outcome for the case at hand, displaying the rationales on which court decisions are based, and calculating the similarity of previous legal precedents.

An understanding of the document structure of breach of contract court decisions is crucial to this research. Ontologies, or formal representations, have been created for many applications of diverse academic disciplines, including Artificial Intelligence and Philosophy. An ontology of the target court decisions, which formally defines the components and their relationships thereof, is therefore first constructed from relevant legal statutes, likely assisted by human legal experts. The constructed ontology will later be utilized in the machine-learning phase of the system.

Court decisions are typically written in natural languages, such as English and Chinese; hence, the proposed system must be able to process natural language. At this early stage of the research we focus on English. In recent years, Natural Language Processing (NLP) has made significant progress. Years of research in linguistic and NLP have produced many industrial-strength NLP Python libraries. Due to its ease of use and generality, spaCy [10] is chosen for this research project. Chosen court decisions are first preprocessed with spaCy. After that, the court decisions, and the information added by spaCy, are fed into the machine learning component of the system.

The rest of the paper is organized as follows. Relevant past research activities are presented in Section 2, which also puts our research into a boarder perspective. The ontology of breach of contract court decisions is given in Section 3. Classification of the sentences with respect to the chosen sample court decisions, based on our implementation of the logistic regression algorithm and that of scikit-learn [5], is demonstrated in Section 4. We present possible research directions in Section 5 and we conclude the paper in Section 6.

2 LITERATURE REVIEW

The organization and formalization of legal information for computer processing to support decision making and enhance search, retrieval, and knowledge management is not recent; the concept dates back to the late 1940s and early 1950s, with the first legal information systems being developed in the 1950s [2, 15]. Knowledge of the legal domain is expressed in natural languages, such as English and Chinese, but such knowledge does not provide a well-defined structure to be used by machines for reasoning tasks [7]. Furthermore, because language is an expression of societal conduct, it is imprecise and fluctuating, which leads to challenges in formalization, as noted in a panel of the 1985 International Joint Conferences on Artificial Intelligence, that include: (a) legal domain knowledge is not strictly orderly, but remains very much an experience-based example driven field; (b) legal reasoning and argumentation requires expertise beyond rote memorization of a large number of cases and case synthesis; (c) legal domain knowledge contains a large body of formal rules that purport to define and regulate activity, but are often deliberately ambiguous, contradictory, and incomplete; (d) legal reasoning combines many different types of reasoning processes such as rule-based, case-based, analogical, and hypothetical; (e) the legal domain field is in a constant state of change, so expert legal reasoning systems must be easy to modify and update; (f) such expert legal reasoning systems are unusual

In: Proceedings of the Third Workshop on Automated Semantic Analysis of Information in Legal Text (ASAIL 2019), June 21, 2019, Montreal, QC, Canada.
© 2019 Copyright held by the owner/author(s). Copying permitted for private and academic purposes.
Published at http://ceur-ws.org

due to the expectation that experts will disagree; and (g) legal reasoning, natural language, and common sense understanding are intertwined and difficult to categorize and formalize [13]. Other challenges include the (h) representation of context in legal domain knowledge; (i) formalizing degrees of similarity and analogy; (j) formalizing probabilistic, default, or non-monotonic reasoning, including problems of priors, the weight of evidence and reference classes; and (k) formalizing discrete versus continuous issues [6].

To address these challenges, expert system developers have turned to legal ontology engineering, which converts natural language texts into machine extractable and minable data through categorization [1]. An ontology is defined as a conceptualization of a domain into a human understandable, machine-readable format consisting of entities, attributes, relationships, and axioms [9]. Ontologies may be regarded as advanced taxonomical structures, where concepts formalized as classes (e.g. "mammal") are defined with axioms, enriched with description of attributes or constraints (e.g. "cardinality"), and linked to other classes through properties (e.g., "eats" or "is_eaten_by") [2]. The bounds of the domain are set by expert system developers with formal terms to represent knowledge and determine what "exists" for the system [8]. This practice has been successfully applied to fields such as biology and family history to format machine unreadable data into readable data and is practiced in legal ontology engineering [4, 11, 14].

NLP can be employed to assist in legal ontology engineering; NLP is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things [3]. After years of research in linguistics and NLP, many NLP libraries have been produced and are ready for real-life applications. Examples are NLTK, TextBlob, Stanford CoreNLP, spaCy, and Gensim. These libraries can parse and add useful information about the input text, which makes machine understanding of legal texts possible. In addition, machine learning has gained tremendous attention in recent years because of its ability to learn from examples. With the additional information added by the NLP libraries, machine learning can be applied to legal ontological engineering because the concepts identified in the ontology can be populated with real-life facts, rules, and reasoning extracted from past legal cases of interest. Many mature machine learning libraries have also been produced. To name a few, there are TensorFlow, scikit-learn, PyTorch, and Apache Mahout.

3 ONTOLOGY

In terms of Computer Science, an ontology is a conceptual model for a certain system, which defines its components, and the attributes of, and the relationships among, those components. Following the notation of [11, 14], the ontology created for this paper is shown in Figure 1.

Each rectangle in the model denotes a set of objects. The rectangle "Breach of Contract Court Decision" represents the set of all the breach of contract court decisions in the state of Alabama. A filled triangle denotes a whole-part relationship, which means that an object of the rectangle connected to the apex of the triangle is a whole while an object of the rectangle connected to the base of the triangle is a part. Hence, a breach of contract court decision is composed of many sentences. An open triangle denotes a superset-subset relationship where the set of objects connected to its apex is a superset of the set of objects connected to its base. Thus, the set "Sentence" has six subsets: "Title", "Law", "Holding", "Fact", "Issue", and "Reasoning". The plus sign + inside of the open triangle means that in addition to the superset-subset relationship, no two subsets can have any common element. This means that no two of the six subsets intersect. A line in the model denotes a relationship set between the rectangles (object sets) connected by the line. A relationship set contains all of the relationships of interest between the set of objects located at one end and the set of objects located at the other end of the line. The connection point of a relationship set and a rectangle is associated with a participation constraint. A participation constraint 1:* means that the minimum is 1 and the maximum is *, where * can mean any positive integer. A participation constraint 1 is a shorthand symbol for 1:1. The pentagon in the figure means a 5-way relationship set, rather than the usual binary relationship sets.

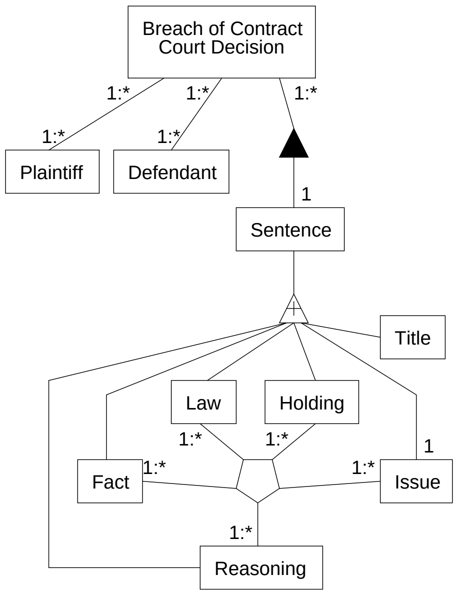

Figure 1: An ontology created for breach of contract court decisions.

Title sentences are not as useful as the other five categories because a title sentence is a part of a court decision's title or format, although it might contain useful information, like the names of the plaintiffs and defendants.

To justify the other categories of sentences, note that legal data is composed of written text that is broken down into sentences. Sentences in legal texts can be further categorized into five sentence types relevant to legal analysis: (1) fact sentences, (2) law sentences, (3) issue sentences, (4) holding sentences, and (5) reasoning sentences. In essence, legal texts arise from disputes between two or more parties that are ultimately decided by a third party arbitrator, which in most cases is a judge. In this study, machine learning will be strictly applied to case law legal texts, which are written decisions by a judge that detail a dispute between two or more parties and the application of law to the dispute to reach a conclusion. In each case decision, there are facts that form the background of the dispute. Fact sentences contain the relevant facts as determined by the judge that underlie any written decision. This is discretionary in nature and relies on the judge to correctly evaluate the facts underlying the dispute and explicate them. Law sentences are statements of law that the judge deems applicable to the dispute. Law sentences are always followed by legal citations. Issue sentences are statements made by either party in the dispute, or by the judge, that can be (a) assertions made by either party of what are correct applications of law or true determinations of facts, or (b) statements by the judge of what he or she determines are the relevant issue(s) underlying the dispute, and how he or she frames such issue(s). Issue statements include the arguments that either party makes to support their respective interests; such sentences are not considered fact sentences or law sentences because only the judge is able to make dispositive statements of fact and law as the final arbitrator of the dispute. Holding sentences are determinations by the judge that apply the law to the dispute and reach a conclusion, also known as a case holding. Holding sentences are the judge's determination of how the law is applied to the facts of the dispute and the corollary conclusion that sides with one party or the other. Holding sentences are actually considered new statements of law as application of law towards a new set of facts—as no dispute is identical—made by a judge that sets precedent for future disputes; holding sentences are arguably the most important sentences of a case decision as such sentences are the only operative sentences that have bearing on future disputes. Only direct application of law to a present set of facts is considered a holding of the case and binding precedent. Reasoning sentences are statements by the judges that explain how he or she reached the conclusion that he or she did in the case. Reasoning sentences will typically detail the steps the judge made to reach his or her conclusion, interpretation of law and/or fact, comparison and differentiation of prior case law, discussion of societal norms, and other analytical methods judges utilize to reach conclusions. Holding sentences and reasoning sentences are similar, but can be distinguished in the regard that holding sentences are direct applications of law to the present facts of the present dispute, whereas reasoning sentences do not provide any direct applications of law to present facts of present disputes and only explain the mental gymnastics the judge has performed to reach his or her conclusion. At times it is even difficult for legal professionals to distinguish holding sentences from reasoning sentences, but any hypothetical application of law to fact is considered reasoning. As such, the five types of sentences are related in a nontrivial way. Hence, the five-way relationship set in Figure 1 denotes such a complicated relationship among the five categories of sentences.

4 CLASSIFICATION OF SENTENCES

4.1 Test Cases

In this early stage of our research project, three sample breach of contract court decisions are considered. As the research progresses, more court decisions will be added.

(1) 439 So.2d 36, Supreme Court of Alabama, GULF COAST FABRICATORS, INC. v. G.M. MOSLEY, etc., 81-1042, Sept. 23, 1983.
(2) 256 So.3d 119, Court of Civil Appeals of Alabama, Phillip JONES and Elizabeth Jones v. THE VILLAGE AT LAKE MARTIN, LLC, 2160650, January 12, 2018.
(3) 873 F.Supp. 1519, United States District Court, M.D. Alabama, Northern Division, Kimberly KYSER-SMITH, Plaintiff, v. UPSCALE COMMUNICATIONS, INC., Bovanti Communication, Inc., Sally Beauty Company, Inc., Defendants, Civ. A. No. 94-D-58-N, Jan. 17, 1995.

The industrial-strength NLP library spaCy is chosen to extract the sentences from the selected court decisions because of its ease of use and cleanliness of design [10].

4.2 Logistic Regression Algorithm

We have not only implemented the logistic regression algorithm, but have also compared our implementation with the one provided by scikit-learn [5], which is highly optimized for performance purposes. The chief finding is that our implementation consistently has more correct predictions than the one provided by scikit-learn.

4.2.1 Recording the key words of the sentences. Classification of the sentences in a court decision requires identifying the features of the sentences. For this purpose, we generate a list of key words. spaCy generates a total of 529 sentences and 1861 key words from the three sample court decisions. We thus use a 529×1861 two-dimensional Numpy array X to store the number of appearances of each keyword in each sentence. In X, X[i,j] records the number of times keyword j appears in sentence i.

4.2.2 Our implementation of the logistic regression algorithm. Although our implementation began with the Adaptive Linear Neuron algorithm in [12], we have made numerous modifications. The following code segment demonstrates the most salient parts of our implementation.

```
def __init__(self, eta=0.01, n_iter=50, slopeA=0.01):
    self.eta = eta
    self.n_iter = n_iter
    self.slopeErrorAllowance = slopeA

def activationProb(self, X, w):
    z = np.dot(X, w[1:]) + w[0]
    phiOfz = 1/(1+np.exp(-z))
    return phiOfz

def fit_helper(self, X, y, w):
    stopVal = self.slopeErrorAllowance * X.shape[1]
    for i in range(self.n_iter):
        phiOfz = self.activationProb(X, w)
        output = np.where(phiOfz >= 0.5, 1, 0)

        errors = (y - output)
        slopes = X.T.dot(errors)
        w[1:] += self.eta * slopes
        w[0] += self.eta * errors.sum()
        slopeSum = abs(slopes).sum()
        if slopeSum <= stopVal:
            break
```

```
def fit(self, X, aL):
    self.ansList = np.unique(aL)
   self.ws = np.zeros((len(self.ansList), 1+X.shape[1]))
    for k in range(len(self.ansList)):
        y = np.copy(aL)
        y = np.where(y == self.ansList[k], 1, 0)
        self.fit_helper(X, y, self.ws[k])

def predict(self, X):
    numSamples = X.shape[0]
   self.probs = np.zeros((len(self.ansList), numSamples))
    for k in range(len(self.ansList)):
      self.probs[k] = self.activationProb(X, self.ws[k])
    self.results = [""] * numSamples
    for i in range(numSamples):
        maxk = 0
        for k in range(len(self.ansList)):
           if self.probs[k][i] >= self.probs[maxk][i]:
               maxk = k
        self.results[i] = self.ansList[maxk]
    return np.array(self.results)
```

Regarding the three sample court decisions, the parameter w of the function activationProb is a one-dimensional Numpy array of 1862 integers. w[1:], of size 1861, stores the current weights for the 1861 key words and w[0] is the constant term of the net input z. The function activationProb first calculates the net input z by computing the dot product of the rows of X and w. In the next step the function calculates the probabilities $\phi(z)$ for the sample sentences (rows) of X based on the net input z. (Technically, the function calculates the conditional probability that an input sentence has a certain sentence type given the sample sentences in X. However, they are simply called probabilities in this paper to avoid being verbose.) It then returns the one-dimensional Numpy array phiOfz, which has the activation probability for each sample sentence in X.

The function fit_helper receives three parameters: the same two-dimensional Numpy array X, the one-dimensional Numpy array y that stores the correct outputs for the sample sentences of X, and the one-dimensional Numpy array w that has the weights and the constant term calculated for the 1861 key words. The function first calls activationProb to calculate the probabilities for the sample sentences of X. It then calculates the output for each sample sentence: 1 if the probability ≥ 0.5; 0 otherwise. The errors for the sample sentences are then calculated accordingly. The most important part of the function is the calculation of the slopes (partial derivatives) of the current weights, which will be updated in the next line. The constant term w[0] is updated after that.

Since there are 1861 weights, there are 1861 slopes. The ideal scenario is that each slope will become zero in the calculation, which is practically impossible. We thus allow an allowance for such a small discrepancy, which is stored in self.slopeErrorAllowance. Thus, the total allowance for 1861 slopes is self.slopeErrorAllowance * 1861 (= X.shape[1]), which is then stored in stopVal. If the sum of the absolute values of the 1861 slopes is less than stopVal, it will exit the for loop and no more updates are necessary. Otherwise, the for loop will continue for self.n_iter times, a number that is set to 500. Note that the learning rate self.eta is set to 0.0001.

Table 1: Comparison of our implementations and scikit-learn's implementation of the logistic regression algorithm

Methods	Tests	Correct Guesses	Total Guesses
Ours	100	8707	15900
scikit-learn's	100	8454	15900

The function fit accepts two parameters: X the two-dimensional Numpy array, and aL the one-dimensional array that stores the correct sentence types of the sample sentences of X. It first finds the unique sentence types in aL. Then, it applies the one versus the rest approach for each unique sentence type to calculate the weights and the constant term for the 1861 key words. To do so, each appearance of the sentence type in y, which is a copy of aL, is replaced with a 1 and 0 elsewhere. The function fit then calls self.fit_helper to calculate the 1861 weights and the constant term for that particular sentence type, which will be used to make predictions.

The calculated weights and constant term for each sentence type are stored in self.ws, which is a 6×1862 Numpy array. To make a prediction for a sentence, we use the predict function, which calculates the probability of each sentence type on the input sentence and assigns the sentence type that has the greatest probability to the sentence.

4.2.3 scikit-learn's implementation of the logistic regression algorithm. scikit-learn's implementation of the logistic regression algorithm can be straightforwardly applied. We first randomly shuffle the 529 sentences and their corresponding sentence types. We then apply the function train_test_split to split the 529 sentences and their sentence types into two sets: 370 training sentences and 159 testing sentences. Afterwards, a logistic regression object is created and fitted with the training data.

4.2.4 Discussions. To compare our implementation and that of scikit-learn for the logistic regression algorithm, we randomly select 370 training sentences and 159 testing sentences from the 529 sentences of the three sample court decisions and feed them to our implementation. The results are shown in Table 1, which indicates that our implementation has more correct predictions than the highly optimized implementation of scikit-learn. However, since we are familiar with our code, our implementation can serve as a platform for future improvements.

Our result is remarkable if we compare our implementation with randomly guessing the sentence types for the 159 sentences with six possibilities for each sentence. Note that the probability of randomly making a correct guess is $p = 1/6 = 0.166667$. Assuming each guess is an independent trial, making exactly 87 correct predictions and making 87 or more correct predictions should follow the binomial distribution. Thus, applying Microsoft Excel's binomial distribution statistical functions BINOM.DIST and BINOM.DIST.RANGE to the problems, we obtain the probabilities in Table 2.

The probabilities for both events are close to zero. Hence, our keyword approach to capturing the characteristics of the sentences of breach of contract court decisions is on the right track, although much improvement remains to be made.

Table 2: Probabilities of randomly guessing with exactly 87 or 87 or more correct predictions for the 159 testing sentences

Event	Formula	Probability
87 exactly	BINOM.DIST(87,159,p,FALSE)	9.08471E−28
87 or more	BINOM.DIST.RANGE(159,p,87,159)	1.08519E−27

5 FUTURE RESEARCH DIRECTIONS

In our experiments, we discover that if the sum of the absolute values of the slopes does not reduce to zero, it usually will go through cycles with respect to the number of iterations determined by the number self.n_iter, which is current set to 500. Our next goal is to find the right number for self.n_iter so that the sum of the absolute values of the slopes is as small as possible.

The ontology in Figure 1 requires more details. More details will result in more accurate characterization of the five types of sentences, which will lead to more accurate classification algorithms.

The word-vector approach of calculating the similarity of two sentences might yield good results. Thus, we will utilize Word2vec, which is one such algorithm for learning a word embedding from a text corpus [16].

6 CONCLUSIONS

In this early stage of the research, we focus on breach of contract court decisions. Five critical contract sentence types have been identified: contract fact sentences, contract law sentences, contract holding sentences, contract issue sentences, and contract reasoning sentences. spaCy, an NLP Python library, is used to parse the court decisions

Three sample breach of contract court decisions are chosen to test our own implementation and the highly optimized scikit-learn implementation of the logistic regression algorithm. Our implementation consistently has more correct predictions than the one provided by scikit-learn and it can serve as a platform for future improvements. Our keyword approach is a good starting-point because the number of correct predictions is far greater than the number produced by randomly guessing the sentence types for the 159 testing sentences. Lastly, many possible future improvements have also been identified.

REFERENCES

[1] Joost Breuker, André Valente, and Radboud Winkels. 2004. Legal Ontologies in Knowledge Engineering and Information Management. *Artificial Intelligence and Law* 12, 4 (01 Dec 2004), 241–277. https://doi.org/10.1007/s10506-006-0002-1

[2] Nuria Casellas. 2011. *Legal Ontology Engineering: Methodologies, Modelling Trends, and the Ontology of Professional Judicial Knowledge* (1 ed.). Law, Governance and Technology Series, Vol. 3. Springer Netherlands. https://doi.org/10.1007/978-94-007-1497-7

[3] Gobinda G. Chowdhury. 2003. Natural language processing. *Annual Review of Information Science and Technology* 37, 1 (2003), 51–89. https://doi.org/10.1002/aris.1440370103

[4] Kim Clark, Deepak Sharma, Rui Qin, Christopher G Chute, and Cui Tao. 2014. A use case study on late stent thrombosis for ontology-based temporal reasoning and analysis. *Journal of Biomedical semantics* 5, 49 (2014). https://doi.org/10.1186/2041-1480-5-49

[5] David Cournapeau et al. [n. d.]. scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/. ([n. d.]). Accessed: 2019-01-25.

[6] James Franklin. 2012. Discussion paper: how much of commonsense and legal reasoning is formalizable? A review of conceptual obstacles. *Law, Probability and Risk* 11, 2-3 (06 2012), 225–245. https://doi.org/10.1093/lpr/mgs007

[7] Mirna Ghosh, Hala Naja, Habib Abdulrab, and Mohamad Khalil. 2017. Ontology Learning Process as a Bottom-up Strategy for Building Domain-specific Ontology from Legal Texts. In *9th International Conference on Agents and Artificial Intelligence.* 473–480. https://doi.org/10.5220/0006188004730480

[8] Thomas R. Gruber. 1995. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies* 43, 5 (1995), 907 – 928. https://doi.org/10.1006/ijhc.1995.1081

[9] Nicola Guarino and Pierdaniele Giaretta. 1995. Ontologies and knowledge bases: towards a terminological clarification. In *Towards Very Large Knowledge Bases*, N.J.I. Mars (Ed.). IOS Press, Amsterdam, 25–32.

[10] Matthew Honnibal et al. [n. d.]. spaCy: Industrial-Strength Natural Language Processing in Python. https://spacy.io/. ([n. d.]). Accessed: 2019-01-25.

[11] Deryle Lonsdale, David W. Embley, Yihong Ding, Li Xu, and Martin Hepp. 2010. Reusing ontologies and language components for ontology generation. *Data & Knowledge Engineering* 69, 4 (2010), 318 – 330. https://doi.org/10.1016/j.datak.2009.08.003 Including Special Section: 12th International Conference on Applications of Natural Language to Information Systems (NLDB'07) - Three selected and extended papers.

[12] Sebastian Raschka. 2015. *Python Machine Learning* (1 ed.). Packt Publishing.

[13] Edwina Rissland. 1985. AI and Legal Reasoning. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 2.* 1254–1260.

[14] Cui Tao and David W. Embley. 2009. Automatic Hidden-web Table Interpretation, Conceptualization, and Semantic Annotation. *Data & Knowledge Engineering* 68, 7 (July 2009), 683–703. https://doi.org/10.1016/j.datak.2009.02.010

[15] Donald A. Waterman, Jody Paul, and Mark Peterson. 1986. Expert systems for legal decision making. *Expert Systems* 3, 4 (1986), 212–226. https://doi.org/10.1111/j.1468-0394.1986.tb00203.x

[16] Wikipedia contributors. [n. d.]. Word2vec. https://en.wikipedia.org/wiki/Word2vec/. ([n. d.]). Accessed: 2019-01-26.