An Investigation into Machine Learning Algorithms for Natural Language Processing

To what extent is machine learning effective compared to naïve algorithms for evaluating whether the content is matching the displayed text of hyperlinks in a series of Wiki pages?

Computer Science

4000 words

Contents

Introduction:

Natural Text Processing (NLP) is the branch of artificial intelligence focused on allowing machines to better interpret human text. As our languages are filled with irregularities and constantly evolve, achieving sufficient accuracy is a complicated task (IBM Cloud Education).

Search engines have made huge leaps in their complexity and ability to provide the most relevant information to the user, and their continual improvements are vital to the success of numerous tech corporations. These engines first began as lexical, meaning that literal matches to the search query were necessary for results to be returned (Berasategi). However, semantic search allowed for better understanding of the intent behind the search phrase, through large amounts of pre-existing data allowing for connections to be made between terms. The algorithms behind creating those connections have continued to evolve, with BERT being the latest innovation in this field (Raghavan). Though, search is the most relevant branch to the focus of this paper, NLP has a variety of other uses, notably in translation, sentiment analysis, and speech recognition, among many others.

A Wiki is a necessary tool for documentation in collaborative computer science projects. Team members can always change, and it is vital to have accessible information about the technical details of the project. The pages of a Wiki also typically have references to one another, through links that relate phrases to webpages. However, as these links have to be manually inserted, and especially since these pages undergo constant change, with many alterations to the content, inaccuracies may be present. Especially in more developed and large-scale wikis, there may be thousands of pages and links present, so manual verification of these would be extremely cumbersome.

Thus, this exploration will focus on practically demonstrating the benefits and drawbacks of NLP techniques and machine learning algorithms in a simpler NLP problem. An investigation will be made into relevant concepts, thus providing the reader with a comprehensive view of NLP techniques and improvements to text analysis that can be gained from an understanding this subject. The problem is similar to semantic search, since the algorithms will be outputting if a match exists between a search phrase (hyperlinked text) and a document (linked webpage), yet is simpler since indexing and ranking matches between all existing pages is not required. This allows for a naïve algorithm to be more viable, thus better highlighting the respective advantages of each approach. However, development of the naïve algorithm will be completed before research into machine learning to demonstrate application of prior knowledge. Overall, this investigation will answer whether simple rule-based algorithms are still applicable for use in NLP tasks, and demonstrate those potential use cases.

Implementing Natural Language Processing:

The main steps of implementing natural language process involves preprocessing the text, vectorizing data, and assembling a classifier for the given goal (Shetty). The naïve and machine learning algorithms are the classifiers in this case, since they will determine whether a match exists. The vectorized features given to each algorithm will also vary by the type, and be discussed in the respective sections.

Text Preprocessing:

Preprocessing first involves tokenizing, which is the process of “separating text into units” (Shetty), in this case separate words. Next, punctuation is removed since tokens such as “word” and “word,” would have the same meaning, and the algorithm should interpret both as the same.

Then stop-words, which are “common words that are likely to appear in any text” (Shetty) are removed to further clean the dataset and improve the model’s detection of meaningful tokens. For example, “an example of a word” would be reduced to “example” and “word”.

Finally, words with differing endings should be reduced to the same normalized form, such as “ending” and “ended” being converted to “end”, since the meaning should be the same. The two main methods of accomplishing this are either stemming or lemmatization.

Stemming:

Stemming involves removing the word ending based on a defined ruleset based on the “morphological structure of the language” (Elia).

The most widely used stemmer is the Porter Stemming Algorithm. It contains a series of five steps, each containing a set of predefined rules for replacing suffixes. These rules are written in notation (*see Appendix A)*, with c denoting a single consonant, and C denoting one or more consecutive consonants. Similarly, v denotes a single vowel and V denotes one or more consecutive vowels. Furthermore, if a set of vowels and consonant are next to each other, this is called a measure (m), such that m = VC. Thus, any word can be written as: $[C](VC)^m[V]$, with the square brackets indicating potential but not essential presence (Mallawaarachchi). An example of a stemmer output would be converting “populated” into “popul”. While “popul” is not a valid word in the English dictionary, and while the output is not easily readable by humans, it is perfect for many machine tasks that don’t rely on existing language conventions.

Naïve Algorithm:

Vectorizing Data:

The first extracted feature for the naïve algorithm is the title match factor, since the heading of the wiki typically corresponds to the contents of the text. As demonstrated in the previous section, the word tokens in the search phrases are often niche and technical; thus, using a predefined dictionary to find synonyms and thus similar between the tokens is impractical. Therefore, this factor is calculated by simply finding the percentage of normalized title tokens that directly matches to the normalized search phrase (all text is only analyzed after pre-processing).

However, for further increasing the certainty of a match, factoring in the content is necessary. A method for extracting features out of a corpus is to find the number of token occurrences, normalized to account for total word count. Since having a single occurrence of a token in a corpus of two words has drastically differing implications than in a corpus of two thousand, those occurrences are divided by a curved word count to compensate, with the resulting value being the token word count factor.

For phrases consisting of multiple tokens, their proximity matters in determining accurate presence of the meaning. For example, if the search phrase is “Gitlab Kubernetes” and the sentence “Using the Kubernetes-Gitlab integration” is present in the corpus, a higher score should be given since an almost exact match of the phrase occurs. If the corpus has the sentence “Create Gitlab clusters and run workloads on Kubernetes Knative”, a penalty should be applied as the tokens are used in separate and unrelated phrases.

Classification:

Once features are extracted as numbers, some weights and thresholds must be applied to convert the numerical value to a Boolean match. Proximity is only a factor if the number of tokens is greater than one, and :

$$tokenWordCountFactor \ *= \frac{1}{numTokens}$$

$$tokenProximityFactor \ *= \ 1 - \left(\frac{1}{numTokens}\right)$$

This final score is then calculated as follows:

$$finalScore \ = \ ((tokenProximityFactor \ + \ tokenWordCountFactor) * (1 - titleFactorWeight) \ + \ titleMatchFactor * titleFactorWeight) \ * \ 100$$

After some manual experimentation with edge cases, the match threshold (minimum final score for link to be considered a match) was set at fifteen percent. The title weight factor was set at forty percent, such that just slightly more than a third of a direct title match was required for a match to be true.

Machine Learning Algorithm:

Feature Selection:

In natural language processing, the three most common methods of extracting features from a corpus are: Bag of Words, TF-IDF (Term Frequency – Inverse Document Frequency), and word embeddings (Waykole and Anuradha 352).

TF-IDF calculates term frequency (TF) and the inverse document frequency (IDF) as follows:

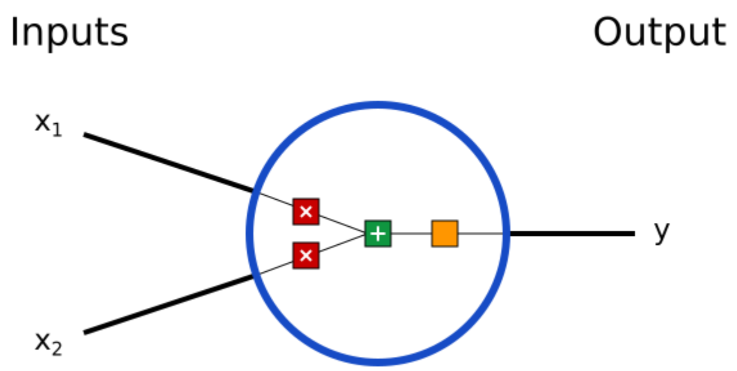
$$IDF = \ln\left(\frac{\text{amounts of documents}}{amount\ of\ document\ containing\ token}\right)$$

Since more frequent and thus less meaningful words would occur throughout the document set, this factor would provide a penalty and thus reduce their TF-IDF score. However, rarer words are more likely to be specific to certain documents, and thus are more vital to determining the meaning of that document. Those words would be given a boost through the IDF score, thus increasing their ranking even if their frequency is lower than the more common terms.

Finally, the most sophisticated technique is word embeddings. The goal of word embeddings is to "help capture semantic, syntactic context" (Gu et al.) and to "help understand how similar/dissimilar it to other terms" (Gu et al). Essentially, it allows for mapping of text into multi-dimensional vectors, while preserving relationships to other words, resulting in similar tokens being mapped closer to each other. One of the most "well-known techniques" (Vo) for word embeddings is known as Word2Vec, and relies on shallow neural networks.

Neural Networks:

To start, a brief explanation of neural networks is in order. As the name implies, neural networks consist of individual neurons. Each neuron takes in some inputs, applies weights, adds those weighted inputs with a bias (for shifting the output function to better match a desired value), and applies an activation function, usually a sigmoid (Zhou). The purpose of this activation function is to normalize the data into a range from zero to one for consistency. The composition of the neuron can be visualized in fig 3.



(Fig 3. Artificial Neuron, Zhou)

With these equations modelling the described changes:

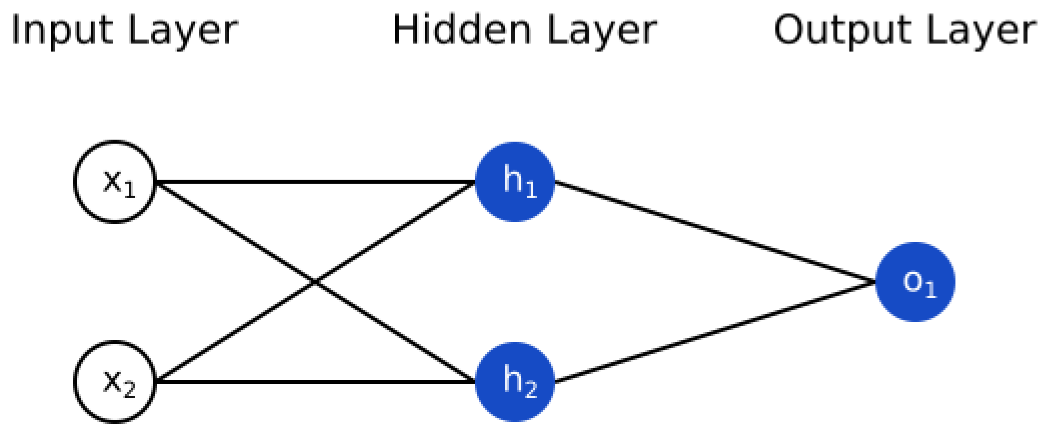
$$x_{1f} = x_{1i} * w_1$$

$$x_{2f} = x_{2i} * w_2$$

$$sum = x_{1f} + x_{2f} + b$$

$$output = f(sum)$$

The network then consists of these neurons arranged in an input layer, n number of hidden layers, and an output layer as seen in fig 4.

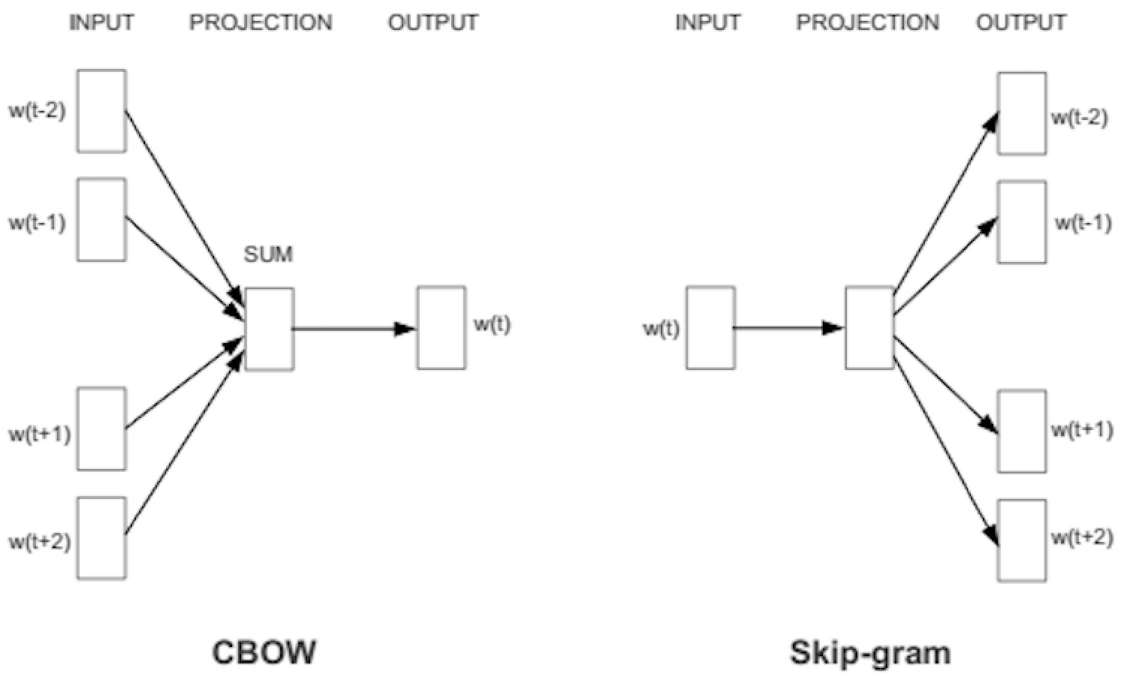


(Fig 4. Neural Network, Zhou)

For training the network, a mean squared error is calculated, known as the loss, with the goal to minimize this as to reduce the overall error. A stochastic gradient descent algorithm is employed to provide information on how to accordingly adjust the weights and biases to minimize the loss. Training data is fed from a sample dataset (with known correct outputs), and after multiple repetitions, the weights and biases would be adjusted sufficiently to provide the correct outputs.

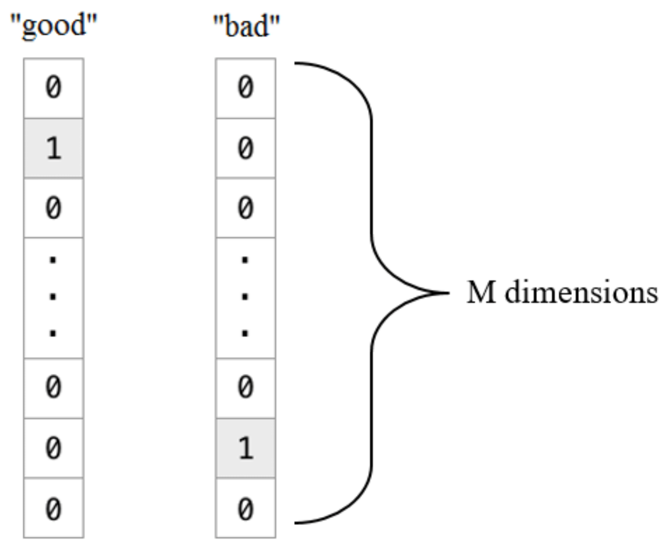
Word2Vec:

Word2Vec is a neural network with a single hidden layer with input and output layers corresponding to word tokens. It works primarily with the assumption that words that occur closer together frequently are likely to have similar meanings. There are two main architectures utilized by Word2Vec, which are the Continuous Bag of Words (CBOW) and Skip-Gram models. The CBOW model is structured so that the output is a prediction of a central word based on the inputted surroundings in an n-sized window. The Skip-Gram model is structured in reverse, with an input being the central token, and the model working to predict the surrounding words within the window. Thus, the Skip-Gram model is able to better represent rarer words in a corpus, though is slower to train and has slightly worse accuracy for the more frequent words than CBOW (Kulshrestha). The representation of these architectures can be seen in fig 5.



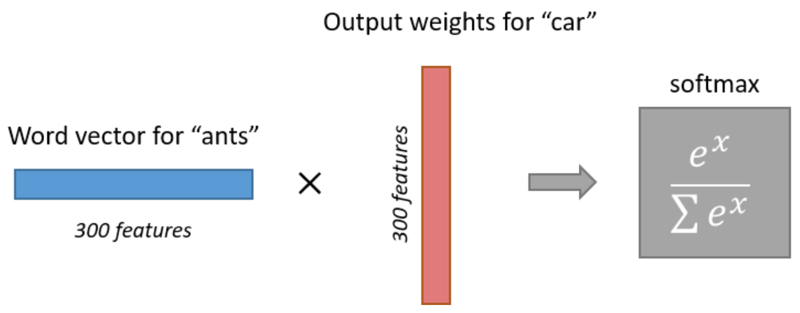
(Fig 5. New Model Architectures, Mikolov et al.)

As numerical values must be passed into the input rather than a String, one-hot encoding is utilized, which contains a vector with a size equal to the total amount of tokens. Each index corresponds to a separate word present in the dataset; therefore, the given input into the neural network is a vector filled with zeros, except for the index that matches the indicated token, which appears as a one. This can be visualized in fig 6.



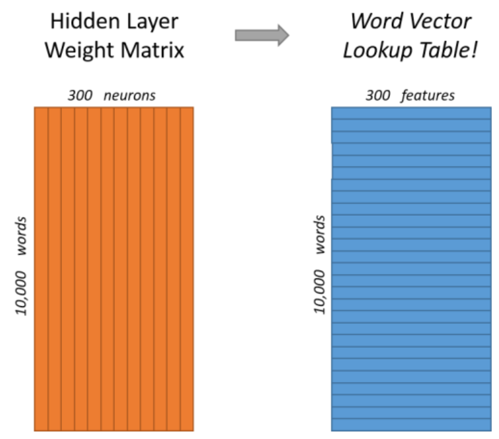
(Fig 6. One-Hot Vector Diagram, Vo)

The output layer goes through a Softmax Activation Function, the result of which provides a probability estimate for a token within the window to be equal to the target word. The entire network can be visualized in fig 7.



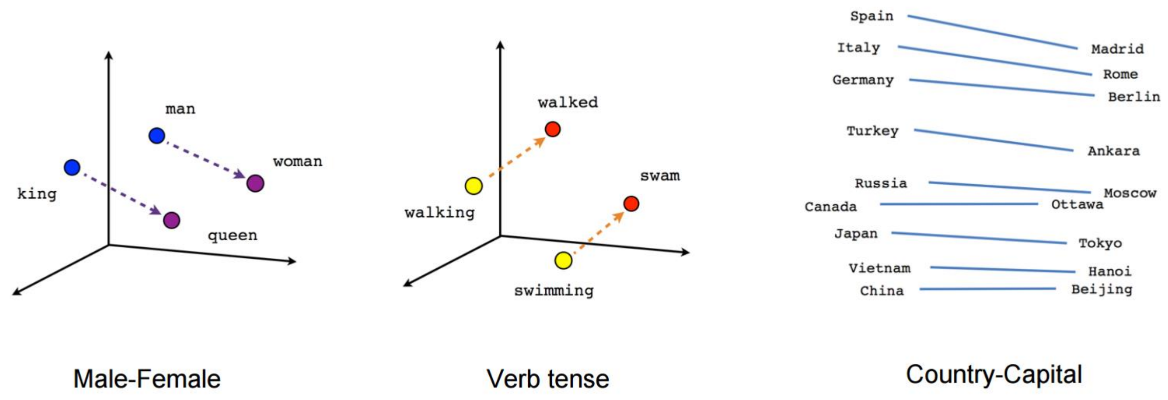
(Fig 7. High Level Architecture Illustration, Gilyadov)

However, the final goal of Word2Vec is to output vectors of the word embeddings, as previously discussed. Due to the design of these models, the weights of the hidden layer in the network simply correspond to the vectors, since these weights have been adjusted towards prediction of similar words. This can be visualized in fig 8.



(Fig 8. Hidden Layer Matrix, Gilyadov)

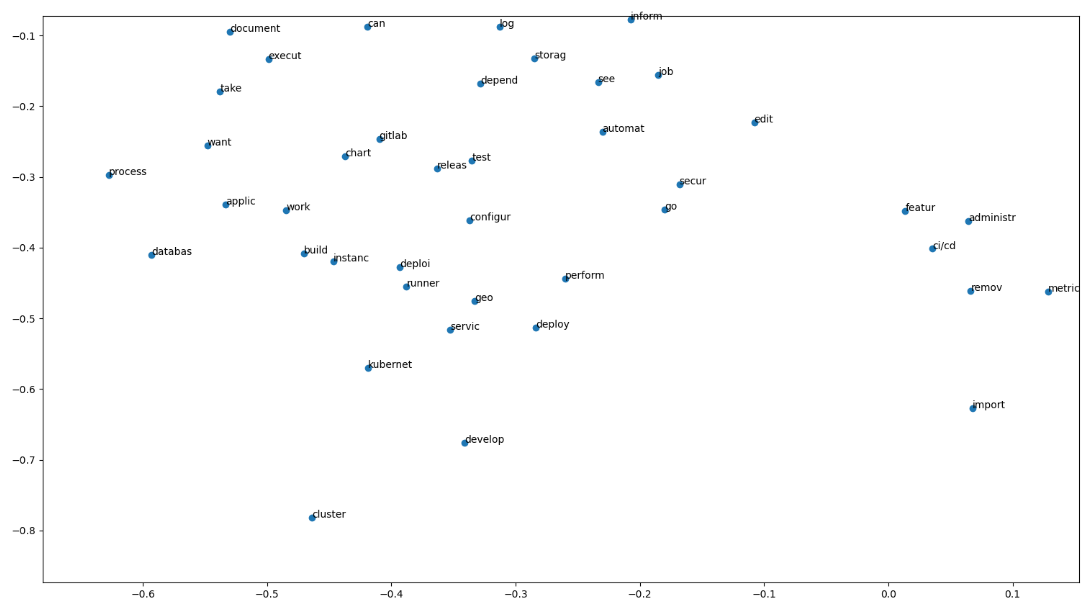
Once these vectors are extracted and assigned to the string interpretation of the corresponding token, various relations can be interpreted through their positioning. For visualization purposes, the number of vector dimension into two or three, though this does result is a slight loss of accuracy. On a sample dataset, some common relations can be seen in fig 9.



(Fig 9. Word Relationships, Liebman)

Notably, the vector positioning allow for prediction of semantic word similarity, along with capturing relationships between them. Algebraic operations can be performed on these vectors to find certain words based on known relations, such as the equation $king - man + women$ returning a vector closest to queen (Mikolov et al). This thus demonstrates the capturing of concepts such as gender and royalty, and is an example of the interesting sophistication that this algorithm achieves.

After training this model on Python, using every sentence present in the Gitlab Wiki using a crawler built for this investigation on Java, the following visualization can be created for a portion of the token stems present. Some notable similar tokens include “build” and “instanc”, “deploi” and “runner”, which are two frequently used pairs. Furthermore, the cluster of tokens surrounding “ci/cd” accordingly relate more to project management tasks, rather than the larger cluster’s focus on more development oriented themes, demonstrating correct interpretation of the present word set (fig 10).



(Fig 10.)

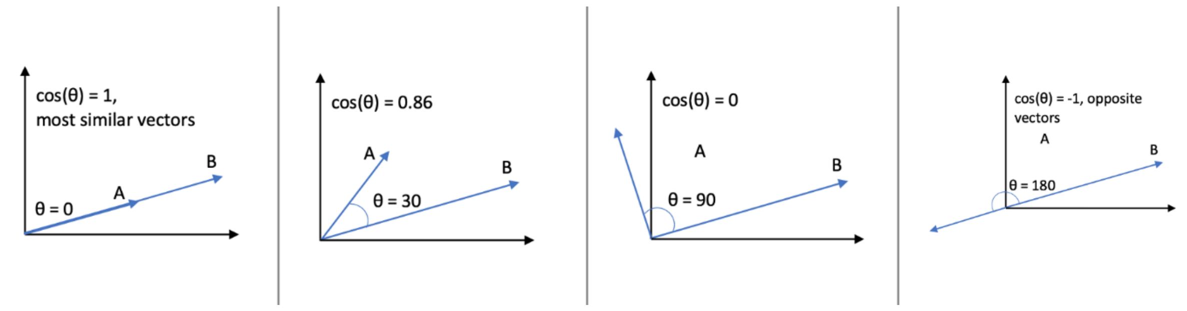
Application to this Investigation:

For this investigation, a useful metric to determine is the similarity of these vectors. There are two traditional methods for calculating similarities: Euclidean distance and cosine similarity. Euclidean distance is simply an extension of the distance formula from two dimensional geometry that allows for it to be applied in an extended number of dimensions. This finds the linear distance between two points, with the formula as follows:

$$distance(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

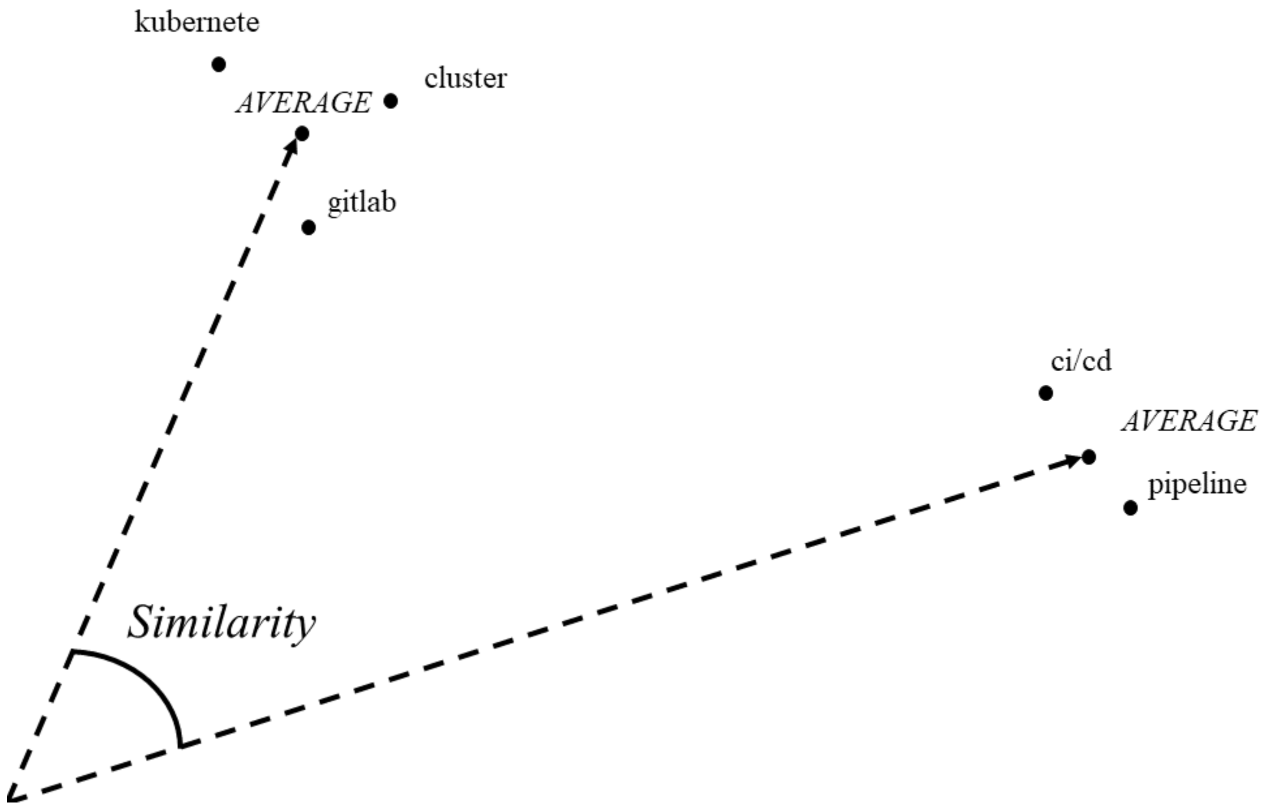
However, at the higher dimensions of the word vectors, “pairwise distance between all of your points approaches a constant” (Victoroff) thus rendering Euclidean distance ineffective. Therefore, cosine similarity is used instead, which focuses on finding the angular difference between vectors, such as that the smaller the angle between them, the higher their similarity. The equation (for points A and B in n dimensions) and visualization for this can be seen in fig 11.

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$



(Fig 11. Cosine Similarity Examples, Varun)

Furthermore, since multiple word tokens are used within the search phrase, the average of those vectors needs to be taken to find the average value of the phrase meaning. The same process will be applied to the title, to check for web page is correctness, and content, in case of a misleading title or specific content reference. Then, similarity will be found between the search phrase vector and those formerly stated, and added as features. This can be visualized in fig 12.

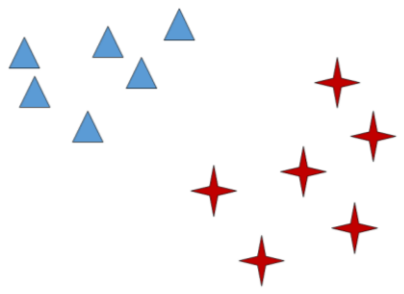


(Fig 12.)

Finally, TD-IDF will also be used as a feature, to ensure that literal matches of the word tokens also occur within the page contents. For instance, the Word2Vec model finds words such as Linux and Windows to be similar, yet in a technical context, there is a crucial difference between those terms and the content that those pages would hold.

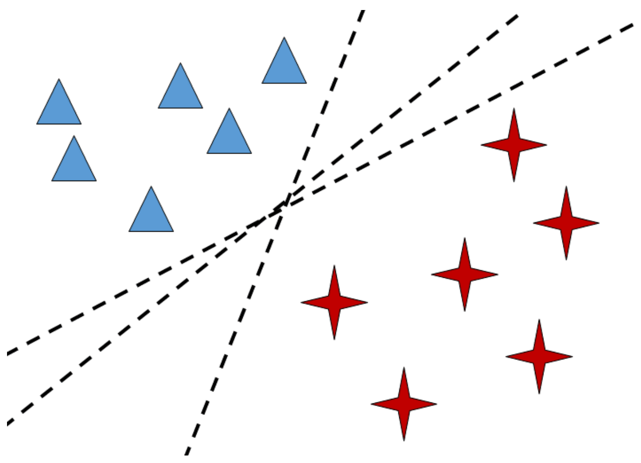
Classification using SVM:

The last step of natural language processing is to create a classifier. For this investigation, the SVM algorithm will be used, since it is able to provide “significant accuracy with less computation power" (Saxena). The main goal of an SVM is to build a linear function to split up data points of differing classes as accurately as possible. Suppose there is a dataset as in fig 13.



(Fig 13.)

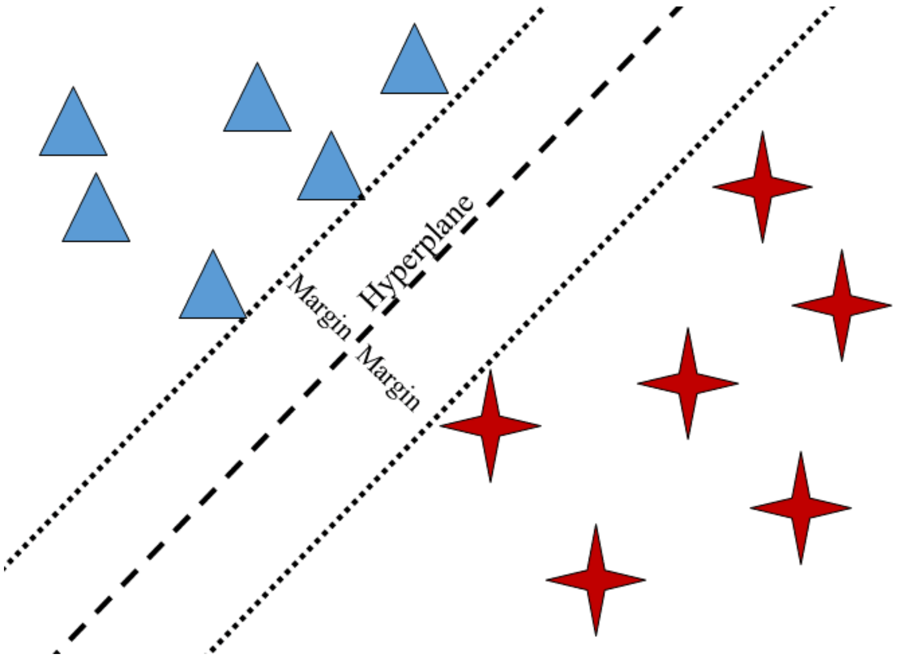
The function line outputted by the SVM is called the hyperplane or decision boundary. However, there are many differing potential lines that can be drawn and yet still classify all points on this training data correctly. For example, all hyperplanes in fig 14 are valid.



(Fig 14.)

Thus, SVM attempts to determine the best possible hyperplane through maximizing the margin, or the minimum distance between the closet points (support vectors) and the hyperplane

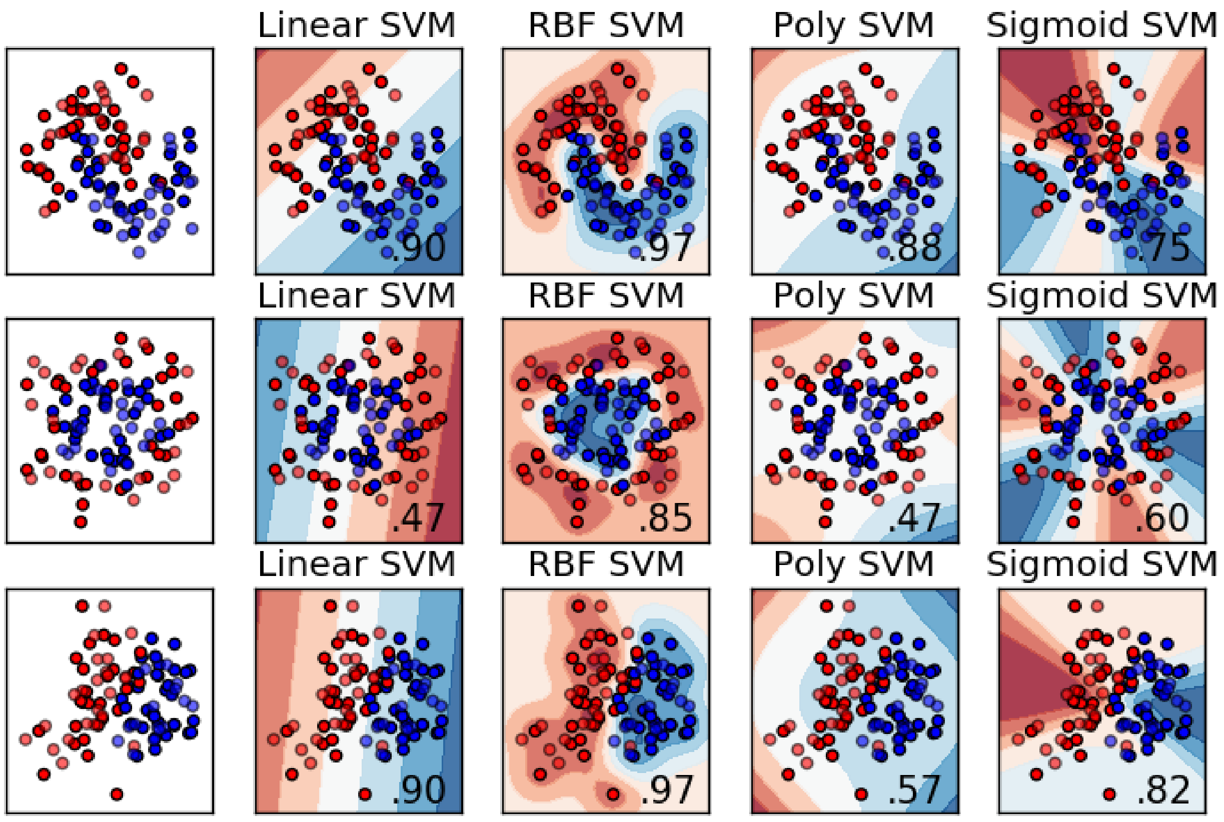
(Dwivdedi). Therefore, the best hyperplane and its margins for this example can be visualized in fig 15.



(Fig 15.)

Furthermore, soft margin allows for data points to be misclassified in exchange for margin maximization, which is more useful in cluttered real-world cases than in this example. This can be altered with the degree of tolerance, which is denoted by the term “C”. Increasing the degree of tolerance increases the soft margin and decreases misclassification penalty, and vice versa.

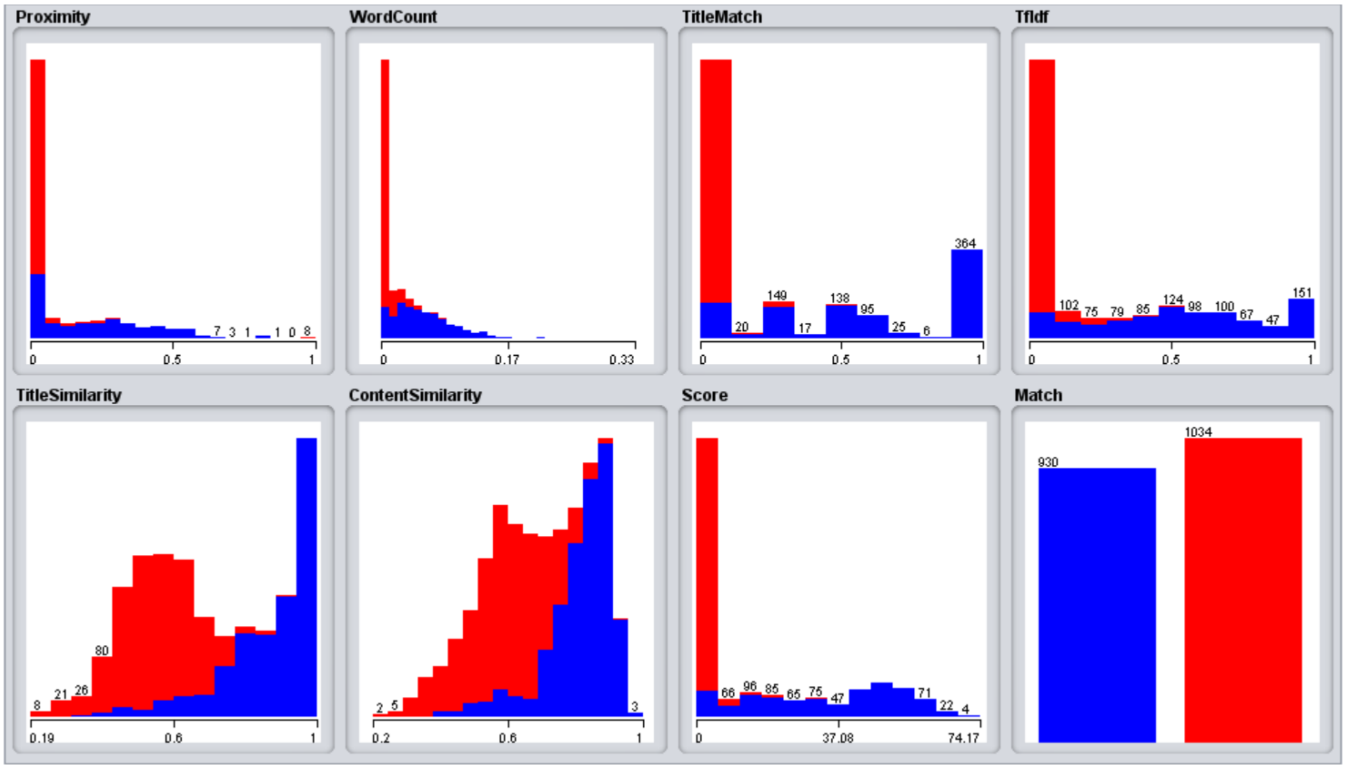
Finally, in some cases the data cannot be separated in two dimensional space with just a linear function. Thus, SVM employs kernels, using Cover’s theorem that such data can be “transformed into a linearly separable training set by projecting it into a higher-dimensional space via some non-linear transformation” (Dwivedi). These Kernel functions refer to the method of calculating the dot product of the vector matrices to essentially create a non-linear function when the data is projected back into two dimensional space. The main Kernel functions in SVM are linear, polynomial, radial basis function, and sigmoid, with each utilizing the function correlating to their name. The differences in classification between the kernels can be visualized in fig 16.



(Fig 16. Comparisons of SVM Kernel Performance, Dwivedi)

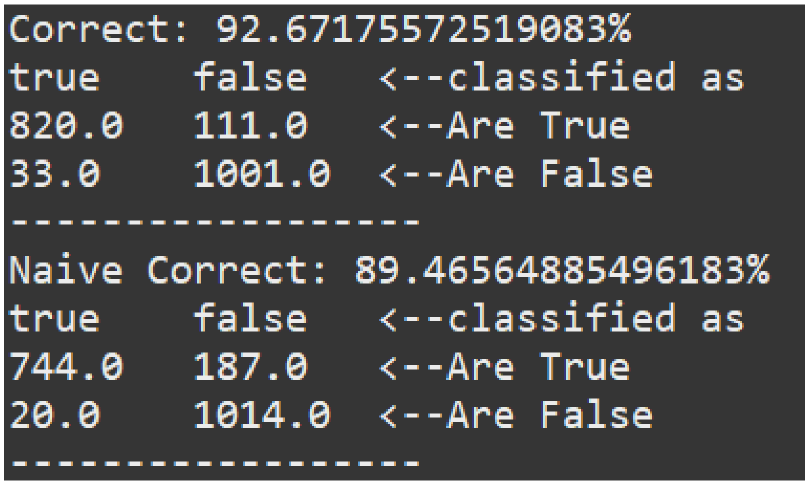
Experimentation:

On the training set, which contained both manual and random links and the largest amount of data samples, the feature value distribution by match can be seen in fig 18, with blue representing true and red being false. The score feature is the final score value of the naïve algorithm. It can be seen that the spread of negative match values is very limited in the naïve features, while the machine learning features (excluding Tf-Idf) contain more even spreads, with slightly closer clustering for the true matches.



(Fig 18. Feature Distributions for both Naïve and Machine Learning by Match in Training)

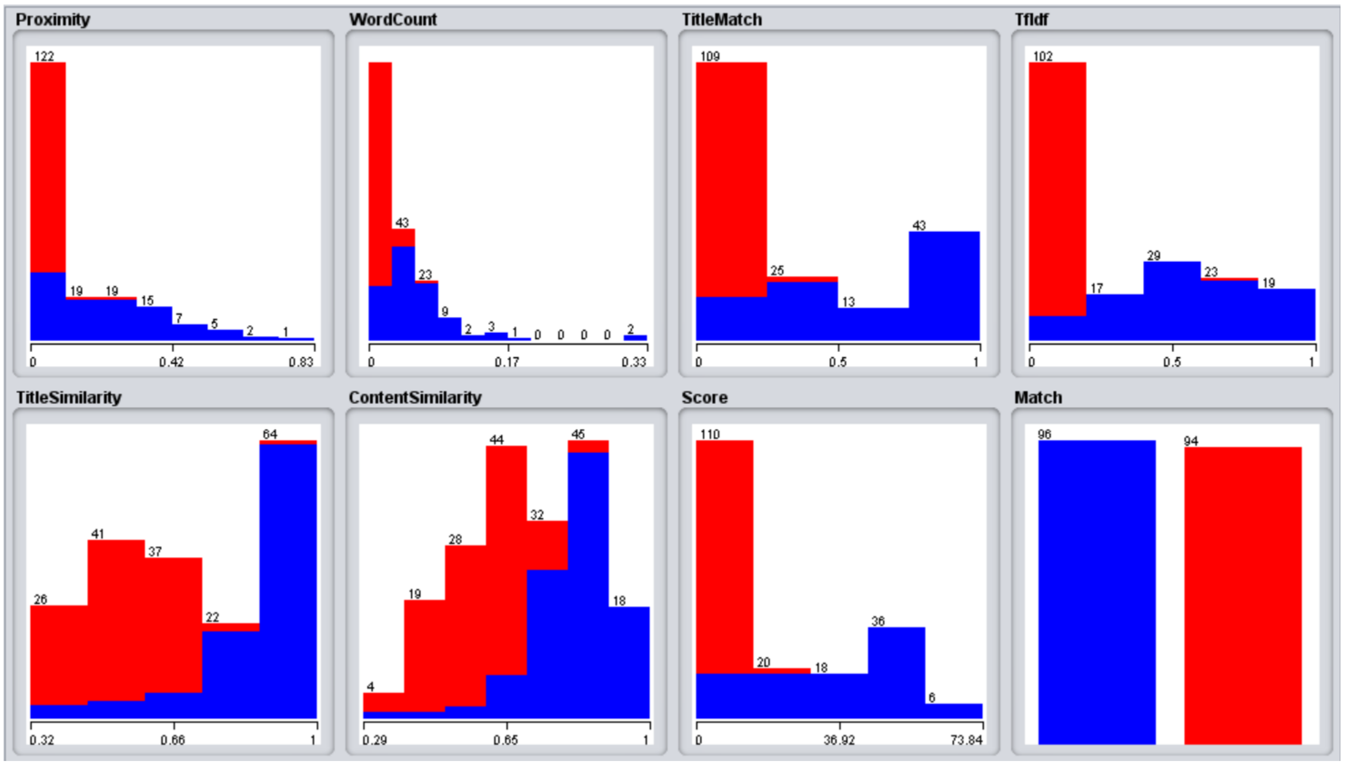
In fig 19, the rate of false positives and false negatives are shown, with the naïve algorithm having slightly superior categorization of false matches, but being outperformed by the machine learning algorithm at differentiation of true matches. However, both algorithms did perform significantly worse at correctly categorizing the true values then negative values, with higher false positive rates than false negative. Furthermore, the machine learning algorithm had a higher overall accuracy by about 3.21%.



```
Correct: 92.67175572519083%
true    false   <--classified as
820.0   111.0   <--Are True
33.0    1001.0  <--Are False
------------------
Naive Correct: 89.46564885496183%
true    false   <--classified as
744.0   187.0   <--Are True
20.0    1014.0  <--Are False
------------------
```

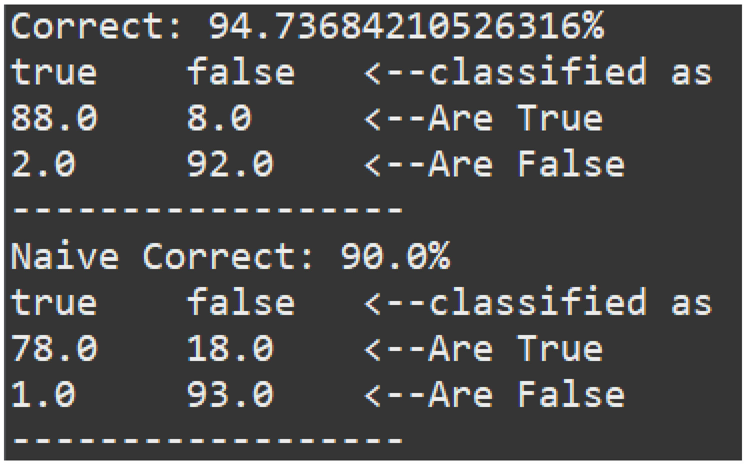
(Fig 19. Confusion Matrices for both Naïve and Machine learning Algorithms in Training)

In the parsed testing data set (without edge cases), a similar feature distribution is present yet with reduced outliers in the data, which can be seen in fig 20. This is expected, as reducing edge cases and the total amount of data should lead to cleaner data with a lower outlier chance.



(Fig 20. Feature Distributions for both Naïve and Machine Learning by Match in Parsed Test)

The confusion matrix for this test dataset can be seen in fig 21, and contains a similar distribution of false positives and negatives. However, the machine learning algorithm performed by about 2.07% better compared to the naïve algorithm's improvement of 0.535%, suggesting that the outliers cause higher errors within the machine learning algorithm. Yet, the increase also suggests a lack of overfitting within the training set, which is a positive indicator.

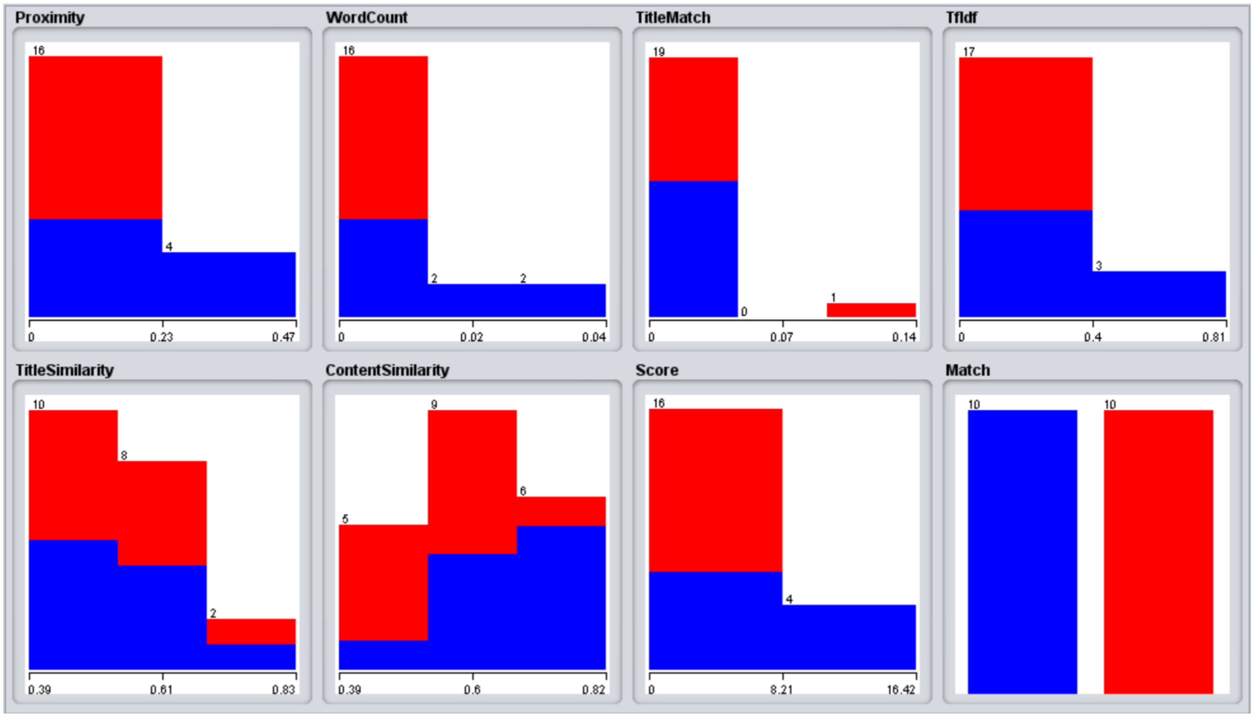


```
Correct: 94.73684210526316%
true    false   <--classified as
88.0    8.0     <--Are True
2.0     92.0    <--Are False
------------------
Naive Correct: 90.0%
true    false   <--classified as
78.0    18.0    <--Are True
1.0     93.0    <--Are False
------------------
```

(Fig 21. Confusion Matrices for both Naïve and Machine learning Algorithms in Parsed Test)

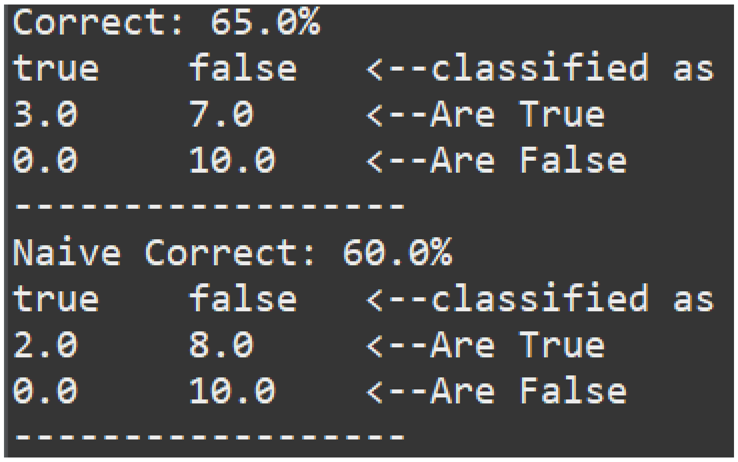
Upon additional investigation of individual cases of false positive outputs, most of the links with exceptionally low scores usually required additional context, with hyperlinked phrases such as “additional documentation” not providing much information about the page. However, the cases with medium titleSimilarity scores usually were marked incorrect due to a lack of match with the heading, despite their presence in the content. Lower titleMatch scores usually occurred due to a presence of synonyms and not exact matches. While contentSimilarity scores were mostly accurate, lower proximity and wordCount scores occurred when a broader topic was referenced, with few keyword matches outside of the title. Thus, the outlier test datasets focus on additional analysis of search phrases unrelated to the title, search phrases synonymous to but not directly matching the title, and search phrases related to but not directly present in the content. Each set also included the same ten false entries as a constant for comparison.

In the unrelated title dataset, containing direct phrases from the content that do not match the title, the distribution of feature scores can be seen in fig 22. As expected, the title scores were not a significant differentiation, and only the content-focused features could produce some separation.



(Fig 22. Feature Distributions by Match in Unrelated Title Test)

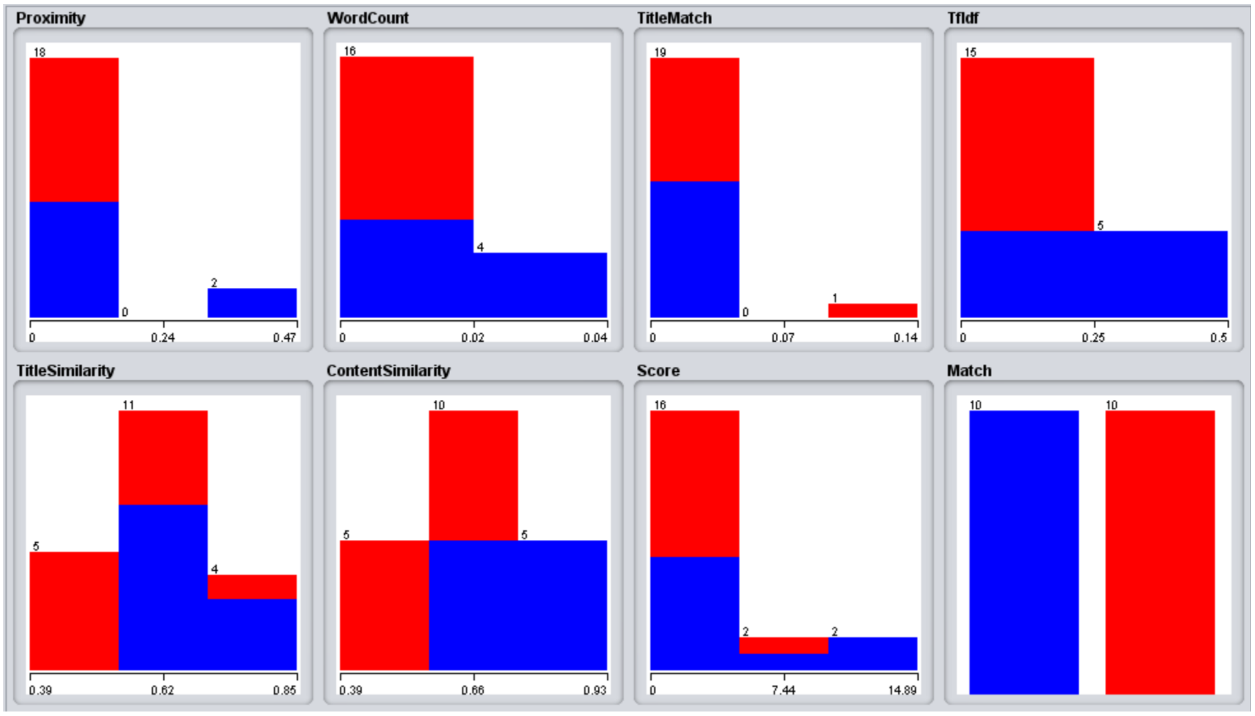
However, the confusion matrix seen in fig 23 shows that the machine learning algorithm outperformed the naïve algorithm by five percent for the true entries, despite clearer separations being present in the proximity and word count features.



```
Correct: 65.0%
true    false   <--classified as
3.0     7.0     <--Are True
0.0     10.0    <--Are False
------------------
Naive Correct: 60.0%
true    false   <--classified as
2.0     8.0     <--Are True
0.0     10.0    <--Are False
------------------
```

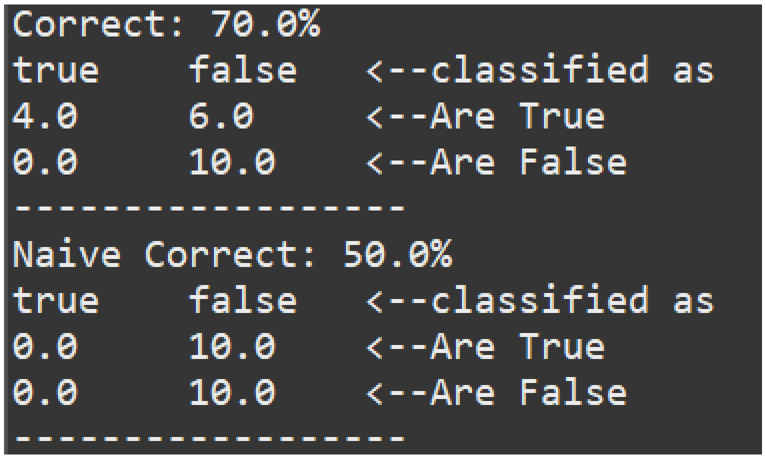
(Fig 23. Confusion Matrices in Unrelated Title Test)

In the synonymous title dataset, containing phrases composed of keywords equaling the title's meaning, taken from child and parent headings, the feature distribution can be seen in fig 24. Both content and title similarity features were higher for the true entries, while the td-idf and naïve features were slightly reduced compared to the previous test.



(Fig 24. Feature Distributions by Match in Synonymous Title Test)

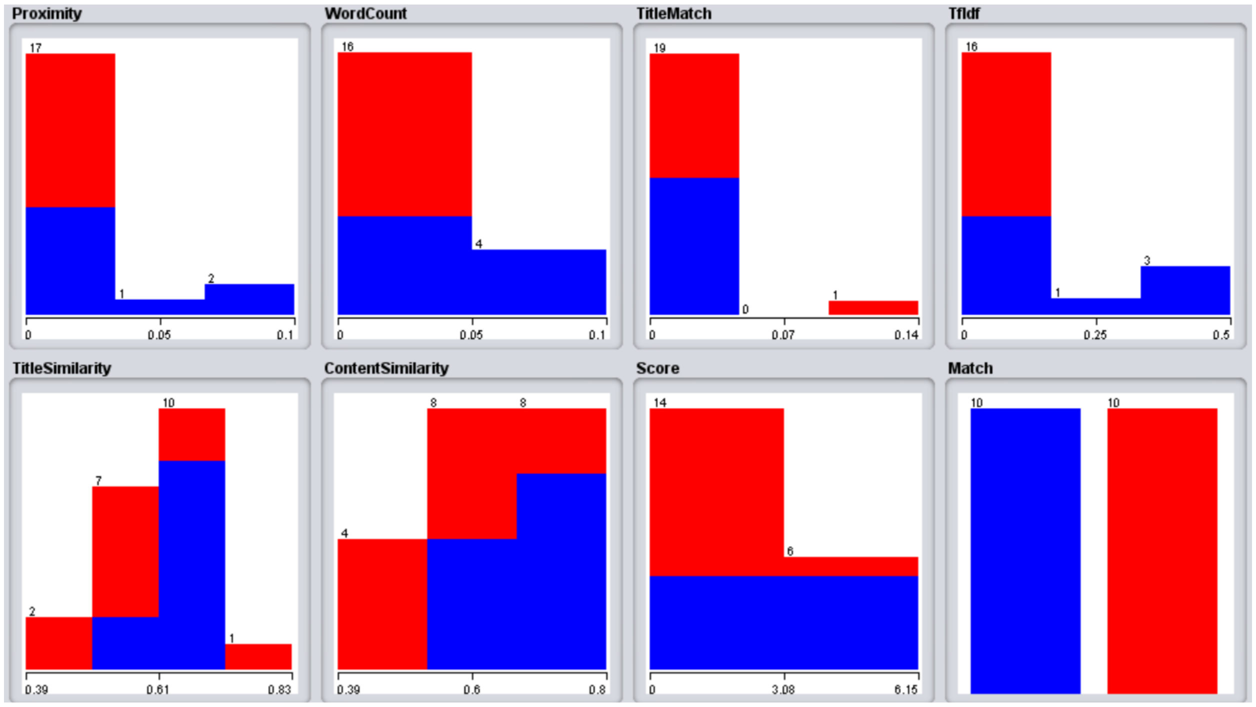
The confusion matrices seen in fig 25 confirm the struggles of the naïve algorithm in this test, with the machine learning algorithm outperforming it by forty percent for the true entries. Notably, the naïve algorithm was also unable to correctly classify any of those entries.



```
Correct: 70.0%
true     false    <--classified as
4.0      6.0      <--Are True
0.0      10.0     <--Are False
------------------
Naive Correct: 50.0%
true     false    <--classified as
0.0      10.0     <--Are True
0.0      10.0     <--Are False
------------------
```

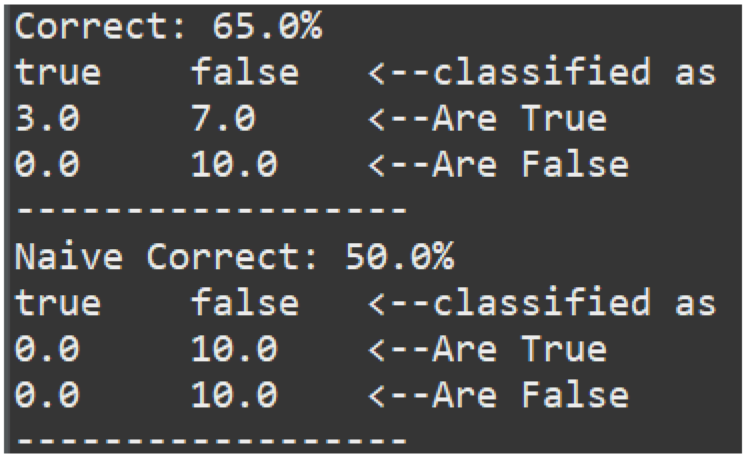
(Fig 25. Confusion Matrices in Synonymous Title Test)

Finally, for the Meta topic dataset, containing descriptive phrases about the document yet without direct matches such as “Package Client Example” for the “Debian API” page, the feature distributions can be seen in fig 26. While proximity scores are significantly lower than both other tests, the word-count scores have doubled. Title similarity and content similarity scores have slightly condensed, and all other features have remained similar.



(Fig 26. Feature Distributions by Match in Meta Topic Test)

However, the naïve algorithm has not been able to successfully classify any entries, seen in fig 27, compared to the machine learning algorithm’s thirty percent score.



```
Correct: 65.0%
true    false   <--classified as
3.0     7.0     <--Are True
0.0     10.0    <--Are False
------------------
Naive Correct: 50.0%
true    false   <--classified as
0.0     10.0    <--Are True
0.0     10.0    <--Are False
------------------
```

(Fig 27. Confusion Matrices in Meta Topic Test)

Analysis:

In every test, the combination of features and a classifier generated by machine-learning allowed for that algorithm to surpass the naïve one in every test. This is due to the semantic similarity approach allowed by the Word2Vec reliant features, providing a comprehensive and more flexible analysis of text through similarity. This allowed the machine learning approach to outperform the naïve one by significant margins in the Title Synonym and Meta Topic tests. While adding a synonym generator through existing dictionaries in to the title match factor might have improved its performance slightly in the former test, it still would likely have been insufficient. A proper understanding of relations between technical terms and word structure specific to this Wiki would have been required to achieve sufficient accuracy, which is difficult without employing a solution similar to Word2Vec.

Notably, this can be seen with the increase of both title and content similarity scores in the Synonymous Title set from the Unrelated Title set, demonstrating the effectiveness of Word2Vec's mappings. Not only were the search phrase tokens identified as title synonyms, but their meaning was found to have a closer match to the overall content of the document that a niche direct excerpt. This shows the success of this approach regarding extracting complex meaning into a computer-comprehensible form. Furthermore, the Meta Topic test would have been difficult for the naïve algorithm to accomplish without being able to extract word definitions, a process that is more effectively done through machine learning.

However, while the strongest performance of the naïve algorithm was in the Unrelated Title test, due to the requirements of finding a direct match, suited to the naïve approach, it was still outperformed. While it can be seen that the naïve features did successfully separate about four of the true entries, only two were correctly classified. This can be attributed to the difficulties with building a classifier from manual thresholds due to the likelihood of overfitting to the training set, since the values are adjusted towards gaining maximum accuracy without considering decision plane margin to the degree of SVM.

Some flaws were present with this investigation, notably the subjective nature of link correctness, the perception of which can vary based on particular knowledge of content or intention. There are few rules with how wiki links should be formatted, leading to numerous cases where insufficient context or non-direct relations are present. Also, this Wiki and the links within were assumed to be correctly formatted, which could not have been the case in some instances.

Therefore, it can be deduced that the Word2Vec model allows for improved interpretation of word relations, thus allowing for improved accuracy in matching. The SVM classifier allows for better separation of training data and improved flexibility for new cases outside of the training set. However, the machine learning approach relies on determining rules from subjective human classification, and is more likely to conform and work around to some human errors and inaccuracies present in the wiki. The naïve approach, while still not significantly worse than machine learning in overall performance, is better suited to enforcing specific and definable rules, and struggles with subjectivity or flexibility outside of the intended use case, where complexity would increase exponentially for every edge case.

Conclusion:

Natural language processing is vital for complex search engines, and has applications in all tasks requiring analysis or prediction of text, with the three steps of preprocessing, vectorizing, and classification. For each step, there are numerous algorithms to choose from that best suit the use case, with stemming vs lemmatization for preprocessing, TD-IDF and Word2Vec for vectorization, and SVM or manual thresholds as a classifier, to name a few.

Word2Vec allows for generating and mapping vectors for each word, such that similarity and relations can be determined. The ability of word2vec to extract meaning and store it numerically was demonstrated, such that it could be mapped to numerous words for conversion into a human-interpretable form, allowing for increased accuracy of meaning and similarity.

SVM provides a classification model focused on maximizing margin between the decision boundaries, and while the hyperplane function was linear, various kernel methods allowed for non-linear modification of the dataset into a more linear output. This approach provided higher accuracy and application to data outside the training set, compared to the naïve threshold approach which cannot feasibly account for all edge cases.

However, the machine learning approach is created to and suited most towards transforming human created text and logic to machine-understandable classifications. This can lead to adoption and acceptance of human errors present within the training set, leading to subjective inaccuracies. Thus, in cases with specific rulesets for separation, a naïve approach would be more effective. Overall, machine learning exists mainly to decode the complex nature of humans, and its complexity is due to our irregularities; understanding perfect order is significantly simpler.

Works Cited:

Benner, Jonas. "Cross-Validation and HYPERPARAMETER Tuning: How to Optimise Your Machine Learning Model." *Medium*, Towards Data Science, 6 Aug. 2020, towardsdatascience.com/cross-validation-and-hyperparameter-tuning-how-to-optimise-your-machine-learning-model-13f005af9d7d.

Berasategi, Ane. "Semantic Search." *Medium*, Towards Data Science, 8 Apr. 2019, towardsdatascience.com/semantic-search-73fa1177548f.

Biswas, Eeshita, et al. "Exploring Word Embedding Techniques to Improve Sentiment Analysis of Software Engineering Texts." *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, May 2019, doi:10.1109/msr.2019.00020.

Brownlee, Jason. "A Gentle Introduction To k-Fold Cross-Validation." *Machine Learning Mastery*, 2 Aug. 2020, machinelearningmastery.com/k-fold-cross-validation/.

Brownlee, Jason. "What Is the Difference between Test and Validation Datasets?" *Machine Learning Mastery*, 14 Aug. 2020, machinelearningmastery.com/difference-test-validation-datasets/.

Ciaburro, Giuseppe. "Regression Analysis with R." *O'Reilly Online Learning*, Packt Publishing, www.oreilly.com/library/view/regression-analysis-with/9781788627306/6bb0d820-6200-4bfe-aa91-e7b7ffa2a9c1.xhtml.

Duque, Tiago. "How to Build a Lemmatizer." *Medium*, Analytics Vidhya, 1 Sept. 2020, medium.com/analytics-vidhya/how-to-build-a-lemmatizer-7aeff7a1208c.

Dwivedi, Rohit. “How Does Support Vector Machine (Svm) Algorithm Works in Machine Learning?” *Analytics Steps*, 3 May 2020, www.analyticssteps.com/blogs/how-does-support-vector-machine-algorithm-works-machine-learning.

Elia, Francesco. “Stemming vs Lemmatization.” *Baeldung on Computer Science*, 24 June 2020, www.baeldung.com/cs/stemming-vs-lemmatization.

Gilyadov, Julian. “Word2Vec Explained.” *Hacker's Blog - Get a Beer and Join Me down This Geeky Rabbit Hole Adventure.*, 23 Mar. 2017, israelg99.github.io/2017-03-23-Word2Vec-Explained/.

“GitLab Docs.” *GitLab Documentation*, Gitlab, docs.gitlab.com/ee/.

Gu, Fangyu, et al. “Nlp: Word Embedding Techniques for Text Analysis.” *Medium*, SFU Professional Master's Program in Computer Science, 4 Feb. 2020, medium.com/sfu-cspmp/nlp-word-embedding-techniques-for-text-analysis-ec4e91bb886f.

IBM Cloud Education. “Natural Language Processing (NLP).” *IBM*, 2 July 2020, www.ibm.com/cloud/learn/natural-language-processing.

Joshi, Atharva, et al. “Figure 1 from Modified Porter Stemming Algorithm: Semantic Scholar.” *Figure 1 from Modified Porter Stemming Algorithm | Semantic Scholar*, 1 Jan. 1970, www.semanticscholar.org/paper/Modified-Porter-Stemming-Algorithm-Joshi-Thomas/0b64bc14783269e815dde120c79097a17210b8f9/figure/0.

Kuhn, Max, and Kjell Johnson. Applied Predictive Modeling. Springer, 2016.

Kulshrestha, Ria. “NLP 101: Word2Vec - Skip-Gram and CBOW.” *Medium*, Towards Data Science, 26 Oct. 2020, towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314.

Liebman, Sam. “Mapping Word Embeddings with word2vec.” *Medium*, Towards Data Science, 28 Aug. 2018, towardsdatascience.com/mapping-word-embeddings-with-word2vec-99a799dc9695.

Madan, Rohit. “TF-IDF/Term Frequency TECHNIQUE: Easiest Explanation for Text Classification in NLP with Python.” *Medium*, Analytics Vidhya, 27 Nov. 2019, medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3.

Mallawaarachchi, Vijini. “Porter Stemming Algorithm – Basic Intro.” *Vijini Mallawaarachchi*, 10 May 2017, vijinimallawaarachchi.com/2017/05/09/porter-stemming-algorithm/.

Mikolov, Tomas, et al. “Efficient Estimation of Word Representations in Vector Space.” *ArXiv.org*, 7 Sept. 2013, arxiv.org/abs/1301.3781v3.

Mujtaba, Hussain. “Pos (Part-of-Speech) Tagging with Hidden Markov Model.” *Part of Speech (POS) Tagging with Hidden Markov Model*, My Great Learning, 20 May 2021, www.mygreatlearning.com/blog/pos-tagging/.

Prince, Mudda. “Categorizing and POS Tagging with NLTK Python.” *Medium*, Medium, 25 Sept. 2019, medium.com/@muddaprince456/categorizing-and-pos-tagging-with-nltk-python-28f2bc9312c3.

Raghavan, Prabhakar. “How AI Is Powering a More Helpful Google.” *Google*, Google, 15 Oct. 2020, blog.google/products/search/search-on/.

Saxena, Shipra. “SVM: Support Vector Machine: How Does SVM Work.” Analytics Vidhya, 12 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/.

Shetty, Badreesh. “Natural Language Processing(NLP) for Machine Learning.” *Medium*, Towards Data Science, 24 Nov. 2018, towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b.

Varun. “Cosine Similarity: How Does It Measure the Similarity, Maths behind and Usage in Python.” *Medium*, Towards Data Science, 27 Sept. 2020, towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db.

Victoroff, Slater. “Is Euclidean Distance Meaningful for High Dimensional Data?” *Indico*, 10 July 2018, indico.io/blog/is-euclidean-distance-meaningful-for-high-dimensional-data/.

Vo, Khuong. “Figure 1.: Representing Words BY One-Hot Vector.” *ResearchGate*, 24 Apr. 2020, www.researchgate.net/figure/Representing-words-by-one-hot-vector_fig1_328161921.

Waykole, Resham N., and Anuradha D. Thakare. “A REVIEW OF FEATURE EXTRACTION METHODS FOR TEXT CLASSIFICATION.” *International Journal of Advance*

Engineering and Research Development, vol. 5, no. 04, Apr. 2018, pp. 351–354., doi:10.21090/ijaerd.

Zhou, Victor. “Machine Learning for Beginners: An Introduction to Neural Networks.” *Medium*, Towards Data Science, 20 Dec. 2019, towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9.

Works Consulted:

bib

Appendix:

Appendix A: List of rules in notation for the Porter Stemming Algorithm

Step 1a

SSES	→	SS
IES	→	I
SS	→	SS
S	→	

Step 1b

(m>0) EED	→	EE
(*v*) ED	→	
(*v*) ING	→	

If the second or third of the rules in Step 1b is successful, the following is performed.

AT	→	ATE
BL	→	BLE
IZ	→	IZE
(*d and not (*L or *S or *Z))	→	single letter
(m=1 and *o)	→	E

Step 1c

(*v*) Y	→	I

Step 2

(m>0) ATIONAL	→	ATE
(m>0) TIONAL	→	TION
(m>0) ENCI	→	ENCE
(m>0) ANCI	→	ANCE
(m>0) IZER	→	IZE
(m>0) ABLI	→	ABLE
(m>0) ALLI	→	AL
(m>0) ENTLI	→	ENT
(m>0) ELI	→	E
(m>0) OUSLI	→	OUS

(m>0) IZATION	→	IZE
(m>0) ATION	→	ATE
(m>0) ATOR	→	ATE
(m>0) ALISM	→	AL
(m>0) IVENESS	→	IVE
(m>0) FULNESS	→	FUL
(m>0) OUSNESS	→	OUS
(m>0) ALITI	→	AL
(m>0) IVITI	→	IVE
(m>0) BILITI	→	BLE

Step 3

(m>0) ICATE	→	IC
(m>0) ATIVE	→	
(m>0) ALIZE	→	AL
(m>0) ICITI	→	IC
(m>0) ICAL	→	IC
(m>0) FUL	→	
(m>0) NESS	→	

Step 4

(m>1) AL	→
(m>1) ANCE	→
(m>1) ENCE	→
(m>1) ER	→
(m>1) IC	→
(m>1) ABLE	→
(m>1) IBLE	→
(m>1) ANT	→
(m>1) EMENT	→
(m>1) MENT	→
(m>1) ENT	→
(m>1 and (*S or *T)) ION	→

(m>1) OU	→	
(m>1) ISM	→	
(m>1) ATE	→	
(m>1) ITI	→	
(m>1) OUS	→	
(m>1) IVE	→	
(m>1) IZE	→	

Step 5a

(m>1) E	→	
(m=1 and not *o) E	→	

Step 5b

(m > 1 and *d and *L)	→	single letter