Content-based and link-based methods for categorical webpage classification

Shushman Choudhury
shushmac@andrew.cmu.edu

Tanmay Batra
tbatra@cmu.edu

Christian Hughes
cahughes@andrew.cmu.edu

Abstract

Through this project evaluate numerous methods for categorizing webpages. We utilize both the textual content of the webpage, and data about the hyperlinks on the the webpage. We first investigate the performance of various classification methods that are only content-based - Multinomial Naive Bayes, Support Vector Machine, Decision Trees and Word2Vec embedding. We then augment some of these classifiers with information from the hyperlinks of a webpage, using a single-hop neighbour lookup. We also make a comment on using graphical models. Our results show that all of the content-only classifiers perform quite well without a great deal of parameter tuning on our test set. Furthermore, we observe a general decrease in accuracy due to using the information from hyperlinks, but there are some cases where the content-based methods are incorrect on their own, but are correct when augmented with link-based information.

1 Introduction

With the ever-increasing amount of textual content stored electronically on the internet, automatic text classification is now an essential application of machine learning. Categorizing webpages based on their topics is a specific instance of text classification. Webpage categorization can be more challenging due to the less curated nature of the content, but it has more information in the form of hyperlinks to other webpages. There are several applications, including topical analysis of the internet, organizing a World Wide Web directory, and improving user browsing experience.

In this project, we address the specific problem of categorizing webpages into classes. We consider typical text classification methods that are based only on the webpage text content and explore ways to leverage webpage link information. Our objective is to gain insights into the correct way to represent and reason about linked webpages, and the effect of this additional information on the performance of the classification system. We initially focus on establishing a baseline of content-based methods and then integrate link-based methods.

2 Related Work

Webpage classification is an important topic in machine learning given the amount of data on the internet. A typical webpage contains text, images, hyperlinks and HTML tags. With supervised learning, where the number of categories is fixed, the simplest approach is to use text present in the webpage for classification. Initial works [6, 5] focussed on encoding simple features based on TF-IDF counts and basic Multinomial Naive Bayes technique for classification. [7, 4, 8, 9, 11] increase the complexity of classifiers by introducing heuristic based improved Naive Bayes, SVM,

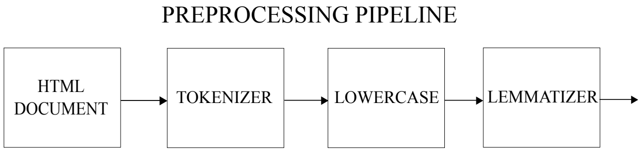


Figure 1: A schematic of the preprocessing for each webpage document

and Bayesian logistic regression respectively for webpage classification. Feature selection is as important as choosing the classifier and [10] conducts an extensive empirical study on feature metrics for text classification. Recently word2vec embedding [16] has also been used for classification of documents.

In addition, we can incorporate information from webpage hyperlinks to generate a structure among linked pages. PageRank [15] is a purely link based ranking algorithm based on a graph constructed by web pages and their hyperlinks. [13, 14] show some other algorithms which use only hyperlinks for web classification. [12] shows how hyperlinks and text from current and sibling pages improve performance. Exploiting the inherent structure among webpages can lead to a better understanding of web data.

3 Approach

Our approach proceeds in two stages. Initially, we explore techniques that utilize only the content of the webpage. Subsequently, we take into account the neighbouring webpages and examine their effect on the overall accuracy.

3.1 Content-based Methods

We have implemented four different supervised learning methods that only consider the text of the webpage: Multinomial Naive Bayes, Support Vector Machine, Decision Trees and Word2vec embedding. We subject each webpage document to a considerable amount of pre-processing to reduce noise and aid the performance of the learning algorithms.

3.1.1 Preprocessing

We use the Python Natural Language Toolkit (NLTK) [1] for pre-processing the webpage content. This lets us generate data and features that are usable by our various learning algorithms. Given a webpage HTML document, our pre-processing step extracts the vector of counts of word lemmas. In the context of Natural Language Processing, a word *lemma* or *headword* is the canonical form of a set of related words. For example, the lemma for *sit*, *sitting*, *sits* and *sat* is *sit*.

For the pre-processing pipeline, the raw text of the HTML file is read in, followed by splitting the character stream into tokens based on whitespaces and other delimiters. Tokens with anything other than alphabetic characters are removed. Every word is converted to lower case and then to its corresponding lemma according to the WordNet [2] lemmatizer. This is summarized in Figure 1.

3.1.2 Multinomial Naive Bayes

Naive Bayes classifiers are a simplistic but effective way to classify documents. The standard Naive Bayes approach assumes that each word is independent of the rest of the words in the document. Previous research has shown that the frequency with which a word occurs can be particularly important in the classification research, especially in regards to the WebKB dataset [5]. For the first classifier, we employ a multinomial version of the Naive Bayes classifier. The multinomial prior is, $P(D|\theta) = \theta_1^{\alpha_1}\theta_2^{\alpha_2}...\theta_n^{\alpha_n}$. This allows the classifier to incorporate word frequencies.

If a word does not occur in the training set, i.e. the word is unseen, it is disregarded by the classifier. The approach we used is the standard one of calculating the maximum likelihood estimate for the set of word frequencies in a document. The MLE is based on the conditional likelihood of each word appearing in a document of each class. The initial estimate $\pi_{y_i} = P(Y = y_i)$, where $i = 1, 2, 3, 4$, is based on the proportion of webpages of each category in the training set. For a given test webpage d, the class chosen is the class y_i with the highest posterior probability, where w is each unique word lemma in d and $n_{w,d}$ is the count of w in d,

$$Y(d) \leftarrow \arg\max_{y_i} \pi_{y_i} \prod_{w \in d} P[w|Y(d) = y_i]^{n_{w,d}}$$

3.1.3 Support Vector Machine

Support Vector Machines have been shown to work quite well for text classification, due to the large dimensionality of the feature space, and the sparsity of feature vectors [4]. For generating feature vectors from webpage documents, we use the popular *Bag-Of-Words* approach, where we consider the frequency of word lemmas appearing in a document, irrespective of its position. Therefore each document is represented as a $|V|$-dimensional feature vector, where $|V|$ is the size of the vocabulary. We use the counts transformed by TF-IDF (*term-frequency inverse-document-frequency*). This is a weighting scheme that emphasizes words that occur more in a particular document, while de-emphasizing words that occur in many documents. A typical instance of TF-IDF is

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) = f_{t,d} \times \log \frac{|D|}{N_{D,t}}$$

where $f_{t,d}$ is the count of term (word lemma) t in document d, D is the total set of training documents, and $N_{D,t}$ is the number of documents from D in which the term t appears.

3.1.4 Decision Trees

Decision tree classifiers create a set of thresholds over features, based on which to split the data. As for the SVM-based classifier, the feature vectors used are length normalized TF-IDF counts. We fit a binary decision tree to the data based on information gain criteria.

Ablation study: Initially we let the tree grow as deep as possible and use testing data to calculate the classification accuracy. Then we prune the tree level-by-level and observe the change in classification accuracy. We aim to study how overfitting takes place if the trees are allowed to grow to maximum depth. We also aim to understand how the pruning of a decision tree decreases the chances of overfitting and increases classification accuracy.

3.1.5 Word2Vec Embedding

Word2vec is a model that is used to produce a word embedding. It is a shallow embedding model that learns to represent a word as a vector of a fixed length. The vector is usually low dimensional. The space of vectors is defined in such a way that words which are semantically closer to each other (the relationship is strongly dependent on the training data) will have similar representations and thus smaller cosine distances from each other in vector space. Word2vec is not a single monolithic algorithm. It contains two distinct models, Skip-gram and CBOW. Skip-gram is an n-gram with some gaps in between. A word can be excluded from any word window. CBOW is trained to predict the target word t from the contextual words that surround it, c, i.e. the goal is to maximize $P(t|c)$ over the training set which is inversely proportional to the distance between the current vectors assigned to t and to c.

Word2vec works similarly to any online learning algorithm; it takes one example at a time and modifies the vector representations until the distance between the current vectors (provided by the training data with labels) is minimized. Note that Word2vec itself does not learn any semantic relationship; it simply brings words that co-occur closer together. Thus the Word2vec embedding depends highly on the training data. The Word2vec architecture is shown in figure 2.

Word2vec needs an extremely large amount of good training data in order to generate good representations. The webpage data we have is not enough to train such a model. So the model we used has been trained on Google News corpus containing about 3 billion words. Even though our dataset is comprised on webpages and the news corpus is not semantically similar to webpages, the trained model seemed to be good enough. The size of the vector representation is 300, and the model uses 16GB RAM to store its dictionary.

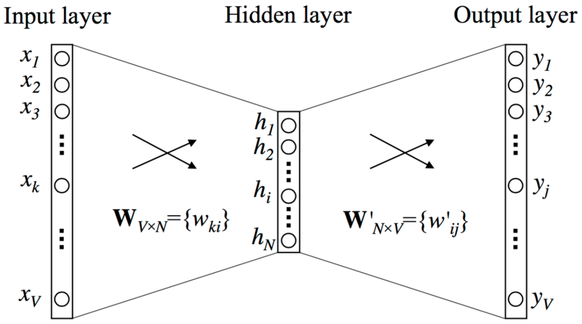


Figure 2: A simple model with context containing only one word.

3.2 Link-based Methods

We now discuss how we utilize information related to the neighbouring webpages of each webpage that we wish to classify. Note that our training-testing split is such that *none of the neighbours of any webpages in our test set is itself in the training set*, i.e. the neighbours of an unknown webpage are unknown as well. This is relevant to the choices we make for utilizing link-based information.

3.2.1 Link-pointing MLEs

In the training phase for using link-based methods, we compute the likelihoods of webpages of one category pointing to those of another - we call these quantities *link-pointing maximum likelihood estimates* or $link_{MLE}$. Let the categories of webpages be denoted as y_i, and let the partitions of the training set D induced by the categories be $D(y_i)$. We compute $link_{MLE}$ for one category pointing to another by iterating over the training set webpages and obtaining the following:

$$link_{MLE}(y_i, y_j) = P\Big[y_i \text{ points to } y_j\Big] = \frac{\sum\limits_{p\in D(y_i)} \sum\limits_{q\in D(y_j)} \delta(p,q)}{\sum\limits_{p\in D(y_i)} \sum\limits_{r\in D} \delta(p,r)}$$

where $\delta(a, b) = 1$ if webpage a points to webpage b. For convenience we will hereby denote $link_{MLE}(y_i, y_j)$ as $\Phi(y_i, y_j)$.

3.2.2 Single-hop neighbour lookup

We adopt an approach related to the idea presented in [12], modified appropriately based on the dataset that we have. We attempt to classify each webpage individually, in one pass through the test set. We only consider the information embedded in the particular webpage we are trying to classify, which is the content and outlinks, if any. Though this approach uses less information than a large test set potentially has (like inlinks and nested connections), it can be used at runtime to classify a single unknown webpage in an online setting, versus relying on a batch of webpages and the extra information it provides. Note that it is different from only content-based methods because now we are also considering the outlinks from each webpage.

Given a new test document d, we first run some content-based algorithm A that outputs the *probability* of d belonging to each of the categories $(P[d|y_i])$. Furthermore, let $L = \{l_1, l_2, \ldots l_N\}$ be the set of webpages that d points to. We assume that webpages in L are all themselves in the test set, i.e. that they are *unknown* and that their true category is one of the y_i's. To use the $\Phi(y_i, y_j)$ values, we need to assign some category to the webpages in L. We run the content-based algorithm A on each $l \in L$ to obtain a set of categories $Y(L) = \{Y(l_1), Y(l_2), \ldots Y(l_N)\}$. It might seem that we could go on and look at the out-links of the l_i's and so on, but as has been shown in [14], the influence of even two-hop neighbours on a webpage is very low, and so we stick to one-hop neighbours. We are trying to estimate the quantities $P[y_i|d, Y(L)]$ for each category y_i. Using Bayes' Rule,

$$P[y_i|d, Y(L)] \propto P[d|y_i, Y(L)]P[y_i|Y(L)] \propto P[d|y_i, Y(L)]P[Y(L)|y_i]P[y_i]$$

Note that for the content-based classifier A, $P[d|y_i, Y(L)] = P[d|y_i]$ as the link information L is not considered while estimating the probabilities. The $P[y_i]$ values are the class prior probabilities π_{y_i} as mentioned earlier. For the quantity $P[Y(L)|y_i] = P[Y(l_1), Y(l_2), \ldots, Y(l_N)|y_i]$, we make the Naive Bayes assumption to obtain

$$P[Y(L)|y_i] = \prod_{l \in L} \Phi(y_i, Y(l))$$

Therefore (with log-likelihoods in the actual implementation),

$$Y(d) = \underset{i}{\text{argmax}}\ \pi_{y_i} P[d|y_i] \prod_{l \in L} \Phi(y_i, Y(l))$$

This formulation allows us to utilize the link-pointing MLE values for augmenting the estimates from the content-only classification methods. In practice, we consider two variants of the Naive Bayes assumption. One variant is exactly what we have shown, i.e. multiply $\Phi(y_i, Y(l))$ for every single neighbour l. Therefore if there are multiple neighbours of the same category, that value will be incorporated multiple times in the product. Let us call this the *BagOfLinks* weighting.

The other variant is to only count each category of a neighbour once, if such a neighbour appears. In this setting, $Y(L) \rightarrow \text{set}(Y(L))$ such that duplicates are removed. Let us call this the *SetOfLinks* weighting. The justification for this is similar to that used in several text processing methods that use the presence or absence of some word, rather than the count of the word, as a feature.

3.2.3 A Note on Graphical Models

Intuitively, it seems like the network of linked webpages can be represented as a directed graph with nodes as webpages and edges as hyperlinks. While that representation is certainly appropriate for webpage network analysis tasks, we were unsure of how to exactly do inference, because (i) nodes in a Bayes Net typically represent random variables, of which several samples can be obtained, unlike a single webpage, and (ii) cycles can exist in webpage networks, while the Bayes Net inference algorithm for the Python library that we found, PGMPY [17], only worked with DAGs.

However, we did some edits to frame the test set as a valid (at least operationally) graphical model, and were able to perform inference on it using the Variable Elimination technique [18] of the PGMPY library . We only make a qualitative comment on that here, omitting quantitative results.

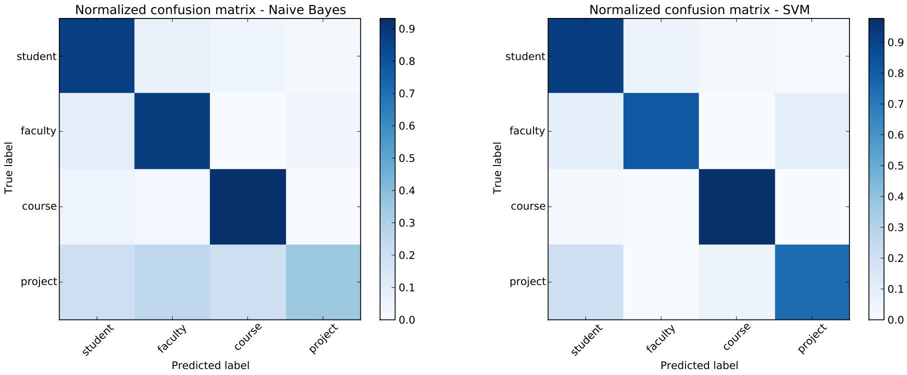
Let us say we have n webpages of m categories, and only consider hyperlinks between webpages in our set. Thus we obtain a *directed graph with cycles*, say G. Firstly, we split G into each of its *weakly connected components*. Secondly, for each connected component g_i, we break cycles (by arbitrarily deleting cycle-completing edges) to get corresponding components g'_i. For a node (webpage) with no parents, the parameter is the $P[d|y_i]$ vector (from some algorithm A like earlier) for the webpage. For a node with p parents, the conditional probability table (CPT) is a $m^p \times m$ matrix derived from the $\Phi(y_i, y_j)$ values described earlier. To incorporate the priors, we multiply each row of the CPT with the $P[d|y_i]$ vector and re-normalize.

Now each g'_i is in a form that allows us to run a joint inference algorithm on it. However, this is exponential in the number of unobserved nodes (and the library memory-overflowed on our system), and so is infeasible for our large test set. In practice, we extract subgraphs (arbitrarily) of size up to 10 nodes from any g'_i with a large number of nodes. Because of the several arbitrary choices made till this stage, we do not report any principled quantitative results. However, qualitatively we did notice that several of the subgraphs that we jointly inferred were all correctly classified. In fact, we found subgraphs containing webpages which the content-based methods (SVM, NB) alone classified wrongly, which were correctly classified (along with the other subgraph nodes) when augmented with the graphical model.

4 Results

We use the WebKB dataset [3] which has a collection of thousands of categorized web-pages from computer science departments of several universities. There are seven categories: *project*, *student*, *faculty*, *course*, *staff*, *department* and *other*. Each class has pages from 4 universities and other miscellaneous universities. The training is done on webpages from 3 of the 4 universities and the miscellaneous ones, while testing is done with the web-pages of the fourth university. Like previous work, we focus on the 4 well-represented classes among the 7: *project*, *student*, *faculty*, and *course*.

4.1 Multinomial Naive Bayes



(a) (b)

Figure 3: Content-based classification results for multinomial Naive Bayes in (a) and Support Vector Machine in (b)

After implementing a Multinomial Naive Bayes classifier with TF-IDF weighted counts, we find the classification accuracy to be $\mathbf{84.07\%}$, despite the simplicity of the approach. The confusion matrix is shown in figure 3a. The values are normalized by the number of examples of each category. The *course* webpages are classified correctly most often. The *project* webpages are the most tricky to classify, as expected because a project can be associated with all three of students, faculty or courses.

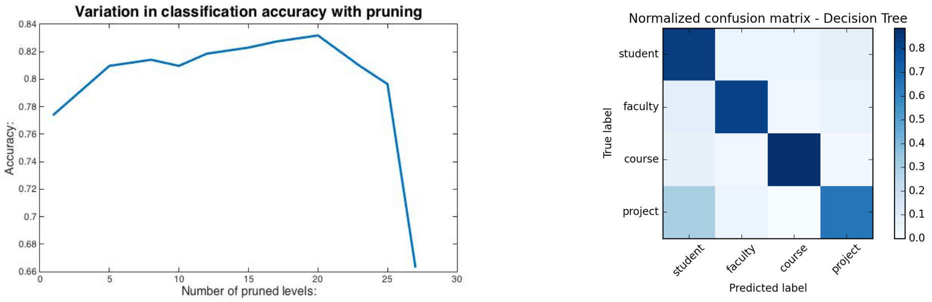
4.2 Support Vector Machine

We train a multi-class Linear SVM on feature vectors which are TF-IDF counts of the known words in the vocabulary. Any words in the test document which were unseen during training phase are filtered out. The multi-class SVM is implemented as 4 *one-vs-rest* classifiers, one for each category. The overall accuracy for the SVM on the test data is $\mathbf{90.26\%}$, the highest among the three methods chosen. The normalized confusion matrix is shown in 3b. Once again, the *course* category is correctly classified most strongly, while the *project* webpages, while better classified by an SVM than by Naive Bayes, contributes the most to mistakes.

4.3 Decision Trees

We perform a 4-way multi-class classification. We observe that if the decision tree is allowed to grow to the maximum depth, the classification accuracy turns out to be $\mathbf{76.99\%}$. We have omitted the confusion matrix for this for sake of brevity. We then study the advantage of pruning the decision tree. We observe that the maximum level the tree can have is 28. We prune the tree level-by-level and observe the change in classification accuracy. The overall results are shown in Figure 4.

As we can see in figure 4a, initially the number of pruned levels is 0 and this is when the tree grows to level 28 (the maximum it can grow) and the accuracy here is 76.99%. We observe that the classification accuracy increases until the number of pruned levels increases to 20 (i.e. the tree has only 8 levels remaining). This shows that the chances of overfitting decreases as the depth decreases to a certain level. After that the tree is as deep as it should be to classify the data properly, as evident by the fall in accuracy observed as the number of pruned levels increases beyond 20. When the number of pruned levels reaches 27, it means that we have only the root left, and the classification accuracy is 67.37%. The maximum accuracy we observe is $\mathbf{83.19\%}$ when the number of pruned levels is 20. The confusion matrix is shown in figure 4b.



(a)

(b)

Figure 4: Content-based classification results for the decision tree. The effect of pruning levels is shown in (a) and the confusion matrix for the best tree is shown in (b)

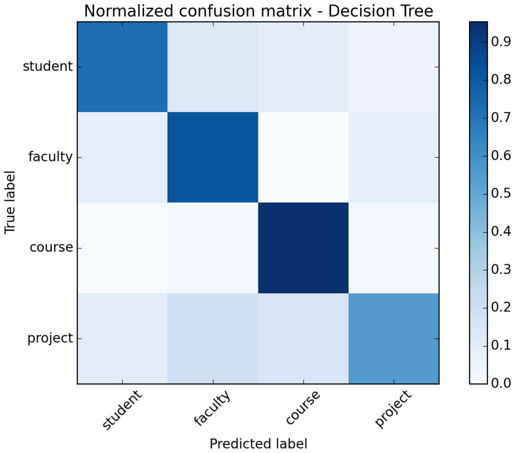


Figure 5: Confusion matrix for Word2vec classification.

4.4 Word2vec embedding

We used a simple strategy of Nearest Neighbor to evaluate our model. We removed irrelevant words like HTML tags, stop words, etc. and then extracted the top 100 most frequent words from each category of the training data. We used our trained model to get the vector representations of these words. For each document in the test set, we similarly extracted the top 100 most frequent words, generated a matrix of word representations (again using the trained model) and used a nearest neighbor approach to obtain the closest category. For each category, first for each word in the test document we obtained the minimum cosine distance from all 100 words in that category and then we took the minima of all these cosine distances corresponding to all the words in the test data. This gives us the score of a test document for each category. The category corresponding to the minimum score is chosen as the predicted category.

The classification accuracy we obtain is $\mathbf{84.6\%}$. The confusion matrix is shown in figure 5. The results are quite good considering that we used just Nearest Neighbor approach for matching. This shows how powerful the trained model is and how good the representations are even though they have been trained on a corpus different from web pages. Even though there is no explicit learning performed for the classification of web pages still the Word2vec embedding is so strong that it can give results comparable to other methods with a simple nearest neighbor approach.

4.5 Single-hop neighbour lookup

As described in Section 3, the link-based method we use works over and above some baseline content-based method that assigns a probability to each test webpage for belonging to each cate-

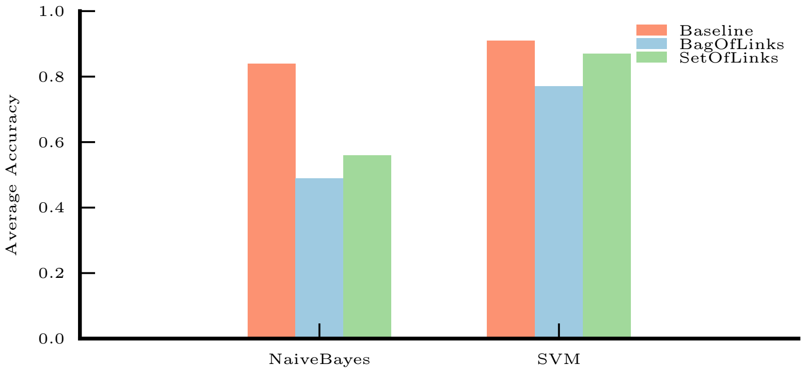


Figure 6: A comparison of the baseline approaches, along with augmented estimates using single-hop link-MLE values. The augmented estimates are generally poorer than the baseline

gory. In our experiments, we use the Naive Bayes and SVM classifiers as the baseline methods, as they have a well-defined notion of this category membership likelihood, while the other approaches typically just classify the document.

We use a subset of the test set for the content-based methods, as not all webpages among the ones used point to another webpage also from the test set. The average accuracy for augmenting the baseline estimates with this single-hop lookup is shown in Figure 6. Unfortunately, we observe that the extra effort does not yield any benefit, as the estimates are poorer than that of the baseline, significantly so for the Naive Bayes baseline.

In general, we believe this is due to the nature of outlink categories being a more noisy and sensitive feature than the words in the document. Furthermore, as described in the approach, we assign categories to d's neighbouring webpages in L based purely on the decision of A. If this decision is incorrect, it could reduce the likelihood of the true category of d, as obtained from running A on d. This would contribute to a poorer performance. The particularly poor performance of Naive Bayes is probably due to many content-based estimates not having been of high confidence to begin with, thereby rendering them more susceptible to potentially noisy link estimates.

Difference Analysis

To determine whether there is any value at all for using the link-based information, we observe the incorrect cases for A, i.e. for the NB and SVM classifiers, to determine if any of them are correctly classified due to the link-based weighting. This would at least provide some positive reinforcement for the link-based weighting. This is indeed the case for both NB and SVM classifiers. For the NB baseline, 35.7% of the incorrect predictions by NB are corrected by the augmented approach. For the SVM baseline, this correction percentage goes up to 50%.

5 Discussion

This project gave us a number of insights into the general text classification problem and webpage categorization. By filtering out non-words and reducing words to their root forms, even a simple classifier like Naive Bayes performs quite well. Furthermore, all the baseline classifiers performed well using just the webpage text content. As we observed, using the hyperlink information in a simple way to augment baseline methods for an individual webpage does not improve the accuracy, and in fact reduces it for a less confident baseline method like Naive Bayes. Therefore, perhaps the value of hyperlinks truly lies in making joint inferences of connected unknown webpages. However, such an approach has several challenges, including a principled way to represent and make inferences from the network, and the tractability of such a process for jointly predicting a large test set.

References

[1] Bird, Steven. "NLTK: the natural language toolkit." *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006.

[2] Miller, George A. "WordNet: a lexical database for English." *Communications of the ACM* 38.11 (1995): 39-41.

[3] Craven, Mark, et al. *Learning to extract symbolic knowledge from the World Wide Web*. No. CMU-CS-98-122. Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 1998.

[4] Joachims, Thorsten. *Text categorization with support vector machines: Learning with many relevant features*. Springer Berlin Heidelberg, 1998.

[5] McCallum, Andrew, and Kamal Nigam. "A comparison of event models for naive bayes text classification." *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1998.

[6] Kibriya, Frank, Pfahringer and Holmes. "Multinomial Naive Bayes for Text Categorization Revisited." 1999.

[7] Rennie, Shih, Teevan and Karger. "Tackling the Poor Assumptions of Naive Bayes Text Classifiers." Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

[8] Sun, Kim and Ng. "Web Classification Using Support Vector Machine." Proceedings of the 4th Int. Workshop on Web Information and Data Management (WIDM 2002) held in conj. with CIKM 2002, Virginia, USA, Nov, 2002.

[9] Tong and Koller. "Support Vector Machine Active Learning with Applications to Text Classification." Journal of Machine Learning Research (2001) 45-66.

[10] Forman. "An Extensive Empirical Study of Feature Selection Metrics for Text Classification." Journal of Machine Learning Research 3 (2003) 1289-1305.

[11] Genkin and Lewis. "Large-Scale Bayesian Logistic Regression for Text Categorization." American Statistical Association and the American Society for Quality TECHNOMETRICS, AUGUST 2007, VOL. 49, NO. 3.

[12] Qi and Davidson. "Knowing a Web Page by the Company It Keeps." CIKM06, November 511, 2006.

[13] Slattery and Craven. "Combining Statistical and Relational Methods for Learning in Hypertext Domains." 1998.

[14] Qi and Davidson. "Web Page Classification: Features and Algorithms." Technical Report LU-CSE-07-010, Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, 18015

[15] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). "The PageRank citation ranking: bringing order to the web."

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In Proceedings of Workshop at ICLR, 2013

[17] Ankur Ankan, Abinash Panda. "pgmpy: Probabilistic Graphical Models using Python". In Proceedings of the 14th Python in Science Conference (SCIPY 2015)

[18] Zhang, Nevin Lianwen, and David Poole. "Exploiting causal independence in Bayesian network inference." Journal of Artificial Intelligence Research (1996): 301-328.