Logo, company name

Description automatically generated

7COM1018 – Data Mining

Practical Text Mining Investigation

Prepared by

Simpson Chiwashira

21017447

Introduction

In my investigation, I have decided to study the impact of using the stemming technique in text mining against the impact of not using stemming. According to the authors Jasmeet Singh and Vishal Gupta who published an article (2016), Stemming can be defined as a process where variant word forms are mapped to their base forms. It is also important to note that stemming is a pre-processing technique that is used before text mining can be performed.

They are many stemming algorithms that have been developed over the past and I used the Lovin stemmer algorithm which is available in Weka the data mining tool I have used. This algorithm works very well like any other stemmer. It simply works by removing suffixes from the text and it also has a large data set with all suffixes. It is really fast and effective this can be supported by authors Bounabi et al in their article where they did experiments using Lovin, iterated Lovin and snowball stemmer in an experiment and the Lovin gave a better performance in the results. The Lovin achieved the best accuracy of up to 97% in the first database hence the reason behind my choosing this technique.

I performed my experiment to investigate the impact of stemming over not using stemming and the results that I achieved convinced me that it has an impact. This was a result of my observation of the results. My results were convincingly conclusive that stemming has a greater impact in the text mining experiment that I performed.

I have chosen to investigate this topic because I find it very interesting. After all, it is something that is used in Search Engine Optimization (SEO), and it has made queries easy to predict. Companies like Microsoft and Google use this for information retrieval, machine learning, and natural language processing.

Methodology and Outcomes

Now in this section, I am going to talk about the procedures and results that I observed and give an evaluation. Firstly, the first thing was to get my data set and load it into Weka. The dataset on its own cannot be loaded into the Weka as it is since the system does not know what to do with it. So, it gives me the option to select ways in which we can load this data. And I selected the option to load it as a TextDirectoryLoader. And in this file, there are two classes labelled business and tech.

So now the next thing I had to do was go on filters and select the Unsupervised and then it branched into Attributes and instances. I went on to choose the attributes and selected the StringToWordVector and after that, I clicked to apply the filter. After that, I was able to see all attributes and they were 1,486 in total. The next thing I had to do was to create two Arff files that I was meant to use for training. By visualising the data that I had the two classes business and tech are not balanced at all. So, for me to have better accuracy I needed to have two class labels with the same number of instances which led me to perform another pre-processing step to make sure we have balanced data. There are two ways that I know how to make the data balanced and that is either to undersample or oversample the dataset.

Balancing the dataset in Weka is also something that can be done since it supports that and has the feature to under-sample (this means reducing the number of data items in one major class to match the minority class). Oversampling means we populate the minority class and also try to match the other class. Weka supports SMOTE and SpreadSubSample which can be used to oversample and under-sample our training dataset respectively.

For me to be able to do that I selected the class label from the drop-down on my hand panel and I went on to select filters whilst I am still on the pre-processing tab. In the filters, I selected the supervised and underneath it, I selected the instances and from there we can see what options we have. I went on to under-sample the training set. To do that you select SpreadSubSample and a dialogue box will appear, I had to input the parameters. I will be showing my parameters through the use of a table and explaining why I would have done so.

Fig 1.1 Table Showing parameters for SpreadSubSample

|  |  |
| --- | --- |
| adjustWeights | False |
| DistributionSpread | 1.0 |
| maxCount | 0.0 |
| randomSeed | 1 |

I changed the distribution spread from 0.0 to 1.0 and the other parameters I left them as they were. So, what it means is just I want to under-sample the first-class label which had 510 so that it matches the minority class that had 401, after that, I clicked OK and then applied it to the dataset and I ended up having the same number of instances in both classes which are 401.

Now the next step is where we have to split our two files into two Arff files. So, since I am investigating if a model performs well when stemming is applied against when it is not applied. My first step is to apply a stemming filter before any classifier is applied. Under the filters I went to the Unsupervised and then under attribute, I selected the StringToWordVectors. Then I clicked on it and a dialogue box appeared and this is where I made the changes.

These are the parameters I put in place.

IDF Transform - False AttributeIndices – First - last

TF Transform - False LowerCaseTokens – False

Stemmer – Lovin Stemmer Stopwords - weka

Usestoplist – True WordsToKeep – 1000

The other parameters are set by default so I did not need to make any changes to them and since I selected the Lovin stemmer it means my dataset will have to reduce in size, and my attribute count. And also enabled the Stop list to be true.

The results after clicking apply came as the Attribute count changed from 1,486 to 1,445. And this means that our stemmer has been applied after that I clicked save on the panels on top and saved the file in a directory of my choice. And I named the file Stemmer, and it was saved as an arff. I closed the application and went on to redo the process of loading the file and converting it and balancing the dataset. Then I just went on to save it after that and renamed it non-Stemmer and the setting on the parameters for StringToWordVector was set to null stemmer and saved it an arff and closed the weka, this is how I converted my two datasets into Arff files.

In the next stage now I had to work with the two datasets I had created, and I worked on them one by one starting with non-stemmer, and I applied some parameters and used three different classifiers and I explained each step and the outcomes and included visuals to see the changes so as to evaluate the performances.

Non-Stemmer

In the Weka, I loaded the dataset that I had created with the name non-stemmer. arff file which I saved in my directory of choice. Now the next step is to start applying the classifiers. I started with the Naïve Bayes classifier. So, on top of the screen, there was a tab which named classify and under that are a lot of functions available. I opened the file named Bayes and I selected Naïve Bayes as required by the assessment guideline.

Before training the model, I had to find out if I change parameters if there was any change and how the model performs. So, I did I changed the split percentage and made a table where I recorded every accuracy after selecting a certain percentage. Split is simply the percentage of the data set that you give to the model to use as training data. I went on to select from 10 - 90 wherein each instance I record all the details as they are coming under the accuracy header. For example, the split is 10 and I selected the random seed to be 1 and the accuracy to 1 on the same split I changed the random seed to 50 and record the accuracy under accuracy 2. I did this for all the splits and all the classifiers to follow.

Then after that, the next thing is I had to calculate the mean of accuracies and that was my overall accuracy for Naïve Bayes when stemmer is not applied. Below is a table which shows all the splits and accuracies.

Naïve Bayes Performance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Accuracy 1 (Seed 1)** | **Accuracy 2 (Seed 50)** | **Mean Accuracy** |
| 10 | 98.25 | 98.25 | 98.25 |
| 20 | 97.04 | 96.72 | 96.88 |
| 30 | 96.96 | 96.61 | 96.785 |
| 40 | 96.88 | 96.88 | 96.88 |
| 50 | 96.88 | 96.5 | 96.69 |
| 60 | 97.5 | 95.95 | 96.725 |
| 70 | 97.92 | 95.43 | 96.675 |
| 80 | 96.87 | 94.37 | 95.62 |
| 90 | 96.25 | 96.25 | 96.25 |

The overall performance of the Naïve Bayes after calculating the average mean accuracy of all the ten splits will give a percentage of 96.75% which is not bad at all. According to the table above the lesser, the training split the higher the accuracy.

The next one after this is the Decision tree: J48. While on the classified table I clicked classifier and there were subfolders, and I opened the one that is named tree and that’s where I found the J48 and selected it. The next thing is to choose my parameters for the j48. And I did not have to change any of the parameters I just used the training set with different splits and different random seed values which I was alternating between 1 and 50 as I did with the naïve Bayes. These are the parameters for the j48 tree: J48 -C 0.25 -M 2.

After that came to the results and they were not as good as from the naïve bayes they were fairly low. At an average of 86.45% that was the overall performance and below is the table of all accuracies and splits for this classifier.

Decision Tree J48

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Accuracy 1 (Seed1)** | **Accuracy 2 (Seed 50)** | **Mean Accuracy** |
| 10 | 81.57 | 76.31 | 78.94 |
| 20 | 84.26 | 83.33 | 83.795 |
| 30 | 91.62 | 83.77 | 87.695 |
| 40 | 86.69 | 85.86 | 86.275 |
| 50 | 88.27 | 84.03 | 86.15 |
| 60 | 90.03 | 82.24 | 86.135 |
| 70 | 84.64 | 87.96 | 86.3 |
| 80 | 88.75 | 90.62 | 89.685 |
| 90 | 91.25 | 95 | 93.125 |

The best performance in this classifier can be seen from the split 90 that is then we had a better performance of 93.125%. It did not perform very well therefore we need to find out about the other classifier.

The next classifier I used is a Support Vector Machine (SVM). On this classifier, there are parameters that I used to come up with the best performance. I used two kernels to train the model and see which one was better. These two kernels are Radial Basis Function (RBF) and the Linear Kernel. For the best performance, I had to determine which values would I use for the cost and the gamma values. However, our Weka data mining tool will do that for us easily by the use of a special function called the grid search. I selected the classifier tab and then from the list I selected meta and from that, I was able to choose grid search. The next step was to select my parameters for this and the first kernel I used is the RBF. When I clicked the grid search here are the parameters used.

Xbase – 10.0 XExpression – pow(BASE,I) XMax - 5.0

XMin - -2.0 Xproperty – classifier. cost Xstep -1.0

Ybase – 10.0 YExpression – pow(BASE,I) YMax - 5.0

YMin - -2.0 Yproperty – classifier.gamma Ystep -1.0

Evaluation – Accuracy debug – false Filter – All Filters

Classifier – LibSVM

After that, I clicked on the LibSVM and selected the parameters for that as well and I only changed the kernel to RBF and normalisation turned to true and these are the rest of the parameters set.

LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z

After training the model by trying all the split percentages and the random seed I came up with a table and the mean accuracy. The overall mean of the RBF kernel came to be 96.92% accuracy. And the grid search found the cost and the gamma values for us as well. The cost was 100,000 and the gamma was 0.01.

Next, I used the same parameters as above, however, the only change was that I selected the linear kernel instead of the RBF kernel and trained the model using the same training set as before and I achieved an accuracy of 97.12% which is way better than the RBF kernel. In conclusion, I decided to take the kernel with the best performance and that is the linear kernel.

Here are the results I recorded in form of a table.

Support Vector Machine

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  | Radial Basis Function | |  |  |  | Linear kernel | |
| split | accuracy 1 | accuracy 2 | Mean |  | accuracy 1 | accuracy 2 | mean |
| 10 | 100 | 100 | 100 |  | 96.95 | 97.22 | 97.085 |
| 20 | 96.1 | 95.01 | 95.555 |  | 96.88 | 97.04 | 96.96 |
| 30 | 96.25 | 96.61 | 96.43 |  | 97.68 | 97.68 | 97.68 |
| 40 | 96.04 | 96.88 | 96.46 |  | 96.88 | 96.88 | 96.88 |
| 50 | 96.01 | 96.25 | 96.13 |  | 96.75 | 97.25 | 97 |
| 60 | 96.57 | 96.57 | 96.57 |  | 97.19 | 97.19 | 97.19 |
| 70 | 97.92 | 96.26 | 97.09 |  | 98.34 | 96.26 | 97.3 |
| 80 | 98.12 | 95 | 96.56 |  | 98.75 | 95.62 | 97.185 |
| 90 | 100 | 95 | 97.5 |  | 98.75 | 95 | 96.875 |

And here is an overview of all three classifiers to visualise how they performed on each split.

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Naïve Bayes** | **Decision Tree: J48** | **SVM (Linear Kernel)** |
| 10 | 98.25 | 78.94 | 97.085 |
| 20 | 96.88 | 83.795 | 96.96 |
| 30 | 96.785 | 87.695 | 97.68 |
| 40 | 96.88 | 86.275 | 96.88 |
| 50 | 96.69 | 86.15 | 97 |
| 60 | 96.725 | 86.135 | 97.19 |
| 70 | 96.675 | 86.3 | 97.3 |
| 80 | 95.62 | 89.685 | 97.185 |
| 90 | 96.25 | 93.125 | 96.875 |

Above are the performances of the three classifiers that I used, and the following is the graph that was plotted for the overall mean accuracy against the split. And we can see that the two classifiers, naïve bayes and the SVM were performing well and the decision tree showed a poor performance.

Stemmer

After training my model on a dataset that I labelled non-stemmer I closed my Weka and loaded the dataset labelled Stemmer. arff. I used the same methods as I have mentioned above and used the same parameters on just a new dataset which has fewer attributes than the other. I started with the Naïve Bayes as before and using the same parameters I was able to achieve the best accuracy of 97.31% which is slightly higher than that of the non-stemmer. The figures of how each split was performed are given in the following table.

Naïve Bayes

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Accuracy 1 (Seed 1)** | **Accuracy 2 (Seed 50)** | **Mean Accuracy** |
| 10 | 97.5 | 96.39 | 96.945 |
| 20 | 97.97 | 97.19 | 97.58 |
| 30 | 97.68 | 98.03 | 97.855 |
| 40 | 97.71 | 97.5 | 97.605 |
| 50 | 97.25 | 96.75 | 97 |
| 60 | 97.5 | 96.57 | 97.035 |
| 70 | 97.09 | 97.09 | 97.09 |
| 80 | 96.25 | 98.12 | 97.185 |
| 90 | 96.25 | 98.75 | 97.5 |

After the Naïve Bayes, I went on to train the model using a decision tree (J48) classifier with the dataset I had used the Lovin stemmer, and I was able to achieve an accuracy that was slightly higher than that of the previous one. I achieved an accuracy of 89.28% and it is much better because of that, I could see how stemming was useful. And here is the table with the values.

Decision Tree: J48

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Accuracy 1 (Seed )** | **Accuracy 2 (Seed 50)** | **Mean Accuracy** |
| 10 | 85.04 | 73.68 | 79.36 |
| 20 | 85.35 | 80.06 | 82.705 |
| 30 | 88.41 | 92.15 | 90.28 |
| 40 | 91.68 | 86.48 | 89.08 |
| 50 | 93.76 | 89.02 | 91.39 |
| 60 | 90.03 | 89.4 | 89.715 |
| 70 | 94.19 | 90.45 | 92.32 |
| 80 | 93.75 | 95 | 94.375 |
| 90 | 92.5 | 96.25 | 94.375 |

Lastly, I used the SVM with both linear and RBF kernels. I used the same parameters also the same as I did in the previous experiment. After using the grid search to calculate the best cost and gamma value it was able to report the values to be cost – 100,000.0 and gamma value – 100,000.0. Now the RBF and the Linear Kernel gave me different results in their accuracies the Linear kernel came out on top with a higher performance than the RBF and the model achieved an overall mean accuracy of 97.3% with the Linear Kernel ahead of the RBF with 97.10%. So, in this case, again I went on to choose the Liner Kernel which proved to have done even better than the previous SVM with no stemmer. Here are the results of the SVM classifier.

Support Vector Machine

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **RBF Kernel** |  |  |  |  | **Linear kernel** |  |
| Split | Accuracy 1 | Accuracy 2 | Mean Accuracy |  | Accuracy 1 | Accuracy 2 | Mean |
| 10 | 96.26 | 94.45 | 95.355 |  | 95.84 | 96.39 | 96.115 |
| 20 | 96.88 | 95.95 | 96.415 |  | 97.19 | 97.81 | 97.5 |
| 30 | 97.32 | 96.96 | 97.14 |  | 96.96 | 97.32 | 97.14 |
| 40 | 98.33 | 97.5 | 97.915 |  | 97.71 | 97.92 | 97.815 |
| 50 | 98 | 97.5 | 97.75 |  | 97.75 | 97.75 | 97.75 |
| 60 | 98.13 | 96.57 | 97.35 |  | 98.13 | 97.19 | 97.66 |
| 70 | 97.51 | 97.09 | 97.3 |  | 97.92 | 97.92 | 97.92 |
| 80 | 97.5 | 98.12 | 97.81 |  | 96.87 | 98.75 | 97.81 |
| 90 | 95 | 98.75 | 96.875 |  | 95 | 98.75 | 96.875 |

Now overall the table with the performances of these classifiers is shown below with the graph that I have also plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| **Split** | **Naive Bayes** | **Decision Tree: J48** | **SVM (Linear Kernel)** |
| 10 | 96.945 | 79.36 | 96.115 |
| 20 | 97.58 | 82.705 | 97.5 |
| 30 | 97.855 | 90.28 | 97.14 |
| 40 | 97.605 | 89.08 | 97.815 |
| 50 | 97 | 91.39 | 97.75 |
| 60 | 97.035 | 89.715 | 97.66 |
| 70 | 97.09 | 92.32 | 97.92 |
| 80 | 97.185 | 94.375 | 97.81 |
| 90 | 97.5 | 94.375 | 96.875 |

And now I will show a graph to visualise the performances and the changes between the two methods. The graph shown below tells us how better performance using a stemmer is unlike not using it in text mining. From the graph, we can see that naïve bayes and SVM seem to be doing better and there is only a slight difference between them but they are all above unlike the decision tree which is performing poorly it is the only one in the 80s region and much lower than any other classifier.

Conclusion

When I started my investigation, my primary objective was to find out the impact of Stemming on a data set over not using it at all. I am happy to report that I have used the appropriate methodology and parameters and come up with conclusive results. These are the mean performances of each algorithm for both Non-Stemmer and Stemmer.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Naïve Bayes | Decision Tree J48 | SVM Linear Kernel |
| Non-Stemmer | 96.75% | 86.45% | 97.12% |
| Stemmer | 97.31% | 89.28% | 97.39% |
| **Difference** | 0.56% | 2.83% | 0.27% |

We can conclude that yes, the use of stemming had an impact on the performances since it is visible that naïve bayes was able to increase its performance by 0.56% after applying it. And the Decision tree J48 had a major increase which was an increase of 2.83%. Which shows how much of an impact it has made on the model. Lastly, the SVM (Linear Kernel) achieved an increase of 0.27%. I believe the best algorithm that has shown the best results and performance is the SVM with linear kernel.

References

1. [ACM Computing Surveys](https://dl-acm-org.ezproxy.herts.ac.uk/toc/csur/2017/49/3) [Volume 49Issue 3](https://dl-acm-org.ezproxy.herts.ac.uk/toc/csur/2017/49/3) September 2017 Article No.: 45pp 1–46
2. [BDCA'17: Proceedings of the 2nd International Conference on Big Data, Cloud and Applications](https://dl-acm-org.ezproxy.herts.ac.uk/doi/proceedings/10.1145/3090354) March 2017 Article No.: 43Pages 1–9