Webpage classification using machine learning

Anirudh K. Muralidhar (anikamal@iu.edu)

Abstract *- The purpose of this task is to classify a given webpage into a particular category using machine learning. The university dataset [1] from webkb has been used for this task.*

Introduction

The task of webpage/website classification can be really useful for maintaining information about websites, and for retrieval tasks.

But, this problem is a real challenge to the existing experts in Computer Science. This is primarily because of their unstructured nature of the data.

In recent times, it's well known that machine learning has started to have a profound influence in driving businesses, and solving complex problem. So, with the help of machine learning, we try to solve this task.

Dataset

The webkb University dataset [1] has been chosen for this particular task.

There are totally 8,282 web pages collected from computer science departments of Universities such as Cornell, Washington, Texas, Wisconsin, and misc. And each of these pages are categorized into one of the following

1. Student
2. Faculty
3. Staff
4. Department
5. Course
6. Project
7. Others

Webpage classification using machine learning

Anirudh K. Muralidhar (anikamal@iu.edu)

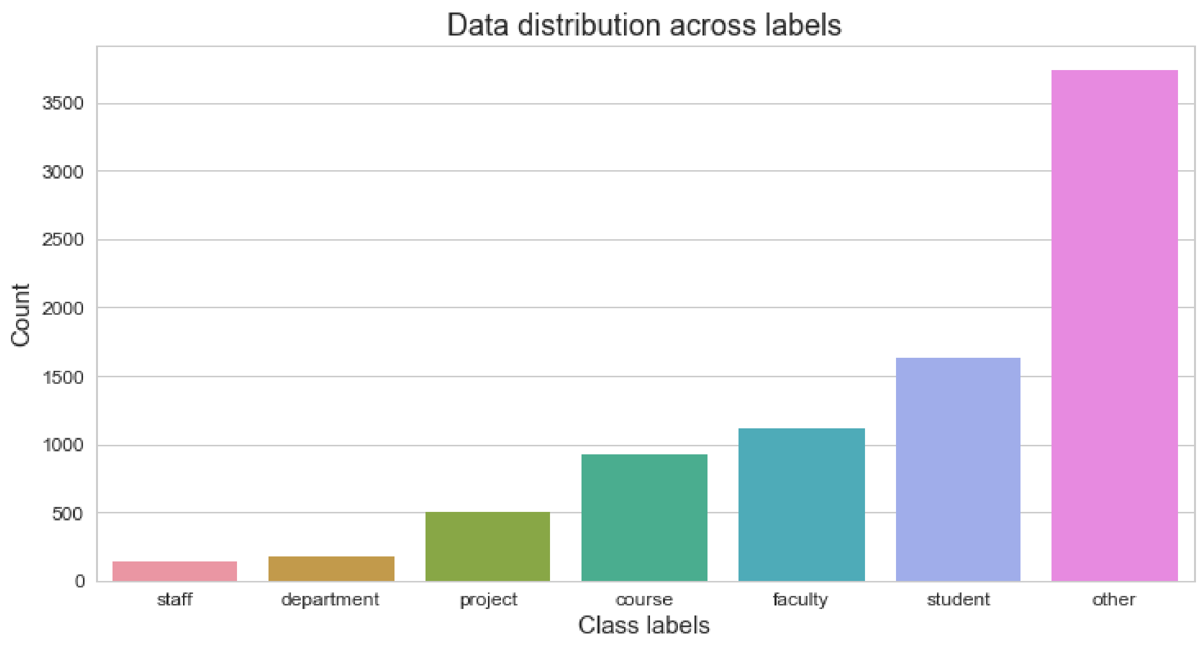


Fig 1. The class label distribution of the data

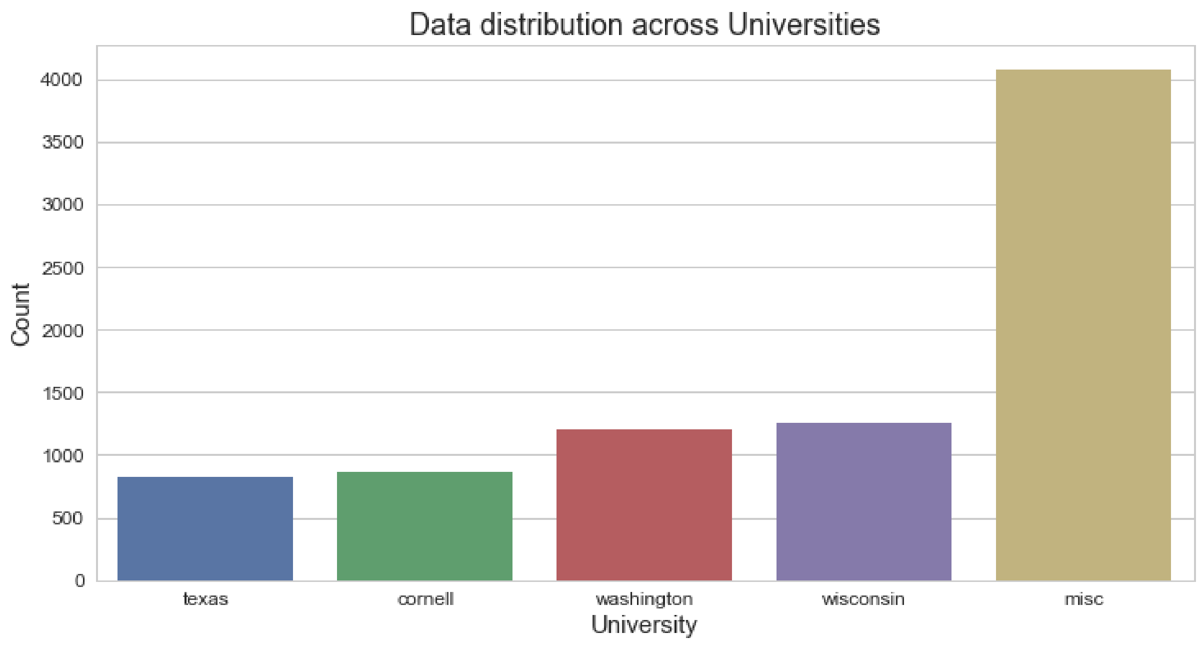


Fig 2. Data distribution across Universities

We could see that “other” label serves as the base class for this dataset.

Webpage classification using machine learning

Anirudh K. Muralidhar (anikamal@iu.edu)

Data fetching, and pre-processing

The dataset is in HTML format, so the required data are extracted from the HTML file. The following steps are done during the process,

1. Remove all HTML tags.
2. Extract text data alone.
3. Ignore punctuations, digits in the text.
4. Handle spacing issue.
5. Convert text to lower case.
6. English stop words have been removed.
7. Ignore words that are of length one.
8. Lemmatize words in the text.
9. Add, keywords from URL to the text.

Finally, store the processed data to a CSV file, for further process.

Finally, bi-gram, and tri-gram words are generated from the text, and from further inferences we include the bi-gram data into the feature set as well.

Exploratory data analysis

Basic EDA has been conducted on the dataset to gain better understanding of the data.

Results are presented below,

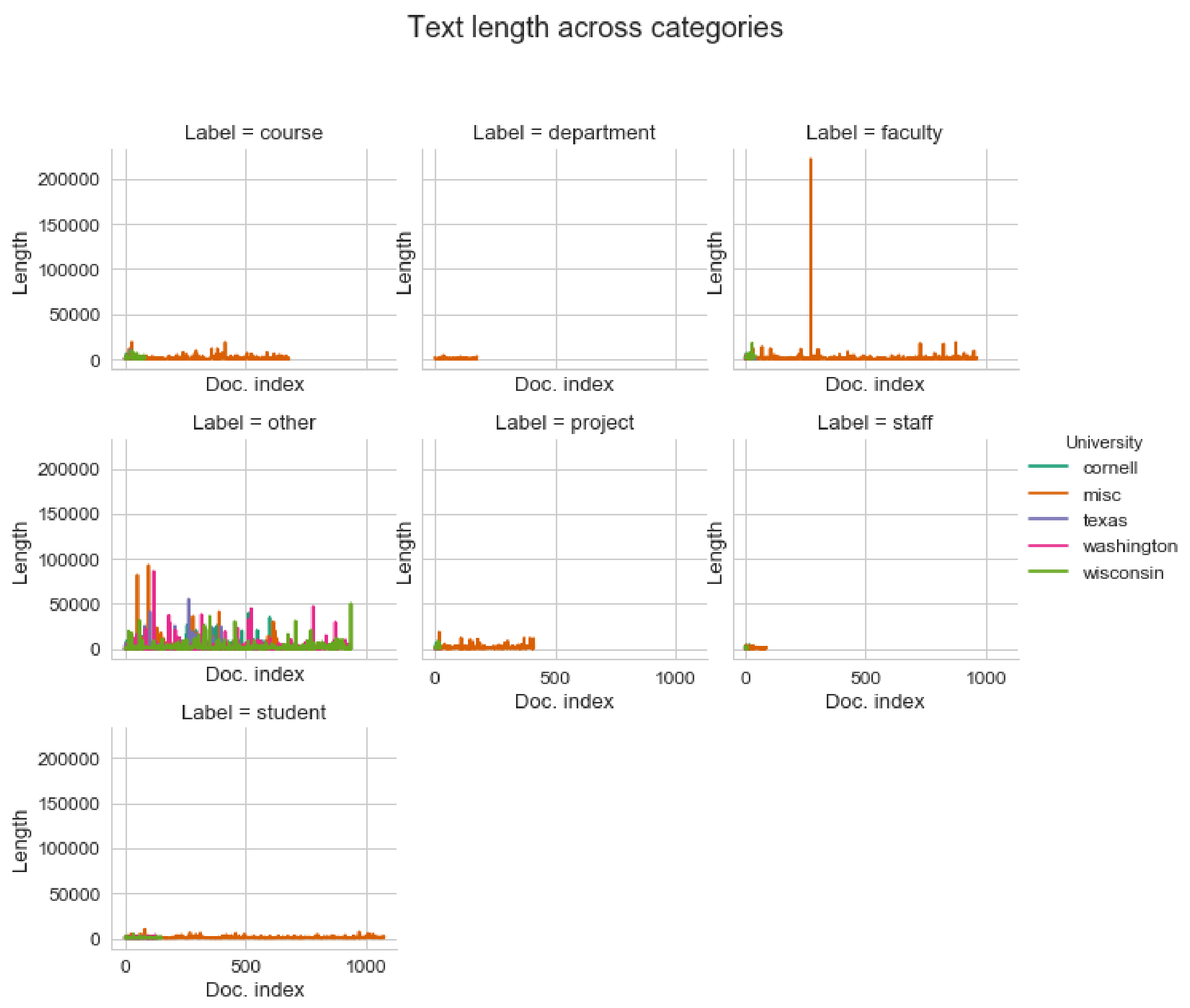


Fig 3. Length of processed documents across categories

We could see that the “other” category seems to have lengthy documents compared to other categories. This gives us an intuition to include the length as a feature as well apart from the text features.

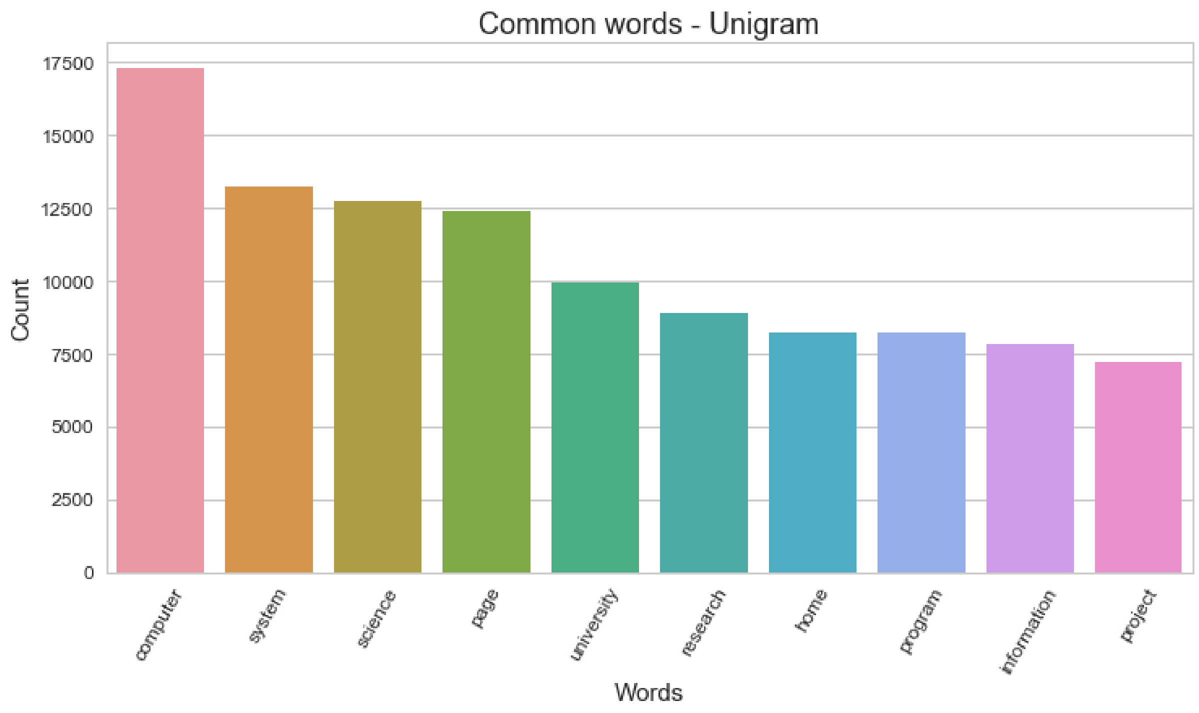


Fig 4. The most common uni-gram words in the data

The word “computer” stands out compared to the other words in the text. This is expected since the dataset has been derived from computer science departments.

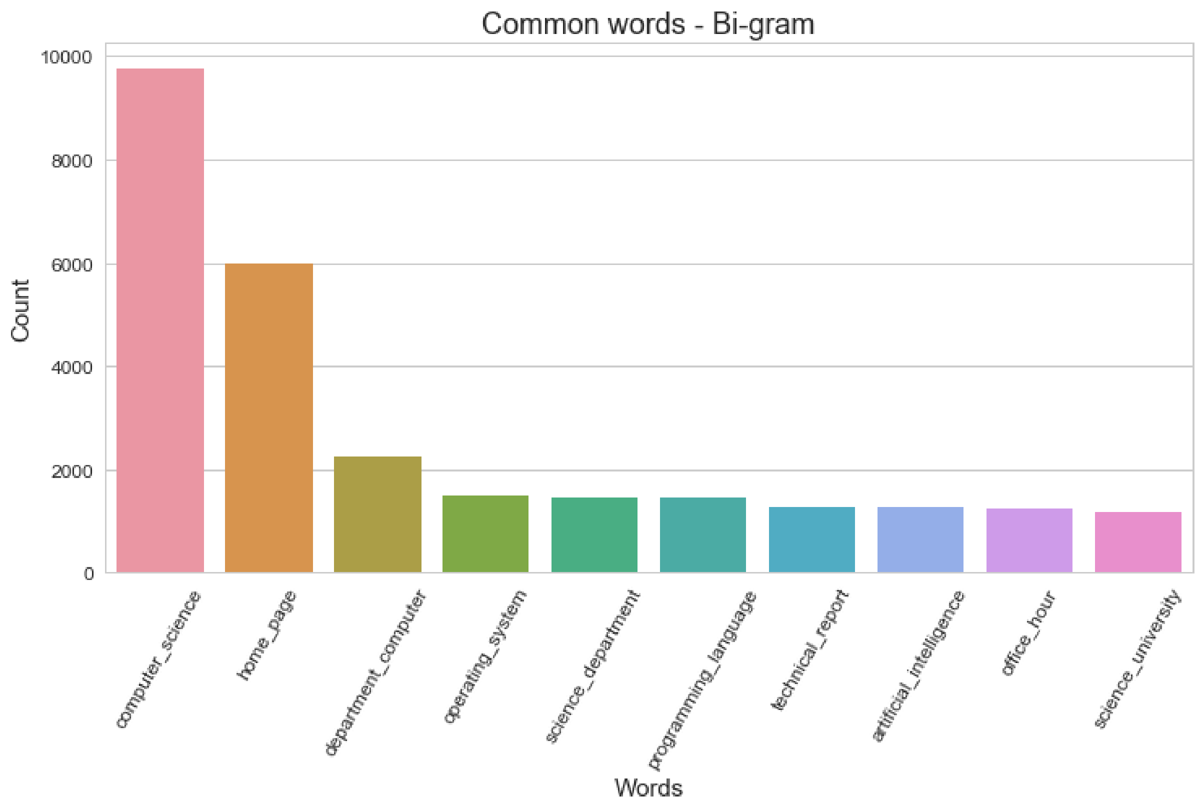


Fig 5. The most common bi-gram words in the data

The above graph shows the most common bi-gram words in the data, we could see that "computer_science", and "home_page" dominating the bi-gram.

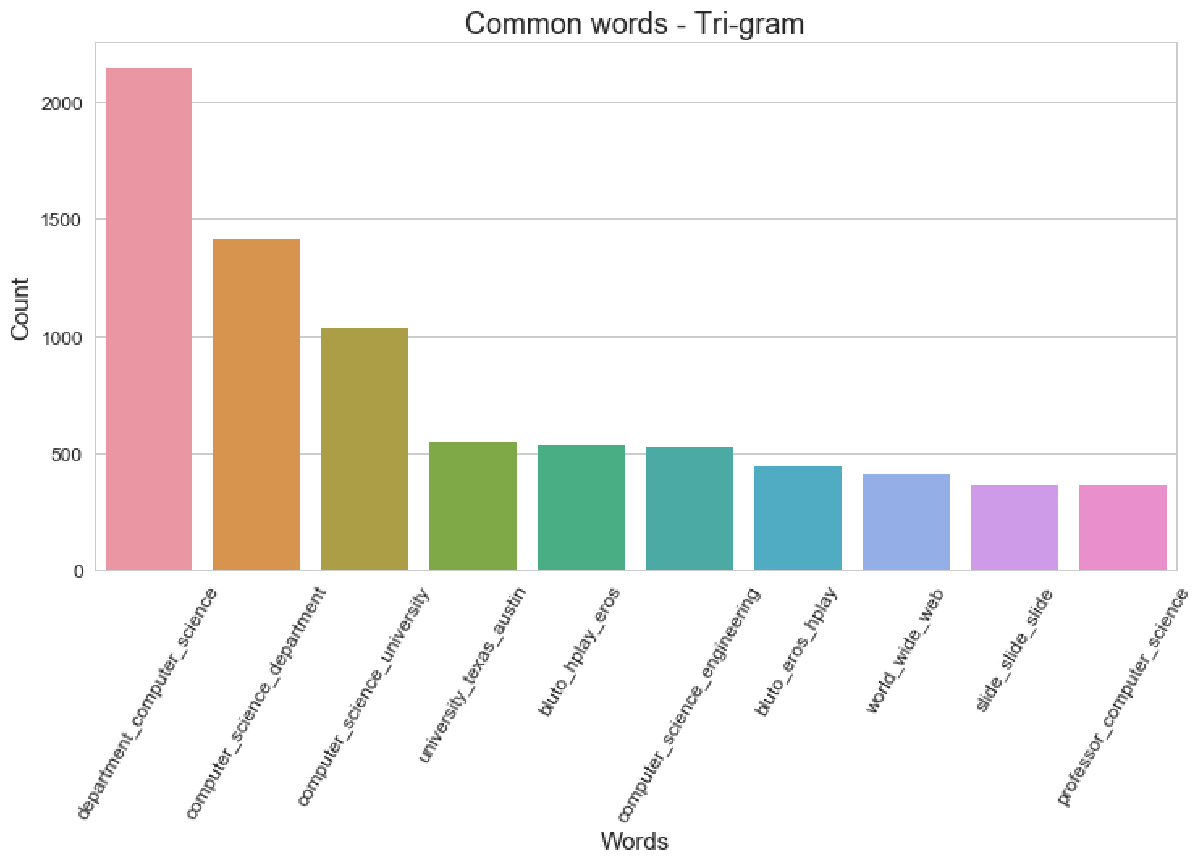


Fig 6. The most common tri-gram words in the data

It's obvious that the word count of tri grams is way lower when compared to uni-gram and bi-gram words.

So, combining all the words that we have, we visualize them through the wordcloud as below

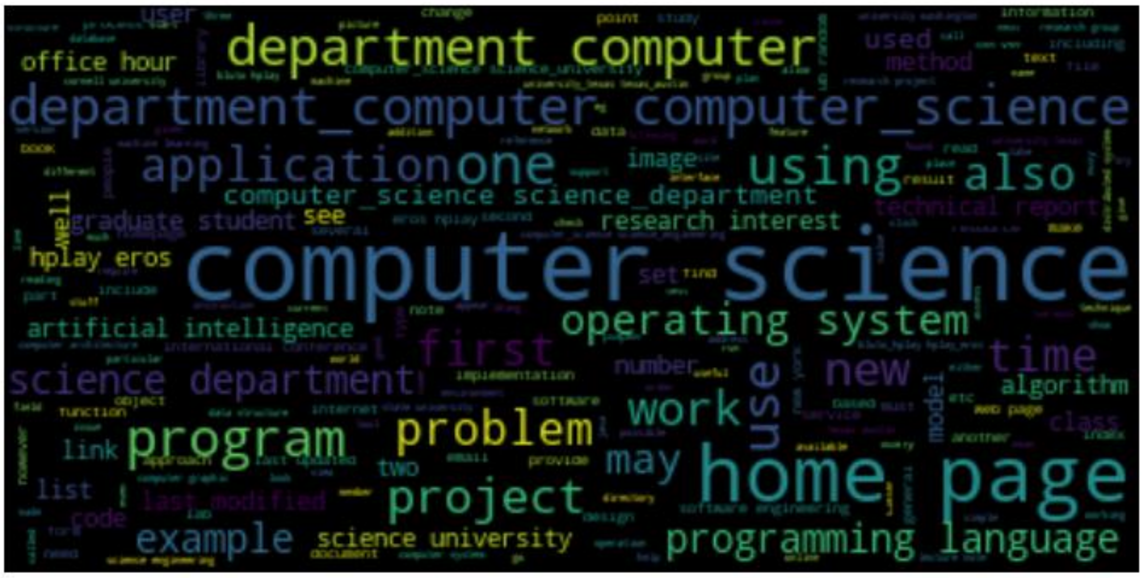


Fig 7. Wordcloud of the input data

From this basic common word analysis, we see tri-gram words aren't really making any useful contributions, so for our final feature set, we will include the bi-gram alone with the text data.

Splitting into train, and test dataset

For the purpose of train, and test, we split the dataset in the following ways

1. We consider all the documents from the "Wisconsin" University as our test data. So, we use the train data from other Universities, and tune parameters using cross validation with stratified split.

Feature extraction (TF-IDF)

The machine learning algorithms works with numerical input data. So, text data can't be directly fed to the algorithm.

A simple bag of words would be suffice enough to feed data to the algorithm, but this doesn't work well in most cases. So, a bit complex method TF-IDF (Term Frequency - Inverse Document Frequency) approach is used.

TF-IDF, not only considers the count of a word appearing in a document, but also its count across the whole corpus as well.

Scaling of data

Then, the data is scaled, so as to make computation much effective.

Feature selection

Once, we have all features converted into a numerical form, we need to find the optimal number of features for which the model gives best results. Our input data has 808161 features with 6964 rows.

Building a model with all features isn't a good idea because it's not guaranteed to give best results, and it's computationally expensive as well. So, using ANOVA, we select the optimal number of features.

Machine learning model

Finally, once we have the data ready, we feed to the algorithm to build the model. We use the Random Forest classifier.

The parameter, number of trees (estimators) for the random forest algorithm is estimated using the cross validation technique with stratified sampling of the data. The graph under **model selection process,** shows the result for various parameter values, and various number of feature values as well.

Metrics used

The dataset we have is multiclass, and unbalanced as well. So, we use weighted F-1 score for evaluation. The accuracy can't be a good measure, when the data is unbalanced, skewed towards a particular class label.

Model selection process

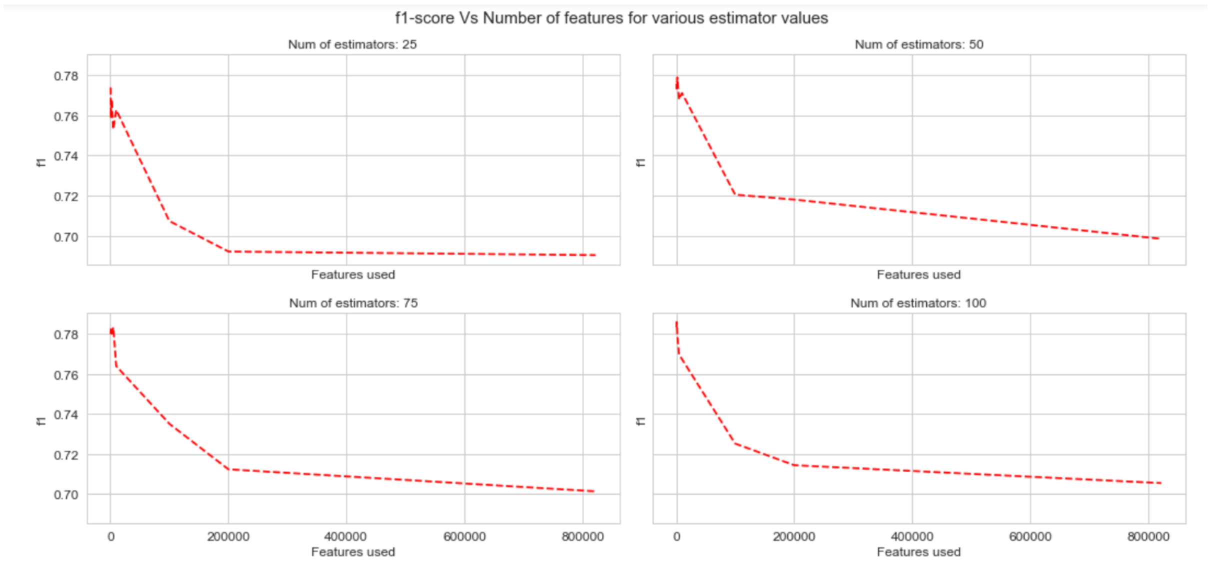


Fig 8. Model selection process with cross validation technique

From, the model selection process, we achieve best results with using top 1000 features, and using 100 number of estimators for the random forest classifier. The F-1 score is close to 0.78 for the best model.

We could see that the model can be further simplified down with 25 estimators as well, because the F-1 score, doesn't differ much. But, we will just go with the best estimator the model process gave.

Also, as we increase the number of features, the model performance starts to dip down. Also, the feature set has reduced from 808161 to 1000, which is a huge saving in terms of computation.

Validation curve

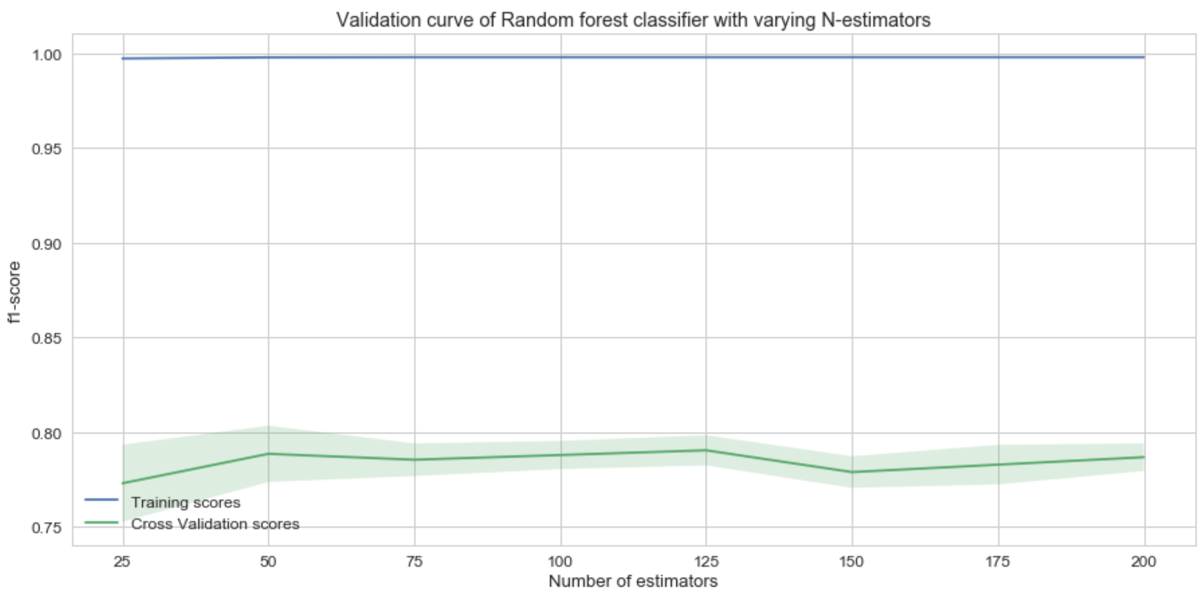


Fig 9. The validation curve of random forest model

The validation curve speaks how well the model performs with change in the input parameter (number of estimators) to the model.

We could observe that random forest almost has no training error, and performs best when the input is 100, and then it saturates.

Learning curve

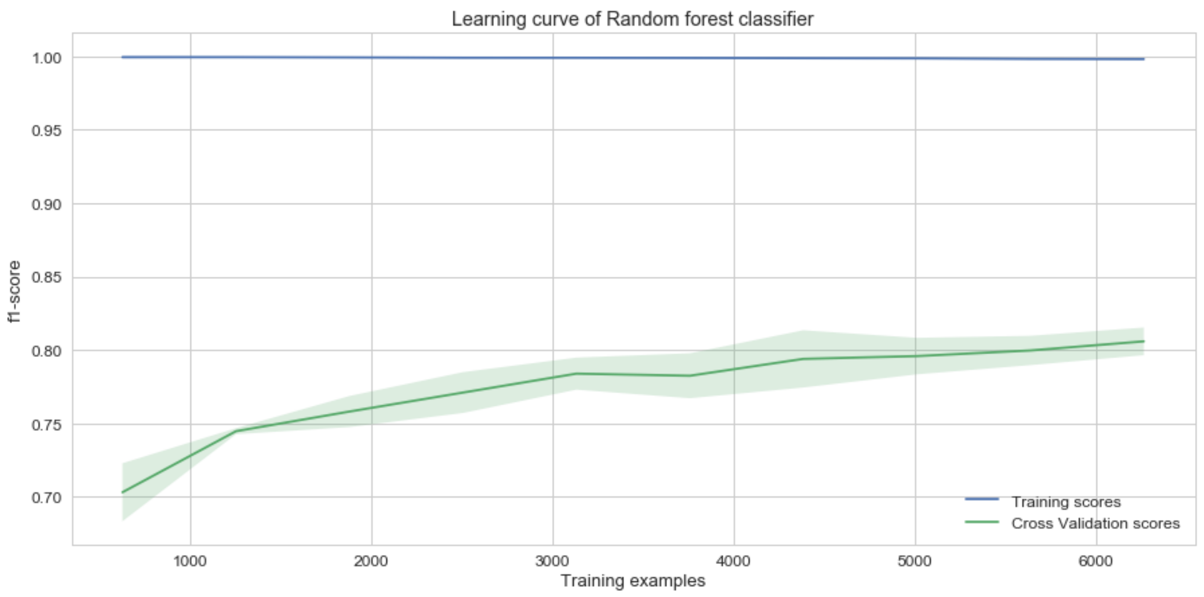


Fig 10. Learning curve of random forest model

The learning curve represents how the model learns with the input training data. We could see that as the training samples are increased, the model's test score improves.

This shows that, if we could have more data to feed in to the model, we could further improves it's score.

Optimizing the tree depth (Bias - Variance trade off)

Till now, we considered only the parameter of number of estimators for random forest model. There is another important parameter, the depth of the tree. We could have optimized this parameter during the model selection process as well. But, we take the bias – variance approach to optimize this.

We know in a tree, when the depth is 1 (decision stump), the model built is going to be under fitted and this leads to high bias, and as the depth increases the variance increases, and bias decreases.

So, in order to find the optimal depth value, we plot the bias-variance graph for the model using bootstrapping. We plot the results as below

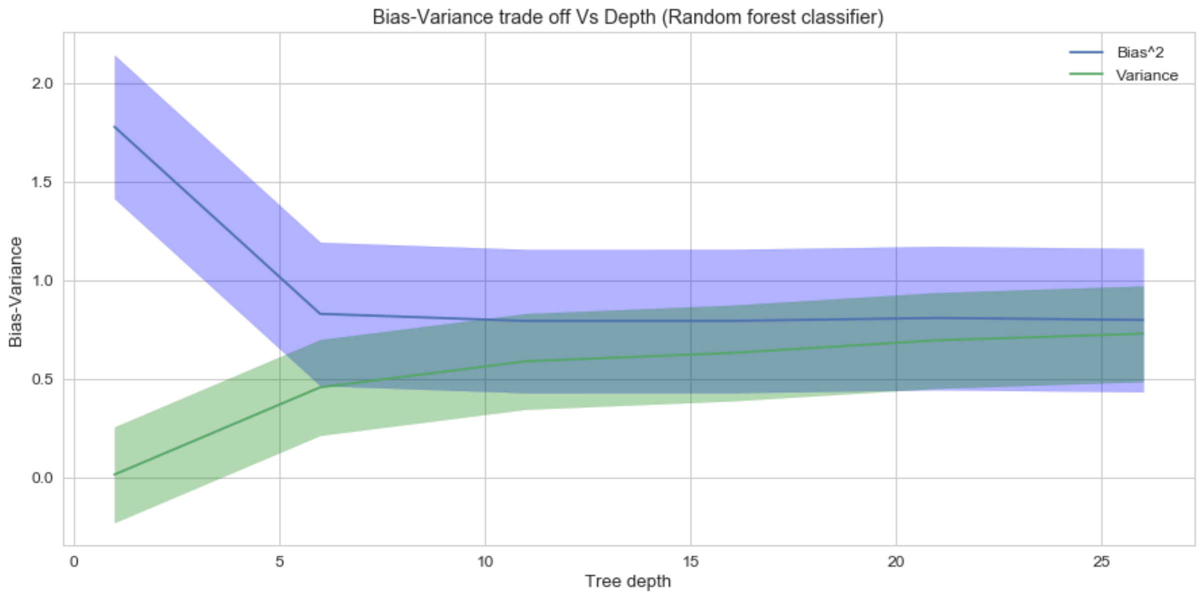


Fig 11. Bias-Variance trade off

From the graph, we see that when the depth is 1, the model has low variance, and high bias, and as depth increases the bias decreases, and variance increases. We pick the depth value of 5-6, so that the model performs at its best.

Building the final model

Based on the all the parameter tuning, we finally build the random forest classifier with parameters, number of estimators: 100, and depth: 5 with top 1000 features.

We build on the train data, and test on the Wisconsin University data.

Webpage classification using machine learning

Anirudh K. Muralidhar (anikamal@iu.edu)

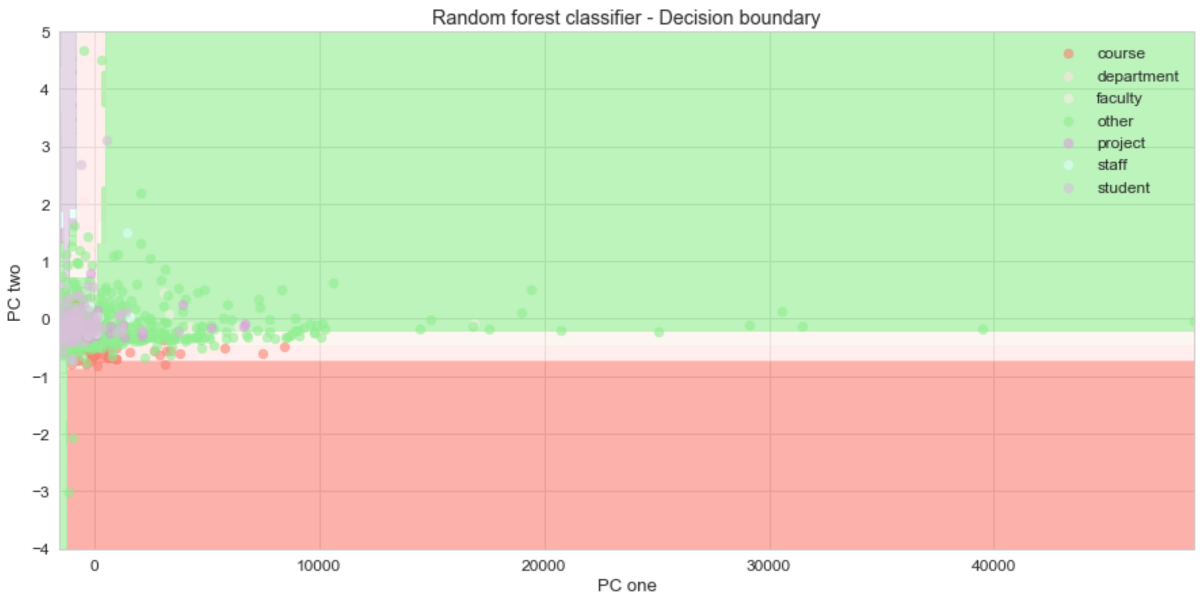


Fig 12. Decision boundary for random forest classifier

The graph, shows the decision boundaries of the random forest model, we convert the data to a 2 dimension using PCA, and then plot this graph. Each color on the graph corresponds to a particular class label.

We could observe that the “other” followed by “course” category dominates.

Evaluation metrics when model tested on test data (Wisconsin University)

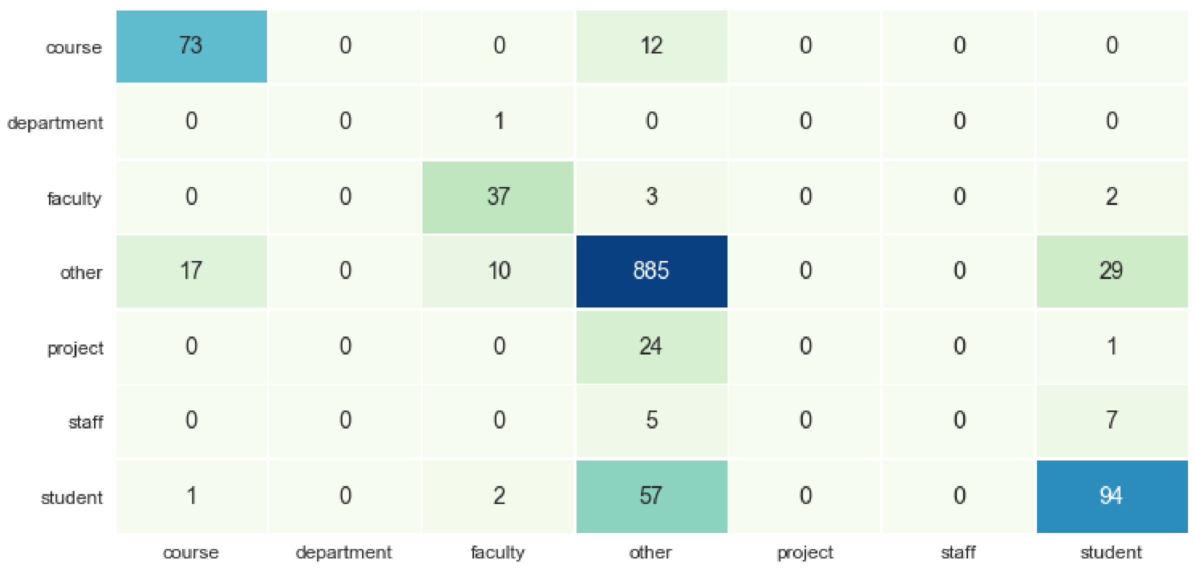
Overall Accuracy: 0.869

Overall weighted F-1 Score: 0.85

Label	**Precision**	**Recall**	**F1-Score**	**Support**
Course	0.79	0.82	0.80	85
Department	0.00	0.00	0.00	1
Faculty	0.88	0.86	0.87	42
Other	0.88	0.96	0.92	941
Project	0.00	0.00	0.00	25
Staff	0.00	0.00	0.00	12
Student	0.81	0.55	0.66	154
Avg / Total	0.84	0.87	0.85	1260

Table 1. Scores across labels

Confusion Matrix



	course	department	faculty	other	project	staff	student
course	73	0	0	12	0	0	0
department	0	0	1	0	0	0	0
faculty	0	0	37	3	0	0	2
other	17	0	10	885	0	0	29
project	0	0	0	24	0	0	1
staff	0	0	0	5	0	0	7
student	1	0	2	57	0	0	94

Fig 13. Confusion matrix

Conclusion

- The random forest classifier performed better as well tuned its parameters.
- Key observation would be, the model didn't do well with labels such as "department", "project", and "staff". From confusion matrix, we see most of them have been classified into "other" category. This might be because of less data available for those class labels.
- An interested choice would be to build a one vs rest classifier one this data, and further performing some intense feature engineering can further improve the model as well.

Further studies

Other interesting thing to study would be the social media data related to a website. This would be really useful for classifying websites than webpages though.

Most of the websites, has a social media account on Twitter, Facebook, and more. Analysis their posts, description can really give an idea of what exact category they belong.

Without social media, we don't know what text of the website to pick for analysis, but social media data provides us some concrete data to work, saves us on computation, and features to be used.

Once, we find out key words from the posts on social media, we could use the Wikipedia dataset, to categorize them into specific buckets. There are several APIs which facilitates in this process.

Let’s say for example, two electronics website, might have different website structures, so crawling their content can be bit different, but their social media posts are going to be of same structure, and when trained on one, it can be tested on other easily as well.

References

[1]. University data set - http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/.

[2]. Reference paper -
https://pdfs.semanticscholar.org/8dd8/8f4555ba56ca54bab923e1f4168759440bb4.pdf