Classification of Journal Abstracts into Categories

Kaggle Team Name: SaJaSha

Sayantan Datta
McGill University
260670123
sayantan.datta@mail.mcgill.ca

Shabir Abdul Samadh
McGill University
260723366
shabir.abdulsamadh@mail.mcgill.ca

Jaspal Singh
McGill University
260727323
jaspal.singh2@mail.mcgill.ca

Abstract

This projects investigates different classification methods to predict the categories of various research papers from scientific journals. We evaluate three different machine learning techniques to compare and contrast their suitability for the task of text classification. The data-set comprises of approximately 90000 *samples with each entry having a unique ID, the abstract of a scientific journal and the actual category the journal falls into. The categories are either* ***math, cs, physics*** *or* ***stat****. An additional test-set comprises of approximately* 15000 *entries with their categories being left out for prediction. We study the given abstracts/documents in the data-set to identify and extract suitable features to be used in the classification process. The report explains the feature engineering methodologies used and their outputs. We experiment and evaluate the accuracy & error for three different classifiers: Logistic Regression, k-Nearest Neighbors and Support-Vector-Machines(SVM). We produce our findings with respect to each classification above. Finally we explain and discuss the reasoning behind our design choices & the corresponding results.*

1. Introduction

Logistic Regression is a linear classifier where the dependent variable is binaray. The predictor output is probability of an example belonging to either positve or negetive class. In our case, since we have four classes, we use one-vs-all training procedure where we associate probabilties for each class with the all our examples and choose the one with highest probability. On the other hand k-NN reports the closeness of a test sample to other k data points in our training set. We select the class of our test sample as the most commonly occuring class among the k closest data points. SVM is a binary classifier with the modus operandi of separating the data points with widest possible margin. Again we use one-vs-all approach for SVM to classify our data into four categories.

2. Training Methods

The data-set comprised of approximately 90000 sample journal abstracts with their categories. An additional 15000 documents were presented on which the predictions were to be made. However, the predictions were to be submitted to the online learning platform *kaggle in class*[2] to validate its accuracy and error. Hence, we divided the given train-set into two portions of 80% (70000) and 20% (20000) except for Logistic regression where the split was $70\% - 30\%$. The 80% of the samples were used to predict the categories of the remaining 20%. Then we used our predictions on this 20% to validate the accuracy of our model. Once we had reached a level of satisfactory accuracy *(after testing with different parameters for the model)*, we used the complete set of 90000 samples to classify the original test-set of 15000 documents. We submitted the results of this predictions to the online learning platform to validate how good our predictions were.

We experimented the same with all three different classifiers and made our final submission to be the predictions from the classifier for which the best rate of accuracy was reached.

2.1. Logistic Regression

We used one vs all method for classification using Logistic Regression. More precisely, we train 4 models to classify

cs/non-cs, math/non-math, stat/non-stat and physics/non-physics pairs. We then choose the prediction with highest probability among the four models for reporting the final outcome. For example, if the probability reported by the four models are 0.2, 0.3, 0.1, 0.6, then we choose our final outcome as 'physics' since it has the highest probability among four. We start with cleaning our data using python NLTK package. For each abstract, we first lower case all text, remove punctuation, remove stop-words(using NLTK English dictionary) and finally stem the words using NLTK-Snowball stemmer. Both training and prediction data are fed through same pipeline.

2.2. k-Nearest Neighbors (kNN) Classification

kNN takes into account a distance *(nearness)* metric of a given test-entry against all the samples of the train-set. Then the $top - k$ train-set entries with the maximum distance/nearness values are filtered out. From the filtered set of k train-set entries we determine how many instances of each *class* has fallen within this top list. We then pick the *class with maximum number of occurrences* in the top-k list and assign it to the test-entry under consideration. It is noteworthy that the meaning of *distance* is a metric of closeness between the test-entry and each of the train samples. Thus, maximum distance above does not necessarily mean how far apart they are but the value of a score indicating their closeness. For a classification problem where this distance function is *Euclidean distance (ED)*, then the nearest neighbors will be decided based on the minimum values of the ED.

For the case of text classification it was required of us to decide on a correct measure for the distance/nearness function. The nearness metric *(between two abstracts or text documents)* we considered in this project was the number of matching words in both the documents. However, more analysis was done to assign an ideal nearness value ***(score)*** for each training-sample *(against the new abstract to be predicted)* without only taking the count of matching words between the two documents.

For every matching pair of words between a test-abstract and a train-abstract, we calculate the ***tf-idf***[1] of that word, to be the nearness measure contributing to the score of that training-sample. We then sum all such individual word scores per matching word and take the total as the *final score* of that specific training-sample against the given test-abstract. For every new abstract to be predicted, we do this calculation against each and every sample in the training set. Thus, we have a score metric indication of nearness of each sample in the training set to the new-abstract. We then filter out the training samples with the $top - k$ scores calculated in this fashion. Finally we determine the **abstract-category** *(cs, math, stat, physics)* with the most frequency in the filtered $top - k$ results and assign it as the category of the new-abstract. In our experiments we have kept k to be 10. We don't do any further experiments for different value of k given that we are able to achieve very good accuracy for predictions with $k = 10$.

In the occasional instance of more than one category being present in the $top - k$ set with equal frequency *(i.e: math:3, physics:3, cs:3, stat:1)*; we calculate the **total-score-per-category** by adding the individual scores of train samples in the $top - k$ for a specific category. We then assign the category with the *highest cumulative total score* to the test-abstract.

2.3. Support Vector Machines (SVM)

For SVM, we used the nltk toolkit *nltk*[1] to clean (preprocess) the dataset and then used the in-built packages of *scikit-learn*[4] to classify the abstracts. We used the one vs all method to train the dataset for 4 classification categories. Multiple experiments were done with the type of feature used, the number of features selected and the type of kernels to be used. We finally selected the one which gave the best accuracy in the validation set.

3. Problem Representation

3.1. Data Handling and Encoding

The data-set provided was 3 csv files. One of them was the *training-set* which had 88639 abstracts from research papers. The 2^{nd} one was the category of each of these abstracts in the training-set. A common **id** column was maintained in both documents to map the category in the latter csv to the abstract in the former one. The categories were one of the following 4: *math, cs, physics* or *stat*. A 3^{rd} test-set was also given which included another 15644 abstracts with **id's** *(these id's were not related to the ones in the previous csv files)*. The task at had was to predict the category of each of the abstracts in the 3^{rd} csv based on a model constructed using the abstracts in the training-set and their categories in the 2^{nd} file.

Before going into calculating the word frequencies, we cleaned all the abstracts to remove every unnecessary details as follows:

1. removed all punctuation marks and symbols.
2. removed the redundant *stop-words*[2]. The stop-words collection used here was that of the *nltk* language processing tool-kit.

[1] 'term frequency-inverse document frequency' (tf-idf) is a measure of importance and level of contribution of a word to a document from the collection of all possible words *(also known as the Corpus).[9]*

[2] words that are very common in the English language and those which occur in every document[8]. (ex: the, is, a, are and etc.)

3. *stemmed* all the remaining words to reduce them into their root form. Stemming reduces different patterns of the same word to its original form (by removing suffixes), hence allowing us to treat them all as one[7] *(ex: message, messages, messaging and messaged will all be treated as one word of the root form.)*

4. *lemmatized* the words to transform them to its *lemma*[3] form. It is different from stemming in that lemmatizing does a morphological analysis in converting the words whilst the former just does a crude reduction[3].

3.2. Feature Creation per Algorithm

3.2.1 Logistic Regression

From the cleaned data set, we choose the first 'n' most commonly used words as our feature set. The first 3 most frequently occurring words were removed from our feature set as they were far too common to make any difference with our prediction. The choice of n was kept as hyper-parameter. With increasing 'n', the trade off was between accuracy vs over fitting and execution time. We tested using n = 100, 200 and 500. The first big jump in accuracy was noticed when increasing number of features from 100 to 200. Similarly, increasing features from 200 to 500 also resulted in stellar increase in accuracy rates. However, we didn't try to increase the number of features further as it would take too long to compute and also increase the chances of over fitting.

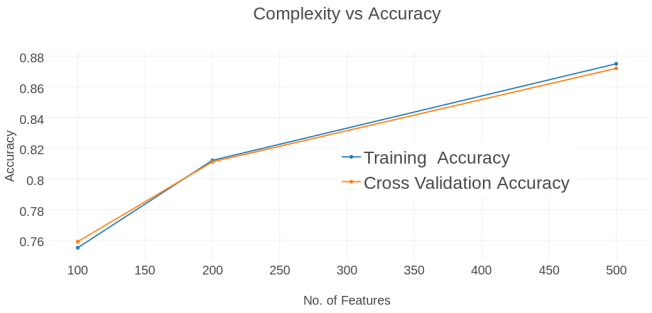

Figure 1: Complexity Vs Accuracy

Regularization parameter for Logistic Regression: For the choice of regularization parameter, we split our training data into 70-30 ratio and used L2 regularization. We trained our model using 70 percent of the data and varied lambda between 1e-1 and 1e4. For, all values of lambda, our training and cross validation accuracy remained almost same. This most likely suggests that the weights are already very small to warrant any significant change when using regularization.

[3]the dictionary or canonical form of a word[6]

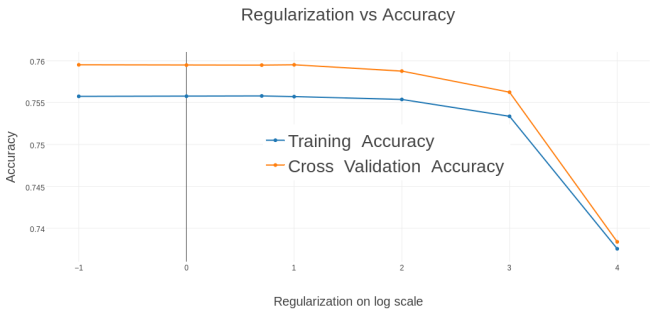

Figure 2: Regularization Vs Accuracy

3.2.2 kNN Classification

We extracted the distinct set of words/tokens that appears in any of the training-document by using the $CounterVectorizer()$ class of the *scikit-learn* toolkit. We set the *max_features* parameter of the above class to 10000 initially. This created a feature vector consisting of only the top 10000 most frequent words in the entire training set. We also did a second prediction cycle by setting this parameter to 15000 to see if we achieve greater prediction accuracy.

We then assigned the feature-value per feature *(per word/token)* in all the documents by calculating the **term_frequency-inverse_document_frequency *(tf-idf)*** of the word with respect to each of the document it appears in. Thus, for every abstract in the training-set the $tf - idf$ values per word in the feature vector was calculated. For a word that does not occur in a specific abstract in question, its feature-value is 0 for that abstract. The $tf - idf$ value per token per document was calculated using the *scikit-learn* tool-kit's $TfidfTransformer()$ class. The $tf - idf$ values are internally calculated as follows by the scikit-learn toolkit:

$$tf - idf(t, d) = tf(t, d) \times idf(t)$$

Here, t is the specific term/word under consideration and d denotes the document in which the term occurs whose feature-value for t is to be calculated. $tf(t, d)$ denotes the number of time the word t appears in the document d. Also, $idf(t)$ is calculated as follows:

$$idf(t) = \log \frac{n_d}{df(d, t)} + 1$$

In the above equation for idf, n_d denotes the total number of documents, whilst $df(d, t)$ represents the no.of documents in which the term t appeared.

3.2.3 Support Vector Machine

Both the word-frequency and the tf-idf feature extraction techniques were tried separately while implementing SVM.

Table 1: Features of a document

x_i	example $(tf - idf)$	representation
$word_1$	0.2408	continuous
$word_2$	0.2503	continuous
$word_3$	0.2446	continuous
...	...	continuous

With the word- frequency feature it was observed that the running time increased many folds, if the number of features crossed 900. It was computationally expensive to calculate the validation set accuracy for features above 900. However, selecting the tf-idf feature (generated by the TfIdfTransform funncion in scikit learn), it was possible to check the validation accuracy for higher number of features. For both the feature selection strategies 2 different kind of kernels (linear and Radial Basis Function (rbf)) were experimented with. The results are discussed in the next section.

4. Results and Discussion

4.1. Quantitative Results

4.1.1 Logistic Regression final prediction scores.

Based on our 500 feature Logistic Regression model, score on final prediction set as reported by Kaggle is 0.77. Also, computationally cheaper, 200 feature model scored 0.72 on Kaggle. Please note that this is neither our final model for submission nor our final score. The scores mentioned in this section are for reference only.

4.1.2 kNN Results

We opted to carry-out an initial prediction on a split-set of the given training data. The predictions on the above split resulted in an accuracy of 91%. Thus, the entire 100% of the given training set was used to predict the categories of the original test-set. These final predictions were submitted to the online *kaggle* system. The *accuracy-scores* of submissions to *kaggle* under several model complexities is shown in $Table2$.

Table 2: kNN - Scores for predictions

k	features	execution-time	score
10	10000	23 hours	0.80530
10	15000	25 hours	0.80491
20	5000	21 hours 30 mins	0.80568

In addition to the above submissions we also compared the prediction scores generated via the kNN implementation and also the *Random Forest* methodology of the *scikit-learn* package. The results of these submissions are reflected in $Table3$. k was set to 15 and the number of features considered to 5000, in both these cases.

Table 3: Scores for predictions with sci-kit learn

k	features	execution-time	score
15	5000	≈1 hour	0.79072
15	5000	≈12 hours	0.79749

4.1.3 Support Vector Machine

Below are the results obtained for the various features and kernels used for SVM classification. Note that the entire data set was divided into training and validation set of 70000 and 18639 respectively. Please refer the graphs present at the appendix for a visual representation of this table.3 & 4

Table 4: SVM Scores for Word-Frequency and tf-idf

$kernel$	features	score (word-frequency)	score (tf-idf)
$linear$	100	0.7622	0.7510
	500	0.8672	0.8728
	900	0.8748	0.8958
	2000	-	0.9101
	5000	-	0.9161
	10000	-	0.9203
	15000	-	0.9207
rbf	100	0.7644	0.7439
	500	0.8592	0.8164
	900	0.88749	0.8993

With word frequency as the feature, it was not possible computationally to increase the number of features above 900. However using tf-idf as the feature and sparse matrix representation, it became possible to increase the number of features. Since the dimension space is huge it was expected for linear kernel to perform better than RBF[5], as evident in the results. Therefore the further tests were only performed using the linear kernel. The increase in accuracy after 10000 features was minuscule, and thus the prediction on the final test set was made with 15000 features, which scored 0.81488 on kaggle. The precision, recall and the f1-score parameters for the validation set are listed in Table 5

4.2. Discussion

The classification task at hand was experimented with four different classification algorithms - Logistic Regression, k-

Table 5: Performance Metrics for SVM

Category	Precision	Recall	f1-score
Math	0.95	0.96	0.95
Physics	0.92	0.92	0.92
Stat	0.90	0.87	0.88
CS	0.91	0.92	0.91

Nearest Neighbors, Support Vector Machines and Neural Networks. Predictions with first three algorithms were submitted to the *kaggle* platform. And the predictions from Neural Networks was used in for an **ensemble** prediction of all algorithms. However, the best prediction scores (for the algorithms which were coded) in terms of the $80 - 20$ split of the training set and the original test set was observed for the k-Nearest Neighbors classifier. It was observed that the prediction score was increasing with k. The higher the number of closely related samples taken into account for prediction, the better the results were. One important point in terms of increasing k was that, this reduced the chances of an abstract being closely related to equal number of training samples for more than one specific category. A higher value for k always ensured that one specific category out of the four occurred as majority among the filtered k samples. Hence, it made the final classification step straight forward.

Another important metric in terms of the classification problem was the number of features to be considered. In this problem the features were words in a given abstract. More specifically it was the *tf-idf* values of these words. It was noticed that there was no significant increase or change in the score by increasing the no of features/words taken into consideration for the model. The only advantage in terms of tuning the feature count was that we were able to achieve better execution times for the predictions. By reducing the number of features the predictions over the entire test set could be completed at a much lesser duration. However, it was observed that by having a higher and a richer set of features/words taken into account we could assure that for every matching pair of words between the test-sample and the train-sample, that this word was present in the feature set. Reducing the feature-words meant that for some of the matching words between the test and train set, it was possible that this word is not present in the feature set. Hence, this affected how this train sample was scored for the given test abstract.

In addition we also studied the total number of abstracts in the training set against which no matching pair of words *(i.e:matching pairs of words between the two* $= 0$*)* were found per test-abstract. We noticed that by having a rich feature/word set we could reduce this count. I.e. having more words in the feature set meant that the chances of not even a single pair of matching word being found between a test abstract and a train-abstract was minimized. This ensured that more of the samples actually contributed and were taken into consideration in the overall prediction.

4.3. Limitations of approach and findings

4.3.1 kNN Classification

One of the key limitations of the kNN classifier is that the entire training data is to be evaluated for every new prediction to be made. Even-though kNN has the benefit of 0 training time, it is at the expense of higher prediction times. This resulted in a single prediction over the entire test-set to run for long hours. Also having to submit results to the *kaggle* platform restricted the amount of validation we could do on the predictions.

4.4. Execution Time & Computation

Given that the data-set is a compilation of almost 90000 documents with the average word count per document being $200 - 250$ words, the number of features involved is quiet high; so is the execution time taken for predictions. The initial pre-processing step where the training set is cleaned, tokenized and word frequencies are calculated takes up considerable execution times. We had reduced the number of tokens *(words in the entire corpus)* taking into account only the top most frequent words that were encountered in any training-set document. Hence, the training involves processing over a $2D$ matrix with approximate dimensions of $90000 \times features$. With this setting each prediction took almost $5 - 6$ seconds.

5. Conclusion: Bagging for improvement

Our final submission to Kaggle is a combination of different predictors using bagging procedure. We used 5 different predictor output namely LR, K-NN, SVM, Naive Bayes and NN to select the majority class as final prediction.

Table 6: Score for individual algorithms

Algorithm	Kaggle Score
kNN	0.80568
Logistic-Regression	0.77435
SVM	0.81488
Naive Bayes	0.80210
Neural Networks	0.79852
Ensembled Prediction	0.81705

The final output after bagging of predictions with 5 algorithms improved our score by 0.003. Even-though this may seem too small an improvement we were able to go up by few positions in the leader-board on Kaggle.

5.1. Contributions

We hereby state that all the work presented in this report is that of the authors. Each author played a role in implementing a specific classifier out of the three. SD implemented the Logistic-Regression model whilst SAS implemented the kNN algorithm for text classification; JS worked on the predictions via SVM classifier and NN; all authors contributed to the approach, design of the feature sets and the written report.

5.2. Acknowledgments

We have used the *scikit-learn*[4] machine learning tool-kit libraries to achieve various tasks in the implementation of the above algorithms. We have used the $CounterVectorizer()$ class to generate the feature matrix from a list of *cleaned documents*. Also we have used the $TfidfTransformer()$ class of this tool-kit to calculate the tf-idf values of each word/token in the feature matrix.

We have also used the *nltk*[1] language processing tool-kit to clean the abstracts in the training-set & the test-set before feeding them into the algorithm. The $SnowballStemmer$ and $WordNetLemmatizer()$ are used to stem & lemmatize the cleaned words. Also the *stopwords* dictionary of nltk is used to identify and remove all stop words from the documents.

5.3. Appendix

The category classifications for the given journal abstracts are provided in a separate file. The following figures reflect the scores obtained for SVM classifier with two different feature sets: *Word Frequency & tf-idf values*. It is a graphical view of $Table$ 4

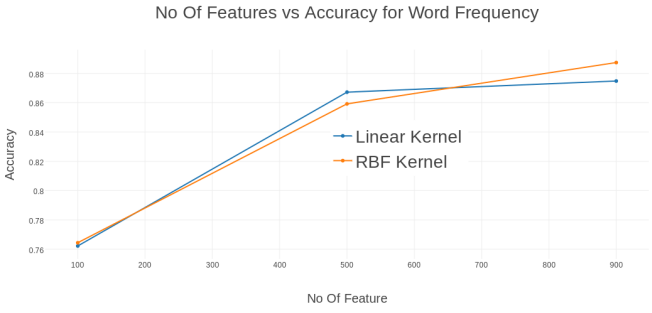

Figure 3: SVM Score with Word Frequency as features

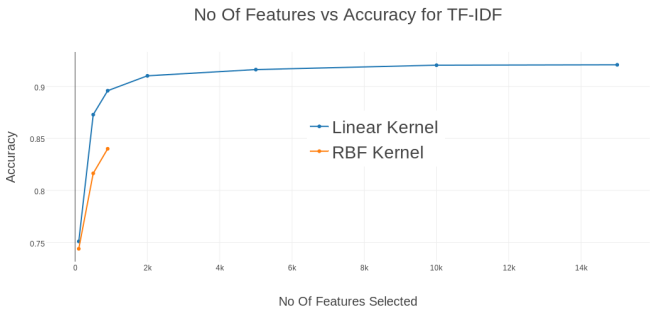

Figure 4: SVM Score with tf-idf as features

References

[1] Bird, Steven, E. Loper, and E. Klein. Natural language processing with python. *OReilly Media Inc*, 2009.

[2] C.-. A. M. Learning. Abstract classification 2016, 2016. [Online; accessed 02-October-2016].

[3] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Stack-Exchange. What are the limitations of kernel methods and when to use kernel methods?, 2016. [Online; accessed 14-October-2016].

[6] Wikipedia. Lemma (morphology) — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-October-2016].

[7] Wikipedia. Stemming — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-October-2016].

[8] Wikipedia. Stop words — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-October-2016].

[9] Wikipedia. Tfidf — wikipedia, the free encyclopedia, 2016. [Online; accessed 3-October-2016].