ArXiv abstract subject prediction: Naive Bayes, Decision Trees, and SVM approaches

THE PREDICTORS
Dimitrios Gallos, dimitrios.gallos@mail.mcgill.ca, 260498740
Arnaud Massenet, arnaud.massenet@mail.mcgill.ca, 260608613
Carlos G. Oliver, carlos.gonzalezoliver@mail.mcgill.ca, 260425853

I. INTRODUCTION

The goal of this project is to use machine learning algorithms in order to analyze abstracts, and classify them according to their topic. The four possible topics in our dataset are computer science, math, statistics and physics. We are provided a labeled training set and an unlabeled test set. The classification predictions are to be uploaded to Kaggle in order for our results to be ranked versus other groups in the class. We chose three different types of classifiers to make prediction on the data : Naive Bayes, Decision Trees, and Support Vector Machines. The first two were implemented by the team, while for the latter we used the scikit-learn implementation [4].

II. RELATED WORK

The process of text classification can be broken up into four parts: document representation, feature selection, constructing a classifier and evaluating a classifier.

The most common method of document representation in the literature is bag of words [7]. In this model, the text is represented as the set of the words it contains, disregarding word order and maintaining multiplicity. A widely used approach to limit the influence of common words on the BOW representation is to remove common words and perform stemming [14].

Subsequently, feature selection is performed order to further reduce the dimension of the bag of words representation. Such methods use an evaluation function applied to each word. Examples of that are : Document Frequency (DF), Term Frequency (TF), Mutual Information (MI), Information Gain (IG), Odds Ratio (OR), CHI- square statistic (CHI) and Term Strength (TS) [14]. The aforementioned techniques rank features by assigning them a score and only keep the highest scoring ones as features. Following that, a vector space model is constructed, with one vector for each document. The weights of the vector are usually determined using TF-IDF.Some common classifiers used for text classification are Neural Networks, SVMs, Naive Bayes and Logistic Regression [3]. Until recently, Gaussian-kernel SVMs achieved state of the art performance in text classification [15]. However, in recent years they are being surpassed by Convolutional Neural Networks [16].

The most common metrics used in order to evaluate classifiers in text classification are Precision and Recall. The F1-Score which is a combination of Precision and Recall is also used often. In general, the higher the Precision the lower the Recall. The point where the recall equals the precision is called the Break Even Point, which is used to compare evaluation results [14].

III. PROBLEM REPRESENTATION

A. Data Preprocessing

The data we were given consisted of a `csv` file containing strings representing the text of over 88000 abstracts. We used the `nltk` toolkit [5] in order to remove punctuation and stop words. We also used a stemmer in `nltk` in order to remove morphological affixes from words. The stemmer maps words that have slight differences for grammatical reasons to the same stem. We chose to use the implementation of Porter's algorithm for our stemmer [6], because it has proven to be very effective in the past and it is the one that is most commonly used in text classification. The results of the data pre-processing were then stored in a `csv` file in order to avoid re-computation.

B. Feature Selection

The feature encoding we used initially in order to train our models was bag of words. We also experimented with using time frequency-inverse document frequency in order to normalize the features created by bag of words. The rationale for using TF-IDF was that there are some rare terms specific to the category that need to be emphasized. TF-IDF emphasizes those terms by scaling down words that are frequent across all documents.

In **Table I** you can see the mean training and validation set error with using TF-IDF versus not using TF-IDF for our linear SVM classifier. Not surprisingly, the mean validation score with using TF-IDF increases significantly. It is interesting to note that the training score without using TF-IDF is almost 100%, while it drops when using TF-IDF.

TF-IDF Used	Mean Validation Score	Mean Training Score
TRUE	91.87%	98.02%
FALSE	89.78%	99.81%

TABLE I
IMPACT OF TIF-IDF NORMALIZATION ON TRAINING AND VALIDATION SCORE USING AN SVM.

IV. ALGORITHM SELECTION AND IMPLEMENTATION

A. *Naive Bayes*

We chose Naive Bayes for its simple implementation and good expected prediction accuracy relative to its complexity. Since we are classifying multi-category texts, we are using a Multinomial Naive Bayes. Naive Bayes is a good baseline with which we can compare our more complex classifiers.

1) Hyper-parameter setting: For each selected feature, we compute the probability $P(x_i|c_i)$, where x_i are the word frequencies and c_i are the categories. However, in order to further reduce the number of words to process, we decided to select words based on their frequency in each class. For each class we decided on a limit on the number of words we compute the conditional probability for each class In order to evaluate the best possible number of words to use we fixed the smoothing parameter α at 1.0 and then performed a 10-fold cross validation on the training data using different limits for the number of selected features **Table II**.

Feature number Limit	Average Error percentage
100	39.98%
500	21.74%
1000	18.83%
2500	14.10%
3000	13.56%
4000	12.98%
5000	12.65%
6000	12.59%
7500	12.60%
10000	12.85%
11000	12.97%
15000	13.58%
20000	14.37%

TABLE II
ERROR ON NAIVE BAYES TRAINING METHOD OVER VARIOUS FEATURE NUMBER LIMITS.

It is clear that for a fixed $\alpha = 1$ the optimal limit lies between 6000 and 7500. Using different α values could alter the results described in **Table II**. For this reason, we computed the *Error percentage* of many combination of alphas and feature number limits **Fig 1**. The results here change quite a bit, in fact we can see that the best combination is reachable with $\alpha = 0.1$ and feature number limit = 20000. The 10-fold cross validation gave an average error-percentage of 11.9% which means that on average 88.1% of the predictions were correct. Something interesting we can observe is the evolution of the average error percentage when we increase the feature number limit and the α at the same time. It seems that for small limits (less than 2000) higher the values of α, limit the average error percentage. After this, the optimal value for α starts decreasing until it reaches 0.1 when our feature number limit is 20000.

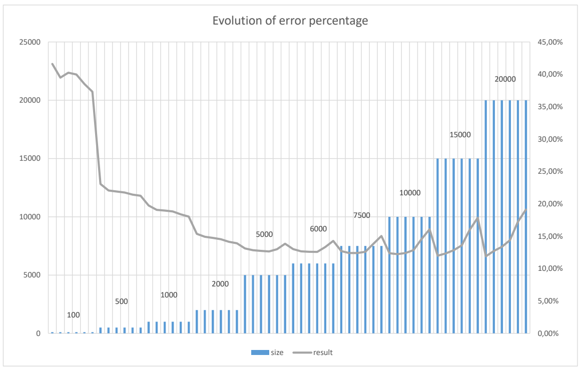


Fig. 1. Error Percentage Graph for the Naive Bayes classifier. We represent the evolution of the error with respect to the feature number limit and the alpha. The alpha is not represented, but goes from 0.1 on the first bar of a column to 3.0 for the last bar of the column.

B. *Decision Trees*

As our second approach, we constructed a decision tree, which is a commonly used tool in text classification tasks [3]. The decision tree was grown recursively, creating decision nodes based on optimal information gain over a set of features. For this approach we use the labeling methods from the `nltk` library to identify grammatical context of each word and isolate only the nouns, this was done in an effort to reduce computation time and we show that it has a positive impact on performance **Fig. VII**. We use a set of words as features, where each feature for a given abstract can take a value of 1 if present or 0 if absent. Nodes were added to the tree until a stopping criteria was met, at which point the current node is assigned to be a leaf with the majority category as its label, and the recursion is stopped. The resulting tree can then be used to classify unlabeled abstracts based on their word content. There were a some important considerations to be made when constructing the decision tree over this particular data set.

Tree Construction: While the decision tree algorithm in theory does not impose many hyper-parameters, given a time limit and computation power constraints, we had to control the depth of the decision tree. We found that the data set was much too large to compute a full decision tree that reduced to pure nodes (containing only examples from one category), and that used the set of all words (features) at a given node for creating the children node. Therefore, we had to choose the stopping criteria for creating a leaf to limit the depth of the tree. This was defined as the maximum entropy allowed at a terminal node, we call this quantity ω. We also had to limit the maximum number of features considered at each node in order to reduce computation time, we call this quantity ϕ. Features at each node were sorted according to frequency, so we would use the ϕ most frequent features for partitioning the data. This is equivalent to implementing a pre-pruning of the tree. We therefore perform a validation step where we consider a set of possible ω and ϕ values to select the optimal combination of parameters. A final constraint we had to impose on the trees was the size of the training set, which greatly influenced the computation time of the learning phase. We took the largest feasible training set for our computation resources which consisted in a training set of $20,000$ labeled abstracts. While these choices will likely result in sub-optimal prediction power, it is the best we can do given the computational resources and time constraints.

C. Support Vector Machine

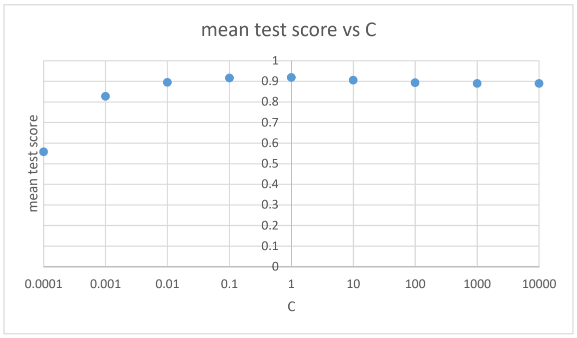
We decided to consider the scikit-learn implementation of both a liner and Gaussian-kernel SVM. The main advantage of using a linear kernel is that evaluation is much faster, thus we can evaluate more hyper-parameters in order to find the optimal ones for this problem. Furthermore, linear-kernel SVMs tend to have strong performance when the feature space is large. This is because one may not need to map data on a higher dimensional space [8]. However, the predictive performance of a non-linear kernel is typically better or equal to that of a non-linear kernel, especially if the feature space is not linearly separable [9].

1) Hyper-parameter setting: The main hyper-parameter needed to be set in scikit-learn's implementation of linear SVM is C. The parameter C represents a trade-off between the miss-classification of training examples versus the simplicity of the decision surface. A high C tries to classify all training examples correctly, while a low C tries to create a more smooth decision surface at the expense of miss-classification of training examples. Thus, a high C can make the classifier more susceptible to over-fitting.

We determined the optimal C by performing multiple sets of 3-fold cross validations and examining the testing error and validation error. In the beginning, we tried to determine the scale of C so we tried values from 10^{-4}) to 10^4 that differed by one order of magnitude between them. As can be seen in **Fig. 2**, as C increases, the training score increases. However, the validation score starts to decrease at around 1. Next, we further explored the values of C around 1 and determined the optimal value of C to be 0.6. The results for this investigation can be seen in **Fig. 3**

The main hyper-parameters needed to be set in scikit-learn's implementation of gaussian-kernel SVM are C and γ. The parameter C is the same as the one in the linear-kernel. γ is the parameter of the Gaussian-kernel that handles non-linear classification [4]. The larger the γ the smoother the curve in the higher dimension. Thus, a low γ gives you low bias and high variance, while a high gamma gives you high bias and low variance.

In order to determine the optimal parameters for the Gaussian-kernel SVM we performed a 5x5 exhaustive grid search on 5 values of C and 5 values of γ. The results can be seen in **Table III** and **Table IV**.It can be seen, in the case of a high γ and a C that is not small, the classifier overfits on the training data, as training score is close to 100% while validation score is just a little better than chance. On the other hand when γ is low the training score is also low, meaning the classifier cannot find the pattern in the data. The sweet spot seems to be when C = 10 and γ = 1.



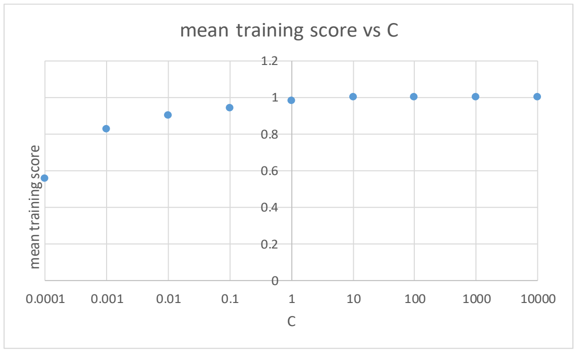


Fig. 2. Effect of C setting on validation and training score.

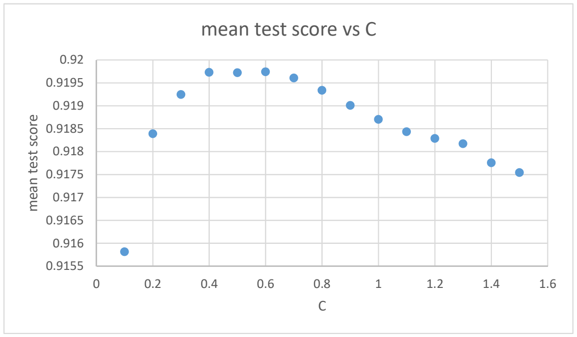


Fig. 3. Effect of C setting on validation score at around C =1.

Training score % over gamma and C						
		γ				
		0.001	0.1	1	10	100
C	0.1	31.7	86.8	90.5	31.8	31.7
	0.9	43.1	92.6	98.1	99.8	99.8
	1	48.5	92.9	98.4	99.9	99.9
	1.1	52.3	93.0	98.7	99.9	99.9
	10	87.6	97.9	99.9	99.9	99.9

TABLE III

TRAINING SCORE VARIATION WITH DIFFERENT VALUES OF γ AND C

Validation score % over gamma and C						
		γ				
		0.001	0.1	1	10	100
C	0.1	31.7	86.7	88.9	31.7	31.6
	0.9	44.3	90.9	91.1	31.9	31.8
	1	49.2	91.1	91.9	32.0	31.1
	1.1	52.9	91.1	92.0	32.1	31.8
	10	87.4	91.8	92.1	32.1	31.7

TABLE IV

VALIDATION CORE VARIATION WITH DIFFERENT VALUES OF γ AND C

V. TESTING AND VALIDATION

Category	Precision	Recall	F1-Score
math	78%	77%	85%
cs	82%	79%	80%
physics	84%	88%	79%
stats	81%	87%	81%
overall	82%	82%	82%

TABLE V

PRECISION, RECALL AND F1-SCORE FOR NAIVE BAYES

A. *NAIVE BAYES*

From the `scikit-learn` validation performance metrics, we show that the Naive Bayes approach is a good predictor overall. We achieve precision, recall and F1 measures of 82% which is well above a random classifier and near the expected accuracy for a naive bayes text classifier [3]. We submitted some attempts to Kaggle using the Naive Bayes classifier and achieved an accuracy of 78% which is a small decrease, meaning that our model is fairly good for generalization.

B. *DECISION TREES*

In order to choose the optimal entropy allowance (ω) and number of features used for decision nodes (ϕ), we perform a validation step with a training set of $20,000$ and the rest of the abstracts are used for the validation accuracy (**Fig. 4**). We note that the higher entropy thresholds, meaning that the trees have less nodes, or pruned more heavily, perform best on the validation set. This is evidence of the phenomenon of over-fitting when nodes are forced to be fully pure. We also note that the number of features used has a lower impact on accuracy, therefore we may be able to afford training on less examples in exchange for faster computation speeds. Finally, we note that while we are well above a random predictor (25%), the overall accuracy is rather low (0.53 – 0.57), compared to other reported instances of decision trees for text classification [11]. This lack of validation accuracy is likely due to the limited size of training set.

Next, we evaluate various performance metrics on the tree with $\omega = 1.5$ and $\phi = 400$ using the scikit-learn.metrics module **TABLE VI**. Overall precision, recall, and F1 are at 60% meaning our predictor is fairly well balanced. However, looking closer at performance in individual classes, we see that recall is reduced in the physics and stats categories. This may be due to the distribution of words over different classes, and the depth of the tree may not have been sufficient to capture the variations in the stats and physics classes.

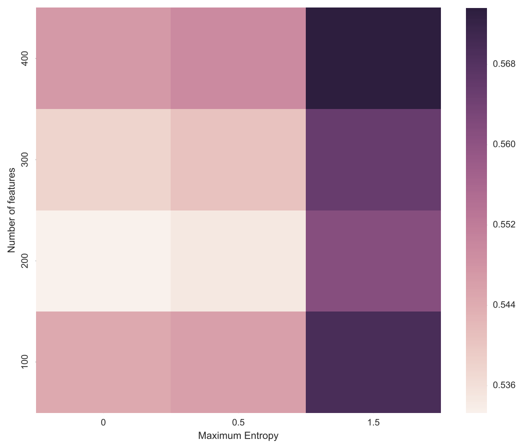


Fig. 4. Heatmap of accuracies on different entropy limits and used features.

Category	Precision	Recall	F1-Score
math	64%	79%	71%
cs	59%	61%	60%
physics	62%	46%	53%
stats	54%	46%	49%
overall	60%	60%	60%

TABLE VI

PRECISION, RECALL AND F1-SCORE FOR DECISION TREE, NOUNS ONLY

TABLE VII

PRECISION, RECALL AND F1-SCORE FOR DECISION TREE, ALL WORDS

Category	Precision	Recall	F1-Score
math	53%	79%	64%
cs	51%	47%	49%
physics	32%	46%	36%
stats	47%	32%	38%
overall	49%	50%	49%

C. *SUPPORT VECTOR MACHINE*

Once the optimal parameters were determined, a 10-fold cross validation was performed in order to validate the predictive performance of the classifiers on the validation set. The evaluation of both SVM predictors can be seen in TABLE VI and TABLE VII.

There are two interesting observations we can make from looking at the results. One is that both classifiers performed much better on the validation set, than on the test set on Kaggle. The difference was approximately 10%. The other one is that the linear-kernel SVM and Gaussian-kernel SVM have almost the same numbers for Precision and Recall across all categories. Nevertheless, it seems that we have reached a good Break Even Point with both classifiers.

Category	Precision	Recall	F1-Score
math	90%	92%	91%
cs	95%	96%	95%
physics	93%	92%	92%
stats	90%	86%	88%
overall	92%	92%	92%

TABLE VIII
PRECISION, RECALL AND F1-SCORE FOR LINEAR-KERNEL SVM WITH C = 0.6

Category	Precision	Recall	F1-Score
math	90%	93%	91%
cs	96%	96%	96%
physics	93%	92%	92%
stats	90%	87%	88%
overall	92%	92%	92%

TABLE IX
PRECISION, RECALL AND F1-SCORE FOR GAUSSIAN-KERNEL SVM WITH C = 10 AND γ = 1

VI. DISCUSSION

A. NAIVE BAYES

The major limitation of our Naive Bayes classifier lies in the method of deciding the number of features to consider. The Naive Bayes we implemented only used word-frequency in order to select the number of words. Perhaps a statistical method such as Mutual-Information or Chi-Square could have yield better results, as these methods are commonly used in the literature.

B. DECISION TREES

While decision trees were not our best performing approach, it is likely that an increased training set and deeper trees have promise for this application. Furthermore, an advantage of the decision tree predictor is that the resulting model can be easily interpreted and can provide insights into the structure of the data that can be readily understood by humans. If we were to improve something in this work, we would try to allow for a larger training set and full feature sets for node creation. Then, we would implement post-pruning methods such as reduced-error pruning which have shown better performance in similar tasks [12]. Lastly, we believe that other tree based methods such as random forests would circumvent many of the limitations in our approach [13].

C. SUPPORT VECTOR MACHINE

A major limitation of our SVM predictive power is the incomplete tuning of of parameters for Gaussian-kernel SVM. The reason for that is the big running time of this classifier. In our four-core machine, a parallelized grid-search involving a 5 by 5 grid takes approximately 25 hours. Due to the lack of time and computing resources we only performed a coarse search for both C and γ.

Lastly, the fact that the validation results for both Gaussian-kernel and Linear-kernel SVM are very similar needed to be investigated more. The lower score in the stats category leads us to believe that the data representation might be the reason. However, further investigation needs to be performed.

VII. STATEMENT OF CONTRIBUTIONS

Dimitri Gallos: Worked with linear and gaussian-kernel SVM, feature selection and report writing.
Arnaud Massenet: Worked on the Naive Bayes, data preprocessing and report writing. Carlos G. Oliver: Implemented the decision tree algorithm and conrtibuted to report writing.

We hereby state that all the work presented is entirely of the authors D.G., A.Y.M, C.G.O.

REFERENCES

[1] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze, Introduction to Information Retrieval, Cambridge University Press. 2008.

[2] Sebastian Raschka, Naive Bayes and Text Classification Introduction and Theory Oct 4, 2014

[3] Yang, Yiming. "An evaluation of statistical approaches to text categorization." Information retrieval 1.1-2 (1999): 69-90.

[4] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12.Oct (2011): 2825-2830.

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

[6] Willett, Peter. "The Porter stemming algorithm: then and now." Program 40.3 (2006): 219-223.

[7] Jindal, Rajni, Ruchika Malhotra, and Abha Jain. "Techniques for text classification: Literature review and current trends." Webology 12.2 (2015): 1.

[8] Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. "A practical guide to support vector classification." (2003): 1-16.

[9] Keerthi, S. Sathiya, and Chih-Jen Lin. "Asymptotic behaviors of support vector machines with Gaussian kernel." Neural computation 15.7 (2003): 1667-1689.

[10] Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47.

[11] Yang, Yiming, and Jan O. Pedersen. "A comparative study on feature selection in text categorization." ICML. Vol. 97. 1997.

[12] Mohamed, W. Nor Haizan W., Mohd Najib Mohd Salleh, and Abdul Halim Omar. "A comparative study of reduced error pruning method in decision tree algorithms." Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on. IEEE, 2012.

[13] Svetnik, Vladimir, et al. "Random forest: a classification and regression tool for compound classification and QSAR modeling." Journal of chemical information and computer sciences 43.6 (2003): 1947-1958.

[14] Jindal, Rajni, Ruchika Malhotra, and Abha Jain. "Techniques for text classification: Literature review and current trends." Webology 12.2 (2015): 1.

[15] Joachims, Thorsten. ”Text categorization with support vector machines: Learning with many relevant features.” European conference on machine learning. Springer Berlin Heidelberg, 1998.

[16] Zhang, Xiang, Junbo Zhao, and Yann LeCun. ”Character-level convolutional networks for text classification.” Advances in Neural Information Processing Systems. 2015.