Mini-project #2: Abstract classification

Koustuv Sinha - 260721248 - koustuv.sinha@mail.mcgill.ca - ksinha
Yu Luo - 260605878 - yu.t.luo@mail.mcgill.ca - YuTTT
Jeremy Georges-Filteau - 260713547 - jeremy.georges-filteau@mail.mcgill.ca - jgeofil
Team: StatBioComp

Introduction

This project will focus on a Kaggle competition for text classification. Each text to be classified is the abstract of a research paper corresponding to one of four categories: computer science, statistics, physics or mathematics. The goal of this project is to utilize machine learning algorithms and models to learn from the training dataset and automatically make predictions on the testing dataset. We implemented a few models such as: Naive Bayes, k-Nearest Neighbor and Logistic Regression. An SGD Classifier from the sciki-learn package (Python) was used to make our best prediction.

Related work

Many machine learning techniques have been successfully used in for text classification. Some examples include: Naive Bayes [1], support vector machines [2], random forests, logistic regression [3] and KNN [4]. The Naive-Bayes model is very often used as a baseline comparison method [6]. Applications of text classification are often related to scientific or technical domains where the number of instances is too large for human classification and there is a need simplifying search.

Feature preparation is often as important as the choice of classification algorithm. A common approach is the bag-of-words, where words in the text to be classified are counted out of context and their frequency is calculated. It has been shown that using Inverse Document Frequency (IDF) with this approach considerably improves performance [5].

Methods

Feature extraction

We chose TF-IDF or Term Frequency Inverse Document Frequency as a metric of vectorization of text features over the simpler Bag-of-Words model. The later counts the number of times a word appears in a document. In our case we need a metric that could account for relative global occurrences of words. The TF-IDF approach measures the relevance of a particular word. The first part, TF or term frequency, is the same as for bag-of-words. However, the IDF or Inverse Document Frequency takes into account the words which appear more commonly in all abstracts (such as “a”,”and”, “the”) and reduces their impact in the vector produced.

Before processing the abstract into the TF-IDF table, we first removed the numbers, punctuation and stopwords (i.e. a, an, the etc.) from them, changed all words to lowercase and stripped the whitespaces. Then, using a Lemmatization procedure we removed all inflectional endings from the words to return them to their base form, e.g.: offer, offered and offering will be lemmatized to offer. Finally, using the scikit-learn TfIdfVectorizer package, we vectorized the entire training and test set separately using the TF-IDF metric. We tweaked the vectorizer to extract one gram, bi-gram and tri-gram words, which creates features based on simple one word, two consecutive words and three consecutive word combinations. We found the n-grams approach in vectorization to work considerably better, because combinations of consecutive words provide contextual information pertaining to the subject of the abstract. Also we used sublinear tf scaling, ie replacing term frequency (tf) by 1 + log(tf), to smooth out the vector form. The resulting vector was then normalized for better fitting by the classifiers. The number of terms, or features, was limited to 40000. For some specific cases such as Linear SVM, Random Forests we limited the number of features by using Chi Square estimate to choose the best 1000 features.

Algorithm selection and implementation

Based on the litterature, we implemented two baseline classifiers that tend to produce good results for text classification. These are K-Nearest Neighbor and Complement Naive-Bayes, in Python on R, respectively. To produce competition winning we also tested and tweaked multiple models from the scikit-learn package. More details about this are given in the following sections.

K-nearest neighbor (KNN)

The K-nearest neighbor method was used as a baseline method to test classification. The distance metric was defined as the euclidean distance.

The key issue with KNN is choosing the number of neighbors. Cross validation was implemented and used on the training dataset to choose the number of neighbors among the set {1, 5, 10, 15, 20} which produces the smallest misclassification error. For each run, the test data was classified based on its the distance to the k-closest instances in the remaining training data.

Once the best number of neighbor was chosen, we used KNN on the training dataset to produce a confusion matrix to better understand how the algorithm performs and determine whether we should use this method to make the prediction for the final submission.

Complement-Naive-Bayes

A Naive Bayes classifier was also implemented as a baseline for the task at hand, as is common for text classification. Complement Naive Bayes was used, as opposed to Regular Naive Bayes, considering it usually fares better with uneven training data [1]. The distribution was modeled as TF-IDF. The complement was calculated according to:

$$\theta_{ci} = \frac{\sum_{j:y_j \neq c} d_{ij} + \alpha}{\sum_{j:y_j \neq c} \sum_k d_{ij} + \alpha}$$

Where d_{ij} is the TF-IDF frequency for word i in document j. The categories are represented as c and the output for each document as y_j. For any document t with raw word counts i, the output category was obtained such as:

$$C(t) = argmin_c \sum_i t_i \log \theta_{ci}$$

Cross-validation (with k=5) was used to evaluate the effect of the number of features on the error rate. Due to constraints on the processing time, only 10 000 entries were used for this test. Values ranging from 1000 to 30 000 were evaluated for the number of features.

A confusion matrix was also produced to better understand the distribution of the results produced by the model. For this purpose, 40000 entries were used for training and 10000 for validation.

Scikit library

For the third part, we used various scikit-learn methods such as Logistic Regression, Random Forest and Extremely Randomized Trees, of which the best performance was achieved was on Logistic Regression and Stochastic Gradient Descent Classifier with modified huber loss as the loss function. It has been reported by several practitioners [7] that Logistic Regression works best with fairly large dimensions, which is the case for text data. For our case we have normalized 40,000 features generated from TfIdfVectorizer, which we feed into the Logistic Regression classifier as well as SGD Classifier with modified huber loss. We used L-BFGS solver to optimize the Logistic Regression, which is a limited memory quasi newton method for convex optimization [8] and has been successfully used in other text classification challenges. [9]

For the SGD Classifier, we tried out various loss functions and found that modified huber gave the best results, very closely followed by Logistic Regression. Huber loss is another form of linear least square loss which is less sensitive to outliers in the data, and its modified version is used for text classification purposes. Mathematically, for a binary classification problem, the loss is defined as

$$L(y, f(x)) = \begin{cases} \max(0, 1 - y\,f(x))^2 & \text{for } y\,f(x) \geq -1, \\ -4y\,f(x) & \text{otherwise.} \end{cases}$$

Where f(x) is the given prediction score.

The hyperparameters were determined by exhaustive Grid Search over several choices. The gradient descent was run for 1000 iterations in both cases, with optimal learning rate for SGD Classifier, which adjusts itself based on distance to global optimum. L2 regularization was used in order to prevent the classifier from overfitting and the hyperparameter in regularization was also determined by Grid Search.

Results

K-Nearest Neighbor

Due to the memory constraints on the server (Yu implemented this algorithm in R), the implemented KNN algorithm was very computationally intensive. Therefore, we only used 962 features after removing 1% of sparse terms. The number of feature was reduced a lot, even specifying 1% reduction of sparse terms, because most of the abstracts have some uncommon words which will be regarded as sparse terms. Also, we only use 3-fold cross-validation to ease the computation.

This results for this method are not satisfactory. As shown in Figure 1, 10 nearest neighbors yielded the smallest misclassification error among the number of neighbor 1, 5, 10, 15, 20; however, the error rate was still very high probably because we shrunk our features by removing the sparse terms. Anyway, 10 nearest neighbors was used for the following analysis.

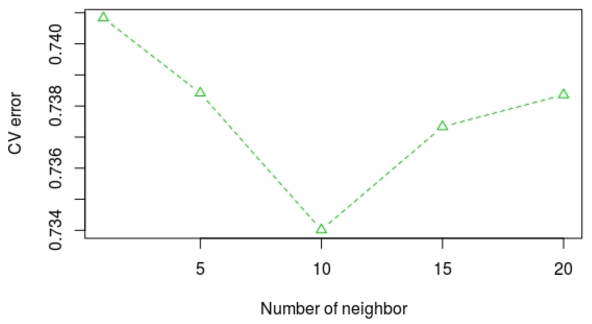

Figure 1 - K-Nearest Neighbor 3-fold cross-validation misclassification error rate versus number of neighbors.

Table 1 - Confusion matrix produced with the K-Nearest Neighbor model using 10-nearest neighbors to predict on the training dataset

Predicted class	Actual class			
	Stat	Phys	Math	CS
Stat	1214	1373	1888	2045
Phys	1897	1922	2877	3227
Math	5215	5834	8412	9223
CS	8316	8813	12744	13633
Precision	0.073	0.107	0.325	0.485

The 10-nearest neighbor method was used on the whole training dataset to make the predictions. The confusion matrix produced is presented in Table 1. From this table, we noticed that this 10-nearest neighbor method based on 962 features tended to wrongly classify the abstracts into the CS category, and is least likely to classify the abstracts into the Stat category. Furthermore, in CS category, only half of the abstracts are correctly classified with only around 48% precision.

To confirm that our implementation was not at fault for the high error-rate, we tested 5, 10 and 20-nearest neighbor in *scikit-learn (Python)* with around 2000 features. The misclassification errors rates were 41%, 42.3% and 42.6% respectively. The use of a library still did not yield much improvement, even though scikit-learn chooses the feature by its own rule instead of simply removing spare terms. We decided not to use this method for the final submission.

Complement-Naive-Bayes

The confusion matrix produced with the complement Naive-Bayes model is presented in Table 2. It is interesting to note that some misclassifications are considerably more common than others. This is the case for: Physics predicted as CS, Stat predicted as CS and CS predicted as Math. Furthermore, in each of these cases the reverse is not true, e.g: CS is not predicted as Stat considerably more than for the other categories. Precision rates for the four categories are similar, but Math is slightly higher.

Table 2 - Confusion matrix for the Complement Naive-Bayes model trained on a subset (n=40000) of the data and validated on another (n=10000)

Predicted class	Actual class			
	Stat	Phys	Math	CS
Stat	1573	59	31	80
Phys	40	1773	18	49
Math	45	49	2800	217
CS	211	168	64	2822
Precision	0.842	0.865	0.961	0.890

The result for the evaluation of the effect of the number of features retained on the error rate is presented in Figure 2. The error rate is reduce sharply from using 5000 features to 10000, but the gain is less important from 10000 to the maximum available of 22521.

When trained on the on the whole training set provided with 50000 features for submission on Kaggle, the Complement Naive-Bayes model obtained a score of 0.80184.

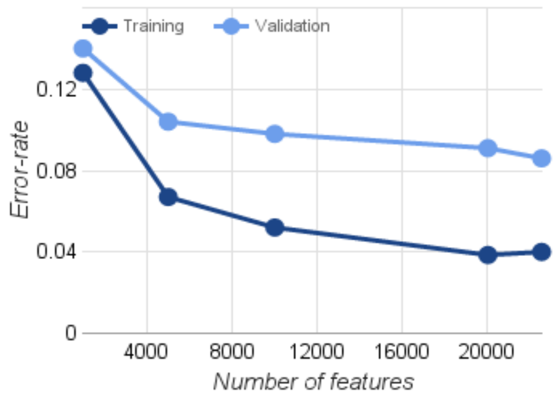

Figure 2 - Training and validation error-rate for the Complement Naive-Bayes model trained on a subset (n=10000) of the data with 5-fold cross-validation versus maximum number of features retained.

Scikit library

For the third part, we tested various classifiers from the scikit-library and finally submitted the result from the SGDClassifier. In this section we will briefly touch upon the results from other classifiers and then inspect the result from our final submission in detail. All results are taken over validation set. The hyperparameters used in all these classifiers are given in Appendix A. The results of each test are presented in Table 3. In each case the training dataset was split into training and validation portions according to a 75% and 25% ratio.

Table 3 - Accuracy, precision, recall and f-score for various models of the scikit-learn library training on 75% of the training data and predicting on the remaining 25%.

Hypothesis	Accuracy	Precision	Recall	f-score
Logistic Regression	0.9228	0.92	0.92	0.92
SGD Classifier	0.9286	0.93	0.93	0.93
Random Forest	0.8769	0.88	0.88	0.88
Extremely Rand. Tree	0.8921	0.90	0.89	0.89
Linear SVM	0.9022	0.90	0.90	0.90

We also tried out various ensemble methods such as Bagging, Voting, Adaboost on one or a combination of the above classifiers. The results of are presented in Table 4.

Table 4 - Accuracy, precision, recall and f-score for various ensemble models of the scikit-learn library training on 75% of the training data and predicting on the remaining 25%.

Ensemble	Accuracy	Precision	Recall	f-score
Adaboost - RF	0.8897	0.88	0.88	0.88
Bagging - ERF	0.8999	0.89	0.89	0.89
Voting - LR, NB, SGD	0.9273	0.93	0.93	0.93

Logistic Regression and the SGD Classifier with modified huber loss generated the best results. For Logistic Regression, we got C=1.25, (inverse of regularization function) and penalty l2, and for SGD Classifier we got alpha or regularization parameter to be best for 0.00004. The confusion matrix generated with respect to SGD Classifier is given in Table 5. When we submitted the results on the training set on Kaggle, SGD Classifier returned a score of 0.82805

Table 5 - Confusion Matrix for SGD Classifier with modified huber loss

Actual class	Predicted class			
	Stat	Phys	Math	CS
Stat	3677	110	56	265
Phys	101	4131	64	123
Math	53	64	6195	163
CS	354	165	163	6535
Precision	0.87	0.92	0.96	0.92

Discussion

K-nearest neighbor

The KNN method is a typical lazy learner method which does not rely on training the model but just stores the data. It is easy to implement but the query time is very slow when the number of samples and the number of features are large. Also, this method requires more memory for text mining, considering many words are unique to a single abstract and a feature has to be added for each of these words. A way to overcome this problem would be to make use of sparse matrices in the implementation.

Although this method is easy to implement, the results were not as good as the Naive Bayes classifier. In our case, only 28.4% of abstracts are correctly classified in our 10-nearest neighbor implementation with 962 features, which is very slightly better than random. A possible explanation is that after removing the 1% spare terms, some unique terminologies were lost and it became harder to distinguish the abstracts considering they probably share many common terms in related subjects like CS, Math, Stat and Physics. But another possible way to reduce feature dimension is by chi-square test, which would be likely keep some terms that are distinguishable. We did chi square test to reduce the number of features to 1000 and we randomly sampled 10000 abstracts with cross-validated with 10-nearest neighbor classifier. The cross-validation error was around 53.7% which is an improvement. However, due to the computation intensity and time frame, we did not go further to this method to reduce feature sizes.

If we had a faster system, adding more features could improve the classifier as more distinguished terms in different categories show up and the distances among those features start to dominate the decision rule. However, this would require more memory or a different implementation. We decide to concentrate our efforts on different methods for this competition.

Another issue can be addressed here is the choice of distance metric. Euclidean distance is the common choice for this type of model. In our case, the categories are all related and it would be better to assign small weights to the common terms. This could improve the performance of the classifier as more weights will go to the rare terms which would make the abstracts more separable. In our implementation, we have already removed sparse terms and there is no advantage to adding the weights. It might thus be interesting to use weighted Euclidean distance if we could input all the features in the KNN classifier.

Complement-Naive-Bayes

As expected for text classification, the Complement Naive-Bayes model produced satisfying results. The test error rate obtained on Kaggle is somewhat lower, but comparable the Stochastic Gradient Descent Classifier used for the final submission.

The misclassifications can be interpreted in an interesting way. The relatively high number of misclassifications of Stat and Physics abstracts into CS is probably due to the fact that computer science is often applied to these fields. The relatively high number of misclassifications of CS abstracts into Math is probably due to the fact that computer science very often borrows from this field when people are trying to explain the theory of certain algorithm.

Using more than 10 000 features did not yield considerable improvement. It is thus a tradeoff between computational power availability and the need for a few decimal point improvement on the score. It should be noted that the Naive Bayes formulation considers the features to be conditionally independent. This can't be said of words in a text, even if they are taken out of context, by their frequencies. However, if we eliminate features by chi square test, some dependent terms would be removed. This would possibly improve the performance of Naive Bayes classifier.

Scikit-learn library

The Stochastic Gradient Descent (SGD) Classifier with modified huber loss gave the best classification accuracy of all the above three methods. Since huber loss is less sensitive to outliers, it can predict the real classes for more abstracts than other algorithms. Also, we have used maximum iterations as 1000 and alpha or step size to be 0.00004, which instructs the gradient descent to move forward slowly and carefully, but having L2 regularization to be careful not to overfit too much.

Although the method was only able to produce 82.85% accuracy on the public test data in Kaggle. Investigating on the validation set results we find the reason for this accuracy is the semantic and syntactic overlap between CS, Stat, Math and Physics domains. For example for a particular case : *"we construct a homotopy invariant index for paths in the set of invertible tripotent in a jb triple that satisfy a fredholm type condition with respect to a fixed invertible tripotent that index generalizes the maslov index in the fredholm lagrangian of a symplectic hilbert space"*, the classifier predicts it as CS while it is a Statistics paper. Therefore, we can conclude due to theoretical overlap between these domains, lot of technical terms will be common in the classes, making it difficult for the classifier to predict the correct class.

Generalities

The confusion matrices for the Complement-Naive-Bayes and SGD classifiers presented similar precision rates for the categories relative to each other. The precision is highest for Math in both cases. It is also interesting to note that misclassifications types that stand out are the same for both methods (such as Stat misclassified as CS). This strengthens the idea that there is overlap between some categories for the terminology used.

In the end, considering the results obtained, SGD was chosen for the final submission of predictions.

Statement of Contributions

Jeremy researched and implemented the Complement-Naive-Bayes model, tested different parameters for this model, wrote the sections of the report related to this method and wrote the related work section of the report.

Koustuv researched and implemented the feature extraction, tested and selected the models in scikit-learn, tested different parameters for these models and wrote the sections of the report related to these models.

Yu researched and implemented the K-Nearest Neighbor model, tested different parameters for this model and wrote the sections of the report related to this model.

We hereby state that all the work presented in this report is that of the authors.

References

[1] J. D. M. Rennie, Tackling the Poor Assumptions of Naive Bayes Text Classifiers, vol. ICML-2003, ed. Proceedings of the Twentieth International Conference on Machine Learning, 2003.

[2] V. K. Bhalla and N. Kumar, An efficient scheme for automatic web pages categorization using the support vector machine, (in English), New Review of Hypermedia and Multimedia, Article vol. 22, no. 3, pp. 223-242, Sep 2016.

[3] A. Surkis et al., Classifying publications from the clinical and translational science award program along the translational research spectrum: a machine learning approach, (in English), Journal of Translational Medicine, Article vol. 14, p. 14, Aug 2016, Art. no. 235.

[4] K. Drame, F. Mougin, and G. Diallo, Large scale biomedical texts classification: a kNN and an ESA-based approaches, (in English), Journal of Biomedical Semantics, Article vol. 7, p. 12, Jun 2016, Art. no. 40.

[5] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513{523, 1988.
[6] Kim S. B., Rim H. C., Yook D. S. and Lim H. S., “Effective Methods for Improving Naive Bayes Text Classifiers”, LNAI 2417, 2002, pp. 414-423
[7]http://fastml.com/classifying-text-with-bag-of-words-a-tutorial/
[8] D.C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. Mathematical programming, 45(1):503–528, 1989
[9] Gopal, Siddharth, and Yiming Yang. "Recursive regularization for large-scale classification with hierarchical and graphical dependencies." Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013.
[10] Joachims, Thorsten. "Text categorization with support vector machines: Learning with many relevant features." European conference on machine learning. Springer Berlin Heidelberg, 1998.

Appendix

A. Random Forest

Random Forest was tested out on 40,000 features, bootstrapped and with log2 max features and number of estimators set to 100. The resulting score was 88.5% accuracy on our validation set with 0.88 for precision, recall and F-score.

B. Extremely Randomized Trees

Extremely Randomized Trees is a modification of Random Forests where decision thresholds are drawn from each feature and the best of these randomly generated thresholds are used as the final splitting rule. It was used without bootstrap and with 100 estimators, which resulted in slightly better 89% accuracy on our validation set with precision, recall and f-score being 0.9,0.89 and 0.89 respectively.

C. Linear SVM

Support Vector Machines (SVM) have been widely successfully used in text classification problems [10] in the past. We used Linear SVM in the LinearSVC package from the scikit-learn to evaluate our data, as more complex kernels are redundant for text classification which is quite linear in nature. In order to reduce the massive feature space, we used a Chi Square metric and chose the best 1000 features to use with Linear SVM. Using hyperparameters C=1.0, squared hinge loss and L2 regularization penalty we got 90% accuracy on our validation set with 0.9 for precision, recall and F-score, which is better than the decision trees but still lower than Logistic Regression.

D. Ensemble Methods

We tested out various ensemble methods on our existing classifiers to get better results on the data.

Bagging

Bagging ensemble method trains a given classifier over several subsets of the features and aggregates the individual performance of them into the final prediction. It works best with complex models with high variance, hence we used it with Extremely Randomized Trees, with 100 estimators. We got 89% accuracy on our validation set with 0.89 for precision, recall and f-score.

Adaboost

Adaboost, a special Boosting ensemble method, uses several weak learners and runs them on several modified dataset versions. For every prediction, it boosts or adjusts the weights for misclassified labels. Using basic Random Forests as weak learner, we got 88.28% accuracy on our validation set, with 0.89 for precision, recall and f-score. Hence we see that neither Bagging nor Boosting was able to provide better results than most of our top performing models.

Voting

Voting ensemble method trains a bunch of classifiers over a given dataset and predict the class based on majority vote. This is useful for a set of equally well performing models, and hence we used it with our best models, such as Logistic Regression, SGD Classifier and Naive Bayes. Voting ensemble performed really well, very close to SGD Classifier results with 92.73% accuracy on our validation set, with precision, recall and f-score as 0.93, 0.93 and 0.93 respectively.