Text Classification - Applied Machine Learning

Michael Golfi
McGill ID: 260552298
Email:michael.golfi@mail.mcgill.ca

Shruti Bhanderi
McGill ID : 260724575
Email: shruti.bhanderi@mail.mcgill.ca

Zahra Khambatys
McGill ID: 260577706
Email:zahra.khambaty@mail.mcgill.ca

***Abstract*—The project studies the effectiveness of different machine learning algorithms on a text classification problem to analyze short conversations extracted from the Reddit website, and classify them according to their topics, which include hockey, movies, nba, news, nfl, politics, soccer and worldnews. Multinomial Naive Bayes, Support Vector Machines, K-Neighbors and Decision tress are considered. The best model is chosen after application of each algorithm. This study is performed towards Project-2 of Applied Machine Learning at McGill University in Winter 2017.**

I. Introduction

Classification problems are an integral part of machine learning. Text classification is a very important subset of classification. While natural language programming and machine learning are pushing the frontiers in practical applications, we present a study on text classification of eight categories: world news, news, politics, nba, soccer, movies, hockey, nfl. Naive Bayes, k-NN, Support Vector Machines and Decision Trees are used to train models to perform the classification experiment in order to predict unseen documents and yield their predicted categories.

II. Related Work

Text classification is a complex problem to optimize, as there are various feature engineering and feature reduction methods available. We decided to use the knowledge from the existing work done in these fields. Below are some of the choices we made using previous studies.

A. Lemmatization vs Stemming

Lemmatization and stemming are both used for dimension reduction in text classification. Lemmatization reduces the word to its root word by removing only the inflectional part. It does so by using a lookup table to find the proper root word of the derivational morpheme, the complete word containing a root and an inflectional segment. On the other hand, stemming reduces the word to its stem using predefined rules which vary depending upon the stemming algorithm used. Stemming may also result in removal of derivational morpheme into undefined words. As shown in [4], lemmatization performs slightly better but at a large cost of speed. This improvement is more prominent in language modeling, where we dont want to lose information by removing the derivational morpheme. In our problem both should work equally well. We decided to use lemmatization as we only had to process the text once thus freeing speed from being a constraint.

B. Uni-gram and Bi-gram

The most definitive method of feature representation for a text classification problem is the Bag of Words (BoW) approach. The use of uni-grams and n-grams (most frequently called bi-grams) have been studied extensively in literature. Caropreso et al[5] study the quality of n-grams on text classification using the Reuters dataset. After using mutual information to retrieve the highest scoring features, their experiments reveal that bi-grams are generally more useful in text classification than uni-grams. Bekkerman and Allan[6] conclude that while bi-grams are unlikely to provide an improvement in classification performance on unrestricted classification problems, their use in limited lexicon domains with stable phrase constructions can be fairly useful. Tan et al[7] finds bi-grams from uni-gram seeds with a minimum document frequency in at least one of the target categories. They are then sorted using highest occurrences and information gain before being incorporated into the training set. These experiments show significant improvement in classification performance on the Yahoo! Science dataset. Based on the work done in the above cited papers, we decided to use bi-grams and uni-grams in our feature set with a document frequency threshold.

C. Feature Selection

Feature selection helps us to reduce the dimensionality of the dataset by throwing out irrelevant and noisy features while keeping only those which have a good correlation with the output labels or explain most of the variance. Feature selection is domain specific and depends on the kind of data. Three algorithms have been used in this study. Manning et al[1] suggest using Mutual Information to measure how much information is conveyed by the absence or presence of a particular feature in making the correct classification. Pearsons chi-squared [2] test gives an estimate of the dependence between a specific term and a specific class. Principal Component Analysis (PCA) is another widely used method in the Machine Learning community which brings out strong patterns in a dataset and reduces higher dimensional data into lower dimensions [3]. PCA did not work for kNN and therefore was not used. In general, chi-squared correlations give a better estimate than Mutual Information and would be a better choice for this type of classification experiment.

III. Problem Representation

The analysis and problem representation can be divided in following sections.

A. Pre-processing

For the first step of pre-processing the raw text was converted to lowercase since case-sensitivity can be ignored for classification problems. Afterwards, all tags, punctuation and extra whitespace characters were removed from the test. Word screening was performed next. The text was screened for stopwords; words which do not provide meaningful representation to a sentence such as determinants. Words with less than 3 characters were removed to filter out erroneous values (eg: 'u n c l e' was a term found in the raw documents, such terms are not easily fixable). The words were then lemmatized to generate the root words and thus reduce the dimensionality of the classification. This process was done prior to feature extraction and was done only once. The output of the pre-processing stage was cached for use by the classification algorithms.

B. Feature Selection and Extraction

At first, features were selected based on count. However, counting is a skewed feature selection process since it favors words which are more common. Therefore, further metrics were explored and thus Term Frequency-Inverse Document Frequency (TF-IDF) was chosen. TF-IDF will place importance on words that appear infrequently in the document and will rank commonly used words in a document as less significant to a classification algorithm. Within the experiment uni-grams and bi-grams were used interchangeably in order to gauge which method would give better results during the cross-validation stage. The first and second part of the experiment used uni-grams which only considered single words as the smallest unit of text to analyze. These parts also only used counting through simple counting of the words in the document. The third part of the experiment used bi-grams which would consider adjacent words as related to one another in the complete document. The resulting counts of single lemmatized words and of TF-IDF vectors for the documents were used as the sole features for the several experiments. These vectors were generated by the scikit-learn classes: "CountVectorizer", and "TfidfTransformer".

Feature selection was further strengthened through cross-validation in the training phase. Cross-validation was performed using the scikit-learn KFold class for the third portion of the experiment. K folding randomly shuffled and sampled training and test data for the entire dataset to train on, the best distribution for training and testing would be chosen based on the produced model from a fold. For the first two parts of the experiment, data was split by generating random partitions into training and testing data with a ratio of 70% training-testing datasets.

IV. Algorithm Selection and Implementation

Algorithms were selected based on their intended strengths. A Naive Bayes implementation was chosen for part one due to the independence criteria. kNN was chosen because of its simplicity to implement versus a Support Vector Machine and a Decision Tree. Naive Bayes, Support Vector Machines (with linear, radial basis function and polynomial kernels) and Decision Trees were all used in part 3 as they were easily implemented as library function calls.

We tested Naive Bayes with Lidstone and Laplace smoothing [10]. The number of features and hyper-parameters were selected for Naive Bayes method via a 3-fold cross-validation. After implementing SVM, it proved to be extremely computationally expensive. Removing features was considered to reduce the running time however this was neglected. The approach chosen instead was to run the training overnight on a Microsoft Azure Standard DS4 v2 VM with 8 cores and 28 Gb of RAM. The SVM was trained in 2 hours vs over 8 hours on a Microsoft Surface Book Laptop. Several models were trained in parallel with slightly tuned parameters to save time and keep the same amount of data integrity.

A. Naive Bayes

We used a Multinomial distribution for Naive Bayes with a smoothing parameter α according to the following equation:

$$P(Category|Word) = \frac{P(Word|Category)P(Category)}{P(W_{C_i})}$$

In Python, the primitive datatype float is only 32-bits. In places where necessary, float64 was used from the numpy library to handle scenarios needing higher precision. Furthermore, the additive log property was exploited to protect against precision underflow when dealing with incredibly tiny numbers. Thereafter, the following equations were realized:

$$P(Category|Word) = max(log_2(|Word_C|/|Words_C|) + P(Word|Category))$$

$$P(Word|Category) = log_2\left(\frac{\alpha + f_{cw}}{|Words_C| + |Vocabulary| + \alpha}\right)$$

Where:

- f_{cw} represents the frequency of a word in a category
- $P(Category|Word)$ represents the probability of a category given a word.
- $|Word_C|$ represents the amount of times a word appears in a category
- $|Words_C|$ represents the number of words in a category
- $P(Word|Category)$ represents the probability that a word is included in a given category
- $|Vocabulary|$ represents the number of words in the total vocabulary
- α represents the smoothing factor corresponding to Laplace and Lidstone smoothing

The smoothing parameter α redistributes the weight of features, by transferring some of the weight to the features which

have zero weight. α = 0 is no smoothing, α = 1 is Laplace smoothing and for any $\alpha < 1$, it is called Lidstone smoothing. Parameter tuning was performed for α for achieving the best accuracy results.

B. kNN

For the implementation of kNN, the algorithm compares each training article with a new test article. The classifier uses Euclidean Distance to determine the similarity between articles by calculating the frequency/occurrence of unique words per article. This was represented as the vector space model.

$$Distance(a,b) = \sqrt{((x-a)^2 + (y-b)^2)}.(1) \quad (1)$$

The important thing to notice with the above equation is that this does not account for the difference in lengths of words and hinders accuracy, hence, the optimized version uses a cosine similarity function, explained further below.

Since the data is text-based with multi-class labels, the data was vectorized and normalized it using the sci-kit learn "CountVectorizer" and "TfidfTransformer" packages. These are very popular methods used in the field of natural language processing (NLP). The TF-IDF method determines the relative frequency of words in a specific document through an inverse proportion of the word over the entire document corpus. In determining the value, the method uses two elements: TF (f_{ij}) - term frequency of word i in document j and IDF - inverse document frequency of word i. The weight matrix was determined in order to classify text further. This was to gauge the relationship between each unique word and an article using a word-article matrix.

The initial structure of the algorithm searches for the weight of each article which is represented as an n-dimensional vector. This can be illustrated by a matrix W with dimensions NxM, where N is defined by a number of unique words in all the articles and M represents the number of categories in an article.

TABLE I
TEXT CATEGORIZATION OF K-NN CLASSIFIER.

Terms	Category
N	total number of Articles
M	total number of distinct words
n_i	number of times i^{th} word occurs
f_{ij}	frequency of i^{th} word in document j
A_j	j^{th} training document
X	test document

The dimensions of the matrix is equal to the product of the number of different unique words and the total number of articles. This can be mathematically represented as following:

$$w_{ij} = \frac{f_{ij}}{\sqrt{\sum_{l=1}^{M} f_{lj}^2}} \times log\left(\frac{N}{n_i}\right).(2)$$

To classify a class-unknown article X, the k-Nearest Neighbor classifier algorithm ranks the article's neighbors among the training vectors, and uses the class labels of the k most similar neighbors to predict the class of the new article. The classes of these neighbors are weighted using the similarity of each neighbor to X, which is calculated using cosine similarity defined as follows:

$$sim(X, A_j) = \frac{\sum_{t_i \in (X \cap A_j)} x_i \times a_{ij}}{\|X\|^2 \times \|A_j\|^2}.(3)$$

Where:

- X is the test set represented in vector form
- A_j is the jth trainingset
- t_i is the intersection or the commonness of X and A_j
- x_i is the weight of word t_i in X
- a_{ij} is the weight of word t_i in an article A_j
- $\|X\|^2$ is the norm of X
- $\|A_j\|^2$ is the norm of A_j

A cutoff threshold is needed to assign the new document to a known class. To optimize our results we repeat the following steps:

- For each word, we determine the value of W_i that gives the best similarity. Introduce new possible values for W_i by multiplying the original W_i with different multiplication factors.
- Select a word j that gives the best overall W_i and update W_j with it.

Note that the biggest issue with this algorithm is finding the optimum W and the uncertainty of overfitting the data.

C. SVM and Decision Trees

The libraries for SVM and Decision Trees from scikit-learn were used as implementations. The linear, radial basis function and polynomial kernels for SVMs were used in the implementation to discover which would yield the best results. For this implementation a scikit-learn pipeline was used. The steps were: count words, perform TF-IDF transformation, train SVM.

V. TESTING AND VALIDATION

To select the best model, we first chose optimum parameters for SVM and Naive Bayes. This was done using 3-Fold Cross Validation and choosing the best value based upon the F1 scores. The training and validation error were, continuously monitored to check for overfitting. For feature selection, we varied the number of features by selecting a random subset of training data based on a threshold cutoff of 70% training vs validation and also based on 3-Fold. The individual metrics per algorithm are discussed next.

A. Naive Bayes

In Naive Bayes, the optimum smoothing value α was found to be 0.009. The smoothing parameter α was varied from 0.005 to 2.5. Using these parameters we obtained an accuracy of 89%. Table II shows the metrics for the Naive Bayes implementation vs table III which represents the results obtained for the same data using the scikit-learn implementation of Naive Bayes.

TABLE II
NAIVE BAYES IMPLEMENTATION RESULTS

	precision	recall	f1-score	support
hockey	0.96	0.93	0.94	6334
movies	0.94	0.96	0.95	6774
nba	0.98	0.87	0.92	5559
news	0.74	0.76	0.75	6313
nfl	0.92	0.94	0.93	5980
politics	0.82	0.83	0.82	5934
soccer	0.97	0.94	0.96	6373
worldnews	0.82	0.88	0.85	6182
avg / total	0.89	0.89	0.89	49449

TABLE III
SK-LEARN NAIVE BAYES RESULTS

	precision	recall	f1-score	support
hockey	0.97	0.97	0.97	6925
movies	0.95	0.99	0.97	7491
nba	0.99	0.92	0.96	6158
news	0.85	0.85	0.85	6994
nfl	0.97	0.97	0.97	6725
politics	0.89	0.90	0.90	6538
soccer	0.98	0.97	0.97	7091
worldnews	0.90	0.93	0.92	7078
avg / total	0.94	0.94	0.94	55000

B. kNN

Results for kNN were not fully realized to be meaningful in the end due to training shortcomings. Instead, more effort was spent to improve already working models.

C. SVM

Tables IV and V show the results of the experiments for the linear and polynomial SVMs. The results show that the linear SVM shows the best results meanwhile the polynomial did not produce meaningful results. The RBF results were similar to the polynomial results.

TABLE IV
LINEAR KERNEL RESULTS

	precision	recall	f1-score	support
hockey	0.96	0.96	0.96	6287
movies	0.96	0.98	0.97	6740
nba	0.97	0.95	0.96	5516
news	0.81	0.83	0.82	6392
nfl	0.96	0.95	0.96	6001
politics	0.88	0.86	0.87	5927
soccer	0.97	0.97	0.97	6324
worldnews	0.89	0.90	0.90	6323
avg / total	0.93	0.93	0.93	49510

D. Decision Trees

The results of the Decision Trees are in table VI. The method performed against expectations and received an overall accuracy of 78%. No further work was done to improve this as it was decided to allocate further effort towards better pre-processing, Naive Bayes and SVM implementations.

E. Overall Results

The overall experiment results are shown in table VII. Naive Bayes performed the best which was also against expectations. The assumed expectations was for SVM to have the best overall performance.

TABLE V
POLYNOMIAL KERNEL RESULTS

	precision	recall	f1-score	support
hockey	0.00	0.00	0.00	6251
movies	0.14	1.00	0.24	6746
nba	0.00	0.00	0.00	5497
news	0.00	0.00	0.00	6293
nfl	0.00	0.00	0.00	6043
politics	0.00	0.00	0.00	5976
soccer	0.00	0.00	0.00	6477
worldnews	0.00	0.00	0.00	6297
avg / total	0.02	0.14	0.03	49580

TABLE VI
DECISION TREE RESULTS

	precision	recall	f1-score	support
hockey	0.75	0.78	0.77	6942
movies	0.90	0.90	0.90	7517
nba	0.76	0.76	0.76	6178
news	0.67	0.67	0.67	6996
nfl	0.82	0.81	0.82	6712
politics	0.76	0.77	0.76	6575
soccer	0.77	0.77	0.77	7067
worldnews	0.78	0.77	0.77	7013
avg / total	0.78	0.78	0.78	55000

TABLE VII
RESULTS SUMMARY

Method	hockey	movies	nba	news
Nave Bayes Precision	97%	95%	99%	85%
SVM Precision	96%	96%	97%	81%
Decision Tree Precision	75%	90%	76%	67%
Nave Bayes Recall	97%	99%	92%	85%
SVM Recall	96%	98%	95%	83%
Decision Tree Recall	78%	90%	76%	67%
Nave Bayes F1 Score	97%	97%	96%	85%
SVM F1 Score	96%	97%	96%	82%
Decision Tree F1 Score	77%	90%	76%	62%

TABLE VIII
RESULTS SUMMARY CONTINUED

Method	nfl	politics	soccer	world news
Nave Bayes Precision	97%	89%	98%	90%
SVM Precision	96%	88%	97%	89%
Decision Tree Precision	82%	76%	77%	78%
Nave Bayes Recall	97%	90%	97%	93%
SVM Recall	95%	86%	97%	90%
Decision Tree Recall	81%	77%	77%	77%
Nave Bayes F1 Score	97%	90%	92%	92%
SVM F1 Score	96%	87%	92%	90%
Decision Tree F1 Score	82%	76%	77%	77%

VI. DISCUSSION

Based on the test results, overall the best performing method is Naive Bayes. Tables VII and VIII displays the category wise precision, recall and F1 score values while figures 4, 5 and 3 give a better visual representation of the data points for the individual methods.

It is important to mention that the category predictions for Naive Bayes between politics, news and world news was not extremely accurate. It is hypothesized that this is due to the fact that among these categories, a similar vocabulary is used and thus the independence criteria cannot possibly hold.

From the categorical analysis, it was inferred that Naive Bayes has achieved better F1 score for news, nfl, politics and world news. For all other classes, SVM and Naive Bayes is achieving similar F1 score. Both achieved better results than decision trees. However, SVM has achieved better precision for hockey subjects. For the remaining other classes Naive Bayes has achieved higher precision values than both SVM and Decision tree methods. Comparing recall values, Naive Bayes has performed best for all the categories except movies. For movies category, SVM has achieved higher recall value.

These results were unexpected. It was expected that SVM would significantly outperform all other methods but it trailed behind Naive Bayes. This points to perhaps a misuse or mal-optimization of the SVM class. Further experiments would be more wary of handling SVM methods properly and assuring that the proper procedures must be followed in order to produce a better model. Naive Bayes has otherwise proven to be a successful method in cross-validation (previously mentioned 94% accuracy) and also on the Kaggle test input submissions (92.5% accuracy).

VII. Conclusion

This experiment is deemed to be a success since it correctly identified test input with an accuracy of 92.5% and the methods prescribed in the project report were successfully applied. Future work would concentrate on improving the custom models in order to be closer in performance to those of scikit-learn.

VIII. Statement of Contributions

- **Michael Golfi**: Worked on the data pre-processing, worked on Naive Bayes implementation for part 1, worked on Naive Bayes, SVM and Decision Trees for part 3.
- **Shruti Bhanderi**: Worked on SVM implementation for part 2, defining methodology and analysis, Report writing.
- **Zahra Khambaty**: Worked on data pre-processing, Implementation for part 2 with K-nn, creating graphs and report writing.

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

IX. References

References

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

[2] Chernoff, Herman, and E. L. Lehmann. *”The use of maximum likelihood estimates in chi2 tests for goodness of fit.”* The Annals of Mathematical Statistics 25.3 (1954): 579-586. APA

[3] Ding, Chris, and Xiaofeng He. *”K-means clustering via principal component analysis.*Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[4] Johnson, David E., et al. .A decision-tree-based symbolic rule induction system for text categorization.IBM Systems Journal 41.3 (2002): 428-437.

[5] Caropreso, Maria Fernanda, Stan Matwin, and Fabrizio Sebastiani. .A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization.Text databases and document management: Theory and practice (2001): 78-102.

[6] Bekkerman, Ron, and James Allan. Using bigrams in text categorization. Technical Report IR-408, Center of Intelligent Information Retrieval, UMass Amherst, 2004.

[7] Tan, Chade-Meng, Yuan-Fang Wang, and Chan-Do Lee. The use of bigrams to enhance text categorization.Information processing & management 38.4 (2002): 529-546.

[8] *http://www.nltk.org/book/ch05.html*

[9] *http://www.nltk.org/ modules/nltk/stem/wordnet.html*

[10] *http://scikit-learn.org/stable/modules/naive bayes.html*

[11] *http://cvxopt.org/*

Appendix

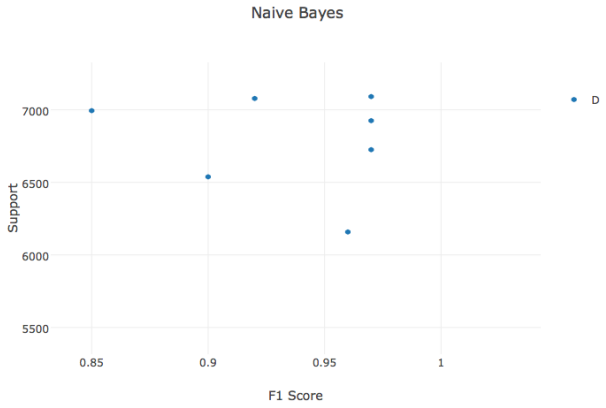

Fig. 1. Support vs F1 Score Comparison

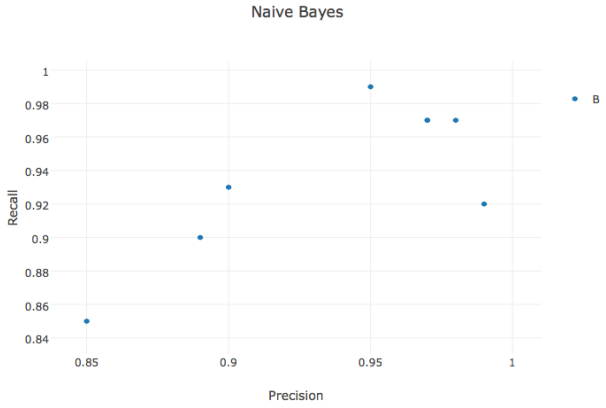

Fig. 2. Recall vs Precision Comparison

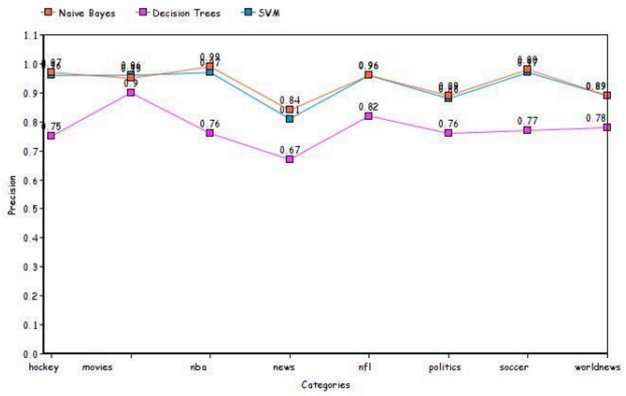

Fig. 3. Precision vs Categories Comparison

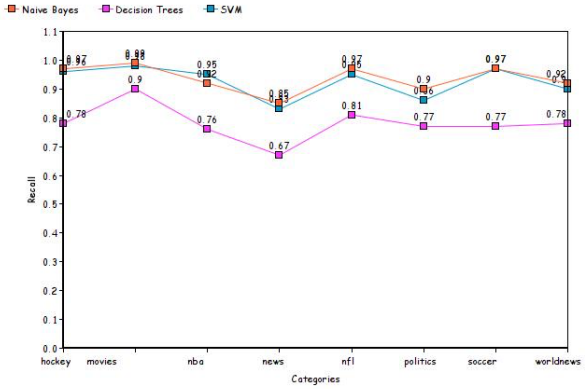

Fig. 4. Recall vs Categories Comparison

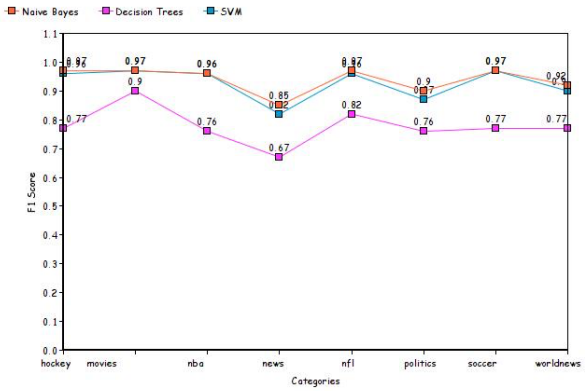

Fig. 5. F1 Score vs Categories Comparison