Comp 551 Project 2

Eric Blachut
260562761
eric.blachut@mail.mcgill.ca

Nirmal Kanagasabai
260716737
nirmal.kanagasabai@mail.mcgill.ca

William Campoli
260636707
william.campoli@mail.mcgill.ca

Kaggle Team Name: "An Unconscious Statistician"

Abstract

In this project, we applied various supervised machine learning methods to the problem of classifying short conversations extracted from Reddit by conversation topic. Each conversation in the dataset belongs to one of the 8 topics: hockey, movies, nba, news, nfl, politics, soccer and worldnews. We implemented ourselves a linear text classification model, Naive Bayes, a non-linear Model, K-Nearest Nearest Neighbors, and used the Scikit-Learn package to implement two additional models: Random Forests and a Stochastic Gradient Descent Classifier [6].

I. Related Work

In the literature, we see several machine learning approaches used for classifying textual data-sets [7]. Both linear and non-linear classification algorithms have been applied for text classification. Some linear classification algorithms include Nave Bayes [8] [2] , and Logistic Regression [3]. Non-linear classification algorithms commonly used are: Decision Trees [9], K-Nearest Neighbors [2], Support Vector Machines (SVM) [4] , and Random forests [5].

Accuracy gains for these models come from two primary sources: feature selection and model parametrization. It is important to pre-process the text to extract features that will be relevant to classification, which we discuss generally in *Data Cleaning* and more specifically in each method-specific sections. It is also important to tune models, for example through fitting hyper-parameters or making distributional assumptions, which we discuss in the *Methods* section for each model we use.

II. Problem Presentation

A. Data Cleaning

The first challenge we had to tackle was that of cleaning the data to generate a set of words for each training/ test example to apply our model to. We used the assistance of the Natural Language Toolkit (NLTK) package in python for assistance with some text processing [1]. The rules that we applied to clean the text are (in order of application):

1. Remove any characters between "<" and ">" (strips out tags, ex. for speakers)
2. Delete words starting with a "@" (strips out usernames)
3. Remove any non-alphabetical characters
4. Remove all stop-words (extremely common words such as "of"), stop-words taken from NLTK [1]
5. Remove all single-letter words

After following this process, we get text that consists almost entirely of sensical words, and can be used to generate relevant features.

B. Feature Selection

The features used for our model are the bag of words: an individual feature for each word in the training example, as well as n-grams: successive combinations of n words (ex. 2-grams are pairs of words that occur in succession in the text). Each of these features is given a weight that represents the significance of this feature in classification. The weighting mechanism used for each classification method is described below. One weighting mechanism that was used for both Naive Bayes and K-Nearest is term-frequency, inverse-document-frequency (TF-IDF) which is discussed in more detail in the Naive Bayes section.

III. Training Methods

A. Linear Method: Naive Bayes

A.1. Method

We implemented a Naive Bayes method for classification. This method works by first training the model to get a conditional probability weighting for each word and each class (conditional probability of seeing that word given the class), then applying these probabilities to the features (words) in training documents to pick the class with the highest probability. The key feature of the Naive Bayes model that must be chosen carefully is the distribution/weighting function used to generate the conditional terms $P(w_i|c_j)$. We fully implemented this model, only using the python *nltk* package for the list of stop-words to remove.

Additionally, we used Laplace smoothing, a method that allows the model to deal with previously unseen words in test documents. In this method, we use the term-frequency inverse-document-frequency (tf-idf) weighting for our probability distribution, with Laplace smoothing. The result is that we estimate the probability of a given class c_j from a test document D with words w_i as:

$$P(c_j|D) = P(c_j) \prod_{w_i \in D} \frac{f(w_i, c_j)1}{\sum\limits_{w \in V} (f(w, c_j) + |V|} \; log(\frac{nd}{nd_{w_i}})$$

where V is our vocabulary (all words found in the training documents), $f()$ is the count function such that $f(w_i, c_j)$ gives a count of the occurrences of word w_i in training documents of class c_j, nd is the total number of training documents, and nd_{w_i} is the number of training documents containing word w_i. This formulation has several attractive features, namely that it is robust to training documents with words not previously seen in our test documents, and it gives a larger weight to words found in fewer documents in our training set (which we would expect to be more predictive). Appendix C shows the 3 words for each class with the highest TF-IDF weight.

We conducted cross-validation on partitions of the training data-set to chose two parameters for our model. The first is the dimensionality of n-grams to use. It was very clear that using 2-grams would be an improvement over the simple bag-of-words case, as the use of 2-grams improved the accuracy of the Naive-Bayes model from 90% to 94% on the test data-set, however we were not sure how 2-grams would compare to 3-grams. The second parameter is the limit with which to exclude uncommon words from our training feature-set. In the literature, often words that occur infrequently (1 or 2 times) in the whole training set are excluded, to speed up training and reduce variance. We tested 5 values of this parameter, from 0 (excluding no features) to 4 (excluding any words that occur less than 5 times across all training examples). For the feature-set that includes 2-grams and 3-grams, the total number of features (unique words, 2-grams, and 3-grams) is: 7,346,398. A higher frequency-exclusion parameter significantly reduces the number of features: for example, excluding features that occur only once (parameter value of 1) reduces the number of features to 1,500,000, and with a value of 2 there are only 780,000 features.

For cross-validation, we used the first 100,000 rows of the training data set with 5 partitions: training on 80,000 rows and testing on 20,000 rows in each iteration. We used the mean accuracy across these 5 iterations to chose parameter values. We found that using 2 and 3-grams with no frequency exclusion had the highest average accuracy across the cross-validation, although excluding words with frequency ≤ 1 had a lower standard deviation of accuracy and similar mean-accuracy. The full results are shown in appendix *A*.

For our final submission on the test data, we used the TF-IDF weighting model with the bag of words + 2-grams + 3-grams, with no frequency exclusion of features.

A.2. Results

The chosen Naive Bayes parametrization performs very well, both the training set with cross-validation, and on the test set.

It is interesting to consider what the most common misclassifications are. In appendix **C**, we show the two most common incorrectly-predicted classes for each true class, as well as the proportion of the true class that were misclassified in this way. This is essentially a confusion matrix that has been truncated for readability to show only the most common mistakes. By far the most common errors are between "news", "worldnews", and "politics", which aligns with our intuition as the discussion topics in these areas are closely related. There is also a high error rate between some of the sports classes (ex. "nba" and "nfl"), which is also intuitive, as these sports share much of the same vocabulary.

A.3. Discussion

We believe that the Naive Bayes method we have used is fairly accurate and robust (low variance), considering the simplicity of Naive Bayes as a classification method (compared to more complex methods such as SVM or ensemble methods).

Further improvements to the Naive Bayes model would be found either with feature selection (ie, dropping some subsets of the words), or with using a different weighting function instead of TF-IDF. However, we have shown through cross-validation that dropping uncommon words/n-grams generally reduces accuracy, so there probably aren't large improvements to be made in this regard, and given the popularity of TF-IDF in literature on text classification, it is hard to imagine another weighting function that would improve our model.

B. Non-Linear Method: K-Nearest

B.1. Method

The second machine learning technique we used to classify our data was k-nearest neighbours. K-nearest neighbors is an intuitive, instance-based supervised machine learning algorithm. Essentially, the algorithm works by storing a set of labeled training data in n-dimensional feature space. When one wishes to classify a test case, the distance of this test instance (in feature space) to every other point in the training data is calculated. The classification of the point is simply the majority class of the k nearest training points to the provided test point. The version of K-nearest neighbors used in this report is a kernel based version, where instead of the k nearest neighbors getting an equal vote, the vote is weighted based on their distance to the test point. The voting algorithm works as follows:

1. For a given test point, x_t, needing classification calculate it's distance to every other point in the training data.

2. Collect the k nearest points to x_t in the training data: $X = x_1, ..., x_{k-1}, x_k$

3. For every class C in X, calculate the score of that class by: $\sum_{i=1}^{N} w(d(x_i))$ where $x_i \in C, X$. d is the euclidean distance and w the weighting function.

4. Classify the the test point, x_t, by choosing the class with the largest score (break ties arbitrarily).

For this model, there are 2 important hyper parameters that need tuning. The first hyper parameter is the choice of K, while the second is the choice of weighting function. K-nearest neighbours is a lazy learning algorithm in the sense that it only does its computing when it needs to classify a point and likewise it needs to store the whole labeled training set in memory. This becomes problematic when the feature space becomes quite large (n on the order of 10^6) especially when it is coupled with a sizable data-set. So due to computational limitations, to select the features for this algorithm we chose to use the 10000 highest scoring TF-IDF value features, which assigns higher scores to features with higher average frequency but normalizes the scores relative to how many total documents that word appears in (in order to remove low information carrying/high noise strings). Like the naive-Bayes model we included bigrams and tri-grams which have the ability to bring more 'context' to the bag of words approach.

To tune our 2 hyper-parameters we used a single-fold cross validation technique on a subset of the provided data (again because of computational limitations). Data was split 66% for training and 33% for validation for 10000 labeled data points. We tested 4 different values for the k parameter: 5, 10, 25, and 50 with the inverse-distance weighting function. Likewise we tried 3 different kernel weighting techniques: inverse of the distance, inverse of the distance squared, and a Gaussian weighting function. The Gaussian weighting function has a hyper-parameter of its own that needs tuning, the sigma value. We tested sigma for the values: 0.25, 2, 5, and 20. All of these weighting functions were tested with 10000 TF-IDF features (including uni-grams, bi-grams, and tri-grams) and a k-value of 50. The details of these weighting functions are available in the Appendix E.

B.2. Results

Observing figure 1 (in the appendix) we notice that a higher k-value yields better accuracy scores on the validation data, with a k-value of 50 achieving an accuracy of 75.85% using a value of 50. To better illustrate the trend of accuracy scores, we fit a log function to show how the improvements in error appear to taper out as we increase k (R-squared = 0.957) . This indicates that while larger values of k will probably give better accuracy scores, we will eventually run into diminishing returns. This is an intuitive result, since for any given test point that we wish to classify, only the closest points will have a huge impact on the overall classification decision due to the weighting function. Likewise, as k-increases we reduce the variance of the classifier, since test points will be classified on the basis of a larger subset of the data instead of a few local neighbors.

Observing appendix F, we notice that the highest accuracy score was achieved with an inverse-distance squared weighting function. This weighting function biases the voting decision towards points that are very close and heavily penalizes points that are further away (relative to the regular inverse-distance weighting function). This suggests that noise from further away points is slightly misleading the regular inverse-distance function into making false classifications.

Observing Figure 2 we see how the accuracy changes with different values of sigma in the Gaussian kernel. Essentially, every value except the first value of sigma (0.25) yielded the same accuracy score of 0.756. This probably means that the sigma-squared values ended up being relatively large compared to the distance-squared value which essentially turns the weight into a constant value for every point, no matter the distance. This is equivalent to using a majority voting version of K-nearest where every point gets equal weight.

B.3. Discussion

Overall, with our chosen feature set, the data suggests that a higher k-value is optimal but one should be weary of making it too large since there appears to be diminishing returns after a certain point. Error seems to reduce at a logarithmic rate while computational costs scale linearly with the size of k. In terms of weighting functions, it was shown that an inverse-distance squared function was optimal compared to Gaussian and simple inverse-distance functions.

Several decisions had to be made due to computational and memory limitations. Firstly, this limited the size of our training and testing sets along with the amount of folds that we could do for cross-validation. This suggests that the accuracy scores generated are probably quite high in variance and should probably be taken with a grain of salt. Secondly, we were also limited in the number of features and size of k to use which probably affected how effective our model was. We hypothesize that with a larger training data set and more features that this model could be comparable to the naive-Bayes approach in prediction accuracy.

The intractability of K-nearest neighbors in this classification problem makes it appear like a sub-optimal classification technique when compared to naive-Bayes or other linear classifiers. The bag of words approach entails a very high dimensional feature spaces and thus lots of computational and memory requirements. With better computational resources such a method could probably provide comparable results to the other methods discussed in this paper. Likewise, perhaps better results could be obtained by using features that take more semantic or contextual content into account, such as the skip-gram approach.

C. Scikit-Learn Packages

C.1. Method

For the third portion of the project, we employed two Scikit-learn methods: Random Forest and Stochastic Gradient Descent (SGD) classifier [6].

As in the two approaches stated above, Data Cleaning and Feature Extraction were carried out before training the models. To do this, Label Encoding (Fit and Transform) and Decoding (Inverse Transform) was carried out using Scikit-Learn Preprocessing Label Encoder. An additional step was to lemmatize the words using the WordNetLemmatizer from the NLTK package. The accuracy of the predictions improved significantly after lemmatization. We also used the TF-IDF approach to feature weighting discussed above.

We decided to go with Random Forest for our implementation because of its performance in handling large datasets with higher dimensionality. It efficiently handles missing data issue and offers methods to balance errors in case of imbalanced classes. This ensemble technique uses averaging which reduces variance.

The second method from Scikit-Learn we implemented is the SGD Classifier: a linear model which estimates the gradient of loss for each sample and updates the model with a decreasing learning rate. The decision to go with SGD was the efficiency it offers and the ease of implementation by offering lots of opportunities for code tuning. The hyper-parameter which is used to tunethe model that the SGD Classifier fits is the Loss Parameter.

The most significant parameter of this family of models we needed to tune are : Loss, Penalty (regularization term), no. of iterations, learning rate and alpha (constant that multiplies the regularization term).

We increased the number of iterations from 500 to 2000 and observed an increase in the accuracy of the predictions. The penalty was set to the default value of l2. Out of all these, the loss parameter, which specifies the loss function to train the model with, had the largest effect on the performance of the model. Initially, we made use of 'Hinge' which provides a linear SVM. Then, we experimented with 'Log' which offers Logistic Regression. Finally, we used 'Modified Huber' which had better accuracy than the other two. The details are available in the Appendix H.

For the Random Forest model, the parameters we considered are: the maximum number of features, number of estimators (trees you want to build before taking the maximum voting or averages of predictions), and class weight.

We set the class weight to balanced, meaning we replicate the smaller classes until we have as many samples as the larger one in an implicit manner. We observe that that the classes, by default, are pretty much balanced and the addition of class weight hardly had an impact in our predictions.

Of these parameters, the most important parameters are the number of features and the number of estimators. Training the random forest model with no restrictions on number of features is too computationally intensive, so we experimented with the maximum features parameter and settled on a value of 0.4. This model considers 40% of the original features, keeping only 6000. While increasing the maximum features parameter increases accuracy, it severely decreases training speed. We have the same trade-off with the number of estimators, which we chose to set to 300.

C.2. Results

The precision, recall, F1-score and Support for both the models were computed using the Scikit-learn library. The precision is intuitively the ability of the classifier to correctly label positive instances of a class. The recall is the ability of the classifier to find all the positive samples. The F-1 score weights recall more than precision by a factor of beta. It can be interpreted as the weighted harmonic mean of precision and recall. The support is the number of occurrence of each class in the training dataset. We trained on 75% of the training data and predicting on the remaining 25%. We calculated these measures for each of the 8 categories in the dataset. The full results were shown in Appendix I (for Random Forest) and Appendix J (for SGD Classifier).

C.3. Discussion

The 'accuracy' of the model is computed by taking the mean of the values. We were able to obtain an accuracy of 93% for Random Forest and 94% for the SGD classifier. With both of them performing similarly, it could be interpreted that the training models were pretty stable. In order to push the accuracy further higher, the data-preprocessing and the feature extraction must be optimized, which is as a future scope or improvement of these classifiers.

IV. Results

Our two most accurate models (on cross-validation of the training data as well as Kaggle's validation of our test-data) are the Naive Bayes model which we have implemented ourselves, and the SGD Classifier model (with Modified-Huber as the loss parameter) from the Scikit-learn package. The Random Forest model (using Scikit-learn) delivered almost the same result as that of the SGD classifier.

The accuracies of these models on Kaggle's validation of our test data are: 95.90% for Naive Bayes, 93.98% for the SGD Classifier and 93.90% for the Random Forest Model. The K-nearest neighbors approach used a subset of the provided data for training and validation and achieved a top accuracy score of 76.12%.

V. Discussion

It is interesting to have a public leaderboard of scores across teams, which allows us to see that some teams have achieved accuracies of above 97% on the Kaggle leaderboard, indicating that there is still room for improvement from our score of 96% with the Naive Bayes model. We theorize that these teams are either using models like SVM that are more complex than Naive Bayes, are using ensemble methods, or are using more sophisticated methods such as neural networks from 3rd party packages. We would be very interested to apply some of these methods to the problem and see how they compare.

Scope for improvement from our models would come from two main sources: improving our models or trying more sophisticated features that take more contextual and semantic information into account. We believe that some of the models we have implemented, in particular Naive Bayes, are already performing very well relative to the model complexity, and there is little scope for improvement of these models. A large increase in accuracy would likely come from implementing a different class of model with higher complexity.

VI. Statement of Contributions

We partitioned the workload by method, and collaborated on the final report. Eric Blachut worked on Naive Bayes, William Campoli worked on K-Nearest, and Nirmal Kanagasabai worked on implementing methods in 3rd party libraries. We hereby state that all the work presented in this report is that of the authors.

VII. Appendix

A. Naive Bayes: Cross-Validation Results

Frequency-Exclusion Parameter Value	2-Grams		3-Grams	
	Mean Accuracy	Std-Dev	Mean Accuracy	Std-Dev
0	94.0%	0.131%	94.53%	0.163%
1	93.5%	0.142%	94.09%	0.127%
2	93.0%	0.173%	93.5%	0.152%
3	92.7%	0.162%	93.0%	0.147%
4	92.3%	0.146%	92.6%	0.133%

B. Naive Bayes: Mis-classification Results

True Class	**Predicted Class**	**Proportion Misclassified (of True Class)**	**Total Error Rate (of True Class)**
news	politics	6.45%	13.19%
	worldnews	5.21%	
politics	news	4.82%	7.62%
	worldnews	1.91%	
nba	nfl	2.78%	6.97%
	hockey	1.25%	
worldnews	news	3.07%	5.72%
	politics	1.50%	
soccer	worldnews	1.14%	3.53%
	nfl	0.72%	
hockey	nfl	1.30%	3.52%
	movies	0.79%	
nfl	news	0.71%	2.11%
	hockey	0.48%	
movies	news	0.51%	1.32%
	worldnews	0.42%	

C. TF-IDF: Highest Weighted Terms by Class

Class	**Words with Highest TF-IDF Weight (in Order)**
news	police, people, get
worldnews	israel, isis, russia
politics	obama, people, republicans
nba	nba, lebron, game
soccer	goal, cup, world
hockey	nhl, game, hockey
nfl	nfl, team, twitter
movies	movie, movies, film

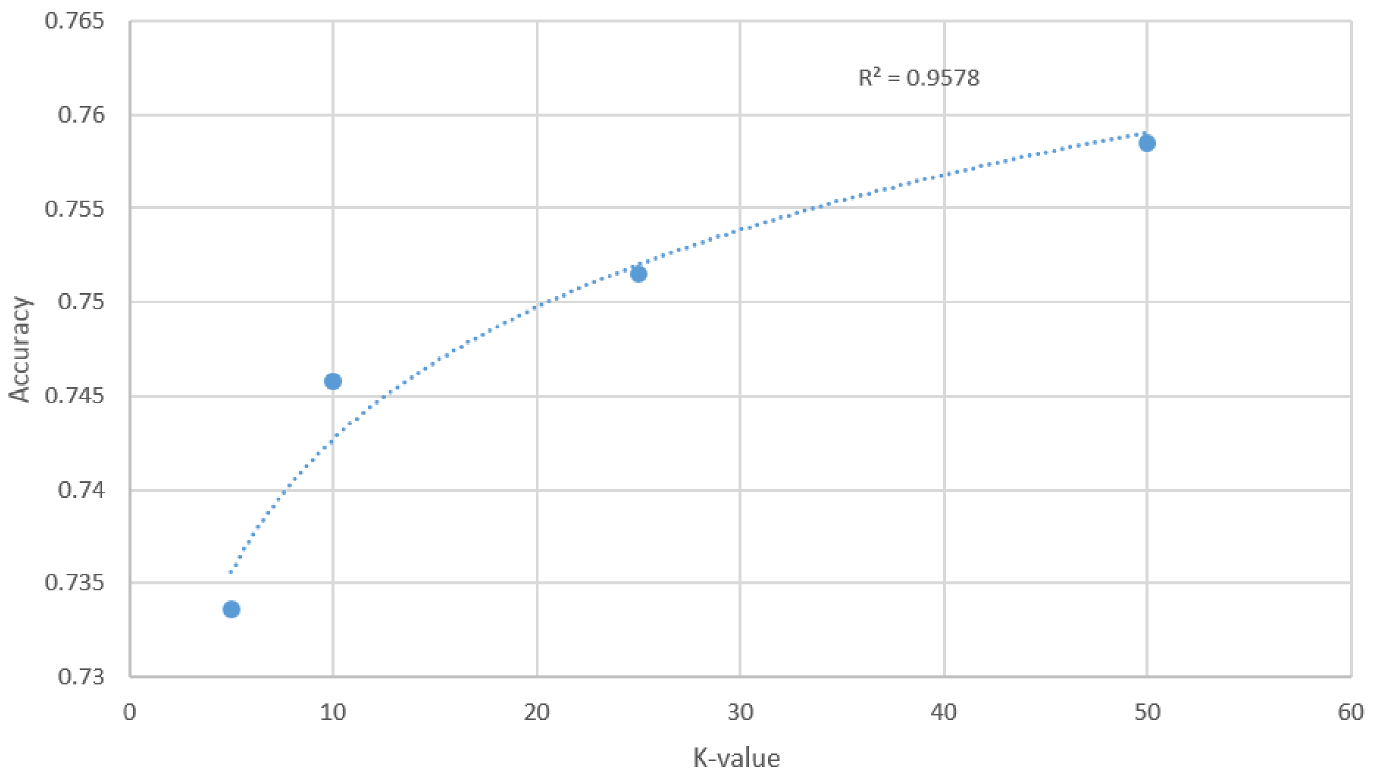

Figure 1: k-NN validation set accuracy results for k-values 5, 10, 25, and 50. 10000 best ranked TF-IDF features with n-gram range of 1 to 3 words. Fit with a logarithmic function to show the trend.

D. *K-NN: Accuracy vs K-value*

E. *Weighting functions*

Name	**Formula**
Inverse-distance	$\frac{1}{d(x_i,x)}$
Inverse-distance squared	$\frac{1}{d(x_i,x)^2}$
Gaussian	$e^{-\frac{d(x_i,x)^2}{\sigma^2}}$

F. *Accuracy vs. Weighting function*

Weighting function	**Accuracy**
Inverse-distance	0.758
Inverse-distance squared	0.761
Gaussian, $\sigma = 0.25$	0.724
Gaussian, $\sigma = 2$	0.756
Gaussian, $\sigma = 5$	0.756
Gaussian, $\sigma = 20$	0.756

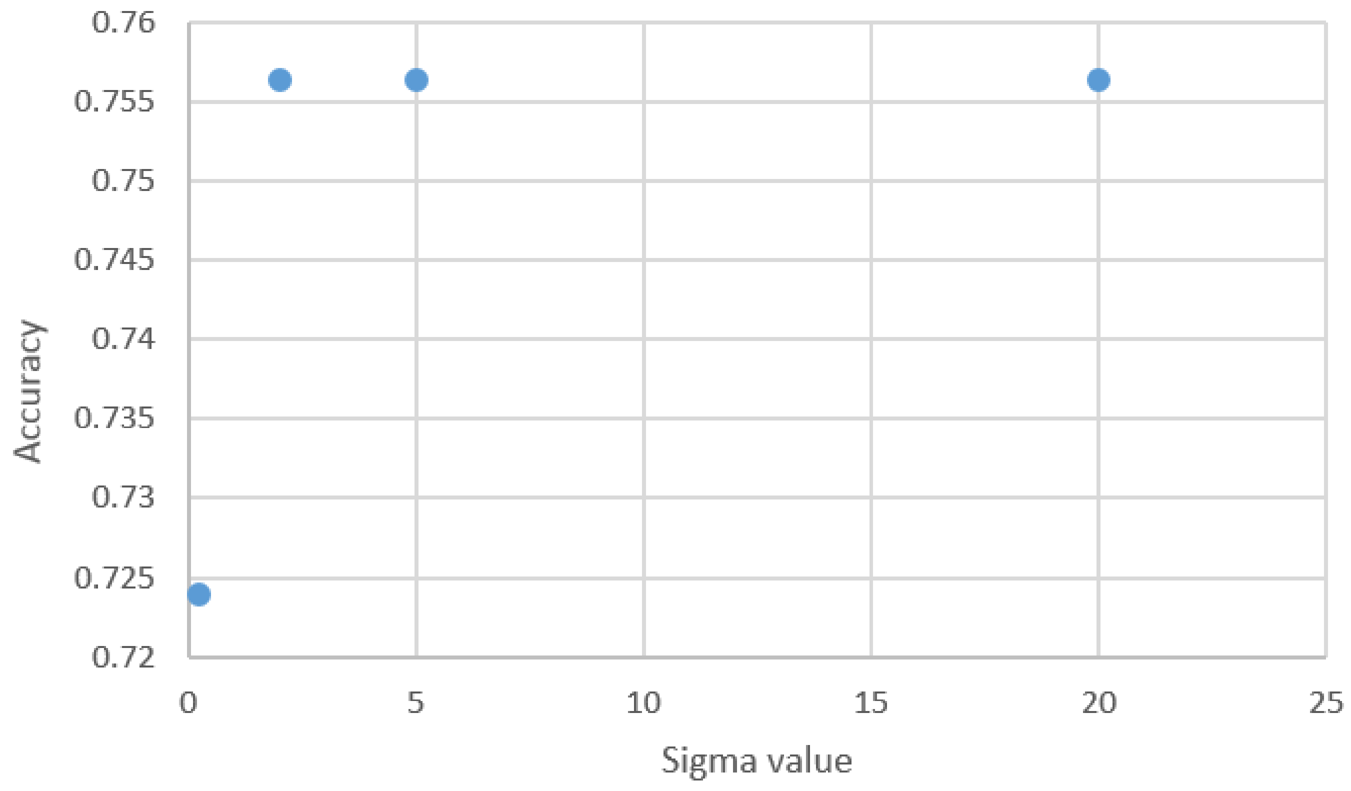

Figure 2: k-NN validation set accuracy results for sigma-values 0.25, 2, 5, and 20 used in a Gaussian kernel. 10000 best ranked TF-IDF features with n-gram range of 1 to 3 words.

G. Accuracy vs. Sigma

H. SGD Classifier: Loss vs Accuracy

Loss Parameter	**Accuracy**
Log	90.85%
Hinge	93.17%
Modified-Huber	94.18%

I. Random Forest: Precision, Recall, F-1 Score and Support

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Hockey	0.95	0.95	0.95	6680
Movies	0.96	0.99	0.97	6750
NBA	0.97	0.94	0.95	6440
News	0.88	0.83	0.85	6704
NFL	0.97	0.96	0.96	6630
Politics	0.89	0.91	0.90	6637
Soccer	0.95	0.97	0.96	6790
WorldNews	0.89	0.93	0.91	6587
Average/Total	0.93	0.93	0.93	53218

J. *SGD Classifier: Precision, Recall, F-1 Score and Support*

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Hockey	0.97	0.97	0.97	6680
Movies	0.98	0.98	0.98	6750
NBA	0.98	0.96	0.97	6440
News	0.85	0.85	0.85	6704
NFL	0.97	0.97	0.97	6630
Politics	0.90	0.89	0.89	6637
Soccer	0.97	0.97	0.97	6790
WorldNews	0.90	0.92	0.91	6587
Average/Total	0.94	0.94	0.94	53218

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[2] Max Bramer. "Introduction to classification: Naïve bayes and nearest neighbour". In: *Principles of Data Mining.* Springer, 2016, pp. 21–37.

[3] Alexander Genkin, David D Lewis, and David Madigan. "Large-scale Bayesian logistic regression for text categorization". In: *Technometrics* 49.3 (2007), pp. 291–304.

[4] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *European conference on machine learning.* Springer. 1998, pp. 137–142.

[5] Sameen Maruf, Kashif Javed, and Haroon A Babri. "Improving text classification performance with random forests-based feature selection". In: *Arabian Journal for Science and Engineering* 41.3 (2016), pp. 951–964.

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

[7] Owen Johnson Samuel Danso Eric Atwell. "A Comparative Study of Machine Learning Methods for Verbal Autopsy Text Classification". In: *International Journal of Computer Science Issues* 10.6 (2014).

[8] Karl-Michael Schneider. "Techniques for improving the performance of naive bayes for text classification". In: *International Conference on Intelligent Text Processing and Computational Linguistics.* Springer. 2005, pp. 682–693.

[9] Celine Vens et al. "Decision trees for hierarchical multi-label classification". In: *Machine Learning* 73.2 (2008), p. 185.