Multi-class text classification: Kaggle Reddit Classification

Hair A. Parra B.[1], Ashray Malleshachari[2] and Hamza Rizwan[3]

Abstract— **Classification in the real world is often more than simple binary classification problems. In this project, we had the opportunity to work on the Kaggle Reddit comments multi-class classification challenge, in which given a training set of approximately 70 000 reddits from 20 different classes, we had to implement several classifiers and submit predictions on a test set on the Kaggle platform. In order to accomplish this, we also had to implement multiple pre-processing steps such as basic text cleaning and text feature engineering, which ended up being one of the most noteworthy challenges of the task, due to the nature of the raw data. For gaining a better picture of our classifiers, we further subdivided the training set to create a small, artificial test set, on which we could obtain certain statistics and metrics on our different classifiers. Amongst all the classifiers were tested (along with a custom pipeline of preprocessing steps that comprised of lemmatization and character filtering), it was observed that the Linear Kernel Support Vector Machine model yielded the best results. Perhaps not as surprisingly, due to the nature of the problem and the given data, the overall accuracy wasn't very high, achieving only a bit over 54% in the best setting.**

I. INTRODUCTION

Reddit is an online social platform in which users post comments in different categories called "subreddits", on a wide range of topics such as news, science, movies, video games, music, books, among others. In this project, we worked on the task of implementing a text classifier for a multi-class classification problem. The training data consisted of 70 000 examples of reddit comments along with their subreddits (i.e. the community where they came from ,and in our case -the classes) and from which there were 20 different ones: 'AskReddit', 'GlobalOffensive', 'Music', 'Overwatch', 'anime', 'baseball', 'canada', 'conspiracy', 'europe', 'funny', 'gameofthrones', 'hockey', 'leagueoflegends', 'movies', 'nba', 'nfl', 'soccer', 'trees','worldnews' and 'wow'. For this task, we implemented our own **Bernoulli Naive Bayes** model from scratch, as well as other popular classic classifiers from the `sklearn` library such as **Multinomial Naive Bayes**, **Linear SVM**, **Logisitc Regression** and **Decision Trees**. As with any raw text used for classification, we also had to implement a series of data pre-processing steps in order to make the text machine-understandable and improve our final performance. The informal nature and variety of commentary done online on forums such as reddit, as well as the number of classes involved makes this a very hard classification task. Therefore, even in the best models, our performance measured in terms of accuracy does not seem to be very high. In the successive section, we describe related work concerning the problem of sentiment classification which is an archetypal problem of text classification.

[1]**Hair Parra** is a 4th year, B.A. student at McGill University, major in Computer Science, Statistics and Linguistics with focus on NLP and Machine learning. Email: jair.parra@outlook.com Site: https://blog.jairparraml.com/

[2]**Ashray Malleshachari** is a second year B.A. Computer Science major and Statistics Minor student at McGill University.

[2]**Hamza Rizwan** is a second year Electrical Engineering graduate student at McGill University

II. RELATED WORK

The problem of sentiment classification , for e.g. knowing whether an email is spam or not, or whether a certain piece of text is of "positive" or "negative" character, or the more general problem of identifying the category of a certain piece of text among a set of different categories, is a widely discussed issue in machine learning. In their paper, *Thumbs up? Sentiment Classification using Machine Learning Techniques*[1] , Bo Pang, Lillian Lee and Shivakumar Vaithyanathan discuss the problem of classifying documents, citing as one of their examples the IMBd dataset, which consists of a collection of movie reviews which could be classified as "positive" or "negative". They display how machine learning algorithms are usually more efficient and even as well performing as human labeling, that too in a substantially reduced amount of time. In particular, they discuss the **Naive Bayes classifier**, **Maximum Entropy** , and **Support Vector Machines (SVMs)**.Excellent expositions of the underlying mathematical formulations of these algorihtms can be found on Standford CS229 notes[2][3], by the renowned AI & ML researcher Andrew Ng. Their paper also discusses that the use of bigrams seems to entangle more relevant information for the classification problem, as well as POS tagging and words positions, among other factors.

III. DATASET AND SETUP

Given 20 subreddit classes, the original training dataset consisted of a collection of 70 000 different comments along with their corresponding labels. As it can be observed in figure 1, the labels are well balanced. We also had at our disposition a testing training set of 30 000 observations to make final predictions on , but no testing labels.Therefore, we decided to further split the original training set and create a mini-testing set. For this purpose, we randomized the testing set and then split it into a 1:9 ratio, creating a training set (test and labels) of 63000 data points, and a testing set of 7000 data points. The reason for this is that we also wanted to obtain further metrics, which will be displayed in a later section.

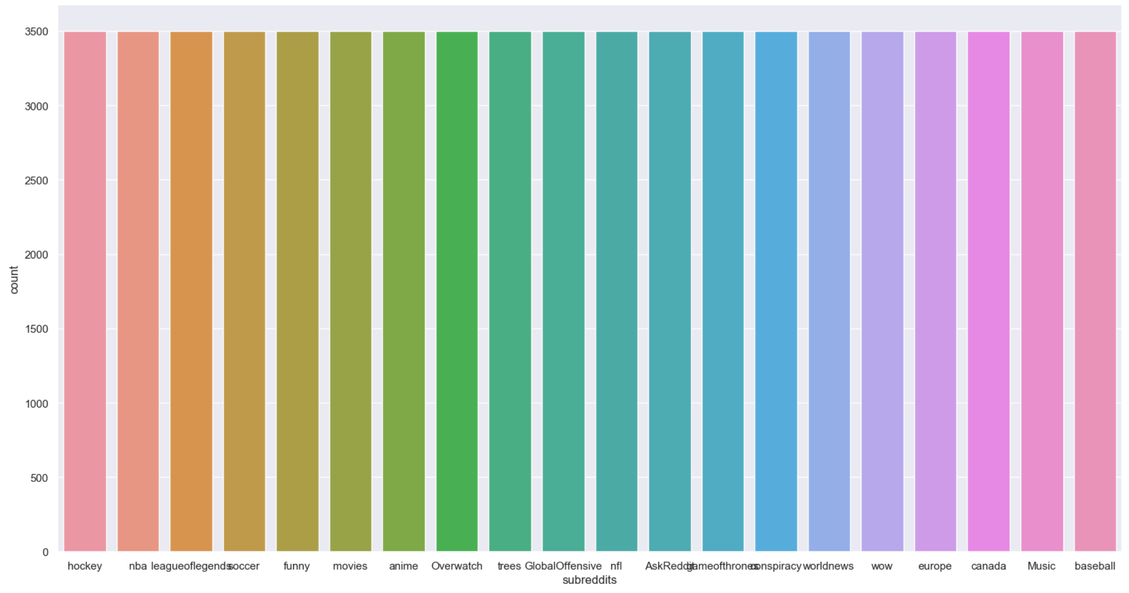


Fig. 1: Distribution of the provided dataset. We observe that these are well balanced.

A. *Text preprocessing*

In general, we applied the following preprocessing steps to all the training and testing feature sets (i.e., the reddit comments):

1) Tokenize the text
2) Stem or lemmatize each of the tokens
3) Convert to lowercase
4) Remove stopwords
5) Filter pronouns, determiners, punctuation, numbers, symbols and other tags we judged not informative enough.
6) Remove tokens of length=1

Tokenizing is a common step applied in order convert sentences to words. In the second step, we created both stemmed and lemmatized versions of the dataset, and for each of these, we applied steps 3-6, with the primary aim to filter all uninformative tokens that would not contribute to the classification problem. In what follows, we discovered that using either tokenization or lemmatization did not affect performance, hence we only report results based on the lemmatized pre-processed feature sets. Although not mentioned here, we also applied vectorization, tf-idf weighting as well as normalization. However, as we implemented these in a pipeline along with the models, they will be described in the following section.

IV. Proposed approach

A. *Initial proposed approach*

Our approach consists of trying to implement a series of classic supervised machine learning models, including **Bernoulli NB**, which we implemented from scratch (as was a key requirement of this project), as well as `sklearn`'s `MultinomialNB`, `LogisiticRegression`, `LinearSVC` and `DecisionTreeClassifier`, as well as the `AdaBoostClassifier`. Note that `LinearSVC` refers to the Linear Kernel SVM.

In general , our approach consisted of the following series of steps:

- `raw data`
- `→ text preprocessing`
- `→ vectorizer`
- `→ tfidf-transformer`
- `→ normalizer`
- `→ classifier`
- `→ randomized search CV`

where:

1) **Text_preprocessing**: as has been described in the previous section
2) **Vectorizer** : transforms the input text into one hot-encoded vectors
3) **Transformer** : assigns a tfidf based weight to the vectors
4) **Normalizer** : normalizes the vectors so they have the same range
5) **Classifier** : the classifier algorithm
6) **Randomized Search Cross-validation**: also called **Grid search**, looks through a set of parameters and 5-fold cross-validating each possible combination.

Here, we reiterate that the first step is the series of cleaning steps described in the previous section. Steps 2-5, in particular, were integrated directly into the `sklearn Pipeline` object, which is why we describe them in this section. The `Vectorizer` step is accomplished by `sklearn CountVectorizer`, with the following relevant parameters, as described in the Sci-kit learn's documentation[1]:

- `ngram_range`: range of n-grams to be considered
- `max_df`: ignore terms that have a document frequency strictly higher than the given threshold
- `min_df`: ignore terms that have a document frequency strictly lower than the given threshold
- `max_features`: build a vocabulary that only considers the top `max_features` ordered by term frequency across the corpus.

Similarly, in step 3 we use **Tf-idf weighting**[2], also from `sklearn TfidfTransformer` with the parameters:

- `norm`: norm for each row of the vectors (e.g. l2-norm)
- `use_idf`: enable inverse-document-frequency reweighting.
- `smooth_idf`: add Laplace priors (Laplace smoothing) to the counts to avoid numerical instability.
- `sublinear_tf`: Consider sublinear term frequency scaling, i.e. $1 + log(t_f)$

After these steps, an appropriate classifier as the one of the ones described before (e.g Multinomial NB) was applied. Finally, we decided to perform **5-fold cross-validation grid search** on the artificial test set, in order to perform hyperparameter tunning.

B. Final approach

After following these steps, we notice the accuracy was not significantly increasing even after the hyperparameter tuning stage. In the end, we realized we had applied **over preprocessing**, thus eliminating information that was relevant to the problem. Then, we chose the best model and once again re-trained the classifier on the full original training set without our custom pre-processing, and instead relied on `sklearn`'s `TfidfVectorizer` module, which is essentially a combination of the previously described `CountVectorizer` and `TfidfTransformer`. From this we calculated and report a 10-fold cross-validated accuracy. Finally, we took our best model (in this case **Multinomial NB** with hyper-tunned parameters as a base estimator to train the popular **Ada Boost Classifier**[3] ensemble model. The respective hyperparameters are reported in the **Results** section.

C. Bernoulli Naive Bayes Implementation

In addition to implementing competitive classifiers with the help of libraries such as `sklearn`, but also we had the chance to implement the Bernoulli Naive Bayes classifier from scratch. Bernoulli Naive Bayes and Multinomial Naive Bayes both are suitable for working with discrete data, with the difference [4] that in Bernoulli Naive Bayes, the word feature vectors are binary values (representing whether the word exists or not), whereas in Multinomial Naive Bayes, the features have information about word frequency.

The `BernoulliNB` class that we implemented contains two methods: `fit` and `predict`. For The method `BernoulliNB.fit`, was implementing in the following fashion:

1) Define
   a) K = Total number of classes
   b) m = Total number of features
   c) n = Total number of training examples

   Then,
2) The $(K,)$-shaped numpy array is the vector or priors $\pi = \{\pi_1, \ldots, \pi_K\}$, i.e. it contains the estimated probabilities $\widehat{\mathbb{P}}(y = i)$ for every class i.
3) The numpy array `ParameterMatrix` with dimensions (K, m), represents the parameter matrix ϕ ,i.e. the entry $\phi_{k,j}$ is the MLE estimate of $\widehat{\mathbb{P}}(x_j = 1|y = k)$ for $j = 1, \ldots, m$ and $k = 1, \ldots, K$, trained on the n available observations.

We calculate `ParameterMatrix` using `np.mean` to calculate a column-wise mean for examples in each class. This vectorized approach in numpy achieves the goal of calculating the conditional probability more efficiently compared to calculating a summation using high-level Python loops, which typically would yield a $O(Kmn)$ running time.

D. Laplace Smoothing

To deal with the issue of numerical instability and division by zero, we also implemented **Laplace smoothing** along with our vectorizer approach.For this purpose, we added 2 additional rows when calculating the column-wise mean: a row of `np.ones` and a row of `np.zeros` for each class. This achieves the effect that the estimates of $\Pr(x_j = 1|y = i)$ for every class k and every parameter m are now given as:

$$\phi_{k,j} = \widehat{\mathbb{P}}(x_j = 1|y = k) = \frac{\sum_{i=1}^{n} \mathbb{1}\{x_j^{(i)} = 1 \wedge y^{(i)} = k\} + 1}{\sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = k\} + k}$$

Similarly, the prior estimates are

$$\phi_k = \widehat{\mathbb{P}}(y = k) = \frac{\sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = k\} + 1}{m + k}$$

Finally, for a new observation $x = [x_1, \ldots, x_m]$, when we make a prediction with the `predict` method, the following calculations are performed:

$$\hat{y} = \underset{k}{argmax} \widehat{\mathbb{P}}(y = k|x)$$

where for $\phi_k = [\phi_{k,1}, \ldots, \phi_{k,j}, \ldots, \phi_{k,m}]$

$$\widehat{\mathbb{P}}(y = k|x) = log(\pi_k) + x log(\phi_k)^T + (1 - x)(log(1 - (\phi_k))^T$$

[1]See https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[2]See https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

[3]See https://en.wikipedia.org/wiki/AdaBoost

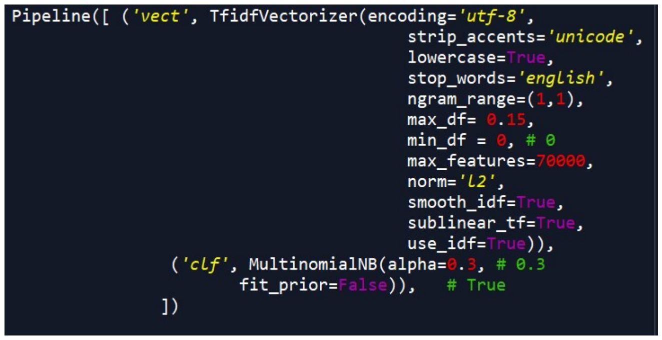
On a final implementation note, the method `BernoulliNB.predict` uses the `ParameterMatrix` we just computed using `BernoulliNB.fit` to estimate $\mathbb{P}(y = k|x)$ i.e. the probability that each example belongs to class k. We know that the posterior probability is proportional to the prior probability times the class conditional probabilities for each class. Starting with the Maximum Likelihood Estimator (MLE) function given in the slides[5], we derived an equivalent matrix form of this equation. Using this vectorized function, we are able to generate prediction labels for the entire dataset at once by performing 2 matrix multiplications. Implementing the predict function using numpy vectorization exponentially reduces our time complexity because we are using highly optimized lower-level numpy matrix multiplications instead of iterating over features, classes, and examples in surface level Python code to generate labels for each test example. This approach reduced our total training time to about 9.98 seconds.

V. Results

Once all the previous steps (i.e. preprocessing, pipeline and hyper-parameter tuning) were performed, we obtained 5 cross-validated accuracy for each of them (on the original full training set, i.e. with 70 000 datapoints.). These, along with the running times for each of the algorithms are displayed in the following table V.

Model	Cross-val. acc	Running-time (s)
Custom Bernoulli NB	50.25%	6.44s
Multinomial NB	55.02%	5.77s
Raw features M.N.B.	**57.10%**	**3.10s**
Logistic Regression	54.02%	24.88s
Linear SVM	55.46%	21.56s
Decision Trees	31.16%	34.16s
Ada Boost	**55.57%**	106.66s

As the reader can observe, our best estimator was the **Raw features Multinomial Naive Bayes**, for which we obtained a local cross-validated accuracy of **57.01%**, for which we applied the following tuned hyper parameter configuration:



```
Pipeline([ ('vect', TfidfVectorizer(encoding='utf-8',
                                        strip_accents='unicode',
                                        lowercase=True,
                                        stop_words='english',
                                        ngram_range=(1,1),
                                        max_df= 0.15,
                                        min_df = 0, # 0
                                        max_features=70000,
                                        norm='l2',
                                        smooth_idf=True,
                                        sublinear_tf=True,
                                        use_idf=True)),
                ('clf', MultinomialNB(alpha=0.3, # 0.3
                       fit_prior=False)),   # True
               ])
```

Fig. 2: Hyper parameter configuration for the Multinomial Naive Bayespipeline.

This specific choice of parameters was obtained through **Randomized Cross-validation Grid Search**, using `sklearn`'s `RandomizedSearchCV` on the artificial dataset (i.e., the original training dataset the was split further). We can also observe the confusion matrix presented in figure 3 (See end of the document). Using this model to make predictions on the full test-set (with 30000 datapoints), we were also able to obtain **57.655%** accuracy on the Kaggle leader-board. It is also worth noticing that before deciding on Multinomial Naive bayes as a final model, our best classifier with lemmatized features was actually AdaBoost, trained on a LinearSVM classifier. This algorithm in fact trains a number of classifiers, penalizing errors of the previous ones and therefore improving them. Moreover, AdaBoost has been conjectured to never overfit[6]. However, this was trained using the **Linear SVM** as a base estimator, and it's performance is only slightly better, and it's running time significantly slower.

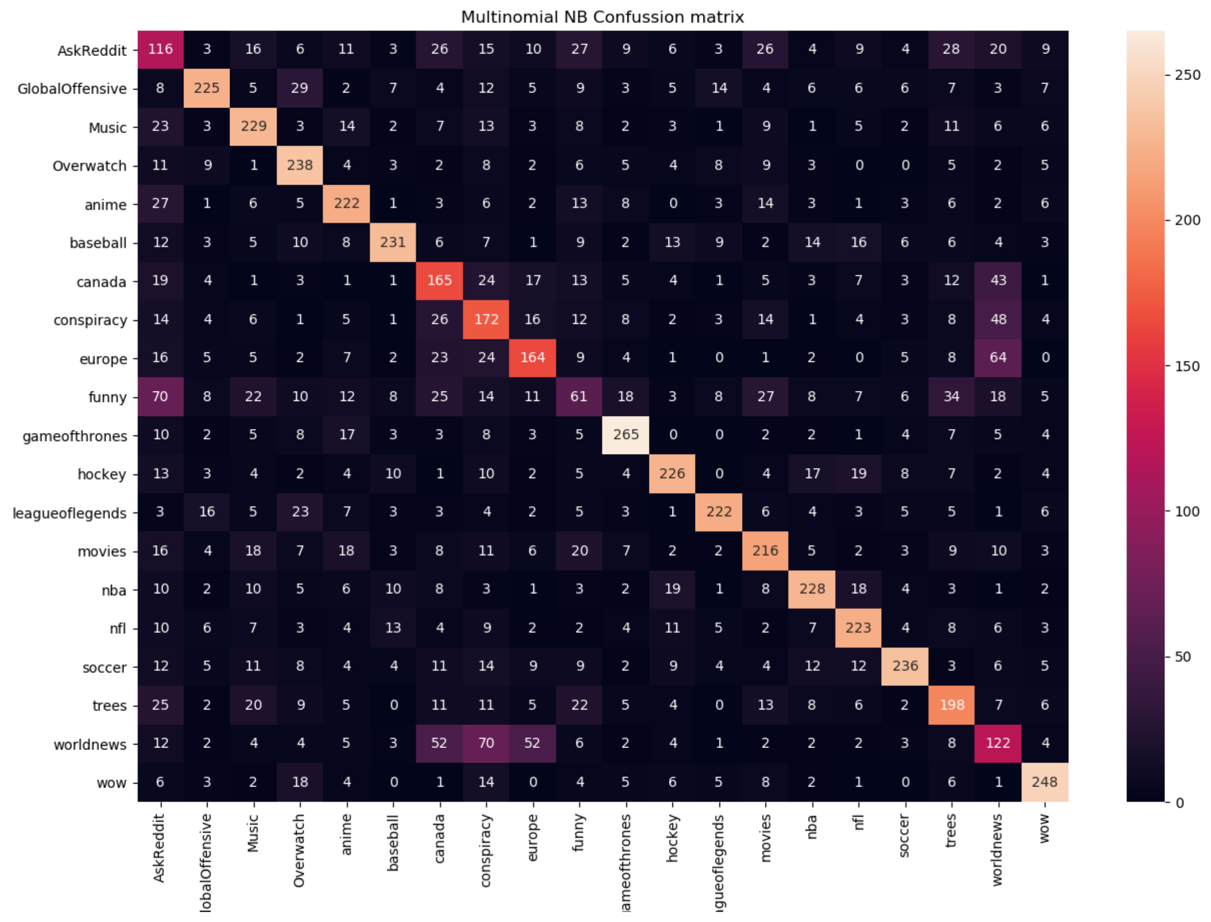
VI. Discussion and Conclusion

Although binary classification is a popular problem that has been widely discussed in the machine learning paradigm, it nonetheless remains an extremely relevant problem for a variety of situations in real life, both in academia and the industry. However, not all problems are of the same nature, and we realize how difficult it can actually be for an algorithm (and perhaps even to a human?) to correctly classify observations which might fall into several classes. We also see, that some algorithms are more adept than others, particularly in the case of Bernoulli Naive Bayes, for instance, in spite of its simplicity, it achieves surprisingly relatively "good" performance as well as great running time in comparison to other classifiers. In this case, however, we notices that the best performance was obtained by the **Multinomial Naive Bayes classifier**.

One interesting thing we notice is that many of the errors might be, in fact, due to a lack of mutual exclusivity between the categories, i.e., many of the reddit comments could in fact belong to multiple categories rather than just one. For instance, referring to the confusion matrix in figure 3, we observe that 36 reddit comments coming from the category "canada", 53 from "conspiracy" and 72 from "europe", were incorrectly classified as "worldnews". However, it is not ridiculous to think that "worldnews" in fact is a superset of these categories. We also observe the opposite: many reddits in the category "worldnews" were improperly classified as "canada","conspiracy" and "europe".

VII. Further work

In order to improve performance, in further approaches we would like to explore consist of employing neural networks and word embeddings such as word2vec and doc2vec to learn more meaningful relationships and features that could potentially improve the classification results, as these capture more semantical and syntactical information rather than simple count features; see [7].



	AskReddit	lobalOffensive	Music	Overwatch	anime	baseball	canada	conspiracy	europe	funny	ameofthrones	hockey	agueoflegends	movies	nba	nfl	soccer	trees	worldnews	wow
AskReddit	116	3	16	6	11	3	26	15	10	27	9	6	3	26	4	9	4	28	20	9
GlobalOffensive	8	225	5	29	2	7	4	12	5	9	3	5	14	4	6	6	6	7	3	7
Music	23	3	229	3	14	2	7	13	3	8	2	3	1	9	1	5	2	11	6	6
Overwatch	11	9	1	238	4	3	2	8	2	6	5	4	8	9	3	0	0	5	2	5
anime	27	1	6	5	222	1	3	6	2	13	8	0	3	14	3	1	3	6	2	6
baseball	12	3	5	10	8	231	6	7	1	9	2	13	9	2	14	16	6	6	4	3
canada	19	4	1	3	1	1	165	24	17	13	5	4	1	5	3	7	3	12	43	1
conspiracy	14	4	6	1	5	1	26	172	16	12	8	2	3	14	1	4	3	8	48	4
europe	16	5	5	2	7	2	23	24	164	9	4	1	0	1	2	0	5	8	64	0
funny	70	8	22	10	12	8	25	14	11	61	18	3	8	27	8	7	6	34	18	5
gameofthrones	10	2	5	8	17	3	3	8	3	5	265	0	0	2	2	1	4	7	5	4
hockey	13	3	4	2	4	10	1	10	2	5	4	226	0	4	17	19	8	7	2	4
leagueoflegends	3	16	5	23	7	3	3	4	2	5	3	1	222	6	4	3	5	5	1	6
movies	16	4	18	7	18	3	8	11	6	20	7	2	2	216	5	2	3	9	10	3
nba	10	2	10	5	6	10	8	3	1	3	2	19	1	8	228	18	4	3	1	2
nfl	10	6	7	3	4	13	4	9	2	2	4	11	5	2	7	223	4	8	6	3
soccer	12	5	11	8	4	4	11	14	9	9	2	9	4	4	12	12	236	3	6	5
trees	25	2	20	9	5	0	11	11	5	22	5	4	0	13	8	6	2	198	7	6
worldnews	12	2	4	4	5	3	52	70	52	6	2	4	1	2	2	2	3	8	122	4
wow	6	3	2	18	4	0	1	14	0	4	5	6	5	8	2	1	0	6	1	248

Fig. 3: Confusion matrix for the Multinomial Naive Bayes model trained on 63000 observations and tests on 7000 datapoints.

VIII. Statement of Contributions

Hair Parra : Github Repository setup, team organization, text preprocessing, pipelines implementation, model selection and testing, hyperparameter tuning, sources research, most of report write-up contribution.

Ashray Malleshachari : Naive Bayes implementation and testing, sources research, report write-up contribution. (Naive Bayes implementation)

Hamza Rizwan : Naive Bayes implementation and testing, sources research, report write-up contribution. (Naive Bayes implementation)

IX. Source Code

The code used for this project can be found publicly at https://github.com/JairParra/Miniproject_2_COMP551 .

References

[1] S. V. Bo Pang Lillian Lee, "Thumbs up? sentiment classification using machine learning techniques," *online:* https://www.cs.cornell.edu/home/llee/papers/sentiment.pdf, 2019.

[2] A. Ng, "2. naive bayes, cs229 lecture notes," *online:* http://cs229.stanford.edu/summer2019/cs229-notes2.pdf, 2019.

[3] A. Ng, "V. support vector machines, cs229 lecture notes," *online:* http://cs229.stanford.edu/notes/cs229-notes3.pdf, 2019.

[4] R. Gandhi, "Naive baiyes classifier," *online:* https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c, 2018.

[5] W. L. H. with slides and content from Joelle Pineau, "Ensemble methods, comp 551 lecture notes," *online:* https://cs.mcgill.ca/~wlh/comp551/slides/10-ensembles.pdf, 2019.

[6] R. E. Schapire, "Explaining adaboost," *online:* https://www.cs.cornell.edu/home/llee/papers/sentiment.pdfhttp://rob.schapire.net/papers/explaining-adaboost.pdf.

[7] K. C. G. C. J. D. Tomas Mikolov, Ilya Sutskever, "Distributed representations of words and phrases and their compositionality," *online:* https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-t pdf.