DV2578 Project

Muhammad Ahmad Imran Rafique
Masters in Computer Sciences
Blekinge Tekniska Högskola
Karlskrona, Sweden
murf21@student.bth.se

Syed Saif Ahmed
Masters in Computer Sciences
Blekinge Tekniska Högskola
Karlskrona, Sweden
saah21@student.bth.se

I. Introduction

A. About the project

This project aims to compare four existing machine-learning algorithms against our proposed model for the task of sentiment analysis via binary text classification. The code for the project has been written in python. A 10-fold cross validation test is performed on the model where accuracy is used as the main metric to collect data for our project. A comparison of accuracy between algorithms will be performed and then evaluation of accuracy using the Friedman and holm-Bonferroni test shall be performed to check the significance of our proposed model (control model) against other algorithms.

B. Data Used

The dataset used in this project is the IMDB 50K dataset. It comprises of 25000 reviews each belonging to a positive and negative label.

C. Algorithms selection criteria

The algorithms for our task of text classification were based on the following criteria

1) Should be able to handle sparse matrices
2) Should compute in a reasonable time when number of features are greater than samples
3) Should be able to operate on high dimensional data
4) Shown some form of utility in prior studies.

A secondary criterion was to have base models from categories like simple, high dimensional, bagging and boosting.

The reason behind the criteria chosen is because text processing in machine learning generally requires or works in these conditions. The textual data is not understood as is by the algorithm and is transformed into a numeric or vector representation of that word or group of words. Naturally, it results in a high dimensional vector representation which, albeit, is understandable by the machine/ algorithm, depending on vector size and physical resources, can take a large time to compute. Prior studies helped motivate our algorithm choice here as well.

With that, our existing models chosen are

1) Gaussian Naive Bayes
2) Support Vector machine
3) Random Forest Classification
4) XgBoost

And Our Proposed model is a Voting Classifier which is an ensemble of

1) Logistic Regression
2) Linear SVC
3) Decision Tree

D. Motivation Behind Algorithms

Naive Bayes is a family of algorithms showing good base performance in classification tasks. They need less tunable parameters and can be used as baseline models although their direct practical nature is arguable given their conditional independence assumption. Chen Et al [1] have described naive bayes in similar terms along with its interpretability and incremental learning feature. For this purpose, they have proposed a method using naive Bayes for feature extraction when data has a large dimension in order to select effective features. The speed of execution along with the use on high dimensions was the interesting aspect in their paper. Jiang Et al [2] used naive bayes to show how a bolstered error approach against cross-validation when dealing with small sample high dimensional spaces. Their interest area was on genomic signal processing and they ran a simulation showing how the approach was better than the existing one they compared against. What was interesting here was the utility of naive bayes in a feature greater than the sample situation. Dhillon et al [3] in their paper discuss a process to reduce loss function in an information theory optimization task. For high dimensional sparse data, they utilized a supervised naive bayes as a prior base to overcome infinite entropy problem with a local search for local minima to create a computationally efficient algorithm for the task. With this use in mind for sparse, high dimensional data, naive bayes was selected as a candidate.

Support vector machines is another algorithm chosen in this project. It has been shown to map data according to appropriate labels even in features greater than the sample and higher dimensions. Although such an arrangement is susceptible to overfitting, Support vector machine with appropriate kernels and automatic regularization, make it a prime choice to use in research works. Liu et al [4] in their paper modified support vector calling it an improved fuzzy support vector machine, where a high dimensional sparse training data has an area of it selected by specifying some radii as representation of the whole data for further training. Albeit they took a subset of

the data, their study showed that tuning certain parameters allows a good trade off between training time, a huge factor in SVM, and generalization ability. Hochbaum et al [5] in their paper discuss that machine learning techniques like K Nearest neighbor or Support vector machine rely on pair wise input for learning, a form of similarity based learning. But with size increasing quadratically, their scalability comes into question. They proposed a method of sparse computation where only relevant information is computed as a sparse similarity matrix without looking at all pairwise similarities which is then projected into a low dimensional space and a grid is used for similarity calculation. Such an arrangement sped up the support vector machine employing a radial basis kernel which showed really good performance on high dimensional data which also translated into less training and tuning time and such that the low dimensional space data could be mapped to higher. The important thing here is to note the consideration for SVM. The while they are good in high dimensional sparse data, the training time is a huge issue and cutting down data as much as possible and still be able to generalize is a point to consider which we did and will further elaborate in the method section.

A Random forest model has shown remarkable performance on high-dimensional, noisy and unbalanced data. Liu et al [6] proposed a semantic aware variation of random forest over 30 dataset and demonstrated better information retrieval results as compared to the state-of-the-art. In natural language processing tasks we have large datasets to work with and since random forest fits multiple models to subsets of data [7] and has shown good results on classification tasks. While high-dimensional sparse matrices do take time, this model has shown to perform in a reasonable time.

XgBoost is a sequential shallow decision tree model that is highly scalable and has been shown to avoid overfitting that may come up with features greater than sample arrangement but does have certain troubles in sparse data. However, XgBoost and its variants have been shown to be fruitful in many practices and research work. Zhang et al [7] in their paper design a stacked sparse autoencoder network XgBoost or SSAE-XGB, that works on unbalanced data and sparsity constraint to achieve generalizability by reducing high dimension enough to get a good data representation. This model was shown to have high performance in terms of F1 score. Preparation of sparse high dimension data was the key here but XgBoost showed its utility in practice.

The model we proposed is an ensemble of Linear Support vector machine, Decision Tree and Logistic regression. Support vector and decision tree were selected along the same reason as discussed above for support vector and random forest (essentially a combo of decision trees). Logistic regression was added to the mix because of its utility as a binary classifier (our project task). Albeit prone to overfitting when it comes to a large enough space, our ensemble proposition aims to leverage the advantages of each model to produce a better result against the state of the art. The ensemble model is part of a voting classifier with hard voting. More details in Method section about hyper parameters.

Other models were considered however for the aforementioned reasons and sources, the algorithms mentioned in previous part were selected. To summarize, these algorithms showed promise in classification tasks even when we have high dimensional data. Some have a time issue and sparse data handling issue, however, in the Method section, data preparation to make things equal for all algorithms will be discussed.

II. Method

A. High level view of process

Fig. 1. illustrates what process is undertaken in the project.

B. Project Environment

The project was conducted using python with jupyter notebooks. The libraries used in the project are numpy, pandas, sklearn, time, re, nltk, xgboost, bs4 and html, and the whole project operates under a seed value of 5678.

C. Data preprocessing

This part explains the details of the first three blocks of Fig. 1. The imdb data is loaded into the project. It consists of 2 columns, the review and its sentiment, either positive or negative. The data is balanced with 25000 rows for each sentiment. For sake of ease, the sentiment have been coded 1 for positive and 0 for negative. The reviews and sentiment columns will be split and sent to a k-fold function later for processing.

To list out the exact procedure

1) Load the data
2) Split column of review and sentiment and label sentiment value 1,0 for positive and negative.
3) In review data, we run through all reviews and from each review remove the html tags, collect alphabet data only,convert review data to lowercase and tokenize the strings (split string into list of strings).
4) With the list of list of string, we lemmatize the data and ignore if its a stop word.
5) Join the lemmatized words back as a single string

To elaborate the procedure enlisted above, we first have to know that the review column is ”dirty” and has to be cleaned up for processing first. It has, on average **1310 characters and 230 words per review**. Those review have alphabets, numbers, punctuation's, html tags etc along with stopwords (words that have high frequency and no discriminative power for NLP tasks which are removed). We also have similar words that are used in different tenses, e.g, cry, crying and cried. Since our task at hand istext classification, we can ignore certain contextual information about tenses and get root word only. The process we used for this root word analysis is lemmatization, which brings word down to its contextual root (Wordnet is used by nltk). E.g, cry ,cried, crying will become simply cry. We first tokenize the review string to get list of strings and we check whether or not it is a stop word. If it is we ignore else we lemmatize the word and add it back to the

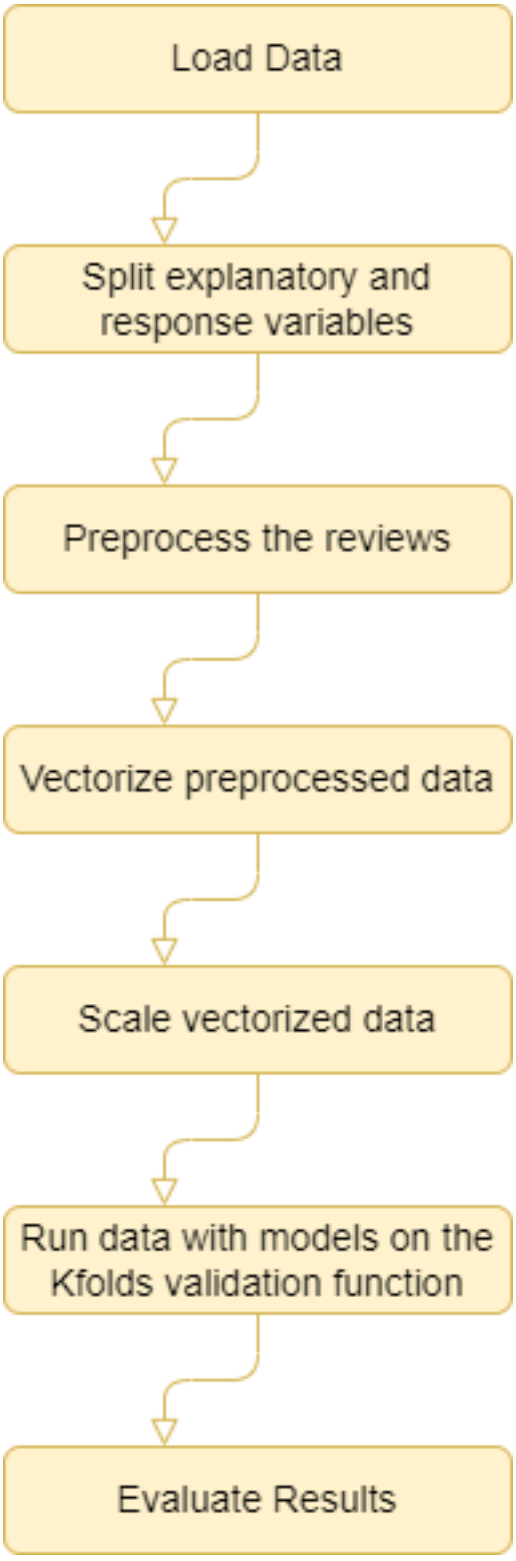

Fig. 1. Example of a figure caption.

list and then convert the list of strings back to a single string. The only stop word excluded from checking was the word "not" because looking at the data, it had certain discriminative properties that made sense to retain in a dataset about reviews.

The result of our preprocessing was that we now had on average **805 characters and 119 words per review**. We now have less noise and more relevant data to work with.

D. Vectorization of preprocessed data

After we finish preprocessing our data, we still have text. Now, a machine learning algorithm needs numbers to work with. There are plenty of algorithms that represent a word or collection of words. We opted for bag of words approach which is realized by the CountVectorizer Function in sklearn. What count vectorizer does is see total unique words in the entire corpus and per sentence breaks the words into columns representing count of the word in that sentence. Fig.2 gives an idea.

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Fig. 2. Count vectorizer illustrated (via vitalflux.com)

The vocabulary can get so large that processing time would become high. Given that we intend to use support vector machine, certain choices were made to give an even playing field with respect to data engineering. From the preprocessed corpus, we found that we have a unique vocabulary of 134282 words. Given that we have 50000 rows, this is a fairly large vectorized dataset to work with. Fortunately, we can specify vocabulary size. Counting words that have exactly 1 occurrence gives us 76297 words which is more than half our corpus. More than one means that the word's discriminative power is less and less. Through trial and error, we collected 1000 as an appropriate vocabulary size which covers 99% of the corpus. By specifying 1000, we mean that the 1000 most frequent words be considered a part of our vectorization. As a result, we get a 50000x1000 matrix. This matrix will be a sparse matrix.

E. Scale vectorized data

Support vector machine, an algorithm we use in this project, is notorious for being sensitive to data scaling. Features with higher values tend to dominate the results. The other algorithms work well without but it's good practice to scale. It will speed up training time. We chose a min-max scaler and scaled data between [0,1]. Now the data is ready to be sent to run with the custom kfolds function.

F. Model Hyper parameters

The models from sklearn were run with default configurations mostly. The following are models with different hyperparameters used after certain trials.

1) SVM uses linear kernel with gamma set to auto and a max iteration value of 2500. SVM takes a large time to execute and setting max iteration value around 2500 executed without error and converged. Linear kernel was seen to perform well on tasks of similar nature in other studies and projects.
2) XgBoost uses the objective parameter of "binary:logistic" for our binary classification task.
3) Logistic Regression also has a convergence problem like support vector which is why the max iteration is set to 1000 and the solver function used is the "lbfgs".
4) The voting classifier of our ensemble employs the hard voting method instead of soft. Hard voting selects highest number of votes while soft voting combines probabilities of each prediction in each model and selects the highest cumulative. Hard voting was deemed more appropriate to use.

G. Run the models

The kfold (10-fold) cross-validation model, data models, vectorized data and corresponding labels are pushed into a custom function for this project called evaluate_model_training(). The models are run in a 10-fold cross-validation and returns information like training time, accuracy and f1 score. The results are converted into a dictionary which is further converted into a pandas dataframe and gets mean and standard deviation appended at the end. We have a mechanism to save the dataframes into excel sheets. The one for this project may be found in the data folder titled "6.FINAL.xlsx".

H. Design considerations

Following design considerations were performed but discarded due to lack of utility or performance impact

1) Tf-idf vectorization was considered however, it resulted in a 50000x91707 size matrix which was not computationally feasible to continue with, given the hardware resources.
2) Principal component analysis was performed on the sparse matrix while trying to keep atleast 90% of explained variance. This was not used because we were left with 800 columns to process which made no significant improvements albeit showed overfitting in case of some algorithms (Refer to 3.PCA applied.xlsx in Data Folder).
3) Vocabulary length 100-800 was also used. (100Vectorizer.xlsx,With800length.xlsx in Data Folder)
4) A grid search CV was run on a support vector machine to find optimal hyperparameters. The processing time took hours without completion so recommended parameters based on research and practice were set.

III. Result and Analysis

A. Data Obtained

Our training function, as previously mentioned, returns training time, accuracy and f1 score. They are shown in the following sections.

TABLE I
TRAINING TIME BETWEEN FOLDS (SECONDS)

Folds	gnb	svc	rfc	xgb	proposed
1	1.3655	208.2012	56.6322	89.7417	43.5214
2	0.5957	194.3387	50.5783	89.3802	40.5604
3	0.5926	208.2895	39.8780	89.6318	45.1023
4	0.5820	193.5719	37.3157	88.8879	44.0547
5	0.5812	208.7499	36.6541	88.9060	41.1707
6	0.5779	194.0285	36.2625	88.9604	42.4192
7	0.5960	202.2217	36.9528	91.2451	46.2521
8	0.5733	204.1414	36.5366	89.6427	43.7436
9	0.5863	196.3817	37.1233	90.0185	44.3786
10	0.5774	191.6305	36.1532	89.0088	44.7451
mean	0.6628	200.1555	40.4087	89.5423	43.5948
std	0.2470	6.8844	7.1771	0.7201	1.7631

TABLE II
ACCURACY BETWEEN FOLDS

Folds	gnb	svc	rfc	xgb	proposed
1	0.8084	0.5742	0.8364	0.8096	0.8682
2	0.8088	0.543	0.8332	0.8144	0.8694
3	0.8094	0.5594	0.8326	0.8026	0.8576
4	0.8156	0.5706	0.8366	0.8088	0.8588
5	0.807	0.5462	0.8212	0.7938	0.8526
6	0.806	0.5502	0.8294	0.8044	0.8652
7	0.8098	0.5494	0.832	0.8032	0.8642
8	0.811	0.5628	0.8314	0.8064	0.8614
9	0.8104	0.5438	0.825	0.7974	0.8602
10	0.8102	0.576	0.832	0.8102	0.8648
mean	0.80966	0.55756	0.83098	0.80508	0.86224
std	0.0026	0.01277	0.0048	0.0062	0.0051

TABLE III
F1 SCORE BETWEEN FOLDS

Folds	gnb	svc	rfc	xgb	proposed
1	0.8057	0.6811	0.8376	0.8211	0.8707
2	0.8072	0.6646	0.8340	0.8259	0.8709
3	0.8062	0.6765	0.8330	0.8143	0.8594
4	0.8135	0.6794	0.8355	0.8193	0.8606
5	0.8073	0.6673	0.8222	0.8051	0.8550
6	0.8038	0.6705	0.8292	0.8155	0.8663
7	0.8093	0.6709	0.8326	0.8143	0.8665
8	0.8082	0.6736	0.8316	0.8164	0.8624
9	0.8086	0.6688	0.8250	0.8086	0.8619
10	0.8081	0.6812	0.8306	0.8204	0.8657
mean	0.8078	0.6734	0.8311	0.8161	0.8639
std	0.0026	0.0059	0.0047	0.0061	0.0050

B. Comments

The results from Table 2 show our proposed model yields a higher accuracy in each fold and overall. The F1 score also corroborates this fact. However, for this project, we are focusing on accuracy mainly. Random forest classification in

terms of performance is a close second. These are merely absolute numbers but now we seek how different or significantly different the groups are. For this, we will perform a friedman test and if required, a holm-Bonferroni test.

C. Applying Friedman Test

Friedman test is used to test for differences in groups when data is ordinal. We will calculate the Friedman statistics with following parameters on accuracy. Table 2.

1) degree of freedom 5
2) number of data line n = 10
3) significance level 0.05

We form a hypothesis

1) h0 - There are no differences between groups
2) h1 - There are differences between groups

The computed Friedman statistic value is 19.85 while the expected value was 11.070. As such, the null hypothesis has been rejected.

D. Applying holm-Bonferroni method

As the Friedman test has indicated that there are differences between groups, a post hoc test has to be conducted that shows pairwise differences. As per Janez [8] since we aim to check whether our proposed method is better than others, it's appropriate to use the holm-Bonferroni method. A separate jupyter notebook by the name posthoctest.ipynb contains the code for the tests. Here, we used the holm-Bonferroni method programmed by Benjamin Patrick Evans. [1] Using rank information from the Friedman test, the holm-Bonferroni test computes an adjusted p-value for pair groups against our control group. If the adjusted p-value is more than the significance 0.05, then the group are not significantly different. Applying the holm-Bonferroni method gives us the following table.

TABLE IV
HOLM-BONFERRONI METHOD APPLIED AGAINST PROPOSED METHOD

Comparison	adjusted p value	less than 0.05
proposed vs svc	6.166903e-08	True
proposed vs xgb	3.018658e-04	True
proposed vs gnb	2.925433e-03	True
proposed vs rfc	1.572992e-01	False

From table 4, we can see that in proposed vs RFC, there is no significant difference between the groups while it exists for SVC, XGB and GNB.

IV. Conclusion

Our proposed method for the task of sentiment analysis via binary classification yielded better accuracy than the ones it was up against. The significant difference between it and Random Forest regression was relatively low. The conclusions when based upon previous research and current data show the results are not so profound. There were factors that impeded work when certain algorithms were dependent on hardware given the size of the input and there was a previous guess that support vector machine might not work as well in these groups, as it showed. Perhaps using more kernels will show other results however for our task, a linear kernel is recommended. What is surprising, however, is how well Naive Bayes performed despite the volume of data. As for our proposed model, certain tuning and data pre-processing might affect it for the better as it still has the best accuracy given the volume of data.

[1]https://github.com/benjaminpatrickevans/MethodComparisonsInPython

V. Contribution

Equal contribution was made to this project. The project was divided into the following parts

1) Problem identification
2) Algorithm selection based on paper research
3) Creating the model
4) Evaluating the model
5) Reporting the results

All tasks were performed in a cooperative and timely manner to ensure completion.

References

[1] H. Chen, L. Chen, Q. Jiang, and S. Guo, "Prior dirichlet distribution based feature selection in naive bayes on high dimensional data classification," pp. 415–420, Dec 2020.

[2] X. Jiang and U. Braga-Neto, "A naive-bayes approach to bolstered error estimation in high-dimensional spaces," pp. 1398–1401, Dec 2014.

[3] I. Dhillon and Y. Guan, "Information theoretic clustering of sparse cooccurrence data," pp. 517–520, Nov 2003.

[4] H. Liu and S. Xiong, "Fuzzy support vector machines based on density clustering," pp. 784–787, May 2007.

[5] D. S. Hochbaum and P. Baumann, "Sparse computation for large-scale data mining," *IEEE Transactions on Big Data*, vol. 2, no. 2, pp. 151–174, June 2016.

[6] M. Z. Islam, J. Liu, J. Li, L. Liu, and W. Kang, "A semantics aware random forest for text classification," p. 1061–1070, 2019. [Online]. Available: https://doi.org/10.1145/3357384.3357891

[7] D. C. N. J. I. M. Nitin Hardeniya, Jacob Perkins, *Natural Language Processing: Python and NLTK*, ser. O'Reilly. Packt Publishing, 2016.

[8] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, p. 1–30, dec 2006.