COMP 551 - Project 2 Report

Omar IBRAHIM 260726420
Justin SUN 260707684
Mohamed BOUAOUINA 260765511

***Abstract*—The purpose of this report is to present the work done on two datasets of textual data for classification tasks. Six different learning models are presented, trained, and evaluated. For each model, a standardized approach to tuning their respective parameters is proposed. In addition, a generalized preprocessing technique for textual data is developed and tested. Key findings indicate that the SVM algorithm is better suited for classification tasks. This report also demonstrates the effectiveness of crossvalidation in preventing overfitting issues.**

***Index Terms*—NLP, classification, text preprocessing, sklearn, SVM, logistic regression, decision trees, random forests, AdaBoost, KNN**

I. Introduction

DOCUMENT classification has been widely studied in the field of Natural Language Processing. In fact, multiple studies compared feature selection techniques or feature space transformation whereas others focused on the performance of different algorithms. Most studies conclude that SVM outperforms other classification algorithms but the problem with SVM is that it does not scale as well compared to Decision Tree or KNN [1]. Additionally, ensemble algorithms like Random Forest and Adaboost seem to be consistently outperforming neural network classifiers in accuracy by 7% [2] and other traditional classification algorithms by 3-4% as seen in the twitter sentiment analysis study [3]. However, these algorithms are faster in training and scale better. In the following sections, We compare a few of the algorithms mentioned above to replicate and confirm the literature on which algorithm has the best speed to accuracy performance ratio.

II. Data preprocessing

In this section, we will explore our procedure for cleaning the datasets. As both datasets are textual data, we are operating with the assumption that the preprocessing methods developed are valid for both of the problems.

Our first approach to extracting features from textual data was to use the TfidfVectorizer from the sklearn.feature_extraction library followed by the Normalizer from sklearn.preprocessing. The vectorizer uses the TF/IDF statistics technique which assigns a weight to a term by considering both its term frequency (i.e. how many times it appears in the corpus) as well as its inverse document frequency that captures how informative the term is.

After vectorizing our data using the technique above, we noticed that a few features were not informative and could be discarded. Indeed, stopwords like ”almost”, ”and”, ”very”, and so on do not particularly help in classifying the data at hand as they are common in every document. We therefore started customizing our vectorizer to ignore stopwords, punctuation, accents, and special characters.

Next, we implemented stemming to break down the terms in the data to their root form. We also explored the use of n-grams: this method enables the model to look at terms in groups of n consecutive occurrences. Therefore, it enables the algorithm to mine more information by considering the context of use of the terms. Theses processing techniques as well as stop words removal seem to be dominant in literature as well [5]. Finally, to prevent our models from overfitting and to speed up their convergence, we limited the number of used features to 20000 picked from the ones with document frequency inferior to 0.95.

To find out more about our preprocessing and cleaning procedure, please refer to our the Cleaner.py file in the src folder of our code submission. You will also find in our code submission the script (MergeIMDB.py) run to merge all the different review files from the IMDB dataset into the train_reviews.csv and test_reviews.csv files used in our classification scripts.

III. Tuning models

In this section, we will be presenting the models implemented for the analysis of the two datasets. Background on each model will be given as well as the approach taken to tune its hyperpamaters. For each model, accuracy was evaluated using 5-fold crossvalidation as implemented by the GridSearchCV class in the sklearn.class_selection package. For details on the code implementation, please refer to HyperTuningExample.py script in our code submission.

A. Logistic Regression

The Logistic Regression model is a classification algorithm similar to the linear regression model, but uses the sigmoid function to map predicted values to probabilities, then draws a decision boundary based on these probabilities.

The hyperparameters used and tested for this model consist of: *penalty, dual, tol, C, solver and max_iter*. *Penalty* is the type of regularization that is to be used consisting of L1 and L2. *Dual* is a boolean indicating whether to solve the primal or dual formation of the problem. *Tol* is the tolerance for the stopping criteria for gradient descent. *C* is the inverse of regularization strength, it retains the strength modification of regularization. *Solver* is the type of solver used to minimize the cost function. *Max_iter* is the maximum number of iterations for the solver being used to converge.

When running cross validation with ranges of values for hyperparameters, between L1 and L2, I found L1 gave higher accuracy for the IMDB dataset and L2 gave a higher accuracy for the Newsgroups dataset. Setting *dual* to false and solving the primal optimization problem yielded higher accuracies on both datasets. A higher value for *tol* yielded higher results for both datasets; values less than or equal to the default value gave worse results and anything higher than 1e-1 gave more or less the same results. *C* was tested with a range of powers of 10 from 10^{-2} to 10^{2}. The default value, 1.0, was shown to yield higher accuracies on both datasets. *Solver* was tested using a range with all the solvers to see which would perform the best; the 'liblinear' solver was best for the IMDB dataset and the 'saga' solver was best for the Newsgroups dataset. *Max_iter* was tested with values 100, 1000, and 5000. Values set to 100 worked fine for Newsgroups dataset, however gave convergence warnings for the IMDB dataset. The best final parameters chosen from cross validation are *C=1.0, dual=False, max_iter=1000, penalty='l1', solver='liblinear', tol=0.1* for the IMDB dataset and *C=1.0, dual=False, max_iter=100, penalty='l2', solver='saga', tol=0.01* for the Newsgroups dataset.

B. Support Vector Machine

The Linear SVM model is a machine learning model widely used in classification tasks, but can also be used for regression tasks. The objective is to find a hyperplane in N-dimensional space, where N is the number of features, and that has maximum distance between points in both classes. This hyperplane is our decision boundary and the data points closest to the hyperplane are our support vectors used to influence to position and orientation of the hyperplane.

The hyperparameters used and tested for this model consist of: *penalty, dual, loss, tol, C, and max_iter*. *Penalty* is the type of regularization that is to be used consisting of L1 and L2. *Dual* is a boolean indicating whether to solve the primal or dual formation of the problem. *Tol* is the tolerance for the stopping criteria for gradient descent. *C* is the inverse of regularization strength, it retains the strength modification of regularization. *Loss* is the type of loss function to be used consisting of 'hinge' and 'squared_hinge'. *Max_iter* is the maximum number of iterations to be run.

When running cross validation with ranges of values for hyperparameters, between L1 and L2, I found L2 gave higher accuracy for both datasets. Setting *dual* to false and solving the primal optimization problem yielded higher accuracy for the IMDB dataset, while setting dual to true gave better results for the Newsgroups dataset. A higher value for *tol* yielded higher results for both datasets; values less than or equal to the default value gave worse results and anything higher than 1e-1 gave more or less the same results. *C* was tested with a range of powers of 10 from 10^{-2} to 10^{2}. The default value, 1.0, was shown to yield higher accuracy on the Newsgroups dataset, while a value of 1e-1 was better for the IMDB dataset. *Loss* was tested using both 'hinge' and 'squared_hinge' to see which would perform the best; 'squared_hinge' was shown to yield higher accuracy for both datasets. *Max_iter* was tested with values 1000, 5000, 10000, and 20000. Values set to 1000 worked fine for IMDB dataset, however gave convergence warnings for the Newsgroups dataset. The best final parameters chosen from cross validation are *C=0.1, dual=False, loss='squared_hinge', max_iter=1000, penalty='l2', tol=0.1* for the IMDB dataset and *C=1.0, dual=True, loss='squared_hinge', max_iter=5000, penalty='l2', tol=0.01* for the Newsgroups dataset.

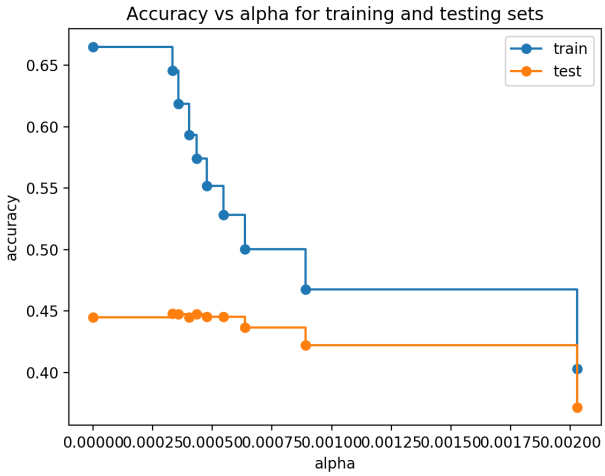
C. Decision Tree

Decision tree is a hierarchical algorithm that builds a tree structure based on an optimization for the best feature splits. We optimized the parameters by doing hyperparameter tuning for all parameters that had a significant effect on the cross-validation accuracy. For both datasets, we started with the default parameters and then tried a set of values for each parameters that are either close or far to the default. The best parameters we found that worked for both datasets were *criterion=gini, random_state=30, min_impurity_increase=0.0001*.

To determine the *ccp_alpha* value for the IMDB dataset, we use sklearn's cost_complexity_pruning_path to determine different pruning paths and their impurity. Figure 1 shows the accuracy on a train-test-split for different alpha values. Both train and test accuracy decrease as alpha increases so we choose *ccp_alpha=0.0025*. We follow the same procedure for 20newsgroups dataset and determine *ccp_alpha=0.00055*.

This model did not require the complete set of features to achieve decent accuracy. For 20newsgroups, we tested values for *max_features* and found that only 40% were needed which also helped increase the speed of training. This could be because of the infrequency of the extracted features for 20newsgroups. As for IMDB, after doing grid search cross-validation, we found that we needed at least 80% of the features.

Fig. 1: Accuracy score versus the alpha pruning value



	IMDB				20newsgroup			
Models	default params	crossvalidation	train	test	default params	crossvalidation	train	test
Logistic Regression	0.933	0.866	0.9	0.884	0.898	0.728	0.898	0.676
SVM	0.989	0.865	0.936	0.888	0.97	0.751	0.97	0.679
Decision Tree	1.00	0.76	0.777	0.765	0.97	0.468	0.489	0.424
Random Forest	1.00	0.776	0.808	0.783	0.973	0.58	0.92	0.531
AdaBoost	0.842	0.853	0.884	0.865	0.492	0.473	0.521	0.449
KNN	0.855	0.781	0.806	0.794	0.379	0.629	0.642	0.599

TABLE I: Results yielded by the different models on the IMDB and 20newsgroup datasets

D. Random Forests

Random Forest is an ensemble learning algorithm that builds on top of decision trees by running multiple estimators.

To tune the parameters we built on top of the parameters we obtained from the grid-search cross-validation for the decision tree, and then we did further tuning for the extra parameters. The parameters *oob_score* and *warm_start* did not affect either datasets. The number of estimators had a very small impact on the cross-validation accuracy for both datasets as well. Accordingly, we picked *n_estimators* to be 50 & 90 for IMDB and 20newsgroups respectively.

E. AdaBoost

Adaptive Boosting, or AdaBoost for short, is an ensemble learning algorithm that builds a strong learner by fitting first a simple learner on the training set (e.g. a decision tree of unit depth) and then keeps adding more learners while increasing the weight of the datapoints that were misclassified by the previous learner. The AdaBoost algorithm is particularly interesting because it specifically selects the features that improve the prediction of the model and is known to be very slow to overfit (cf. COMP 551 Boosting Lecture).

In our subsequent work, we use the AdaBoostClassifier object from the sklearn.ensemble package to test the AdaBoost model on our two datasets. A first run was performed on the train set with the default hyperparameters to get an estimate of the training accuracy. In the following, we present the different hyperparameters and how we tuned them using crossvalidation accuracy scores.

learning_rate: This parameter determines the contribution of each weak learner to the ensemble. We started by trying a range of different values while keeping the other hyperparameters constant. We then select the two learning rates that gave the best accuracies and narrow down the search by trying different increments between these two.

n_estimators: This parameter sets the maximum number of weak learners to stack on top of each other. We applied the same tuning approach as previously while using the best *learning_rate* determined in the previous section. We kept increasing the number of estimators until the accuracy started to drop.

random_state: The tuning of the past two parameters was done using a value of 0 for the *random_state* parameter. We noticed that changing this parameter did not affect the accuracy score nor the training time.

base_estimator: On the news set, we noticed that the model was fitting poorly to the train set which lead us to believe that it was not expressive enough to capture the complexity of the dataset. We tried to complexify the model by trying deeper trees as the *base_estimator* but that did not result in better accuracies.

The final results obtained with the tuned hyperparameters are displayed in Table I in the Results section.

F. KNN

Finally, the last algorithm we considered is the k-Nearest Neighbours classifier. In order to determine the label of a datapoint, this model simply determines the closest points to it from the training set and uses the mode of their labels to get the predicted value. We chose to test this model because we saw in our first COMP551 lecture that it is used for online news recommendations and we wanted to see how it fares against our datasets.

In our subsequent work, we use the KNeighborsClassifier from the sklearn.neighbors library. This class has three hyperparameters that we attempted to tune: *n_neighbors* i.e. the number k of nearest neighbours to consider and *p* which determines the function to use to calculate the distance between the points. For each dataset, we started by increasing the number k until the crossvalidation score started to drop. We then evaluated two different distance calculation methods (Manhattan and Euclidean) and determined that the L2 norm (Euclidean) yielded better results for our experiments.

IV. Results

This section compares and discusses the results presented in Table I. In the table, the performance of each model is reported on the different datasets. We first show the accuracy the train error using sklearn's default paramaters. We then display the crossvalidation, train, and test accuracies obtained using the tuned hyperparameters from the previous section.

In order to reproduce our results, please refer to the src/main.py script in our submitted code and select the model and dataset of interest. In order to recreate the conditions of our experiments, kindly run our script using a conda environment created from the environment.yml file.

A. General observations

First of all, we would like to report that our models have an average accuracy of 0.83 and 0.56 on the IMDB and

20newsgroup datasets respectively. We are therefore off from the TA baseline by 6% and 14% respectively.

Furthermore, this experiment exhibits two fundamental trends in machine learning. First, we can observe the phenomenon of overfitting in many models as we got high train accuracies with default paramaters but these were not selected after hyperparameter tuning because they caused the model to overfit and did not yield good crossvalidation scores. Second, we can see in our results table that the crossvalidation accuracy is a very good estimate of the test error as it is within a couple of per cents in most models.

B. Comparison of test accuracies across models and datasets

Generally, all models performed significantly better on the IMDB dataset, as opposed to the 20newsgroups dataset. In addition, within each dataset, the Logistic Regression and SVM models generally out perform all others with respective accuracy scores of 0.884 and 0.889 on IMDB, and 0.676 and 0.679 on 20newsgroups; with the exception of AdaBoost on the IMDB dataset with a score of 0.865, which is comparable to that of Logistic Regression and SVM. Furthermore, the worst performing across both datasets were Decision Tree and Random Forest, with respective accuracy scores of 0.765 and 0.783 on IMDB, and 0.424 and 0.532 on 20newsgroups.

C. Timing considerations

We found that cross validation would take exceptionally longer on the 20newgroups dataset, as opposed to the IMDB dataset for all models. Where on the IMDB dataset cross validating took on average 5 to 10 minutes, the 20newgroups dataset would take on average 30 to 45 minutes. Individually training the models followed the same trend of taking longer to train on the 20newsgroups dataset, as opposed to the IMDB dataset. We find that this aligns with our previous assumptions, since with the IMDB dataset we are doing binary classification, whereas with the 20newgroups dataset we are doing multiclass classification which would take longer.

V. Conclusion

This project was a valuable experience to familiarize ourselves with the basics of Natural Language Processing. We were introduced to key concepts in the preprocessing of textual data such as stemming, stopwords removal, tf-idf vectorization, and n-gramming which enabled us to improve our models accuracies on our models.

In addition, this project allowed us to confirm some trends observed in literature that suggest SVM's superiority in classification tasks over other algorithms. In future experiments, we should explore more the scalability of these algorithms as the training data size varies as well as the timing taken to train. We also propose trying to combine these models together at the training and prediction time in an effort to improve accuracy.

Finally, this experiment also enabled us to revisit and demonstrate first-hand the fundamental concepts in the field of machine learning such as overfitting, estimation of test error using crossvalidation, hyperparameter tuning.

VI. Statement of Contributions

Omar:

- Data preprocessing
- Decision Tree hyperparameter tuning
- Random Forest hyperparameter tuning
- Report

Justin:

- SVM hyperparameter tuning
- Logistic Regression hyperparameter tuning
- Report

Mohamed:

- Data preprocessing
- AdaBoost hyperparameter tuning
- KNN hyperparameter tuning
- Report

VII. Review of Extra Features

In this section, we list the extra paths that we have explored beyond the project's basic requirements.

- Extended data cleaning and preprocessing beyond the steps shown in the Sklearn tutorial which included Customizing the tf-idf vectorizer with n-gramming, stemming, stopwords removal, etc.
- Explored an additional model that promised to work well on news datasets (KNN)
- Standardized approach to tuning hyperparameters and estimating test error using sklearn's GridSearchCV (cf. HyperTuningExample.py)

References

[1] F. Colas and P. Brazdil, "Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks", 2020. [Online]. Available: https://link.springer.com/chapter/10.1007/978-0-387-34747-9_18. [Accessed: 12- Mar- 2020]

[2] J. Chan and D. Paelinckx, "Evaluation of Random Forest and Adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery", 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0034425708000679. [Accessed: 12- Mar- 2020]

[3] "Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques - IEEE Conference Publication", Ieeexplore.ieee.org, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7050801/metrics#metrics. [Accessed: 12- Mar- 2020]

[4] A. Maas, "Learning Word Vectors for Sentiment Analysis", Ai.stanford.edu, 2020. [Online]. Available: http://ai.stanford.edu/ãmaas/papers/wvSent_acl2011.pdf. [Accessed: 12- Mar- 2020]

[5] V. Srividhya, "Evaluating Preprocessing Techniques in Text Categorization", Sinhgad.edu, 2020. [Online]. Available: http://sinhgad.edu/ijcsa-2012/pdfpapers/1_11.pdf?fbclid=IwAR01NbXcoWHfOY8ZjJoLDTQFFTSHpgBwx_awE3dX9Os9eNSiNOlNJvmGDsI. [Accessed: 12- Mar- 2020]