Final Project Written Report

CS-UY 4563

May 2, 2022

Prof. Linda M.Sellie

Alex Yan (11:00am Section) & Chuanyang Jin (2:00pm Section)

Introduction

The data that we used was from Kaggle. This data set contains 10876 real tweets with keywords, locations as metadata, and the tweet itself with labels. Each tweet is either related to a real-world disaster or describing something irrelevant. The data is already separated into a training set and test set, with respectively 7613 and 3263 samples, but the test set is unlabeled and only for submission on Kaggle, and thus we will further split the training set into training and testing set into 6851 and 762.

Our goal was to use logistic regression, support vector machines (SVM), and neural networks to predict whether a tweet is related to a real disaster or not. The performance of the model is measured by its F1 score, since in real-world applications, disaster tweets vs. all other tweets generally form an imbalanced dataset. We tried several hyperparameters on each model and evaluated their performance to find the best model and hyperparameter combination.

Example of a tweet in the dataset:

Keyword: “ablaze”

Location: “GREENSBORO, NORTH CAROLINA”

Tweet text: “How the West was burned: Thousands of wildfires ablaze in California alone http://t.co/vl5TBR3wbr”

Label: 1

Preprocessing

The raw data was provided in 2 .csv files, with every sample having 5 entries marking its number in the dataset, keyword, location, tweet text, and label. We first extracted the tweet text and decided only to use the text field as our model feature. This is due to the fact that the keyword is an arbitrarily decided metadata and location is considered more likely to be noise and thus cause overfitting.

After a round of manual feature selection, a regular expression is used to filter out non-informative words and letters in every tweet text. Since Twitter uses Hypertext Markup Language, raw tweet text extracted from Twitter contains special characters in HTML.

Figure 1 shows an example tweet, and the way it would be presented in the dataset is "This is a test message &amp will tag @user2 https://t.co/image_url." However, "&amp" and "@user2" here have no actual meaning. The image is transformed into a hyperlink, which is also meaningless. Therefore, the processed text fields removed all ampersands and mentionings, and replaced all hyperlinks with the string "URL."

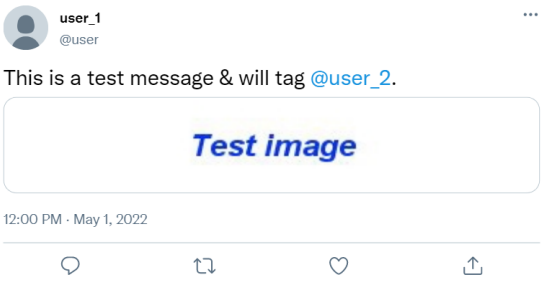

Figure 1: Example tweet

Then, three different methods were adopted to transform text into feature vectors. Sklearn's feature extraction library is used. The first method made use of sklearn CountVectorizer, which transforms the texts into a feature vector with the number of times each word appears in the collection of every tweet text (token count). The second method used TfidfVectorizer, which adds Term Frequency–Inverse Document Frequency (TF-IDF) weighing factor to generate a feature vector with the same shape as method 1. For these two methods, typical English stopwords were removed with sklearn's built-in stopword dictionary. Word embedding is applied to the third method, where we used Bidirectional Encoder Representations from Transformers (BERT) model to conduct a pre-training of the raw text data and thus generate a feature vector.

After applying three methods of feature transformation, the provided training set was split into a training set and a testing set with respectively 6851 and 762 samples. For CountVectorizer and TfidfVectorizer, the feature vector has 14443 features. BERT transformation yielded 384 features.

Logistic Regression

After preprocessing, logistic regression is conducted on all three types of transformed feature vectors. The logistic regression was done using sklearn's LogisticRegression function. In total 63 models were created, including 3 models with no regularization, and 42 models with either L1, L2 regularization applied to 3 types of feature vectors. Each regularization was applied with C values [0.01, 0.1, 1, 10, 100, 1000, 10000], and thus 14 models for each type of feature vector. The results of different hyperparameters used are shown below.

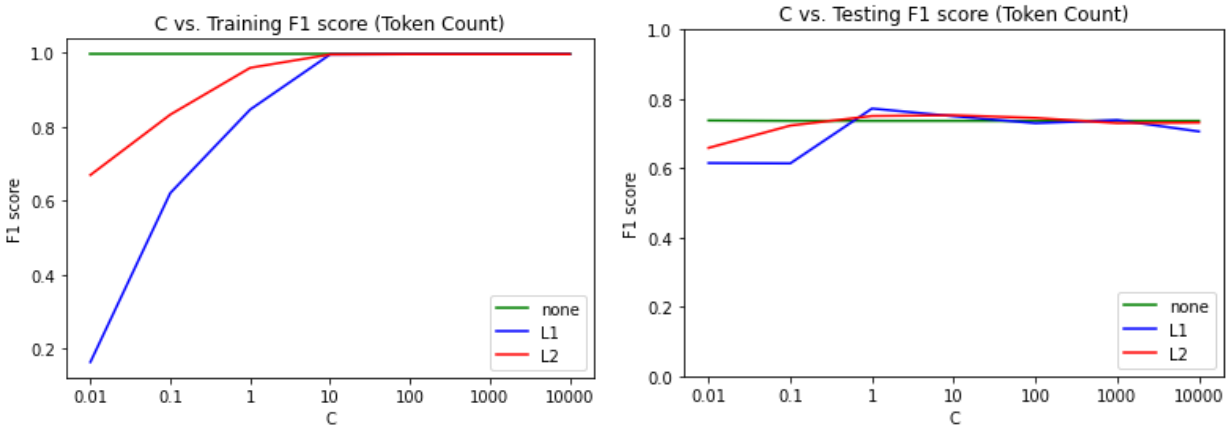

Figure 2: F1 Score for logistic regression with token count transformed features

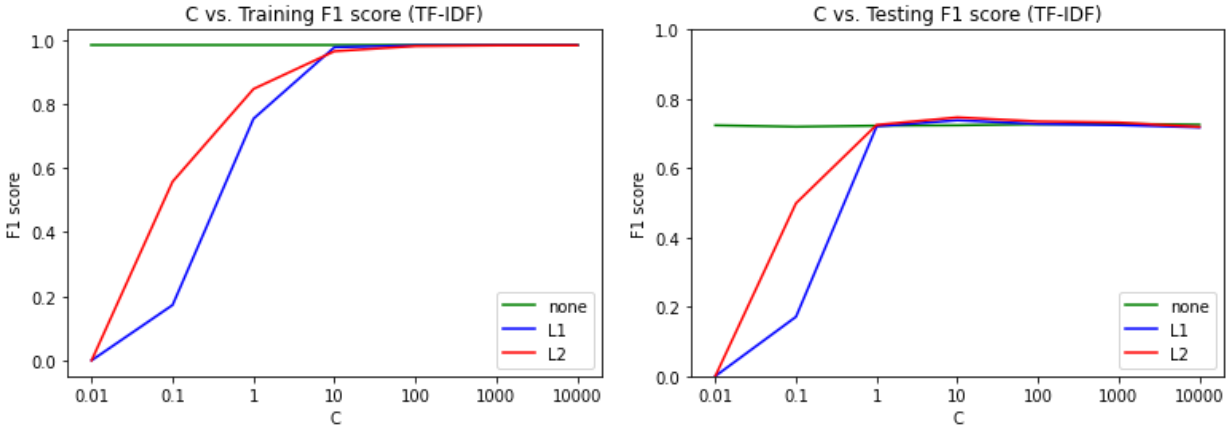

Figure 3: C vs. F1 Score for logistic regression with TF-IDF transformed features

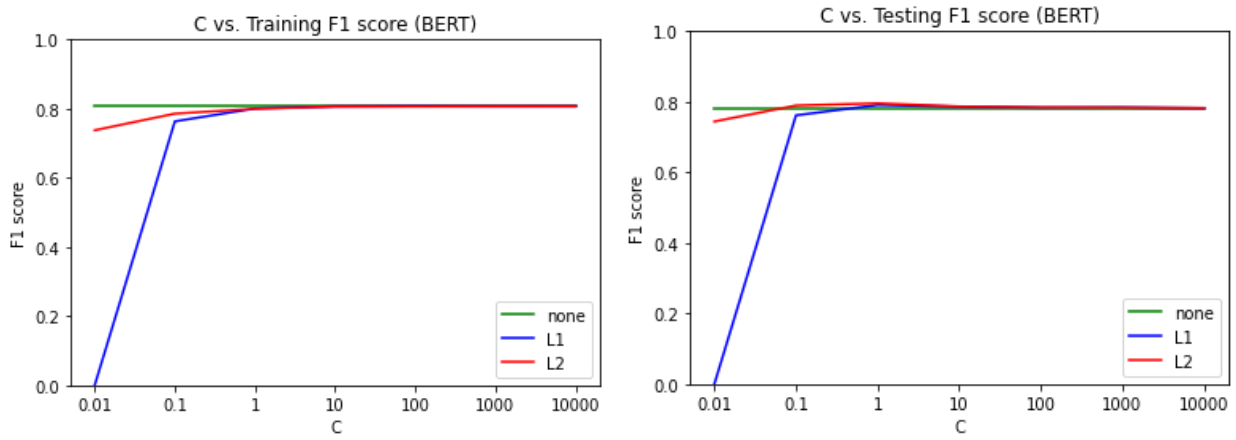

Figure 4: C vs. F1 Score for logistic regression with BERT transformed features

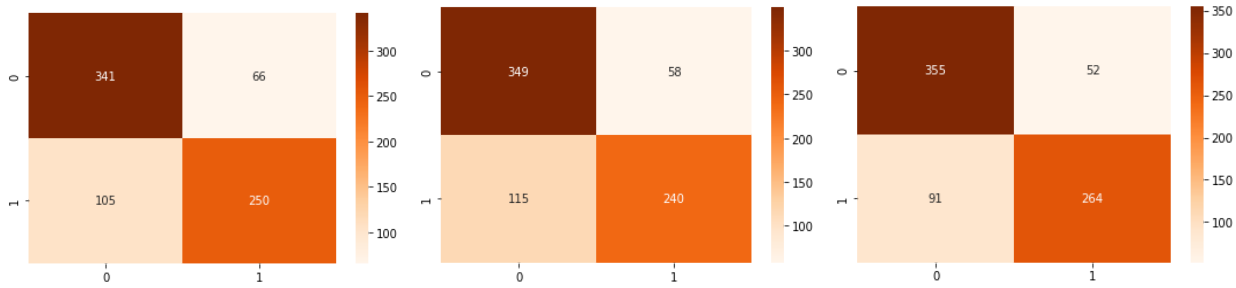

Figure 5: Logistic regression confusion matrix for best models' prediction of testing set (left to right: token count, TF-IDF, BERT)

There are certain models that fail to converge at some given Cs, but are included to show the boundaries.

Then we explored somewhere around the best parameters we obtain from the loops above—C = 1, penalty = "l2", feature transformation being BERT. Finally we got our best result using logistic regression with C = 0.8, penalty = "l2"—an F1 Score of 0.796992. Its confusion matrix is shown below.

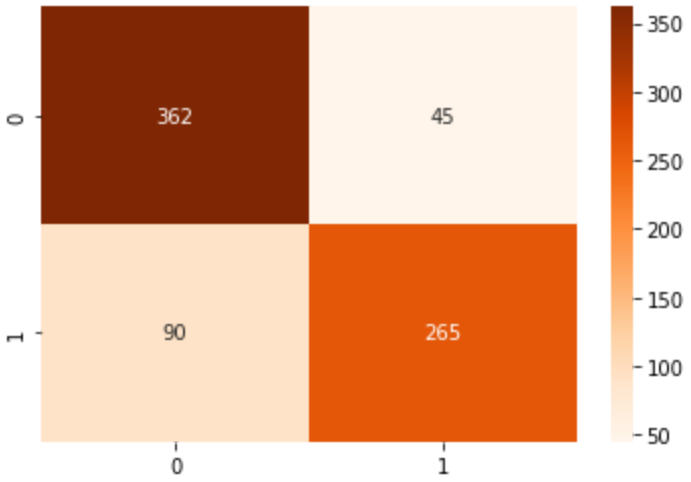

Figure 6: Logistic regression confusion matrix for best models' prediction of testing set

Support Vector Machine (SVM)

SVM is also used to fit the training data. After preprocessing, sklearn's svm library function SVC is applied on all three types of transformed feature vectors to model to form in total 54 models. Three different kernels (radial basis function, linear, polynomial of degree 3) were used. The C values were chosen to be [0.01, 0.1, 1, 10, 100, 1000] to remove too many samples that do not converge (and thus not providing a valid result) caused by too large lambda on L2 regularization (but not all). The results are shown below.

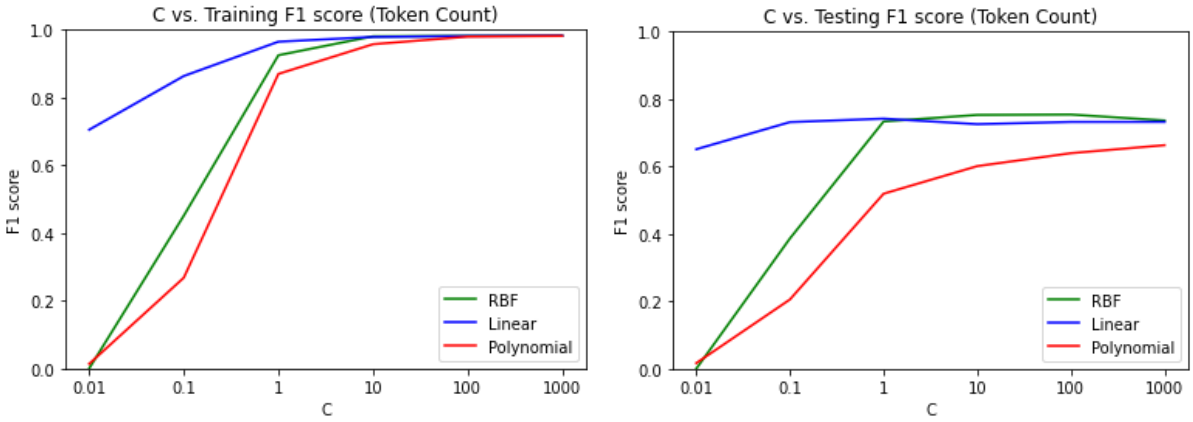

Figure 7: F1 Score for SVM with token count transformed features

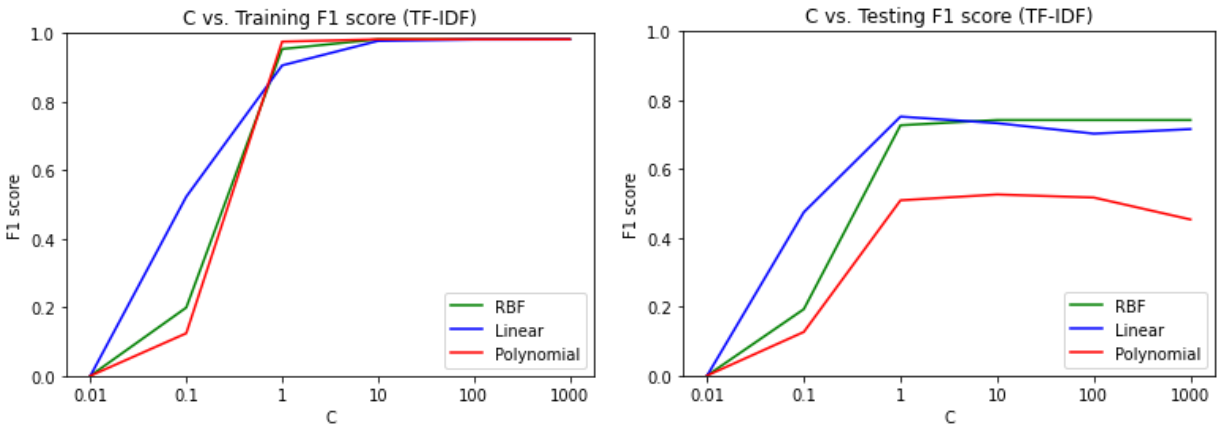

Figure 8: F1 Score for SVM with TF-IDF transformed features

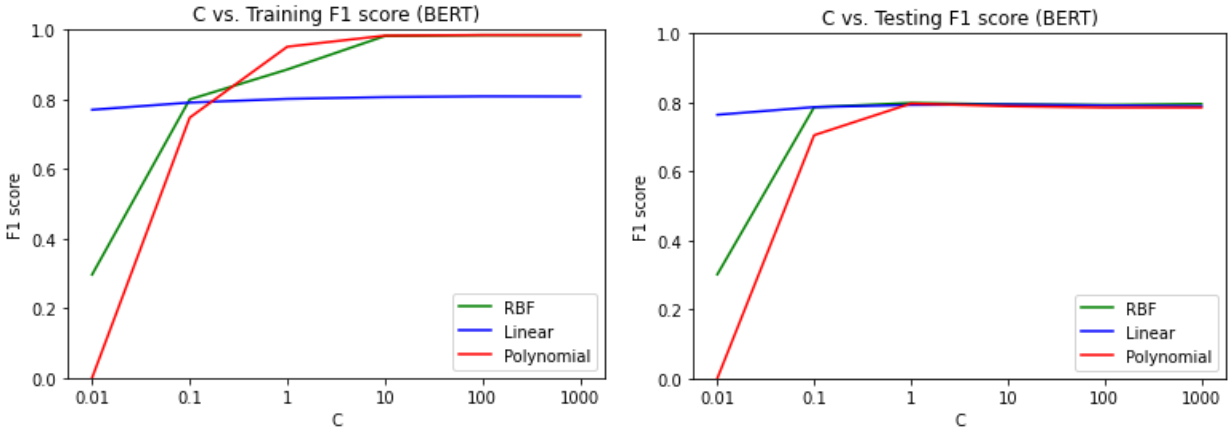

Figure 9: F1 Score for SVM with BERT transformed features

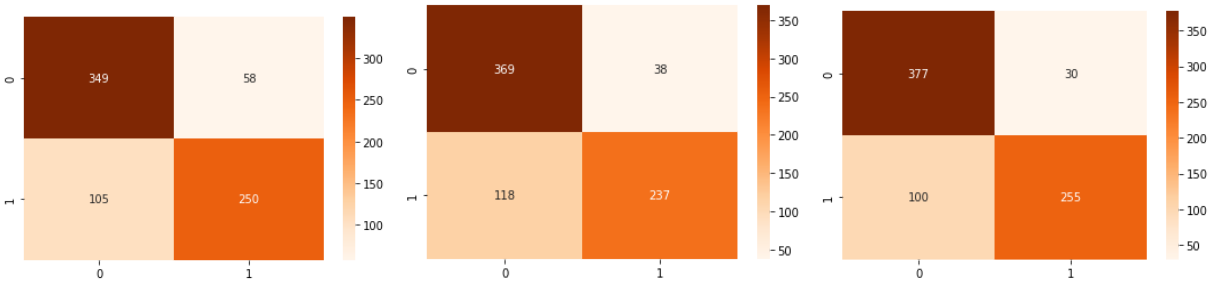

Figure 10: SVM confusion matrix for best models' prediction of testing set (left to right: token count, TF-IDF, BERT)

Then we explored somewhere around the best parameters we obtain from the loops above—C = 1, kernel = “rbf”. Finally we got our best result using SVM with C = 1.7, kernel = “rbf”—an F1 Score of 0.811060. Its confusion matrix is shown below.

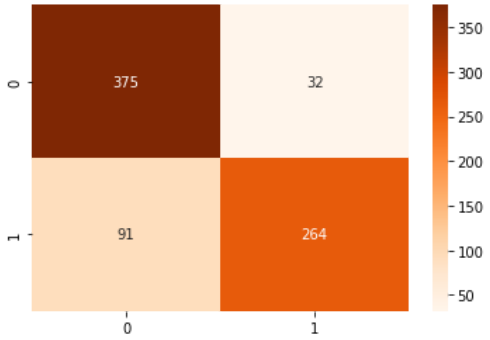

Figure 11: SVM confusion matrix for best models’ prediction of testing set

Neural Networks

Neural networks models were employed to fit BERT transformed data only, due to the extensive amount of time it would take to run on 14443 features generated by token count method and TF-IDF method. Different structures were tried, and the best hidden layer structure (256, 48) was applied to the BERT models with 384 features. Different activation functions were also used, including logistic, ReLU, and tanh function. The choice of lambda was limited to [0.0001,0.001,0.01,0.1, 1, 10] to avoid non-convergence issues. These variations resulted in 18 different models. The results are shown below.

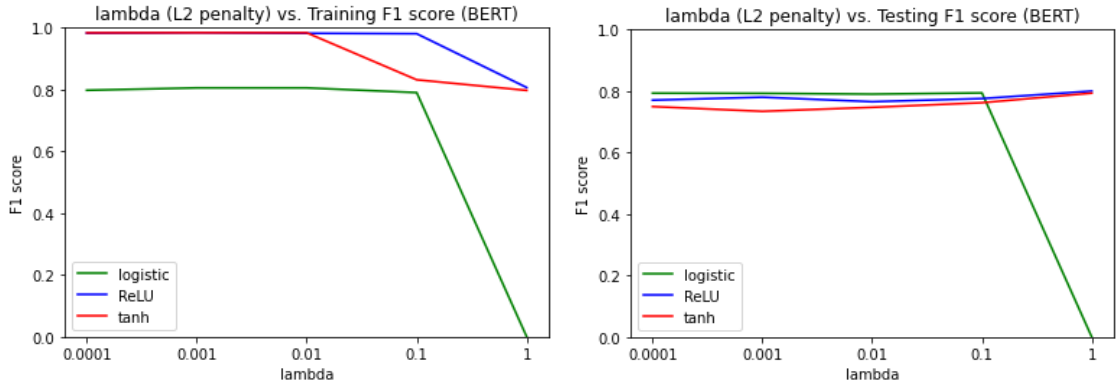

Figure 12: F1 Score for neural network with BERT transformed features

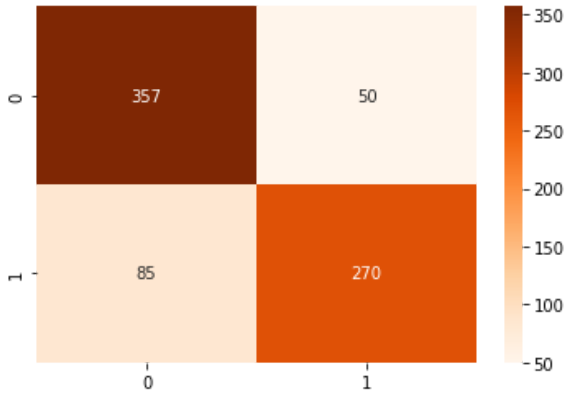

Figure 13: Neural Network confusion matrix for best prediction of testing set

Table of Results

Type \ C	0.01	0.1	1	10	100	1000	10000
TC Train	0.996223	0.996223	0.996223	0.996223	0.996223	0.996223	0.996223
TC Test	0.736686	0.736686	0.736686	0.736686	0.736686	0.736686	0.736686
TC, L1 Train	0.162929	0.620519	0.846494	0.995535	0.996223	0.996222	0.996222
TC, L1 Test	0.614980	0.613970	0.772307	0.749999	0.730015	0.739002	0.706051
TC, L2 Train	0.669252	0.832793	0.959435	0.995535	0.996223	0.996223	0.996223
TC, L2 Test	0.658662	0.723127	0.750769	0.753012	0.745156	0.730205	0.731778
TF-IDF Train	0.982262	0.982262	0.982262	0.982262	0.982262	0.982262	0.982262
TF-IDF Test	0.723926	0.723926	0.723926	0.723926	0.723926	0.723926	0.723926
TF-IDF, L1 Train	0.000000	0.172529	0.753657	0.976478	0.982081	0.982256	0.982238
TF-IDF, L1 Test	0.000000	0.171284	0.720930	0.738317	0.727838	0.724458	0.717868
TF-IDF, L2 Train	0.000000	0.557369	0.846387	0.963930	0.980000	0.982075	0.982262
TF-IDF,L2 Test	0.658662	0.498960	0.725165	0.746875	0.732006	0.730205	0.719135
BERT Train	0.806911	0.806911	0.806911	0.806911	0.806911	0.806911	0.806911
BERT Test	0.782222	0.782222	0.782222	0.782222	0.782222	0.782222	0.782222
BERT, L1 Train	0.000000	0.763314	0.800569	0.806417	0.807473	0.806836	0.806911
BERT, L1 Test	0.000000	0.761760	0.790977	0.786248	0.783382	0.784023	0.782222
BERT, L2 Train	0.737639	0.785701	0.799785	0.806204	0.806267	0.806761	0.806767
BERT, L2 Test	0.744186	0.789634	0.795795	0.786885	0.784546	0.784023	0.782222

Table 1: Logistic regression result table (highlighted best F1 score for each transformation)

Type \ C	0.01	0.1	1	10	100	1000
TC, RBF Train	0.000000	0.451151	0.924972	0.980765	0.982006	0.982456
TC, RBF Test	0.000000	0.386516	0.733552	0.753048	0.754147	0.736535
TC, Linear Train	0.704984	0.863576	0.964973	0.979047	0.981692	0.982299
TC, Linear Test	0.651162	0.731707	0.741984	0.725637	0.732272	0.732394
TC, Poly Train	0.013623	0.268010	0.869799	0.957511	0.979840	0.981975
TC, Poly Test	0.016759	0.205513	0.519269	0.601134	0.639575	0.663265
TF-IDF, RBF Train	0.000000	0.199200	0.953976	0.982244	0.982244	0.982244
TF-IDF, RBF Test	0.000000	0.192893	0.727272	0.742400	0.742400	0.742400
TF-IDF, Linear Train	0.000000	0.522892	0.906103	0.977578	0.981554	0.981905
TF-IDF, Linear Test	0.000000	0.474576	0.752380	0.733333	0.702865	0.716012
TF-IDF, Poly Train	0.000000	0.124155	0.975516	0.982207	0.982207	0.982195
TF-IDF, Poly Test	0.000000	0.126649	0.509164	0.526104	0.517171	0.453389
BERT, RBF Train	0.296533	0.799925	0.885979	0.982075	0.984832	0.985002
BERT, RBF Test	0.301435	0.786271	0.798771	0.794520	0.792738	0.795763
BERT, Linear Train	0.770234	0.790923	0.801532	0.806597	0.808818	0.808387
BERT, Linear Test	0.764331	0.786585	0.792738	0.794029	0.791666	0.789317
BERT, Poly Train	0.000000	0.747562	0.951429	0.983272	0.985002	0.985002
BERT, Poly Test	0.000000	0.705061	0.796875	0.788990	0.785276	0.785276

Table 2: SVM result table (highlighted best F1 score for each transformation)

Type \ C	0.0001	0.001	0.01	0.1	1
BERT, Logistic Train	0.797737	0.805413	0.805357	0.790192	0.000000
BERT, Logistic Test	0.793261	0.792738	0.790123	0.794074	0.000000
BERT, ReLU Train	0.982672	0.984525	0.982152	0.980762	0.806568
BERT, ReLU Test	0.770562	0.780120	0.765432	0.775631	0.800001
BERT, tanh Train	0.983595	0.984006	0.983804	0.831639	0.797312
BERT, tanh Test	0.749642	0.734104	0.747126	0.762195	0.793363

Table 3: Neural network result table (highlighted best F1 score)

Additional Methods

—Naive Bayes, Random Forest, and Gradient Boosting

To practice more and have fun, we adopted three additional common methods not included in our class–Naive Bayes, Random Forest, and Gradient Boosting–without carefully tuning the parameters.

Naive Bayes

Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

We tried the methods of Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes, Bernoulli Naive Bayes and Categorical Naive Bayes.

$$Formula\ for\ GaussianNB\colon\ P(x_i|y) = \frac{1}{\sqrt{2\Pi\sigma^2_y}} exp(-\frac{(x_i-\mu_y)^2}{2\sigma^2_y})$$

We obtained our best result using sklearn.naive_bayes.BernoulliNB, with the best F1 Score of 0.789174. Its confusion matrix is shown below.

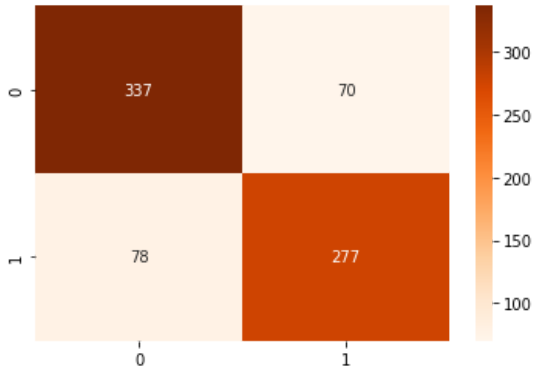

Figure 14: Naive Bayes confusion matrix for best models' prediction of testing set

Random Forest

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks like ours, the output of the random forest is the class selected by most trees.

To our understanding, the training of a random forest is to repeatedly randomly select samples from the dataset, and fit them with decision trees. Then to predict unseen samples, it takes the majority vote of the decision trees.

We employ the method of sklearn.ensemble.RandomForestClassifier(). The parameter N_estimators in the method indicates the number of trees in the forest. Its default value is 100 (increased from 10 in the previous version). We vary a few values (10, 50, 80, 90, 100, 110, 120, 150, 200, 1000, etc.) and find out that n_estimators around 100 is the optimal parameter for our prediction task.

This algorithm includes random processes and every time gives a different result even with the same parameters. We got the best F1 Score of 0.786834. The confusion matrix of a F1 score of 0.786271 is shown below.

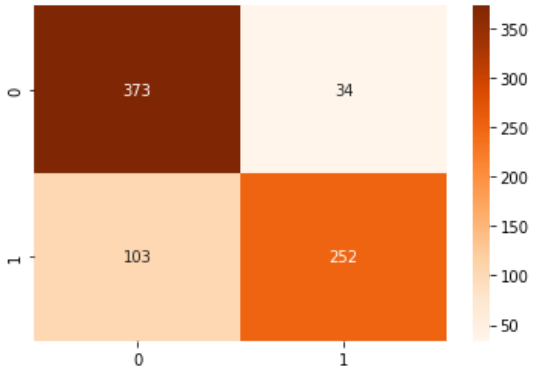

Figure 15: Random forest confusion matrix for best models' prediction of testing set

Gradient Boosting

Gradient boosting gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest.

We employed the method of sklearn.ensemble.GradientBoostingClassifier and got our best result F1 Score of 0.791411. Its confusion matrix is shown below.

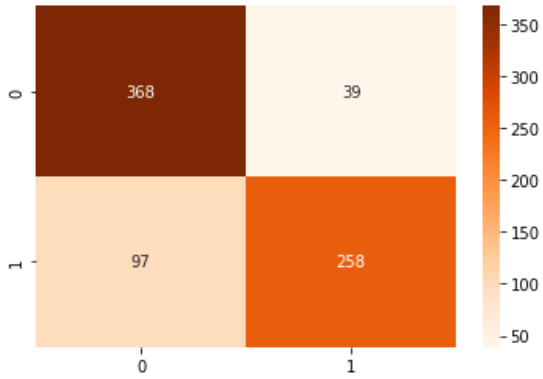

Figure 16: Random forest confusion matrix for best models' prediction of testing set

Ensemble Model

Model	LogReg	SVM	Neural Networks	Naive Bayes	Random Forest	Gradient Boosting	Ensemble Model
F1 Score	0.796992	0.811060	0.800001	0.789174	0.786834	0.791411	**0.812121**

We find that the results of our six models are similar, all in a range of [0.786834, 0.811060]. A single algorithm may not make the perfect prediction for a given dataset. We tried to build a simple ensemble model to take advantage of the different models.

We first chose the 3 models with the highest F1 scores out of our 6 models, namely SVM, Neural Networks, and Logistic Regression. For each tweet prediction, we let the three models perform a majority vote to decide the final prediction. This method gives a F1 score of 0.803571. This score is pretty high, but no higher than the SVM method alone.

We came up with another idea. We let the 4 models with the highest F1 scores–namely SVM, Neural Networks, Logistic Regression, and Gradient Boosting–be Model I, II, III, and IV, in decreasing order of their independent F1 score1. For each tweet prediction, we let Model I to decide the prediction, unless Model II, III, and IV give the same prediction which does not agree with Model I. This method gives a F1 score of 0.812121. This score is higher than any of the models alone.

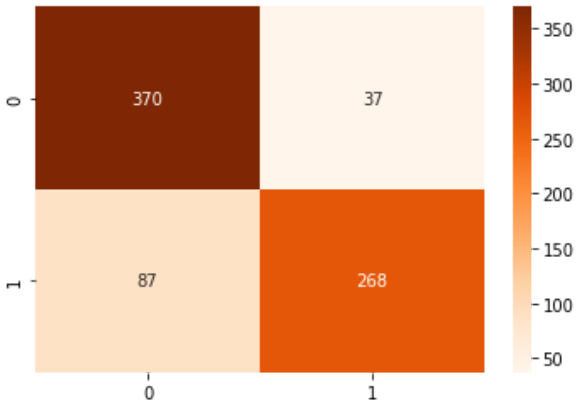

Figure 17: Confusion matrix for our final best models' prediction of testing set

Conclusion

Before drawing conclusions from the results produced by different models, it is worth noting that for this classification problem a larger dataset would be needed and we believe Kaggle's dataset is somewhat inadequate for the training to produce higher performance (F1 score). Detecting if a tweet describes a real disaster may not have 100% human accuracy, and thus the performance of the models and hyperparameters we adopted may be limited not by choice of model but rather the limit of the dataset itself.

For logistic regression, the result seems acceptable. The final hyperparameter chosen to have the highest F1 score does not produce too much overfitting, since there is no significant deviation between training F1 score and testing F1 score. Also, telling from the confusion matrix, this model seems to be classifying most points correctly. Generally there are more false negatives than false positives, but the values are within acceptable range. There are certain C values that are too extreme to produce a valid result, but these values set boundaries for satisfactory C values. By examining the output of 54 different hyperparameters, it is worth noticing that the token counting and TF-IDF method seems to easily overfit even after applying regularization, reaching around 99% of training F1 score while producing low testing F1 score. This could be due to the high variance in the feature vector transformed by token counting and TF-IDF, since BERT transformed features hold a steady training F1 score of around 80%, and thus prevents overfitting.

For SVM, the overfitting becomes more of a problem. The best model which produced the highest F1 score has a 10% difference between its training F1 score. However, it yields the highest performance among all the models we tried. Considering that the confusion matrix also shows that SVM makes the least mistakes, we believe that it is either due to the restriction of the given sample or the overfitting is very slight to affect the overall result. Some other observations are that polynomial kernels of degree 3 caused massive overfit to token count and TF-IDF feature transformation, but not to BERT transformed feature vectors. Considering the linear separation model worked even better, it can be said that the 3rd degree polynomial kernel was too complicated for our training sample with token count and TF-IDF, but more accurate with BERT transformed samples.

For neural networks, a concession is made to only use BERT transformed features to save computation time. It turns out that the hidden layer of (256, 64) provides a satisfactory training result and the best result seems to experience little overfitting. This is supported by the fact that the training and testing F1 score are roughly the same around 0.8. In unrecorded manual tuning of hidden layer structures, one layer and three layers both yielded worse results than two layers, which indicates that the problem complexity is better suited to using two hidden layers. It is also worth mentioning that ReLU and tanh both exhibited high training F1 scores until lambda is increased to 1, showing that increasing the regularization intensity contributed positively to the result by limiting overfitting.

For Naive Bayes, Random Forest, and Gradient Boosting algorithms, there are generally less hyperparameters to tune (regularization may be impossible). Sometimes a randomized algorithm (such as Random Forest) can give significantly different results with the same parameters, which increases the difficulty of tuning hyperparameters. The best hyperparameters are usually found somewhere near the default values, but it's not always true.

For all feature transformations, BERT is revealed to be the best transformation. The best F1 scores of all our six models are similar, maybe due to the property of our dataset. Our best model is a combination of SVM, neural networks, logistic regression, and gradient boosting, producing a F1 score of 0.812121.

Methodology-wise, it is clearly useful to try different C scores and manually adjust hyperparameters after obtaining the highest testing F1 score. This technique yields our best F1 score for all the models. Combining different models also turns out to be useful in improving the performance. It is strongly believed that a larger dataset would yield more accurate results.

Works Cited

Bert:

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Naive Bayes:

Hand, D. J.; Yu, K. (2001). "Idiot's Bayes — not so stupid after all?". International Statistical Review. 69 (3): 385–399.

Random Forest:

Ho, Tin Kam (1995). Random Decision Forests. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

Gradient Boosting

Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009). "10. Boosting and Additive Trees". The Elements of Statistical Learning (2nd ed.). New York: Springer. pp. 337–384.