CS-UY 4563: Machine Learning:

Final Project Written Report

NLP For Stress Detection Based On Twitter Dataset

2:00 PM Section

April 29, 2024

Professor Linda N. Sellie

Garvin Huang, Oliver Sun

Introduction

The purpose of this project is to use machine learning models to accurately classify whether or not a piece of text will elicit a positive stress response or a negative stress response. The dataset was taken from Kaggle, and contained text taken from both Reddit and Twitter and their corresponding labels, with a 1 being a negative stress response and a 0 being a positive stress response. The models used in this project were Logistic Regression, Support Vector Machine, and Neural Network

Data Selection

We opted to use the dataset from Twitter ("Twitter_Non_Advert-Tabelle.csv" to be specific) since the data from the Reddit dataset were highly skewed towards 1. The data was split into a training set, a validation set, and a testing set. Missing values were marked and subsequently dropped. The training set was 70% of the original set, while the validation set and test set were both 30% of the original set.

Preprocessing

For preprocessing, all letters were converted to lowercase and all punctuation was stripped. Stemming, which reduces a word to its root word, was then applied to all the words in the dataset. We did NOT remove stop words, as taking them out may interfere with the original sentiment of the text.

Logistic Regression

Logistic regression was the first model used for stress response classification. Two transformations were used in conjunction with two different types of regularization. Hold-out validation was then performed to gauge the accuracy of the models.

The first transformation was the Bag of Words (BOW) transformation, which assigns each unique word a value based on its frequency across all documents. The second transformation used was Term Frequency Inverse Document Frequency (TF-IDF) vectorization, which assigns each unique word a numerical weight based on how important a term is within a document relative to a collection of documents. Both transformations were included in the sklearn.linear_model library. Then we compared the performance of the model when using L1 regularization and L2 regularization using seven C values ranging from 0.0001 to 100.

C Value	Training Accuracy	Testing Accuracy	Training Weighted Avg Precision	Testing Weighted Avg Precision	Training Weighted Avg Recall	Testing Weighted Avg Recall
0.000100	0.385366	0.373377	0.55634	0.54374	0.38537	0.37338
0.001000	0.385366	0.373377	0.55634	0.54374	0.38537	0.37338
0.010000	0.614634	0.626623	0.76133	0.77046	0.61463	0.62662
0.100000	0.776307	0.785714	0.79178	0.80010	0.77631	0.78571
1.000000	0.945645	0.798701	0.94578	0.80097	0.94564	0.79870
10.000000	1.000000	0.800325	1.00000	0.80179	1.00000	0.80032
100.00000	1.000000	0.795455	1.00000	0.79752	1.00000	0.79545

Table 1: Logistic Regression with BOW Transformation and L1 Regularization

C Value	Training Accuracy	Testing Accuracy	Training Weighted Avg Precision	Testing Weighted Avg Precision	Training Weighted Avg Recall	Testing Weighted Avg Recall
0.000100	0.615331	0.626623	0.76113	0.77046	0.61533	0.62662
0.001000	0.654355	0.649351	0.75076	0.75242	0.65436	0.64935
0.010000	0.798606	0.792208	0.81216	0.80968	0.79861	0.79221

0.100000	0.918467	0.805195	0.91929	0.81244	0.91847	0.80519
1.000000	0.990244	0.808442	0.99023	0.81154	0.99024	0.80844
10.000000	1.000000	0.811688	1.00000	0.81426	1.00000	0.81169
100.00000	1.000000	0.814935	1.00000	0.81702	1.00000	0.81494

Table 2: Logistic Regression with BOW Transformation and L2 Regularization

C Value	Training Accuracy	Testing Accuracy	Training Weighted Avg Precision	Testing Weighted Avg Precision	Training Weighted Avg Recall	Testing Weighted Avg Recall
0.000100	0.385366	0.373377	0.55634	0.54374	0.38537	0.37338
0.001000	0.385366	0.373377	0.55634	0.54374	0.38537	0.37338
0.010000	0.614634	0.626623	0.76133	0.77046	0.61463	0.62662
0.100000	0.619512	0.631494	0.76003	0.76913	0.61951	0.63149
1.000000	0.799303	0.775974	0.80985	0.78904	0.79930	0.77597
10.000000	0.997909	0.811688	0.99791	0.81297	0.99791	0.81169
100.00000	1.000000	0.797078	1.00000	0.79901	1.00000	0.79708

Table 3: Results of TF-IDF Transformation w/ L1 Regularization

C Value	Training Accuracy	Testing Accuracy	Training Weighted Avg Precision	Testing Weighted Avg Precision	Training Weighted Avg Recall	Testing Weighted Avg Recall
0.000100	0.614634	0.626623	1.00000	0.77046	0.61463	0.62662
0.001000	0.614634	0.626623	1.00000	0.77046	0.61463	0.62662
0.010000	0.614634	0.626623	1.00000	0.77046	0.61463	0.62662
0.100000	0.644599	0.649351	0.93174	0.77823	0.64460	0.64935
1.000000	0.890592	0.797078	0.90785	0.86003	0.89059	0.79708
10.000000	0.991638	0.814935	0.99171	0.82173	0.99164	0.81494
100.00000	1.000000	0.819805	1.00000	0.86003	1.00000	0.81981

Table 4: TF-IDF w/ L2 Regularization

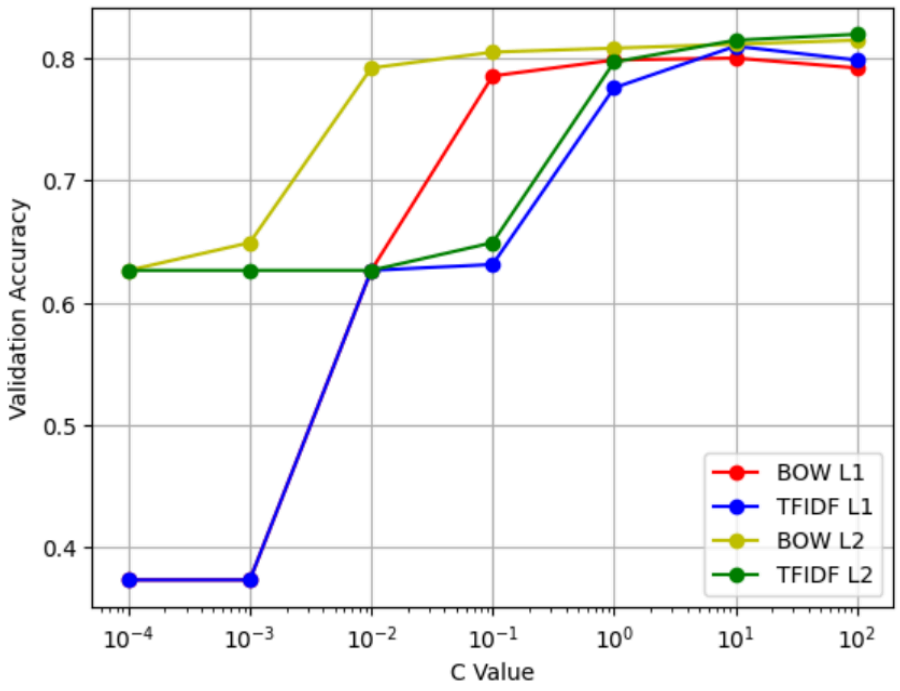

Figure 1: Validation Accuracy of All Models Across Different C Values

Based on the data above, the best model was the one that used a TF-IDF transformation with L2 regularization with a C value of 100, having a validation accuracy of 81.98 percent. It should also be noted that, based on the graph above, accuracy was more correlated with the type of regularization used than the transformation used; In this case L2 regularization worked better. This might be attributed to the fact that text-based data results in higher variance, which would be penalized harder by L2.

Support Vector Machine (SVM)

Support Vector Machines were the second type of classifier used for stress classification. The Scikit-Learn library implements a multitude of different kernels for SVM. We chose to experiment with three different kinds of kernel functions: linear, RBF (Radial-Basis Function), and polynomials of degree 3. All three functions use ridge regression with a squared L2 penalty. Each kernel function was observed with a set of hyperparameters, with C = [[0.1, 1, 10, 100, 1000, 10000]. These hyperparameters were chosen because they represent having either high regularization or minimal regularization.

C	Training Accuracy	Testing Accuracy	Weighted Avg Precision (Train)	Weighted Avg Recall (Train)	Weighted Avg Precision (Test)	Weighted Avg Recall (Test)
0.1	0.642	0.684	0.691	0.642	0.720	0.684
1	0.957	0.822	0.957	0.957	0.820	0.822
10	0.999	0.827	0.999	0.999	0.828	0.827
100	1.000	0.822	1.000	1.000	0.822	0.822
1000	1.000	0.822	1.000	1.000	0.822	0.822
10000	1.000	0.822	1.000	1.000	0.822	0.822

Table 5: TF-IDF with Linear SVM

C	Training Accuracy	Testing Accuracy	Weighted Avg Precision (Train)	Weighted Avg Recall (Train)	Weighted Avg Precision (Test)	Weighted Avg Recall (Test)
0.1	0.630	0.664	0.672	0.530	0.677	0.664
1	0.986	0.834	0.987	0.987	0.840	0.835
10	1.000	0.844	1.000	0.1	0.844	0.844
100	1.000	0.844	1.000	0.1	0.844	0.844
1000	1.000	0.844	1.000	0.1	0.844	0.844
10000	1.000	0.844	1.000	0.1	0.844	0.844

Table 6: TF-IDF with RBF SVM

C	Training Accuracy	Testing Accuracy	Weighted Avg Precision (Train)	Weighted Avg Recall (Train)	Weighted Avg Precision (Test)	Weighted Avg Recall (Test)
0.1	0.630	0.664	0.672	0.530	0.677	0.664
1	0.986	0.834	0.987	0.987	0.840	0.835
10	1.000	0.844	1.000	0.1	0.844	0.844
100	1.000	0.844	1.000	0.1	0.844	0.844
1000	1	0.844	1	0.1	0.844	0.844
10000	1	0.844	1	0.1	0.844	0.844

Table 7: TF-IDF with 3rd Degree Polynomial SVM

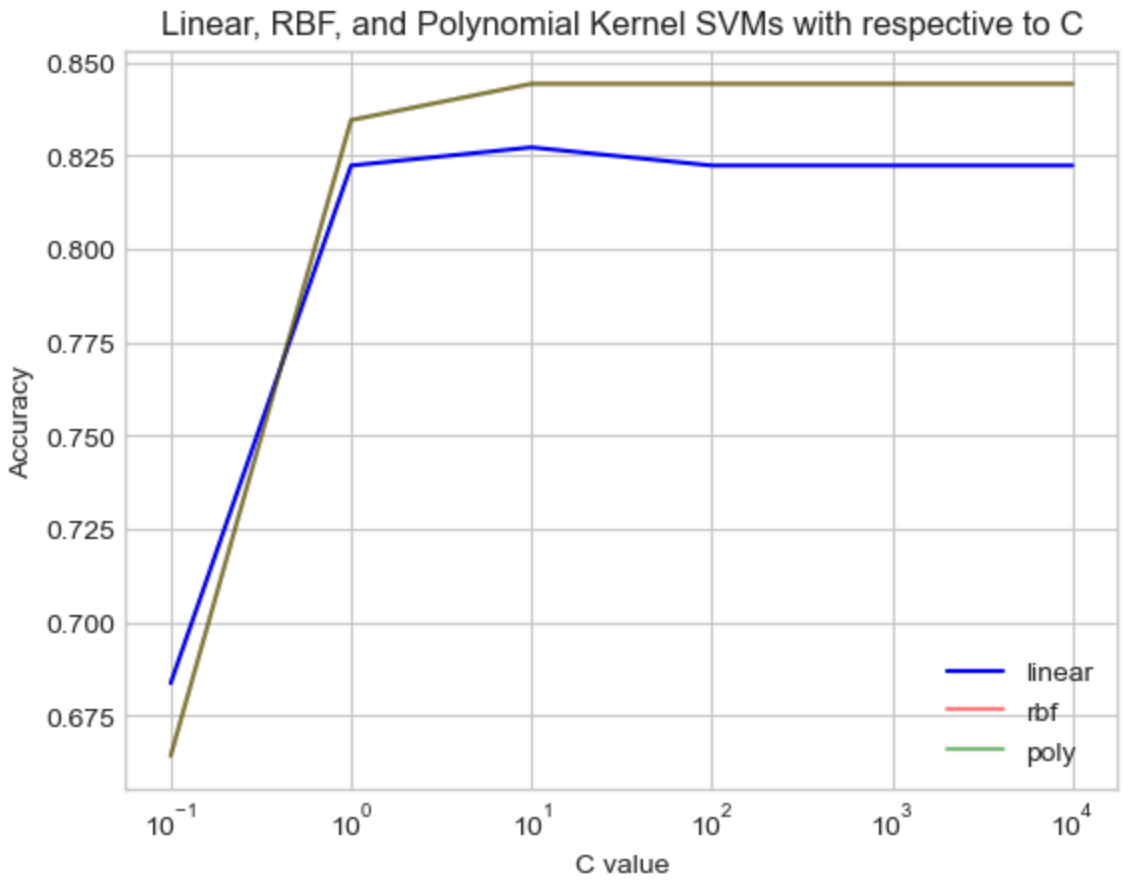

Figure 2: Linear, RBF, Polynomial Kernel SVCs

Based on the result of our graphs, we can see that both RBF and Polynomial kernels outperformed the linear SVC. However, both RBF and Polynomial Kernels are performing the exact same. In addition, we found that C values beyond 10 do not increase accuracy, only remaining the same or worse when C is increased past that. It is suspected that the similarity between the two is the result of Sklearn's implementation of the polynomial and RBF kernels.

The results of the previous test indicated that there was a ceiling in accuracy. In an attempt to further delineate the results of the RBF and Polynomial kernels, we performed K-fold validation to investigate. The C value chosen for the K-fold validation is C = 10, as it yielded the highest accuracy previously.

# of folds	Linear validation	RBF validation	Polynomial validation
2	0.7756	0.7848	0.6585
3	0.7896	0.8006	0.6653
4	0.7902	0.8030	0.6652
5	0.8049	0.8104	0.6738
6	0.8036	0.7993	0.6732
7	0.8043	0.8043	0.6756
8	0.8006	0.8037	0.6756
9	0.7975	0.8049	0.6762
10	0.8067	0.8104	0.6768

Table 8: k-Fold Cross Validation for Linear, RBF, and Polynomial Kernel SVCs

When looking at the results of the k-fold cross validation, all three kernels showed worse accuracy. However, the Polynomial kernel suffers the most from accuracy loss, while RBF validation remains the highest in overall accuracy. For all three, the trend for accuracy increases with # of folds. This follows because training size increases with # of folds, and an increase in training size means that the model should have improved generalization error. Based on the graph above, we have found that the best classifier for modeling stress response in text is the RBF kernel SVC, using a C of 10, only slightly ahead of the linear kernel. This is slightly unusual as linear classifiers are usually preferred for natural language processing, but this tells us that the dataset is likely not very linearly separable.

Neural Networks

We used Neural Networks for our last model. We used three different activation functions for the hidden layer: the sigmoid, ReLu, and tanh. There were three different neural network configurations used: one hidden layer of 100 neurons, ten hidden layers of 100 neurons, and one hidden layer of 10 neurons. The Keras Adam optimizer was used with a learning rate of 0.001 was used to train our neural networks. Each neural network was trained for 10 epochs. A snapshot of the neural network's training and validation accuracy was taken at each epoch and put on a graph.

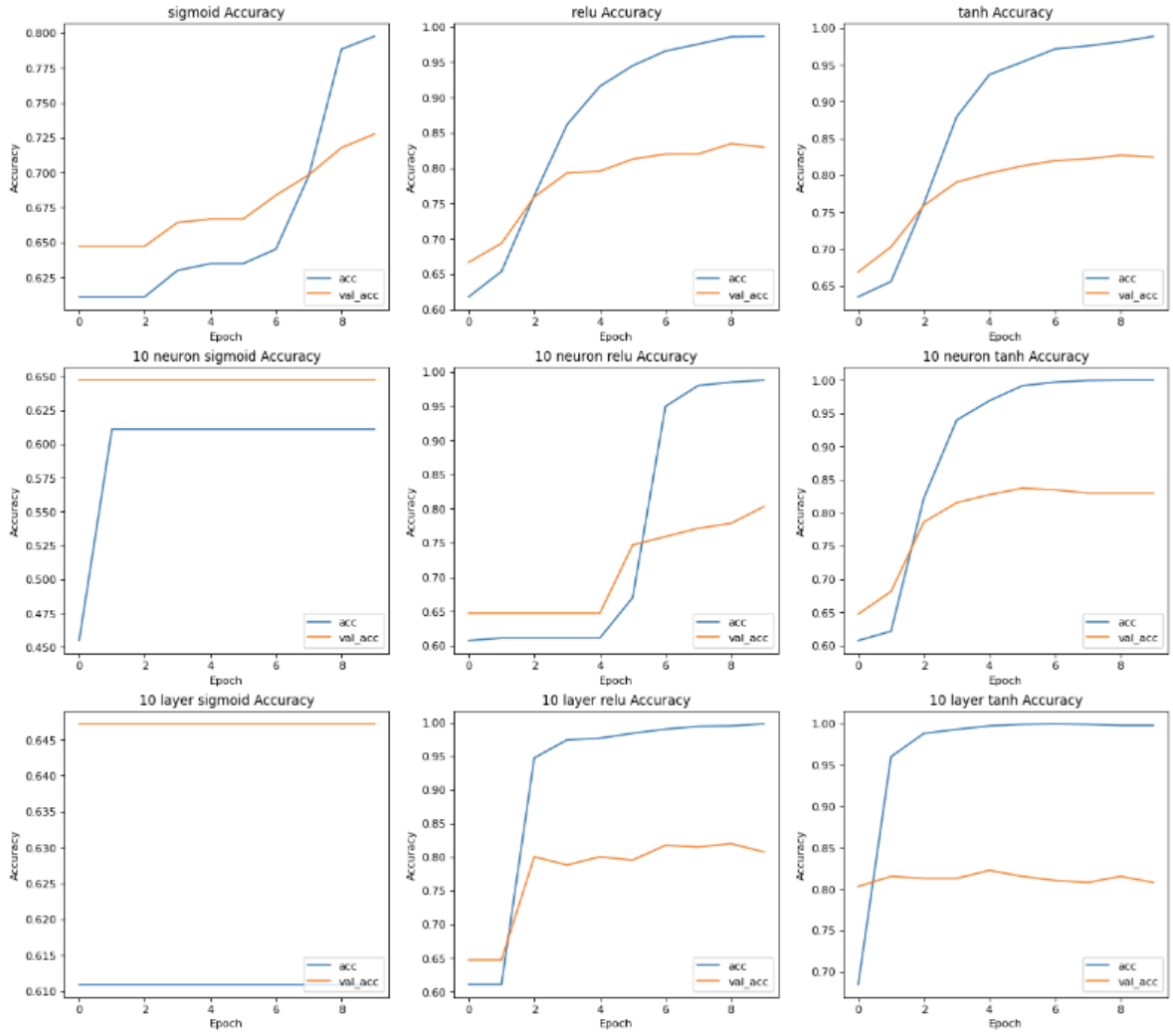

Figure 3: Accuracy Graphs for 3 different Neural Network Structure

The table below depicts the final training and validation accuracies for each activation function given a specific architecture after 10 epochs:

	Sigmoid train	Sigmoid val	ReLu train	ReLu val	Tanh train	Tanh val
1 hidden layer, 100 neurons per layer	0.7976	0.7275	0.9866	0.8297	0.9884	0.8248
1 hidden layer, 10 neurons per layer	0.6110	0.6470	0.9878	0.8029	1.000	0.8297
10 hidden layers, 100 neurons per layer	0.6110	0.6470	0.9982	0.8078	0.9976	0.8078

Table 9: Neural Network Accuracy with Sigmoid, ReLu, and tanh Activation Functions

From this, we determine that the most effective activation function and architecture is the ReLu activation function in conjunction with the 1 hidden layer with 100 neuron neural network.

Final Conclusion:

The best performing model out of all was the SVM using RBF Kernel and TF-IDF vector feature transformation but overall, the validation accuracy of all three models converged to roughly the same value. The reason the SVM outperformed the Logistic model can be attributed to the fact that an SVM maximizes the margin for the decision boundary, leading to more certain and better classifications in general. The reason the SVM outperformed the Neural Network could be due to the fact that the Neural network did not have as many hidden layers, leading to underfitting when it came to high dimensional data, which is also the main motivator for using a SVM.

The main reason that SVM is the optimal choice for our problem is because it is very versatile when it comes to high dimensional data, and is why the Polynomial and RBF kernels outperformed the Linear kernel. However, when working with text-based data, a feature vector can have thousands of features, and the Polynomial Kernel most likely suffered from overfitting even with only a degree of 3, leaving us with the RBF kernel

Logistic	SVM	Neural Network
0.820	**0.844**	0.830

Table 10: Best Results Across Models

Works Cited

Stress Detection Dataset

https://www.kaggle.com/datasets/mexwell/stress-detection-from-social-media-articles/data

Sklearn Logistic Regression

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Sklearn SVC

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Example Keras NN Implementation

https://github.com/zafirhsn/machine-learning/blob/master/FinalProject_Log%2BSVM.ipynb