Text to Emoji

Efraim Paley (ID.327340089),David Goldberg (ID.204507271),Esther Revivo(ID.207171968)

Submitted as final project report for Practical Topics in Machine Learning, BIU, 2021

1 Introduction

In the age we live in with the world so engrossed in social media, a big part of that is communication via messages. Almost every social media platform these days has a messaging system as part of their platform. A big part of the messaging experience is sending emojis and GIF's to one another, which in a sense is supposed to represent the sentiments and the feelings of the sender. This method of messaging has grown in popularity especially over recent years. An issue with sending these emojis however is that, sometimes, people can't always find the appropriate emoji to represent the appropriate feeling that you currently have.

Additionally, it may be difficult for the receiver to understand the senders current feelings and emotions based on the text they sent. In light of this, we decided to create a classifier to help with this issue. The classifier we created helps predicts emotions for associated texts. We created this classifier with the hopes that it will will further help to represent emotions associated with text messages. Furthermore, we hope this can we a useful tool for Product Managers who, when trying to gauge the interest of users in their products, can see how users feel towards the product based on the emotions that are represented in reviews left by the users. The Product Manager, in place of reading the whole review given by the user, can instead use our tool to input the text of the review and see what emojis are outputted, and in essence getting a feel of the users feeling towards the product based on the emojis. This can save valuable time and will allow more reviews to be evaluated, and in turn can help improve the product based on the feedback from those reviews.

1.1 Related Works

We found a similar idea to ours on a website called DeepMoji.

2 Solution

2.1 General approach

We choose a Data set from Kaggle, which is made up of thousands of lines text. In each line there is a text message which is mapped to a certain feeling. The emotions in the data set are
1. Joy,
2. Sadness,
3. Anger,
4. Fear,
5. Love
6. Surprise.

In order to make the data set easier to work with, we first thought it would be smart to translate the data from text form into vectors, where each vector represents a different type of emotion. In the end, we ended up using a different idea where we attached a text label to a sentence. Despite this, we still left the option of representing the emotions as vectors in our code.
We then applied Machine Learning algorithms in order to help predict the emotion associated with a certain text, based on the appropriate vectors. We attempted multiple classifiers in order to find the classifier which gave the best results. The classifiers we tried were 1. SVM, 2. LinearSVC, 3. RandomForestClassifier, 4. DecisionTreeClassifier, 5. AdaBoost. Once we found the best classifier, we then tested the model based on the chosen classifier and got our results.

2.2 Design

As stated above, we chose our Data set from Kaggle. Our Data set was made up of lines where each line was made up of a sentence and then at the end of the line, the appropriate emotion to match the sentiment presented in the sentence. An example of a line found in the data set is as follows:
" I just feel extremely comfortable with the group of people that i don't even need to hide myself "; joy
. You can see that the sentence is followed by the emotion. We wrote the code itself in python. The emotions found in the data set were 1. Joy 2. Sadness 3. Anger 4. Fear 5. Love 6. Surprise We thought of an option to represent the emotions as vectors as such:
1. Joy:"1. 0. 0. 0. 0. 0."
2. Sadness: "0. 1. 0. 0. 0. 0."
3. Anger: "0. 0. 1. 0. 0. 0."
4. Fear: "0. 0. 0. 1. 0. 0."
5. Love: "0. 0. 0. 0. 1. 0."
6. Surprise: "0. 0. 0. 0. 0. 1."
As stated above we also used the option of attaching a label to a sentence, but

we left both options in our code.

We then made NGrams for the given texts, with a range from 1 to 4.
As stated earlier, the 5 classifiers we used where SVM, LinearSVC, RandomForest, DecisionTree and AdaBoost, We will give a brief explanation the different classifiers we tried:

1. **SVM** - A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text. Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it's common to have access to a dataset of at most a couple of thousands of tagged samples.).

2. **Linear SVC** - The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. This makes this specific algorithm rather suitable for our uses, though you can use this for many situations.

3. **RandomForestClassifier** Random forests or random decision forests are 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, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of over fitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees.

4. **DecisionTreeClassifier** - A decision tree is a supervised machine learning classification algorithm used to build models like the structure of a tree. It classifies data into finer and finer categories: from "tree trunk," to "branches," to "leaves." It uses the if-then rule of mathematics to create sub-categories that fit into broader categories and allows for precise, organic categorization. As the rules are learned sequentially, from trunk to leaf, a decision tree requires high quality, clean data from the outset of training, or the branches may become over-fitted or skewed.

5. **AdaBoostClassifeir** - An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

3 Experimental results

The data set we used came already split into a train set and a test set. The data overal was quite large and we had trouble running our code on such a large data set so we ended up cutting down the data a bit. We ended up splitting 20 80 with 20 percent in the test data and 80 percent in the train data.

Below we have a table showing the number of emotions in both the test and the train data

*** TRAIN: DATA INFO ***

joy: 2693
sadness: 2298
anger: 1091
fear: 935
love: 661
surprise: 322

*** TEST: DATA INFO ***

joy: 695
sadness: 581
anger: 275
fear: 224
love: 159
surprise: 66

The training and test accuracy's for each classifier are summarized in the table below. As can be seen in the table, AdaBoostClassifier gave the clos-

Classifier	Training Accuracy	Test Accuracy
SVC	0.896888	0.515
LinearSVC	0.999375	0.8435
AdaBoostClassifier	0.315336	0.32
RandomForestClassifier	0.992001	0.6895
DecisionTreeClassifier	0.999375	0.8385

est results between the test and training accuracy's. All of the other classifiers seemed to be over fitting because the training accuracy was so much higher then the test accuracy. However, we noticed that in the test accuracy the LinearSVC

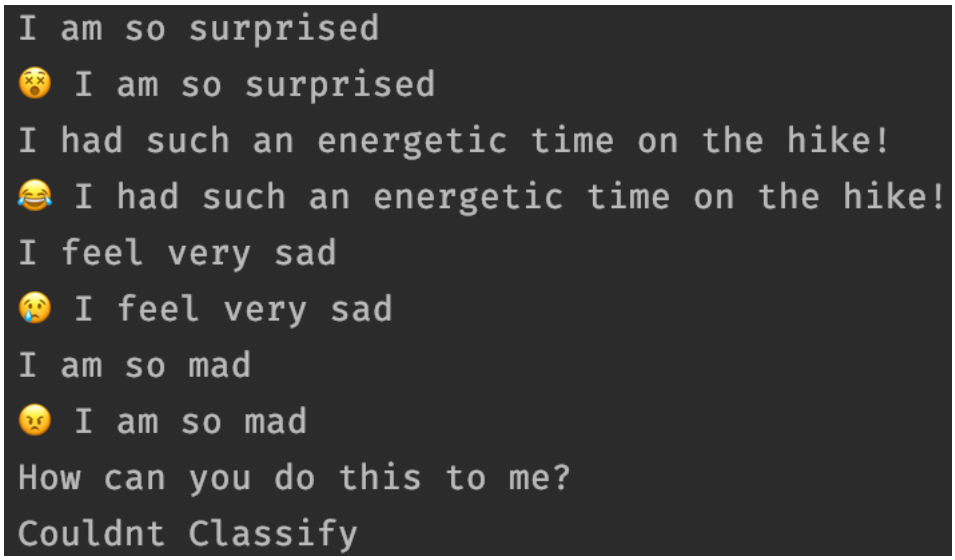
gave the best results so we tried to fine tune the parameters for that classifier. We tired fine tuning with a number of parameters but many of them caused issues with the program, and we couldn't work with them. We ended up using the following parameters because they worked the best:
parameters = 'C':[3, 5, 10], 'tol': [0.1, 0.01, 0.001]

These were the results from the Linear SVC after we tried tuning the parameters:
Validation acc: 0.8147742036227358
Training acc: 0.9993750781152356
Test acc : 0.839
Best parameter: 'C': 3, 'tol': 0.001

We can see that there still seems to be over fitting between the train and the test. Also, despite the fine tuning of the parameters, the test results were similar as to the results without the tuning.

We also preformed an error analysis. Below is the confusion matrix:
[[0 0 0 0 0 0 0]
[1 223 6 22 3 20 0]
[0 11 166 14 0 18 15]
[1 4 5 634 24 22 5]
[0 5 1 39 104 9 1]
[0 15 6 39 2 518 1]
[0 0 11 19 0 3 33]]

Additionally, we put a few sentences of our own and wanted to see how the model preformed with them. Overall the simple sentences performed well but regarding the more complicated sentences, it had a hard time classifying them Below are examples of some of the sentences:



4 Discussion

We really enjoyed doing this project with taking a data set and running our own experiments with different classifiers. We also felt it was cool to add the NLP element in this project with the adding of the n-grams. As stated above, it seemed that the classifiers were over fitting. We were not sure why this was happening but perhaps further fine tuning of the parameters may help solve this issue. Despite this we saw that the Linear SVC did give the best results on the test, but despite fine tuning the parameters, we were not able to improve those results.

Overall, We feel that this is a really cool idea and can be a useful tool to help with getting the sentiments of different texts without having to read the entire text, which can be helpful in many different aspects of social media as well as other fields.

5 Code

Here is a link to our code repository: https://github.com/EtiPeretz/text-to-feelings-project-machine-learning