ORIE 4741 Final Report

Aliyah Geer, Adarsh Mital, Elise Kronbichler

December 2020

1 Background

Sentiment classification of text is a classic NLP task that has been performed in various contexts. Most of these models have been fitted on 'well structured' and fully-formed pieces of text e.g. grammatical sentences that include punctation. In comparison, song lyrics are less structured and logical, and are also missing spoken pauses and emphasis that are important to the meaning.

Thus, it is difficult to find consistent feature transformations which capture their full semantic meaning. In this project we seek to fit models that classify human annotated lyrics beyond sentiment, into one of four moods: Happy, Angry, Sad, and Relaxed.

2 Dataset

Our data was initially comprised solely of the "Moody Lyrics" dataset: Mood annotated lyrics published in a paper by Erion Cano and Maurizio Morisio from the Polytechnic University of Turin. This dataset features about 2000 songs that have been labeled as one of four moods: "happy", "angry", "sad", and "relaxed". Because this dataset only contains title of the song and the labeled mood, we used a package called "lyricsgenius" to scrape lyrics of a specified song from the site Genius.com.

To supplement this dataset, we collected 700 additional human annotated lyrics from a previous study that Cano and Morisio referenced in their paper. The methodology for annotating these songs was extremely similar (though not identical), and the four categories of annotation were the same. We felt it was appropriate to include these 700 additional songs, in order to expand our dataset and prevent overfitting our models.

Below are several visualizations of the most common words found in our data set under specific labels. The two wordclouds demonstrate that there are obvious differences between the words that occur most often in one class vs. another (i.e. "happy" vs. "angry").

Figure 1: angry lyrics

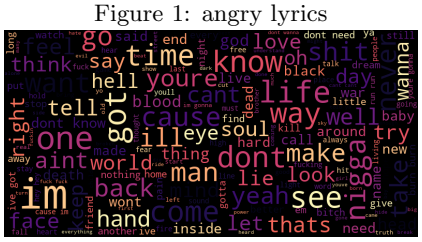
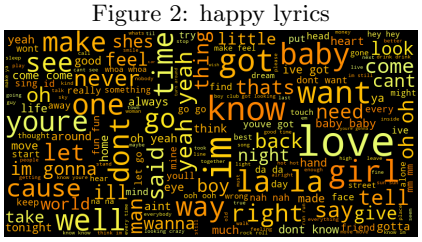


Figure 2: happy lyrics



3 Preprocessing

Before anything else, we needed to clean the lyrics dataset. Here, we addressed a multitude of issues: Null values, upper-case words, non-english lyrics, "stopwords", strangely formatted audio books, and punctuation marks.
We created a function to address these issues which converted all words to lower-case, eliminated rows with null values and non-english lyrics, as well as any punctuation marks and "stopwords" (words common to all songs e.g "I", "and", "that", "the"). After this step, we had a cleaned and comprehensive set of lyrics, with 2655 unique songs. The distributions of class frequencies were fairly even amongst "happy", "sad", and "angry" songs, though we had approximately 5 percent fewer "relaxed" labels.

4 Feature Engineering: Bag of Words

For our preliminary analysis, we decided to use a bag of words model to extract features from our lyrics. In a BOW model, each lyric's vector representation is simply comprised of the frequency counts of each vocabulary word in that sentence. We thought that the BOW approach would give us a basic feature representation of our data, from which we could continue to build more complex representations.

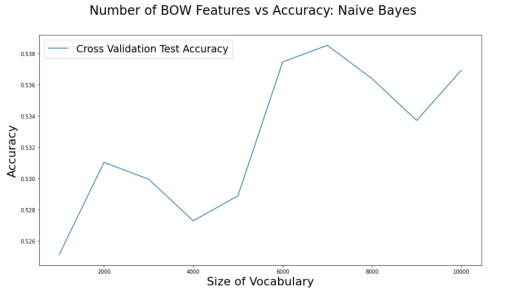
4.1 Model: Naive Bayes

To get a "base" accuracy, we used a probability-based classification function called Multinomial Naive Bayes. This function looks at the probability of each class, as well as the probability of the features given the class. Naive Bayes then uses that to find the probability of the class given the features. Since our parameters are feature counts, it was appropriate to use Multinomial Naive Bayes over Bernoulli or Gaussian.

The decision rule of a Naive Bayes Classifier is the following, where C_k denotes a class and x_i a feature.

$$\text{y} = argmax_{k\in(1\ldots K)} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

To determine the optimal number of features to use for Naive Bayes, we used k-fold Cross Validation to get an estimate of the prediction accuracy of each number of features in a set of predetermined feature numbers (from 1000 to 10000)



There is only a variation of 2% within our worst and best performing BOW models, and the best accuracy we achieved was **53.8%** at 7000 features.

4.2 Model : Logistic Regression

We fitted another relatively simple model using our BOW feature transformation with Logistic Regression. Logistic Regression allowed us to find a more probabilistic mapping from our features to labels, and is generally more explainable than a Random Forest or Neural Network. Logistic Loss is known as

$$l_{logistic}(x, y, w) = log(1 + exp(-yw^T x))$$

Because we have 4 classes, we used Multinomial Logistic Regression, which runs a series of individual binomial logistic regressions. To prevent overfitting and avoid outlier impact, we used l2 regularization with a default regularization strength of 1. The number of BOW features we used was 7000.

We achieved a poor test set accuracy of 49%, but a strong training set accuracy of 91.5%. With both our models on BOW transformations overfitting so much, we decided to try a more complex feature representation in TF-IDF.

5 Feature Engineering: TF-IDF

TF-IDF differs from BOW in that it weights words based on their frequency in each individual document(song) vs the number of documents(number of songs) that they occur in. The mathematical representation is given below

$$\text{tf-idf(t, d)} = \text{tf(t, d)} * \log(\text{N}/(\text{df(t)} + 1))$$

tf-idf(t, d) is the score a term t in a document d gets. tf(t, d) is the number of times t occurs in d over the number of words in d. df(t) is the number of occurrences of t in the entire corpus of documents.

5.1 Model: SVM

With these more complex feature embeddings, we fit a more complex classification model, opting for a Support Vector Machine(SVM). The ability to form non linear decision boundaries through kernel transformations as well as adjust our regularization made SVM an attractive prospect to us. SVM uses hinge loss with l2 regularization. Hinge loss can be expressed through the formula

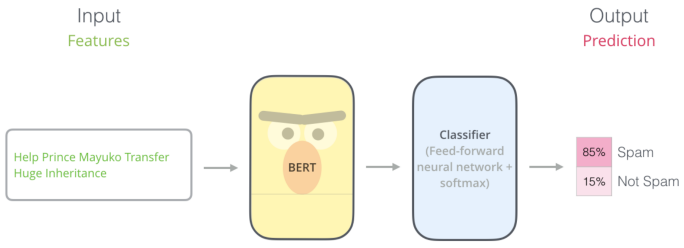
$$l_{hinge}(x, y, w) = (1 - yw^T x)_+$$

We adjusted our SVM model's number of TF-IDF features to use (ranging from 1000-10000), our kernel type, and the 'C' parameter, which conceptually determines the strength of our regularization.

Using cross-validation to estimate our accuracy, we found that regardless of how strong our regularization is, or the number of TF-IDF features, we are still massively overfitting to the training data, with test set accuracies hovering between **58-59 %**. Though our TF-IDF model with SVM resulted in better fit than BOW with Logistic Regression of Naive Bayes, it became clear that straightforward word-based embeddings are not optimal to model this dataset. So we moved on to sentence-based embeddings.

6 Feature Engineering: SBERT Embeddings

BERT (Bi-directional Encoder Representation for Transformers) is a breakthrough language model that has been pre-trained on a large corpus including the Toronto Book Corpus and Wikipedia, created by Google. Because of its extensive training, BERT can be used to transfer knowledge to many tasks through its pre-trained models, with the simple addition of a layer fine-tuned to that task. This is an image of a sentence conceptually passing through BERT's pretrained layers into its final layer.

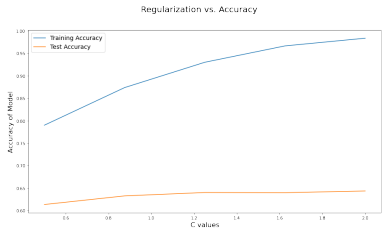


Transformers are models that are able to use the concept of 'attention'(an awareness of the relationship of every element in an input sequence to one another), to create complex representations of sequences. BERT is a bidirectional transformer, meaning it creates understandings of its sequences from left to right and vice versa, allowing for more complex representation than if unidirectional.

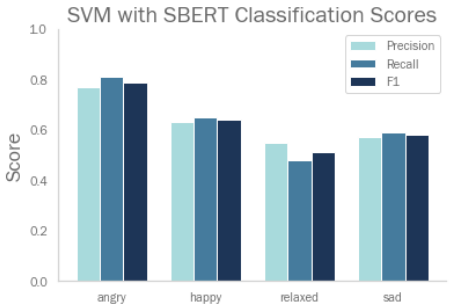
SBERT (Sentence-BERT) is a variant of BERT, which presents recent advancement in generating embeddings for sentences by passing in two sentences into BERT at the same time and performing pooling + additive transformations on their embeddings, resulting in similar vector representations for sentences semantically similar to one another. This is the embedding model we use here, in the hopes of creating embeddings that capture the relationships between words in our lyrics. We used the Sentence Transformers' SBERT package to create these representations, ultimately ending up with 768 features for each of our 2655 songs. In this step we do not fine-tune a final neural network classification layer that would take in these embeddings, as is often done in BERT tasks. Instead, we take SBERT's embeddings of our sentences and then use the classification models outlined below on these features.

6.1 Model : SVM

We again fitted an SVM, now to these SBERT features. We began with Grid Search to find what kernel type and gamma value would reduce our test error. After this, we plotted our regularization strength vs our accuracy via cross validation.

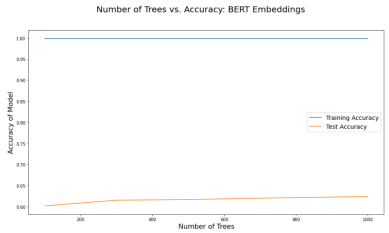


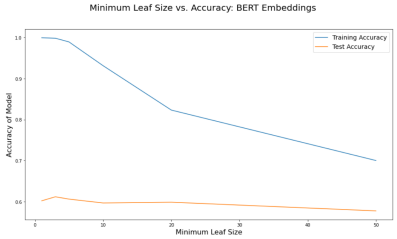
With these superior sentence embeddings, we are able to get a much better sense of how regularization strength affects our ability to generalize. The best generalizing model we came up with had hyperparameter values of C = 2.5 and gamma = .01. Our cross validated test accuracy was **64%**. The following was our model's classification report.



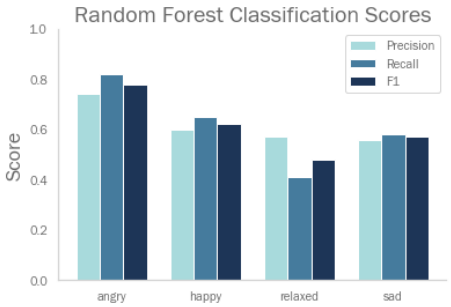
6.2 Model : Random Forest

For our second model on SBERT embeddings, we decided to use a Random Forest. This does well with high dimensionality, does not often overfit, and has many adjustable parameters. Random Forests operate as collections of individual decision trees that vote by committee to classify, each taking a small subset of features into account. Because there are so many parameters to tune, we focused on a select most important ones. Holding other parameters constant, we found the cross validation accuracy when adjusting these parameters.





We saw that as we increased the number of uncorrelated trees, our test cross validation accuracy rose. We overfitted less because our variance decreased across the Random Forest model. As leaf size increased, we began to overfit less until size 3. We created these graphs for other features as well and then ran grid search based on the points in our graphs where we saw best generalization. Our final hyperparameters are: Max Depth = 23, Max Features = 15, Min Trees = 3, and No. of Estimators = 2000.

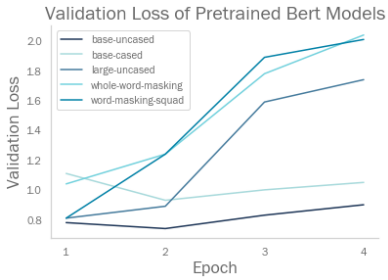


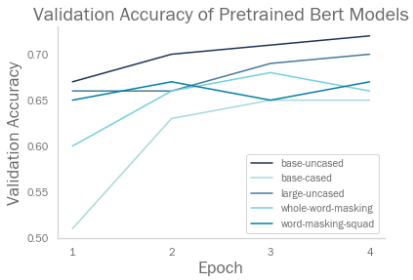
We achieved a cross validated test accuracy of **63 %**. Additionally, this was our least overfitted model so far, though our recall on 'relaxed' songs is quite poor. With such significant jumps in our test accuracy, it is fair to conclude that the BERT embeddings captured the semantic similarities of our sentences considerably better than BOW of TF-IDF. Though some of the credit does go to better hyperparameter tuning as well.

7 Full BERT Process

Lastly, we used a full neural network based BERT approach. In this step, we took the final embeddings of our sentences and added an untrained layer of neurons at the end. We then fine-tuned this final classification layer. We chose Hugging Face's BERT for Sequence Classification model for this.

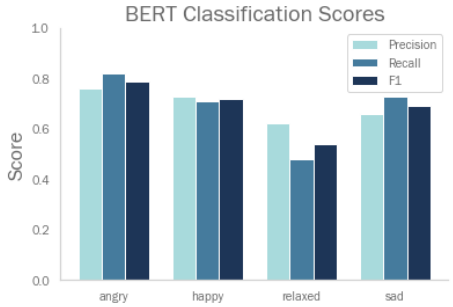
BERT has several pretrained models to choose from, so we trained on multiple of these and compared results in order to determine the most effective pretrained model to adapt to our task. Additionally, we tested the optimal number of epochs, or cycles for fine-tuning our final classification layer. Too few epochs results in underfitting, while too many results in overfitting. The results of these tests are shown below.





Bert-base-uncased resulted in the lowest validation loss values and the highest validation accuracy, meaning that it is the most effective of the pre-trained models on our data. We also determined that the optimal number of epochs when using bert-base-uncased was 2, since it results in the lowest loss value.

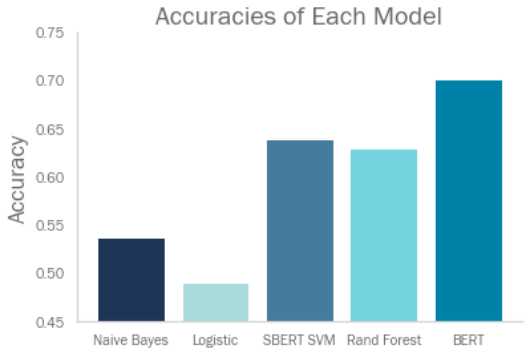
We ran our final model using these parameters and evaluated it on our test data, resulting in a **70.1% test accuracy, the highest of our models**. We also analysed the precision, recall, and f1-score of this model, shown below.



Like in previous cases, the model underperformed on "relaxed" lyrics, with a particularly low recall, meaning that many of the lyrics our model classified as "relaxed" actually belonged to another class.

8 Conclusion and Possible Improvements

The motivation behind our project was to determine how well state of the art NLP models could capture the semantic complexity of more unstructured pieces of text, and how much more successful they would be than traditional feature representations of text. We iteratively progressed through more complex feature transformations and models, concluding with a neural network based embedding and training technique. Using hyperparameter tuning, we were able to overfit less and generalize better throughout our process, with our best model(full BERT based process) coming out on top with an accuracy of **70%**



Though we still do overfit considerably, this was a strong result for us considering we had four separate classes to deal with, and the complex and variable nature of our data. However, our current models are still not strong enough to apply in a generalized industry based scenario for mass mood annotation of song lyrics, at least not a form of annotation that would be most 'human-like'. Further improvements on our process could involve trying out several other vector representation models(GloVe, ELMo, etc). Additionally, provided we find a large enough human annotated dataset, narrowing our song selections to a limited group of artists could preserve a consistent lyrical style across our songs, allowing for more confident feature representations.

8.1 A Weapon of Math Destruction?

One of our group members is writing a paper related to what ethical guidelines should exist for newly emerging AI systems that could create sophisticated models of people without needing to access conventional 'protected attributes'. In this vein, a more accurate, reductive mood classification model like the one we have implemented, could be used to label one's taste in music as predominantly "angry", or "sad"–Then jump to various conclusions about how this reflects on the individual. Though our model does not explicitly deal with personal information that could lead it to encode stereotypical biases, an application like the aforementioned could require AI practitioners to expand the notion of protected attributes.

9 References

"MoodyLyrics Dataset." https://www.researchgate.net/publication/317031495_MoodyLyrics_A_Sentiment_Annotated_Lyrics_Dataset

"MOODetector Dataset." http://mir.dei.uc.pt/pdf/Journals/MOODetector/TAFFC_2018_Malheiro.pdf

William Scott. "TF-IDF from scratch in python on real world dataset." https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

Jay Alamar. "The Illustrated Transformer." http://jalammar.github.io/illustrated-transformer/

Jay Alamar. "The Illustrated BERT." http://jalammar.github.io/illustrated-bert/

"Richer Sentence Embeddings using Sentence-BERT." https://medium.com/genei-technology/richer-sentence-embeddings-using-sentence-bert-part-i

Chris McCormick. "Fine Tuning BERT." https://mccormickml.com/2019/07/22/BERT-fine-tuning/

"Sentence Transformers Documentation." https://www.sbert.net/

"Hugging Face Transformers Documentation." https://huggingface.co/transformers/index.html