Introduction

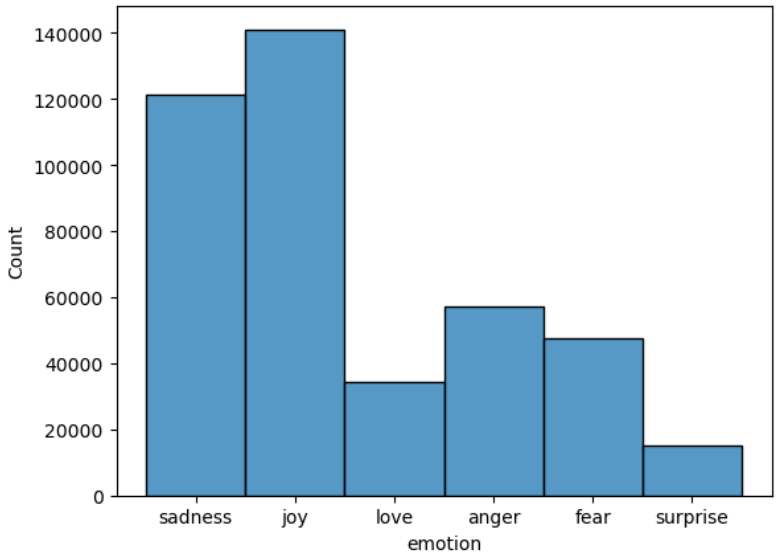
For this project, I am creating an emotion recognition search engine for text. There are many applications of text-based emotion recognition systems, such as opinion mining, personalized recommendation, and content filtering.[1] It is also used for software usability testing, e-education, and reactions to advertisements and websites.[2]

In this project, I built two distinct models used in the same search engine: an emotion classification model and a learning-to-rank model. The classification model allows the user to classify the emotions of queries, which is the main use case. The learning-to-rank model allows the user to retrieve similar messages that exist in the corpus I used. Queries can be inputted either one at a time through the command line, or as a batch of queries in a file. For both the batch option and the retrieval of similar messages in the corpus, I also show the percent distribution of each emotion. In this way, I created a basic interface for opinion mining for emotion recognition, where the "opinion" of a query is the emotion that it is predicted to convey. To conduct opinion mining of certain topics outside the corpus, the batch of queries should all be related to the topic, regardless of the distribution of emotions.

The main contribution of my project is an analysis of the importance of specific features when building models for emotion recognition, including part-of-speech (POS) features and n-gram features. I conducted an ablation study of the classification model for this purpose.

Data

The emotion dataset is available through a GitHub repository[3] that contains a download link. It consists of two columns, one for the text itself and one for the emotion label. There are 416,809 rows and six distinct emotion labels: sadness, joy, love, anger, fear, and surprise. The dataset has been preprocessed with methodologies described in the paper "CARER: Contextualized Affect Representations for Emotion Recognition."[4] The following figure is the class distribution of the dataset.



[1] https://ieeexplore.ieee.org/abstract/document/8625499
[2] https://link.springer.com/chapter/10.1007/978-3-319-08491-6_5
[3] https://github.com/dair-ai/emotion_dataset
[4] https://aclanthology.org/D18-1404/

Related Work

In the paper "Extracting Emotion Causes Using Learning to Rank Methods From an Information Retrieval Perspective,"[5] the authors used a learning-to-rank model to rank Chinese phrases or clauses with respect to certain provoked emotions, which is analogous to query-level document ranking in information retrieval. Specifically, they used the POS ratios of words as features. I drew inspiration from this for the features I constructed, though I am using on English phrases.

Methods

I built two distinct models for my search engine – a Linear Support Vector Classification (SVC) model from SciKit-Learn[6] to quickly predict the emotion of user input, and a learning-to-rank model with PyTerrier[7] to retrieve the most similar documents in the corpus.

Classification model methods

For the Linear SVC model, I constructed text-based features in the form of part-of-speech (POS) percentages, sentiment measures (polarity and subjectivity), and n-grams (1, 2, 3, and 4-grams). I also used a feature (denoted as "er") that is the percentage of single emotion or emotion-like terms that are synonymous with the emotion class labels. I used the NLTK package's synonym expansion functionality for this purpose. Finally, I used a feature (denoted as "neg") that is the percentage of negation terms, which include "no," "not," and all contractions ending with "'t."

The percentage of each POS in a document is calculated as the number of words having that POS divided by the total number of words that fall within any POS. Please note that I only used a subset of POS's deemed important for emotion, so not all words are tagged with a POS. The table below lists the POS's and their Penn Treebank POS Tags,[8] which is what NLTK uses for POS tagging.

POS	POS tag contains
Noun	NN
Pronoun	PR
Verb	VB
Adverb	RB
Modal auxiliary	MD
Adjective	JJ
Conjunction	IN

[5] https://ieeexplore.ieee.org/abstract/document/8625499
[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[7] https://pyterrier.readthedocs.io/en/latest/ltr.html
[8] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

For hyperparameter tuning, I tuned the C regularization parameter, which is typically tuned in a range of log values. I created two dummy baseline models using SciKit-Learn's provided dummy classifier.[9] The first always predicts the most frequent class (joy), and the second uniformly predicts a random class.My classification models uses 25% of the dataset (104,202 rows) for training and the rest (312,607 rows) for testing.

I also conducted an ablation analysis to measure feature importance in my classification model. In other words, I measure the change in performance after leaving out one feature, and I do this for all features. For the n-gram features, this means excluding all 1-grams, 2-grams, 3-grams, or 4-grams features in the sparse representation. Meanwhile, for the POS and other features, I simply exclude that feature.

Learning-to-Rank model methods

For the learning-to-rank model, I used BM25, TD-IDF, and Bayesian smoothing with Dirichlet Prior (DirichletLM) as the relevance features, and the same features in the classification model besides n-grams as the non-relevance features.

Since I did not do my own annotation, I simply assigned a relevance of 3 to the entire dataset to indicate that documents are each relevant to their emotion label, but I used my own custom NDCG scoring function that assigns a relevance of 0 if the emotion does not match. This was necessary because PyTerrier's provided experiment scoring function[10] does not do this; it entirely relies on relevance scores in the corpus that have a range of values (0 through 3).

Evaluation and Results

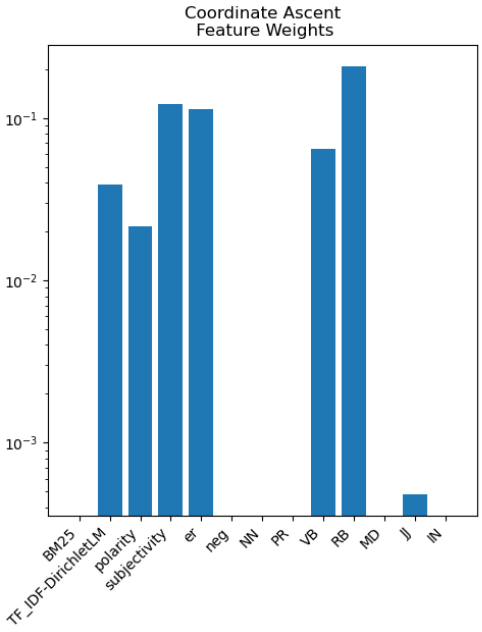
Learning-to-Rank model

I score my learning-to-rank model using NDCG cutoffs calculated with a custom scoring function as described previously. For the learning-to-rank model, besides the BM25 and TF-IDF baselines, I tested random forest regressor and classifier (RF and RFC), fastrank coordinate ascent (CA), and LambdaMart (LGBM). The CA model has the best performance. However, the basic BM25 and TF-IDF models both outperform it by a significant margin.

Model	NDCG, 5	NDCG, 10	NDCG, 20	NDCG, 50	NDCG
BM25	0.9242	0.8811	0.8367	0.7961	0.7880
TF-IDF	0.9263	0.8798	0.8393	0.8006	0.7950
RF	0.5952	0.6320	0.6503	0.6701	0.7596
RFC	0.8373	0.8189	0.7943	0.7713	0.7842
CA	0.8969	0.8569	0.8021	0.7577	0.7856
LGBM	0.5294	0.5770	0.6008	0.6263	0.7463

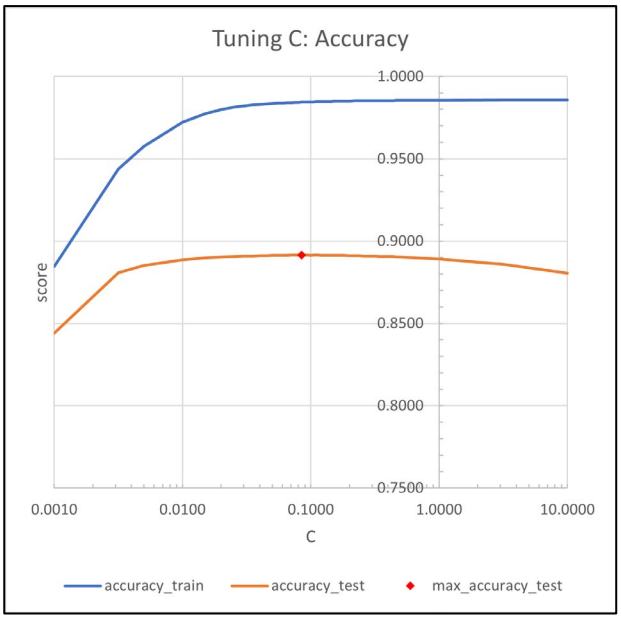
[9] https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html
[10] https://pyterrier.readthedocs.io/en/latest/experiments.html

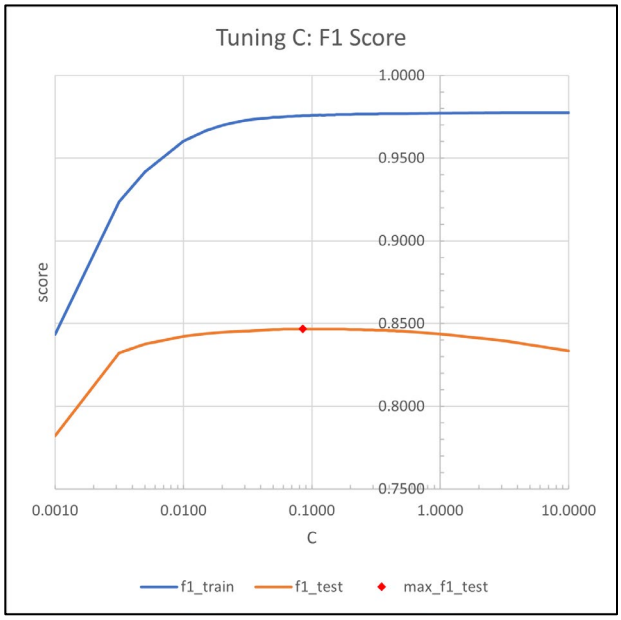
I also looked at feature importance in the model. It seems like verbs and adverbs are the most important POS features.

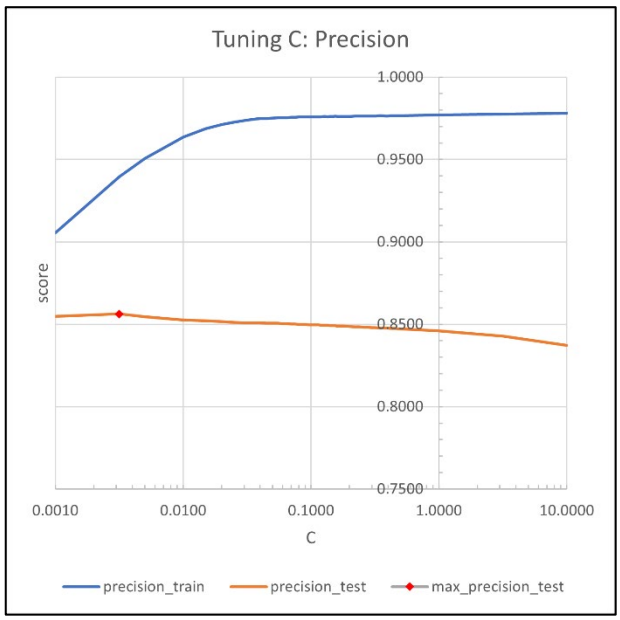


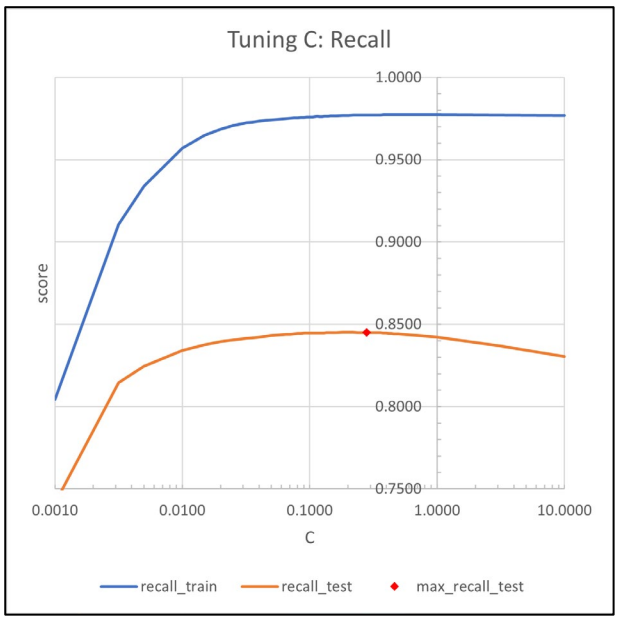
Classification model

I created graphs to visualize the tuning process with the axis denoting the C value on a log scale, with graphs for the macro accuracy, precision, recall, and F1 score across all emotion classes. I also denote where test scores reach a maximum using a red point. As you can see, training scores continue to increase and then plateau as C increase, while test scores reach a maximum. Accuracy and F1 score are both maximized where C=0.085. Thus, this is the model I selected.









The overall metrics of the best machine learning model (Linear SVC) and the dummy baseline models, including testing and training, are shown in the following tables.

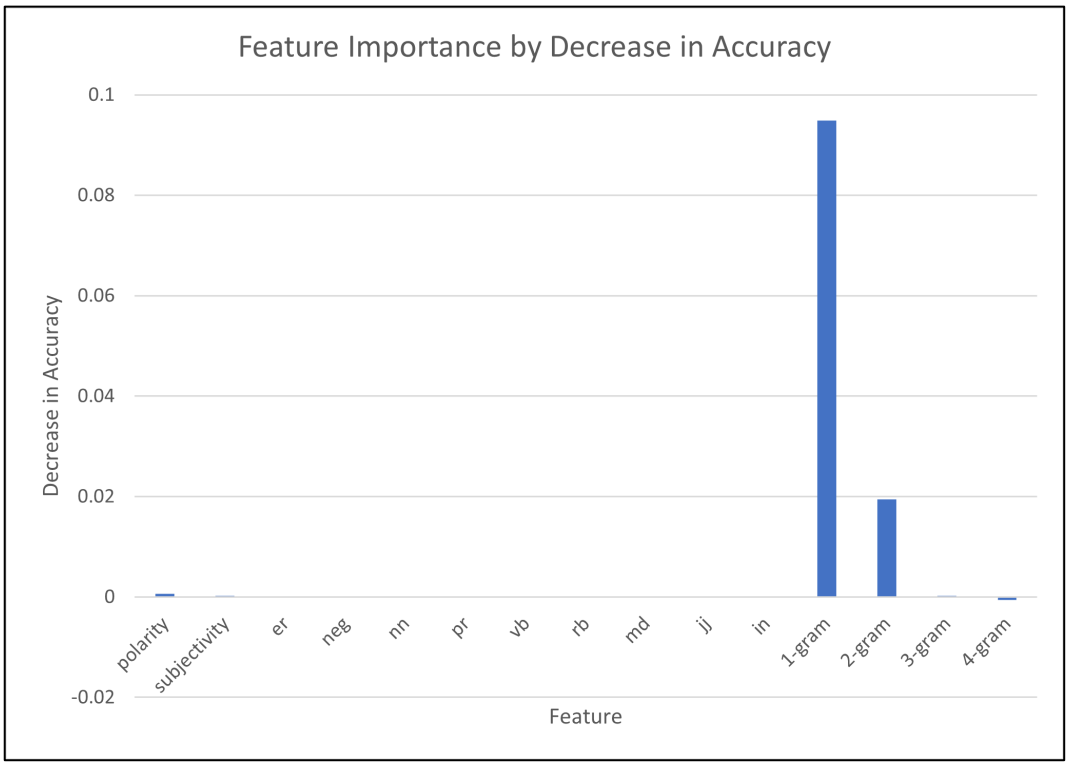
Testing Scores

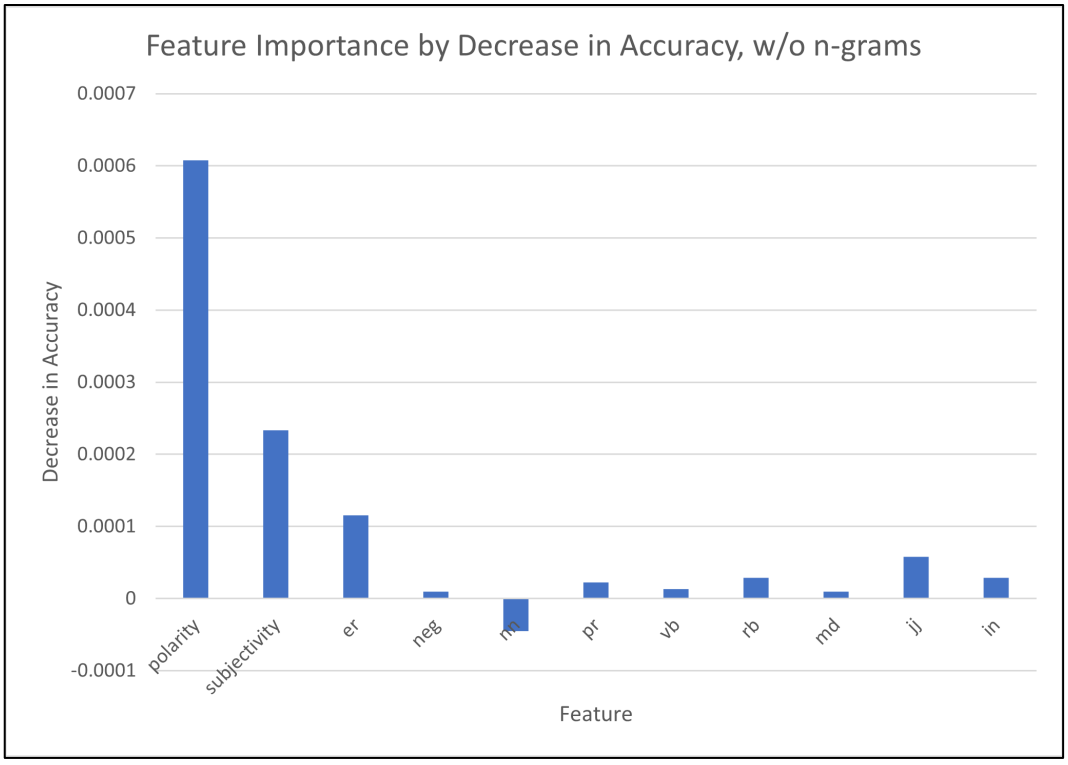
Model	Accuracy	Precision	Recall	F1 Score
Linear SVC	0.8915	0.8500	0.8445	0.8468
Most frequent class baseline	0.3379	0.0563	0.1667	0.0842
Random class baseline	0.1663	0.1664	0.1670	0.1486

Training Scores

Model	Accuracy	Precision	Recall	F1 Score
LinearSVC	0.9844	0.9758	0.9756	0.9757
Most frequent class baseline	0.3401	0.0567	0.1667	0.0846
Random class baseline	0.1653	0.1653	0.1654	0.1474

For my ablation analysis, I created the following graphs. The first one describes the decrease in accuracy by excluding each features, including n-grams, while the second one describes the same but excluding n-grams. Clearly, 1-grams and 2-grams are far more important for the model's performance. However, among the latter features, the sentiment features of polarity and subjectivity, and the emotion term percentage (er) are most important, while the POS tag percentages are the least important.





Discussion

In the paper "Extracting Emotion Causes Using Learning to Rank Methods From an Information Retrieval Perspective,"[11] the authors built a Chinese emotion classification model with only POS term percentages, and found that it performs comparably with their n-gram model. In my analysis, I did not find this to be the case for English emotion classification.

The classification model outperforms the baseline by quite a bit. I attribute this to inability to replace the information conveyed by specific words and phrases through the n-gram features.

Conclusion

In my project, I found that n-gram features, especially 1-grams and 1-grams, still are the most important features for emption classification.

Other Things I Tried

I first attempted to use a dataset of Twitter tweets[12], but my initial models had very poor out-of-sample prediction accuracy. I attribute this to the high percentage of incorrect spellings of words, which would require extensive spelling correction. Thus, I went with the preprocessed dataset as described previously.

[11] https://ieeexplore.ieee.org/abstract/document/8625499

[12] https://www.kaggle.com/datasets/042977506d4b87fe2ce6998514bd60df9ae2bdde98acf973acfd87e758e50d68

I first thought I can just use a learning-to-rank model, but the performance was lackluster, and I could not find a way to incorporate n-gram features into a PyTerrier pipeline, given that n-gram features greatly improved the classification model. Also,

What You Would Have Done Differently

With my chosen corpus, I could only assign the lowest and highest relevance scores (0 and 3) depending on whether the emotion matches or not. Thus, if I were to do the project differently, I would have done data annotation on my corpus, such as reassigning relevance scores within a range (1 through 3), which would have allowed for a more effective learning-to-rank model that may have removed the need for the separate classification model.