Natural Lyrics Processing Project Analysis

Aidan Stoner

September 27, 2025

Introduction

The Natural Lyrics Processing Project is a program written in Python that analyzes song datasets using natural language processing concepts. Using these concepts, the program creates language models of the dataset of songs to perform predictions. The goal of this project is to analyze the performance of a few language models when predicting the artist of a song.

You can visit the GitHub repository of this project here:

`https://github.com/AidanS39/music-nlp`

Datasets

This project currently uses the Song Lyrics Dataset, created by Deep Shah on Kaggle. `https://www.kaggle.com/datasets/deepshah16/song-lyrics-dataset` The Song Lyrics Dataset was parsed from `www.genius.com`. The dataset contains 6027 songs from 21 artists.

Models/Concepts used

Naive Bayes

Naive Bayes is a classification algorithm that is known for being simple yet effective. It uses Bayes Theorem to do classification tasks, and can be used for not just text classification tasks, but also any general classification task.

The project uses Bayes Theorem, which is the equation below:

$$P(classifier|document) = \frac{P(document|classifier) * P(document)}{P(classifier)}$$

N-grams

N-grams are used as a way to model language. N-grams work by taking a count of each N-sequence of words in a specific corpus and creating a probabilistic distribution of each N-sequence using those counts. Naive Bayes actually uses 1-grams, or unigrams, in its training process.

Using the Maximum Likelihood Estimate:

$$P(w_k|w_{k-n:k-1}) = \frac{C(w_{k-n:k})}{C(w_{k-n:k-1})}$$

Smoothing

Smoothing is needed when using n-grams with an n greater than 1 to ensure there are no zeros in our probabilities, as having any zeros will completely ruin our calculations for our models. For example, lets say there is a trigram (3-gram) that doesn't occur in our training set and therefore has a probability of zero. Now, if that same trigram is found in a test set, the likelihood of the entire sentence we are observing would be zero, which is not the intended behavior. The intended behavior is that every possible n-gram in our model should have a nonzero probability.

Laplace

Arguably the simplest form of smoothing is Laplace smoothing, also known as add-1 smoothing. This form of smoothing adds 1 to every n-gram count. This ensures that even if an n-gram never occurred in our training set, that n-gram will still have a nonzero probability and still possibly occur. This smoothing method is known to move too much probability from the more common n-grams to the unobserved/less common.

Using Laplace Smoothing:

$$P(w_k|w_{k-n:k-1}) = \frac{C(w_{k-n:k}) + 1}{C(w_{k-n:k-1}) + |V|}$$

where $|V|$ is the size of the vocabulary of the corpus.

(Stupid) Backoff

The idea with backoff smoothing is that we give unseen n-grams a proportion of their lower order n-gram's probability. In order to distribute this probability evenly and preserve a real distribution, the weight of the proportion of lower

probability has to be calculated. The "stupid" part of backoff is when we give up on trying to preserve the real distribution and just set the weight to a constant factor. This method actually came from a team at Google, and was surprisingly effective. The reason I used stupid backoff in this project is because of the current lack of data. In order to preserve a real distribution, we need held out data to estimate the right weights, which would further cut out more data from our training set.

Stupid Backoff follows this equation:

$$P(w_k|w_{k-n:k-1}) = \begin{cases} \frac{C(w_{k-n:k})+1}{C(w_{k-n:k-1})} & if C(w_{1:n}) > 0 \\ 0.4 * P(w_k|w_{k-n+1:k-1}) & otherwise \end{cases}$$

Dense vs. Sparse Models

When developing a language model using n-grams, a common decision is what and how to store each n-gram. There are two general ways one could go when it comes to what to store, and that is by creating either dense or sparse models. A dense model stores all probabilities for every possible n-gram, no matter if the n-gram hasn't occurred in the training set. This can be a good choice if using n-grams with a smaller size n, and/or if there is a lot of compute and memory available for use. A sparse model on the other hand only stores probabilities occurring n-grams from the training set. Any non-occurring n-grams are not stored, and if the probability is needed for them, it is computed on the fly. This choice is suitable for workloads with higher n-gram counts, and/or if there is not a lot of compute and memory available for use. Because this particular project needs to simultaneously create multiple models for each artist, and the project allows for larger size n-grams, I decided to use sparse models. The amount of memory and compute needed to store every possible n-gram for every artist would get far too large when approaching a higher n.

Experiment

The project exposes the dataset to two different types of models, one I would call a "pure" Naive Bayes model, and another that I would call an "n-grams modified" Naive Bayes model. When applying Naive Bayes to model language, the modeling process uses something called a *bag of words*, which is a set of counts for every unique word in all of a classifier's documents. As an example, if an artist had only one song with the lyrics "the cat ran from the dog", a bag of words would look like this:

$$the : 2, cat : 1, ran : 1, from : 1, dog : 1$$

To clarify, a bag of words retrieves the counts of every word from all songs of a particular artist, so each artist will have its own bag of words that reflects

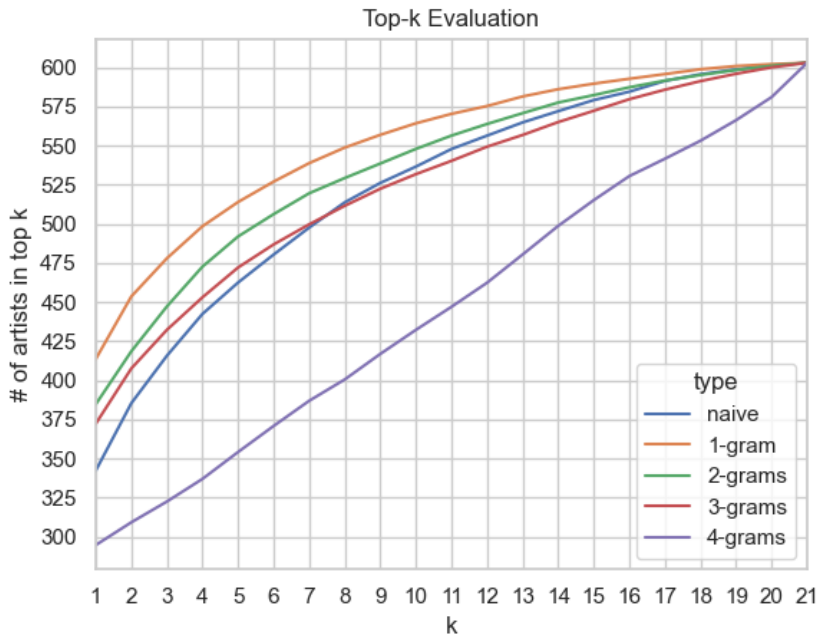
their own song lyrics.

One observation can be made about bag of words, it is just a unigram count. The pure Naive Bayes model uses these unigrams counts. The n-grams modified model instead uses n-grams, but in the same way that the pure Naive Bayes version uses unigrams.

As stated previously, any n-grams higher than 1 need to be smoothed. The n-grams modified model has two cases of smoothing, the first being stupid backoff. This backoff is applied normally to any non-occuring n-gram in the test set. If backing off from a non-occuring unigram in the test set, that n-gram is given a baseline minimum probability that is calculated using the laplace smoothing method. This was applied as an attempt to offset the large amount of unseen words in the test set. As a disclaimer, this decision is not backed by any proven or known methods, it can be thought of more as a hack.

The project analyzes the performance of the pure Naive Bayes model and the four n-grams modified Naive Bayes models, one using 1-gram, one using 2-grams, one using 3-grams, and one using 4-grams. These five models were evaluated using a method called *top-k evaluation*. Top-k evaluation analyzes every test document and counts how many documents are in the top k artists given by the respective model. This is done for every rank position k. Since the dataset used for this project has 21 artists, top-k evaluation will be done for all 21 rank positions. 100 evaluation tests were performed, with each model training and testing on the same data. For each of the 100 tests, a random split was computed, with 90 percent for training and 10 percent for testing.

Results and Analysis



The plot above shows the averages of the results of the 100 top-k evaluation tests for all five models described above.

The 1-gram model performed the best, slightly outperforming the 2-gram, 3-gram, and pure naive model. The 3-gram model performed slightly better than the pure naive model for higher ranked k's (1 to 7), while the pure Naive Bayes started to perform slightly better for lower ranked k's (7 to 21).

The 4-gram model unsurprisingly performed the worst by a significant amount, correctly predicting the correct artist for just under half of the test songs. The 4-gram model also seems to follow more of a linear model, unlike the other models which follow a quadratic model.

What is particularly surprising is that the 2-gram and 3-gram models actually seem to perform better than the pure Naive Bayes model for higher ranked k's. This leads me to believe that the added context of higher order n-grams has a possibility of increasing the performance of the Naive Bayes model. This could also be due to other factors, which is addressed later.

Reflection

There are some factors that should be considered when reflecting on the results and conclusions I made about the experiment and project as a whole. The first one is that the dataset itself is relatively tiny for language modeling purposes.

As stated before, the dataset contains only 6027 songs from 21 artists, which really is not sufficient to be creating a language model that can predict with high accuracy.

This dataset also has very high variability with the amount of songs per artist. For example, Taylor Swift has 479 songs in the dataset, while Khalid only has 64 songs. This matters because the language models tend to correctly predict artists with more songs in the dataset much more than artists with less songs in the dataset. This can be seen directly within the rankings that the language models create, which if interested can be tested in the application itself. This high variability definitely had a negative impact on the performance of the model.

This is not only a shortcoming of this project specifically, but also a general limitation of classifying by artist. An artist can only create so many songs, so there will always be a hard limit to the amount of data you can use from a specific artist. This is different from classifying on an attribute like genre, as artists will always be creating music for (almost) all genres, and a dataset has the ability to have a lot less variability in the amount of songs per genre. Selecting a good classifier matters.

Another factor that should be kept in mind is that the smoothing methods used are not statistically backed. Using both stupid backoff and laplace smoothing does not generally lead to good results. The reason I used them is because of the lack of data. If the dataset was a fair amount larger, held out datasets could preserve a proper statistical distribution and ultimately improve the model's performance. The models that have the potential to improve the most from a more sophisticated smoothing method are the higher n-gram models.

Differing methods of handling unseen words in the dataset could have also had an impact on the performance of each individual model. For the Naive Bayes model, unseen words were simply ignored. For the n-gram models, unseen words were given a minimum probability using laplace smoothing. I believe that is a really important factor in why the 1, 2, and 3-grams models outperformed the pure Naive Bayes model.

I think higher order n-grams in this experiment only hurt the respective model's performance, but not because of the n-grams themselves. Given the circumstances of a small dataset and a less sophisticated smoothing method, higher order n-grams just aren't the right fit when considering the performance of a model. I still do think that higher order n-grams do have the potential to improve the Naive Bayes method of language modeling.