Got Grammys?

Will Harris, Keaton Hubbard, and Clinton Washington

Introduction

"And the record of the year goes too……" Each year, millions of viewers await to see what song will win the record of the year at the Grammys, and each year there are those who are always shocked that certain songs won grammys and those who are shocked that some weren't even nominated. For our final project, we decided to create a dataset consisting of songs that have not been nominated, songs that have been nominated, and songs that won record of the year from the years 1958-2021.

Our original goal was to build a model that could predict the songs that would win Record of the Year. This idea led us on a journey that presented us with many challenges and reconsiderations along the way. These challenges included gathering the data and getting it all in one place, dealing with issues in our data that were naturally present, finding the optimal form of the data, working with different algorithms to find the best fit for our data, and dealing with unexpected results.

Gathering the Data and Creating the Dataset

Website

https://totalmusicawards.com/grammy-awards/record-of-the-year-winners-nominees-archive/

We were lucky enough to find the Record of the Year winners and nominees all in one place. This list went back all the way to the first Grammy awards in 1958 and contained every single year after, including the most recent in 2021. The downside of this website is that the listings were not already in a table form that could easily be exported. So, in order to get our hands on this data we had to employ various web-scraping methods. This involved using Java to code a program that took the desired text from the Html code, organized that text in a table format, and exported an excel file with that data. The data extracted included song title, artist name, grammy award year, and whether or not the song won or was nominated for Record of the Year (class attribute). This dataset has 325 instances of songs.

Spotify API

https://developer.spotify.com/console/get-audio-features-track/?id=06AKEBrKUckW0KREUWRnvT

Spotify offers a wonderful API with many great features. We took advantage of the data they have stored about various attributes of individual songs. They call this data “Audio Features.” These features include Danceability, Energy, Acouticness, Speechiness, and many compelling pieces of information about songs (these attributes will be described in detail a little further down the page). We found a data set online created by Sean Miller through a website called data.world that has 28,000 instances of songs with all of the Spotify API data. Here is a link to that page: https://data.world/kcmillersean/billboard-hot-100-1958-2017.
This table has every song that has ever been on the Billboard weekly top 100 list from 1958 until 2019. Unfortunately there were many songs that appeared multiple times in the dataset, so we had to remove these entries. That only diminished the size of the table to over 23,000 instances, so we still had a lot to work with. Because of its scope in terms of years and also in terms of its attributes, this was the perfect dataset for us to use in conjunction with the table we got from web-scraping.

Overlaying the Data

With our two datasets fully complete, we needed to combine them to create the final dataset. This was done by joining the tables with a common attribute we called “SongID” which consisted of a string of the song title followed by the artist’s name. We could not simply join by the song name because there are multiple songs with the same name. For example, there are 6 songs with the title “Dance With Me”. Also we could not join by artist name because again many artists are in the table multiple times. This is why the “songid” attribute was essential in the process of overlaying. Microsoft Excel has a function that can overlay two tables with the exact attributes that are desired. The final list of attributes is listed below with explanations for each one.

Attributes	**Description**
Song	String, title of the work
performer	String, name of the performer credited on the song
danceability	Numeric, describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
energy	Numeric, a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

key	Nominal, The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, etc.
loudness	Numeric, The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Values typically range between -25 and 0 db in our dataset.
mode	Binary, indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
speechiness	Numeric, detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
acousticness	Numeric, a confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
instrumentalness	Numeric, predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	Numeric, detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
valence	Numeric, a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
tempo	Numeric, the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
Time signature	Nominal, an estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).

year	Nominal, the year in which the song was released
Grammy Fate	Nominal (Class Attribute), the results of a song in relation to the grammys. Values: Won, Nominated, Not-Nominated

The Finishing Touches

The final step in our process to complete the final dataset came somewhat unexpectedly. When we attempted the overlay, some of the data was not able to crossover. This was because the “SongID” for certain songs from the web-scraped data did not have a matching partner in the Spotify dataset. The biggest causes of this were formatting differences between the two datasets. These differences included a slightly different character for apostrophes and slight differences in language such as using the word “featuring” instead of “with”. Once all of these formatting issues were resolved, there were still a few songs that had not been transferred over. We assumed that every Record of the Year nominated song would have been on the Billboard top 100 list at some point, but that was not the case. Due to this issue, we had to manually enter in about 40 songs with all of the appropriate attributes found using the Spotify API Console. After we completed that step, our dataset was ready to be used.

Data Preparation

Splitting up the Dataset: Holdout Method

In testing our data, we used the holdout method because we had so much data to work with. Instead of just using one set of holdout data, we removed 5 years from our data set (2014-2018). As a result, we had 5 test datasets for the 5 years we removed, and 5 training datasets that did one of the mentioned years removed. We chose more recent years due to the data being more relevant and up to date. We also had more songs to experiment with for the later years as well.

Attributes

After the data set was successfully split into various training sets and corresponding test sets, some major work needed to be done to clear up the attributes. Some attributes were easily found to not be predictive and others needed some strategic work to be able to maximize their predictive potential.

Deleting Attributes

In some cases specific attributes served our purposes better if they were left out of the dataset. One type of these attributes are what we like to call “Identifier Attributes.” The values of these attributes serve no other purpose than to tell us what the instance is by using a unique identifier. The two attributes in our dataset that fell under this category were “song” and “performer”. Even though these two attributes served a pivotal role in the overlaying process, from a predictive standpoint they are simply just noise. If they were left in the dataset, certain algorithms would either use them to extremely overfit the data or have a hard time processing the sheer amount of useless values. PRISM for example would think there is a correlation between the name of a song and whether or not it would be nominated for a grammy and it could possibly create a rule for every single one of these names. Therefore the “song” and “performer” attributes were promptly deleted from the table.

Other attributes that needed to be deleted were ones whose values were all in the same small area. In other words, almost every instance practically has the same value for these attributes. The two examples from our dataset would are “Instrumentalness” and “time signature”. The overwhelming majority of the most popular songs from 1958 until the present day have vocals and are in 4/4 time. For all the songs where these claims are true, “instrumentalness” would have a value that is practically 0 and “time signature” would have a value of 4. Because almost every song had these values, these attributes could not provide any distinction between class attributes. For these reasons these two attributes were also deleted.

Fixing Attributes

Attribute Types

Some attributes at first appeared to be one type of value but turned out to better suit another type. For instance, both “key”, “mode”, and “year” have numbers as values, so one would first assume that their type should be numeric, also known as continuous. However upon a closer look, casting these attributes as numeric would actually not be the best option. All three of these attributes use numbers as a category and not as a continuous measurement of some value. This suggests that they should in fact not be numeric and should be something else. Since, “mode” only has two values to denote major and minor, this needs to be cast as binary. “Key” only has twelve distinct categories, so this attribute needed to be changed to nominal instead. Conveniently, Weka has two simple filters aptly named “NumericToBinary” and “NumericToNominal” that do exactly what we want to do to these two attributes.

The third attribute, “year”, required a little bit more complex of a change. If we were to simply recast this attribute as nominal, we run into a lesser version of the problem we had with the “song” and “attribute” values. There would be 63 different categories that would almost certainly lead to overfitting of the data. In fact when the dataset was run in the PRISM algorithm with this formation of the “year” attribute, almost every single rule had individual years as the primary antecedent clause. And on top of that there were multiple rules generated for every year paired with various other attributes. Having this amount of extremely specific rules is the definition of overfitting. So, a middle ground had to be taken in the form of discretization. Instead

of having 63 distinct categories, we could put years in larger groups. Since 63 is evenly divided by 7 and 9, we thought we could perform an equal width discretization with 7 bins of size 9. This also approximates the idea of a decade, which is a common way to group songs in our society. With this new formation of the “year” attribute we no longer run the risk of overfitting, and we can still use its predictive potential.

Accounting for Outliers

On many of the numeric attributes were a few instances on the fringes of the range. These are known as outliers because they stray far from the common distribution of the values. These outliers came from either errors/missing values in the original dataset or just songs that defied the norms.

Overall, the dataset was very clean without many missing values or errors, however there were a few errors that we caught and had to fix. For instance, a song entitled "Hello, Dolly" by Louis Armstrong had zeros in the dataset as values for tempo, time signature, and valance. It is not possible for a song to have a tempo of zero or a time of zero, so this is clearly an error. To fix this issue, the instance was just deleted. We have 23,000 other songs so that one song will not be missed. If errors and missing values were rampant throughout the dataset we would have had to resort to other methods.

Even though the rest of the outliers were not caused by errors, they can still skew the results of our testing. Therefore something needs to be done about these values. Again, the easiest way was to remove them. This can be done on numeric attributes by setting a split point to chop the data and remove one of the two pieces. Weka provides great tools for this action. By looking at the histograms weka provides, one can easily find the outliers on the edges of the ranges. Using these histograms, we can find a reasonable split point that would cut out the outliers while still keeping the predictive instances. Using a Weka filter named “RemoveWithValues” these outliers were removed from the dataset. The specific values and setting used see Appendix 1.1.

After all of the outliers were removed, we could feel much more confident in the predictive quality of our data. The absence of these values is imperative to a few of the steps we took later in the process like oversampling and the nearest neighbor testing algorithm. If the outliers were still present, these two methods could take them into account and skew our results towards the values of the outliers.

Making Every Instance Nominal for Special Circumstances

For certain testing methods, every attribute has to be in a nominal form. Just like how we transformed “year” into a nominal attribute using discretization, we can discretize every other numeric attribute to make them nominal. Equal width binning was also used on these attributes. If equal frequency binning was used, certain attributes like speechinees would have extremely small bins. Having extremely small bins can lead to predictions that are overly specific to the training data and therefore causes overfitting. This version of the dataset was used when necessary in testing, but we also used it on other algorithms to see if it produced better results.

Instances

The largest problem with the nature of our dataset is that there are overwhelmingly more not-nominated songs than nominated and winning songs. A ratio of 23314:263:63 respectively. This is an extreme case of imbalance within a dataset which would cause our results to overly lean towards predicting not-nominated. There was no way around this issue in our data collecting phase as there have only been 63 record of year winners and 263 nominees. There are way more songs that have not been nominated for record of the year than songs that have, and our data set reflects that. However, in order to get predictive results this issue needed a major fix. This fix came in the form of sampling the majority class and oversampling the minority classes.

Sampling

Sampling is the process of only taking a proportion (a sample) of your instances. This can be targeted towards specific classes, so in our case we wanted to target the majority class of “not-nominated”. The instances included in the sample are chosen randomly to ensure that the distribution stays roughly the same to how it was at the start. Using Weka’s resample filter, the sampling process was applied with a bias toward the majority class. This only kept our specified percentage of instances and removed all of the rest. We decided to keep a fairly small percentage of data, but still enough to gain quality information from, in order to limit the amount of oversampling we would have to do to balance the classes. We determined that it would be better to sample more than oversample, which is the process described next.

Oversampling

Oversampling does the exact opposite of sampling. Instead of removing instances, it artificially creates them. This was especially helpful for our minority classes. We mentioned before that there are only 63 possible instances of songs that won record of the year. However, now with oversampling we can fix that issue. We oversampled our data using a process similar to the popular nearest neighbor algorithm. It surveys the minority class and based on variations of the distance formula, creates likely values for a new instance. In this way we can have more than 63 data points of winning songs. However, if too much oversampling is done, it can diminish the reality of the dataset by having too many fake instances. This is why we decided to sample more than we oversampled.

Having Only Two Classes

By using a combination of both of these techniques we were successfully able to balance out each of the three classes. However, we soon realised that having to balance out three severely unbalanced classes can cause accuracy issues for building models. The balanced data seems to be too far removed from the reality of the data. So, our new plan was to group the “won” and “nominated” classes together. Afterall, all of the songs that won were nominated in the first place. This still left us with an unbalanced data set but to a lesser degree. The smallest minority class now has 326 instances in comparison to 63. So for our final

balancing act we sampled the majority class down to 689 instances. This is roughly double the amount of the minority class which allowed us to only have to oversample the minority class by 100%. The decision to move from three to two classes gave us two major benefits: 1. It is a much simpler and true to reality process to balance the data and 2. Testing became much simpler as well by only having to predict two classes.

Data Analysis

When running our Data analysis, it was very evident that we were going to face challenges with producing predictive results with an unbalanced class attribute. Once we combined nominated songs with songs that were won, our class attribute became a lot more balanced. In theory, we presumed that this change would allow us to predict more instances correctly while creating more models to build from. Afterall, with only two classes, we had more chances to get something right. These ideas caused us to use the following algorithms as testing methods on our test data:

IBk

IBk, also known as Nearest-neighbor, is used to compare instances that share similar values to determine the class attribute. This is done by using the euclidean distance formula that works with numerical and nominal values. For numeric values that are on different scales, it allows for normalization of the values. With most of our values being numeric, we thought that this test would be an appropriate algorithm due to our attribute values being on different scales. The test on the training data proved to be quite accurate, but once we implemented the test data we ended up with many false positives. We even used a larger bucket size, which compares new instances with more instances within the data to determine an attribute value, with our large dataset.

OneR

OneR is a simple rule-building algorithm that allows for the use of only one predicting attribute. It finds the one attribute that most accurately predicts the class attribute, and uses that to build its rules for each of the attribute's different values. Because of this, OneR does best with nominal attributes with a relatively small number of different values while still being more than that of the class attribute. We thought that using OneR might give us an idea of whether or not there was one attribute that stood out as being especially predictive.

Naive Bayes

Naive Bayes is a statistical algorithm that assumes that each of the dataset's attributes are all equally predictive. It calculates the probability of classifications for each of the class values given the instances and their classifications that are in the dataset, and then chooses the class value that has the highest probability. Naive Bayes is especially useful when a dataset contains lots of numeric data thats predictiveness is relatively unknown. Because it accounts for every attribute equally, it can give a decent indication of the predictability of your dataset as a whole. We believed that Naive Bayes would be appropriate for our dataset because the majority of our attributes are numeric, and we did not have an idea of certain attributes that would be more predictive than others off the bat.

J48

J48 is a standard tree building algorithm that takes instances and runs their values through a set of rules. These rules are created by using 1R for each attribute to determine overall accuracy. After doing this for each rule, the algorithm calculates a goodness score for each attribute and chooses the highest one as the rules for the trees. Due to some of our attributes being nominal after descritized, we thought this attribute would do well since we had a mix of numeric and nominal attributes and we only had two class attribute values.

PRISM

The PRISM algorithm attempts to cover every instance by formulating rules that apply to their unique features. This seemed like it would be a good fit for our dataset because by assuming that nominated songs can be predicted we assume that there are differences that set these songs apart from the others. Therefore, the nominated songs would likely have common rules that would apply to them and only them. Additionally, on the opposite side, the not-nominated songs would also have unique attributes that would determine their so-called grammy fate. By attempting to cover the dataset and starting large, we believed that we could quickly find the most important commonalities which would lead to very predictive results.

Cost Sensitive

Due to the unbalanced nature of the dataset, cost sensitive classification seemed like a logical choice. This algorithm lets the user add a cost value to predicting instances incorrectly. If it costs more to predict one class wrong, then the algorithm will be less likely to predict that outcome. Cost sensitive classification can be paired with all of the

algorithms we had used thus far, so we hoped that adding costs would improve on their results. We thought of two ways that we could utilize this method: diminishing the amount of false positives and diminishing the amount of false negatives. With certain algorithms many not-nominated songs will end up being classified as nominated. This is called a false positive. By adding an extra cost to predicting this result, the amount of these false positives should decrease. Other algorithms will predict a lot of nominated songs as not-nominated. Intuitively, this is called a false negative. Just like with false positives adding a cost to classifying nominees as not-nominees would diminish the amount of false negatives.

Results

After running our data analysis, it was very evident that there would not be any way to produce results that would predict which songs would be nominated for Record of the Year. All of our efforts in testing had utterly failed. Even though our class attribute was a lot more balanced, we realized that the attribute values for each class were similarly distributed for each class. As a result, all of the algorithms were skewed towards non-nominated since it was the class attribute value that had the most instances in our test data. Each algorithm had unique ways to disappoint us.

IBk

Nearest neighbor proved to be ineffective as many of the class attributes values are similar and are distributed in the same manner. Also, the use of oversampling also does not fix the attribute value distribution issue. Nearest neighbor would also confuse the non-nominated songs as nominated creating a numerous amount of false positives. Even for the few songs we did classify correctly that were nominated, it's highly possible that it was by chance due to us adding the additional grammy winning songs to the nominated section.

OneR

OneR did not work well with our dataset. Likely due to our many numeric attributes with wide ranges of values, OneR chose 'song name' as its one rule to classify by, so it created a rule for every song in the dataset. Because each of our dataset's two nominal attributes (song and artist) were unique to the instance, OneR was not a good fit for our data.

Naive Bayes

Naive Bayes yielded an impressive 94.9366% accuracy, but upon a closer look, we realized the real reason was not the data's predictability. One look at the model's confusion matrix told us that Naive Bayes had classified, whether correctly or not, 96.23% of the test instances as 'not-nominated,' which made up 98.5817% of our dataset. This meant that Naive Bayes, in this case, was virtually acting like ZeroR, which would classify 100% of the test instances as the majority class value (not-nominated) and be correct exactly 98.5817% of the time.

However, this wasn't the end of Naive Bayes in our project. Due to some disappointing results with predicting the results of the Grammy's, we decided to branch off of our main goal to answer one nagging question: Could our data be useful for anything at all? We answer this question in a later section.

J48

J48 produced one leaf before we discretized the year attribute, and that was non-nominated. After we discretized the year attribute, J48 produced a tree attribute that used all of the attributes. As a result, we got very high results. Although, it was not very accurate because it was an extreme case of overfitting.

PRISM

What we thought would be beneficial about this algorithm turned out to be its downfall. We believed that it would form rules based on patterns seen in the data, which it did, but it produced hundreds of rules that all predicted instances as nominated. This tells us that there were not any attribute values or sets of attribute values that were unique to one class. We attempted this algorithm on the 2018 training and test set pair by discretizing all of the numeric attributes. PRISM showed such a lack of predictivity on this set that we decided not to discretize the rest of the sets as it would give much of a benefit.

Cost Sensitive

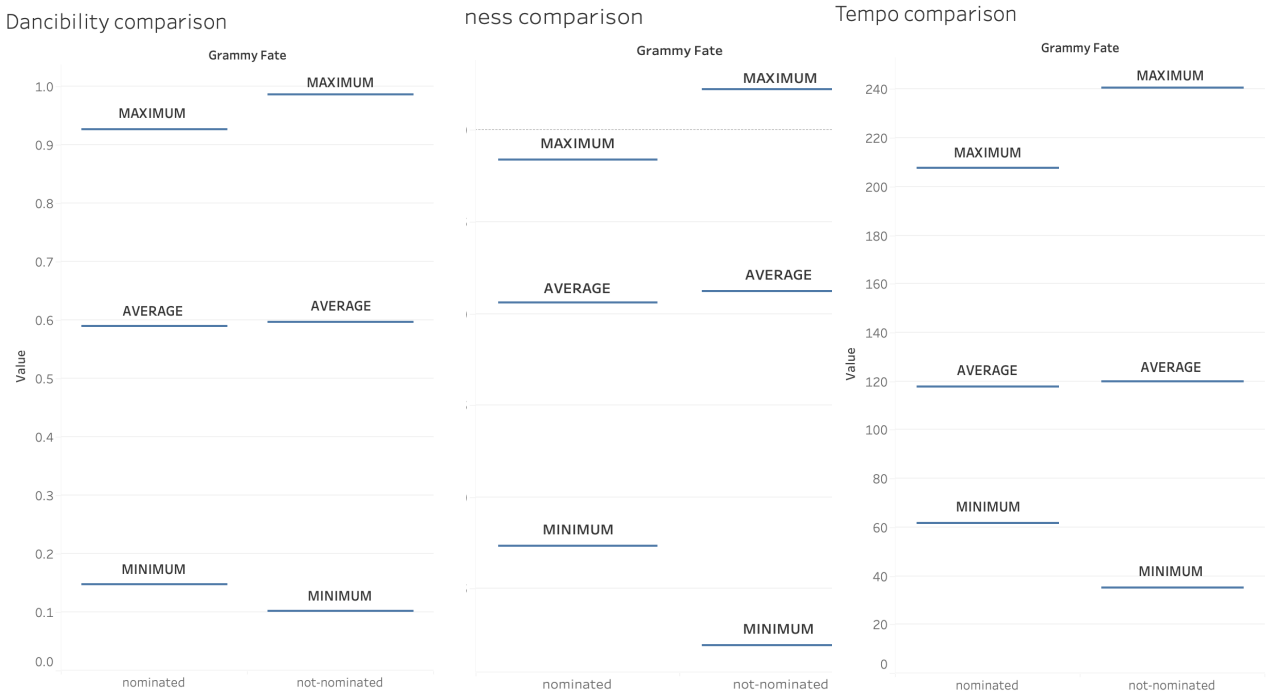
This method was successful in its goal in incentivizing predicting certain classes, but this led to too many of the other class being falsely predicted. For instance if we placed a cost of predicting the false negative case (nominated songs predicted as not-nominated) too many false positive predictions occurred (not-nominated songs predicted as nominated). The inverse of that was also true. One way to picture this process is that whenever the cost caused a nominated song to be predicted correctly, around 50 more not-nominated songs would be falsely predicted as nominated. So even

though cost sensitive learning was able to predict more nominated songs correctly, it predicted far too many false positives for any of the algorithms to be successful.

Using Visualizations to Explain our Findings

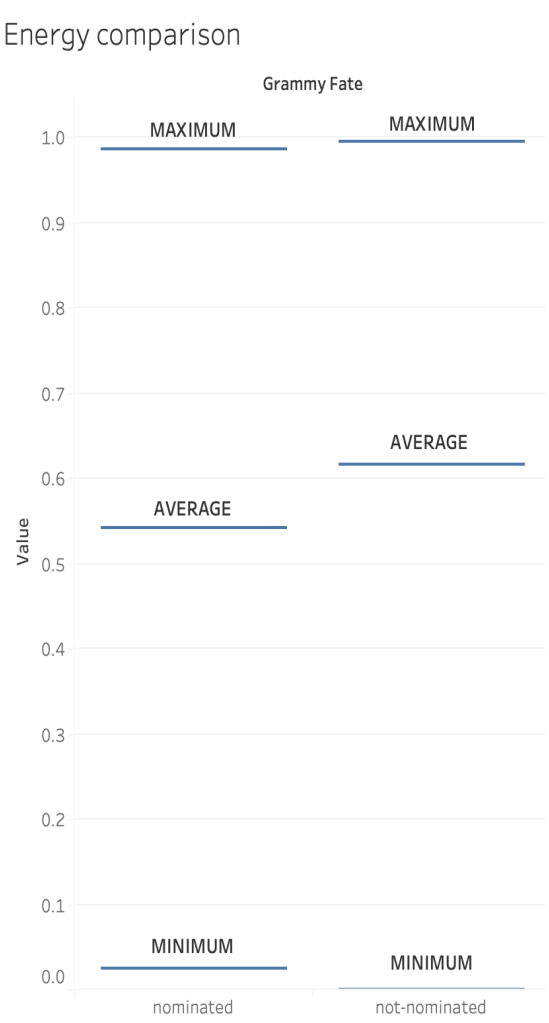
After running our tests, it was evident that there was not enough of a difference between the classes and their attributes. By using visualizations, we were able to showcase the similar distribution of these attributes between the different class attribute values.

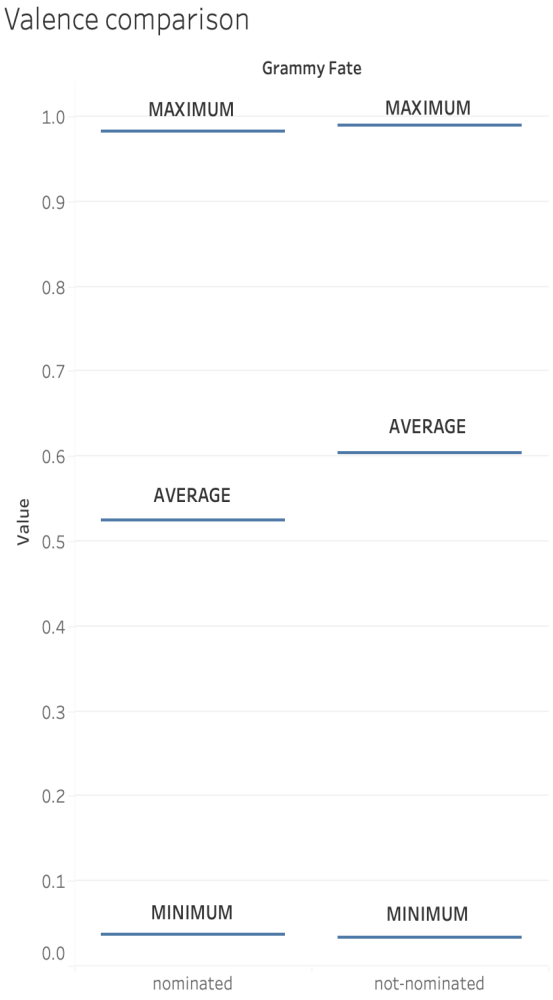
Danceability, Tempo, and Loudness



By looking at the distribution of values among these three attributes, we can see that the averages are very close to the same value. This shows us that for these attributes there is not an overwhelming difference between the classes. However the “nominated” and “not-nominated” distributions are not the exact same. The biggest difference is that the nominated songs stay out of the extreme ranges of these values. This actually informs us of an important finding. Grammy nominated songs do not tend to break boundaries, but instead stay within the norms. At least in respect to all of the attributes we have.

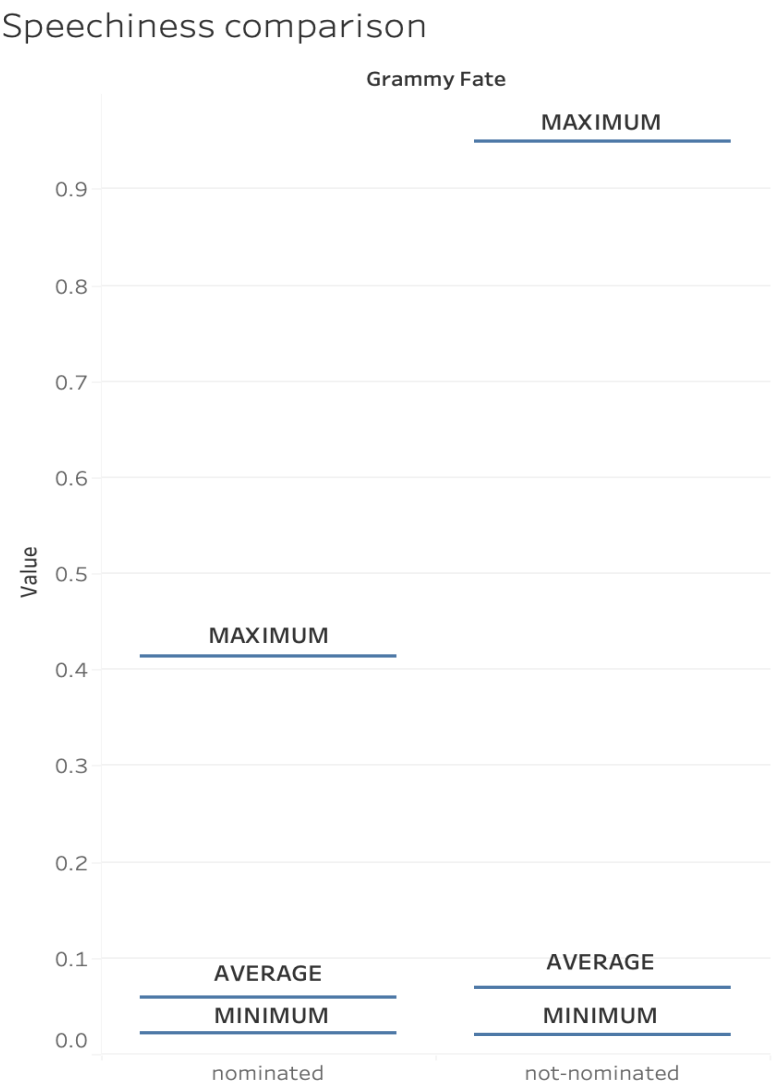
Energy and Valance





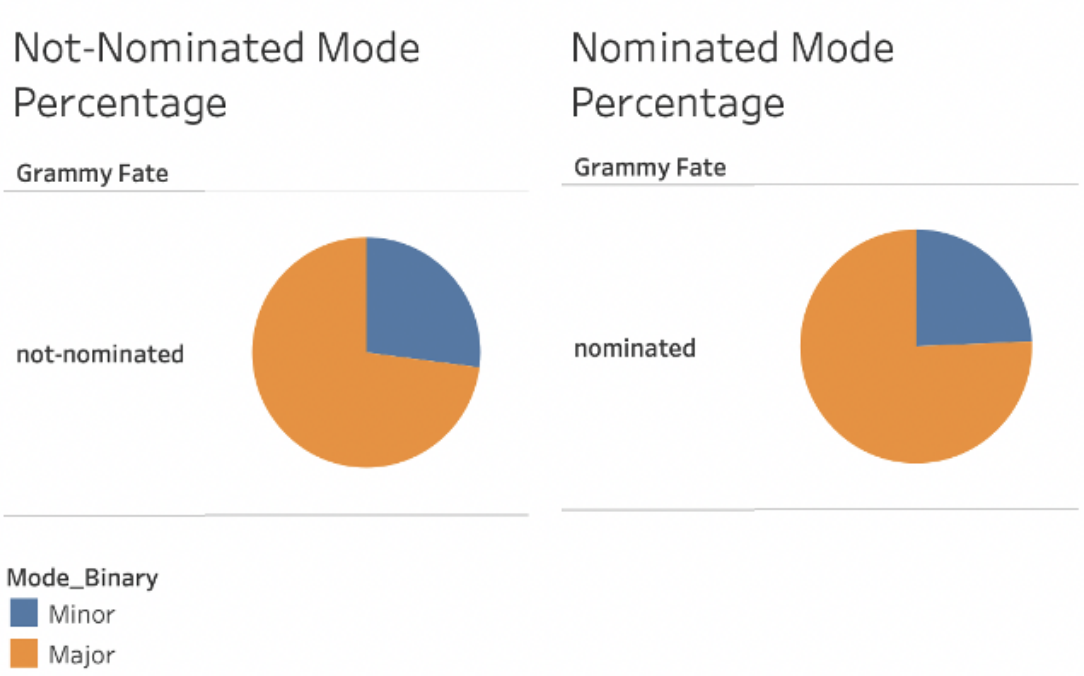
These two graphs look almost the exact same as the previous three, except the averages do have more of a noticeable difference. However, our testing has indicated that this slight difference was still not enough to provide any predictivity. The concept of nominated songs not reaching the extremes is not as prevalent as before but it can still be seen.

Speechiness



Speechiness was nearly the same for the minimum and the average. However, the maximums were drastically different. This is potentially due to outliers within the non-nominated section. Although, it is evident that songs that are popular generally have spoken words. The maximum for the nominated songs is slightly above .4 showing that it's not very common for billboard songs to have high speech content.

Mode



Pie charts, while not effective in many things, are especially effective in comparing dueling percentages. Although the percentage of major and minor modes are not exactly the same for the nominated and not-nominated they are very close. This tells us that “mode”, just like the attributes mentioned before, does not supply a large enough difference to make accurate predictions.

Key

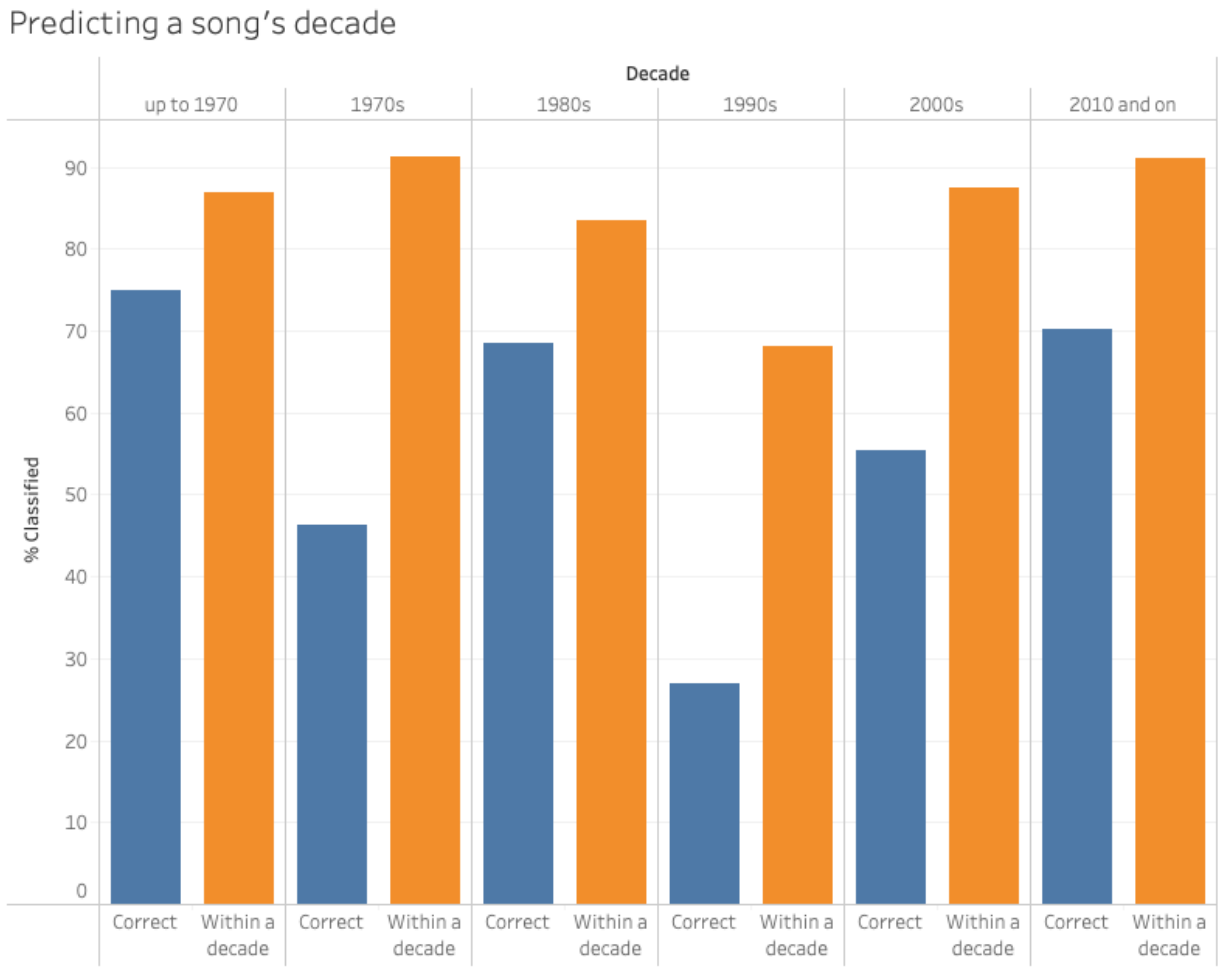
The pattern we have seen in the distribution of values stays consistent for this attribute too. There are very minimal differences between the two classes. The most interesting feature to notice is that keys of 5 and 7 (or better known as the keys of F and G) are the most common among the nominated instances while key 0 (the key of C) is the most common among the not-nominated instances. However, the key of 0 (C) is still common within the nominated songs. These observations are very intriguing, but because the differences are fairly slight, key does not provide much predictive value. Our testing proved this claim.

Reasoning with our Results

These visualizations exemplify exactly why our algorithms had trouble distinguishing between our two classes; there is no attribute that can show any meaningful difference. The slight differences shown between classes in a few of our attributes are not nearly enough to aid in predicting which songs were nominated for record of the year. Even though we cannot accurately predict nominated songs, we can confidently say that in order to have a nominated song, it is extremely helpful to fit within the norms and not reach the extremes.

Attempting to Make Our Dataset Useful

We wondered if predicting the decade of a song would be more possible with our current data. So, we temporarily removed Grammy fate and made Year our class attribute. Then we used unsupervised discretization to convert the numeric value Year into the nominal value Decade, which was grouped into 6 values (up to 1970, 1970s, 1980s, 1990s, 2000s, and 2010 & beyond). Finally, we ran it through Naive Bayes, and the results were surprisingly predictive.



In this bar chart we can see just how predictive our data is of the song's decade in comparison to our Grammy results. What makes these results additionally impressive is the fact that we have better accuracy percentages with 6 class values, as opposed to the 2 class values of our Grammy predictions. Naive Bayes was able to predict a song's decade correctly more than half the time (58.634% to be exact). Since we now had 6 different class values, we knew that this could not be luck. The data was on to something, so we followed it further. Using the model's confusion matrix, we calculated

the percentage of accuracy for each of the individual decades. These percentages are represented by the blue bars in the graph above. We also noticed that, of the incorrectly classified counts on the confusion matrix, numbers got smaller as they wandered away from the treasured diagonal line of correctly classified instances. Of course this is to be expected when you have predictive data and ordinal class values, but after some rough going with the Grammy predictions, we weren't expecting much. So when we were pleasantly surprised by the seemingly predictive nature of the confusion matrix, we decided to give a bit of grace to our model and also calculate the number of instances that were correctly classified or classified within one decade of the correct one (one to the left or one to the right). These values are represented by the orange bars.

The combination of these two calculations made for some fun discoveries. For example, the 1970s and 1990s show great disparages between these blue and orange percentages. This means that, according to our data, songs released in the 1970s and 1990s are less unique to their own decades, but are much more unique to their neighboring decades, or their entire "quarter century" if you will. In fact, the 1970s have the second-lowest blue value (% of instances classified in exactly the correct decade), but have the highest orange value (% of instances classified in the correct decade or within one decade).

Overall, it was fun to see our beloved data bear some fruit when it came to predictiveness, even if it wasn't exactly in the way that we envisioned from the start of our project.

Conclusion

In the end, we found that Grammy nominations and wins are more subjective than they are quantifiable. The ever-changing standards of a culture's taste in music cannot simply be tracked by levels of danceability or acousticness. There is something more to the handing out of Grammy's than just the song's loudness or key. Perhaps discovering these cultural trends would require much more data. Adding attributes about how the song was received in its broader cultural context rather than just numeric data about the mechanics of the song itself may help solve that issue. But this type of data is not quite as quantifiable or available. Retrieving it may require finding out how many times each song is mentioned in a news article, for example. Or, for newer songs, this could mean getting their count of mentions on social media platforms. These more social factors may have more of an effect on the Grammy's than we might have first expected. Maybe Grammy's are not necessarily given to the year's best-tempoed song everytime, but rather to the year's most impactful song. Or perhaps the Grammy's are just full of it.

Appendix

1.1) Preprocessing Specifics

- Training Data
    - song, performer, instrumentalness, time signature removed
    - key to nominal
    - mode to binary
    - year in 7 bins of 9 years each
    - danceability remove instances with values < 0.1
    - loudness remove instances with values < -25 & > -0.35
    - speechiness remove instances with values > .5
    - tempo remove instances with values < 55 & > 215
    - Not-nominated class taken down to 689 instances using the Supervised/resample filter
        - class bias: 1.0
        - noReplacement: true
        - sampleSizePercent: 6
    - Nominated boosted up to 634 using the supervised/SMOTE filter
        - default settings
- Test Data
    - song, performer, instrumentalness, time signature removed
    - key to nominal
    - mode to binary