Disclaimer: *The insights and analysis expressed in this paper are those of the authors and do not necessarily reflect the views, official policy, or position of Georgetown University and its stakeholders.*

GEORGETOWN UNIVERSITY
School of Continuing Studies

Predicting Music Genre with Lyrics and Machine Learning Algorithms

Github: https://github.com/georgetown-analytics/Music-Lyrics

Georgetown University School of Continuing Studies
Cohort 23 — Capstone Project

Author: Tony Wright
Project Coordinator: Tony Wright
Capstone Advisor: Karen Belita

June 26, 2021 — Capstone Project in Partial Fulfilment of the Requirements for theCertificate in Data Science.

TABLE OF CONTENTS

1 ABSTRACT

1.1 Using algorithms to identify a product, a corresponding consumer, and uniting the two is the dominant use of advanced algorithms today (metric: 'money generated from the activity'). Identifying attributes of a new song is a necessary first step in finding a consumer for that song. The most fundamental of song attributes is genre. Fortunately, consumers readily identify themselves publicly by their favorite genres making it a powerful Foreign Key to complete the sale. This capstone used machine learning and the song lyrics to identify a song's musical genre.

1.2 Data Science has several tools to convert language into consumable data for various algorithms. Natural Language Processing (NLP) adds several unique steps to the front of the data science process. NLP strips away the pieces that help language make sense for people but add nothing to a machine's understanding. These NLP tools are rightly focused on literature and common media products. Applying NLP to songs faces several unique issues, however. First, words are often used as another musical instrument. Repeated words, in repeated lines of a chorus help keep the beat. But a chorus makes a mockery of n-gram analysis. Second, words are often chosen more for their sound than their meaning.

'Don't fix your lips like collagen.
And say something when you goin' end up apologin [sic]"
Kanye West's 'Can't Tell Me Nothin'[sic]" -2016

The man's a genius, but how does a machine identify that sentiment? This capstone project attempted to sharpen the NLP tools by using a large corpus of songs to create domain-specific NLP sentiment analysis variants, among other things.

1.3 Much of this is artificial, of course. The music industry has no limit to the people who promote, critique or sell music products that can 'Name that Genre!' in three notes or less. And an acute manpower shortage in this group made the *Data Science Journey of Discovery* an end in itself. Several necessary steps to lay the foundation for a legitimate data product were skipped to stay aligned with the syllabus overall and the rest of the groups. The data scrapping and much of the code fine tuning required to create something sustainable are not here.

2 HYPOTHESIS and FRAMING

2.1 Given a song's lyrics it is possible to identify a genre for that song using machine learning. Hip Hop is more distinct and machine learning will have more success classifying Hip Hop songs.

2.2 This capstone conducted training using a corpus of pre-labeled data with three genres. Rock, Pop, and Hip Hop. We assumed the genre label was correct.

2.3 There were two major concerns apparent at the start of this effort. First, song lyrics are not Natural Language (or they are hyper-natural depending on your nature), and NLP tools have limitations as a result (discussed above). Second, the corpus spanned decades of music and the subjective genre label may well have shifted, creating a moving target [variable].

2.4 The capstone attempted to mitigate both issues by using the data to create the tools required to evaluate the data. The tools are a derivative of existing NLP tools, created with domain-specific knowledge, in turn created by the corpus itself.

3 METHODOLOGY

3.1 **Logistics.** The software environment was created and constantly updated using the full spread of available options (Anaconda, pip and Homebrew). A current environment.yml is kept in the 'cfg' folder of the Git repository (https://github.com/georgetown-analytics/Music-Lyrics). That repository contains the recommended format and various pieces, including 'notebooks' and 'sample' folders. The 'sample' folder contains python scripts created first in Sublime Text, which were then tested piecemeal in a Jupyter Notebook and then run in their entirety in Terminal. The 'notebooks' folder is a series of Jupyter Notebooks outlining the data science journey. As data was cleaned, wrangled and munged it was kept in an Amazon Simple Storage Service (S3) 'bucket'. Several buckets were configured as WORM with 'Object Lock' enabled. These held the original dataset and specific canonical dataframes created at defined checkpoints along the way.

3.2 **Data Ingestion and Wrangling.** The datasets came from Kaggle. This set off a series of dataset-to-exploratory-data-analysis (EDA)-to-hypothesis-modification-to-dataset cycles, a necessary artificiality. We had turned the first steps on their head as one should start with a hypothesis and then go find, or make, or scrape the data. We thoroughly explored three different sets. We chose '6 Musical Genres' because it had the key features (lyrics and corresponding genre) and the lyrics were not yet pre-processed. This enabled us to fully work NLP.

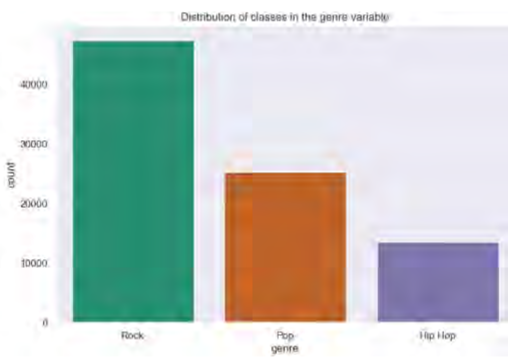
Song Lyrics from 6 Musical Genres	Rock, Pop, Sertanejo, Hip Hop, Funk Carioca	artists-data.csv	**Artist**	# Songs	# Popularity	Link	**Genre**	Genres
167499 tracks		Lyrics-data.csv	Alink	**Sname**	Slink	**Lyric**	Idiom	
Link								
Music Dataset: 1950 - 2019	pop, country, blues, rock, jazz	tcc_ceds_music.csv	#	**artist_name**	**track_name**	**release_date**	**genre**	**lyrics**
23689 tracks						Plus 24 other other classifications ->		
Link								
Song Lyrics	No Genre Information	album_details.csv	#	id	**singer_name**	name	type	**year**
25000 tracks		Lyrics.csv	#	link	artist	**song_name**	**lyrics**	
Link		songs_details.csv	#	song_id	**singer_name**	**song_name**	song_href	
		Pertinent Data:	**Artist**	**Song**	**Lyric**	**Genre**	**Year**	

3.3 **NLP Pre-Processing.** This process is unique to NLP and required some self-education[1] [2] and Teacher Assistance (TA) assistance.[3] NLTK, Gensim, and regex tools were used to create initial features (word and letter counts for the full set of lyrics) and process the lyrics for more advanced parsing. Before removing highly repetitive / low information words (stopwords) the group decided to look at multiple NLP options. The half-pre-processed lyrics through a standard path and the group started to send the same data, at the same point in cleaning, through a spaCy set of analysis.[4] The standard path included genism stopwords and NLTK lemmatization. This created a reduced set of lyrics with their own word/character counts. Additional feature engineering added a sentiment score and label (positive, negative and neutral). The group also appended a different sentiment analysis feature using the AFINN lexicon. While it was clear that spaCy was a powerful and clean way to accomplish most NLP things, the

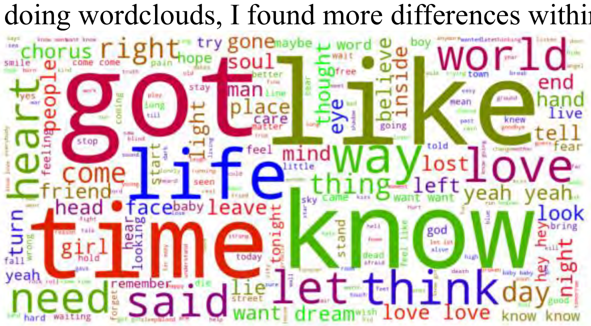
[1] Bengfort, Bilbro, Ojeda "Applied Text Analysis with Python", O'Reilly Media, Inc. 2018.
[2] Sarker, Bali, Sharma "Practical Machine Learning with Python", Apress Press, 2018
[3] L Carter, Sansui, Holland, Tanner, Scaramella, Johnson *"Article Classification Between Real & Fake News"* https://github.com/georgetown-analytics/From-Russia-With-Love-fake-news-
[4] spaCy API, Explosion 2016-2021, https://course.spacy.io/en

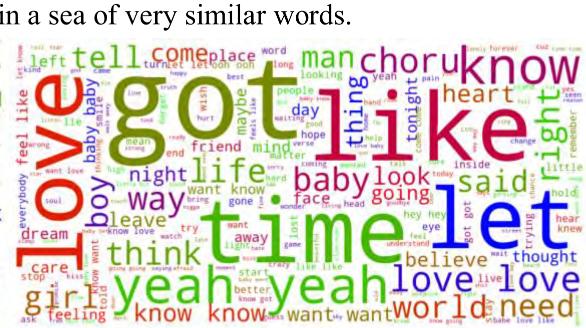
group failed to find a repeatable way to append the nlp.doc to my growing dataframe. The group had to run spaCy nlp.doc again with each new Jupyter session. We dropped the spaCy avenue of investigation.

3.4 **EDA and Visual Analysis.** The second time through EDA looked at the earliest features created and identified some promising facts (Hip Hop counts and sentiment are different, visually, from the others) and some concerning ones (Rock and Pop are similar). There is a disparity in the number of examples of each genre in the total data set of 86,290. Down sampling was required.



While doing wordclouds, I found more differences within a sea of very similar words.





Digging deeper in n-grams (CountVectorizer) we found some differences when looking at individual words and their frequency. There was real divergence between genres at the df.head(200) level of detail.

	Hip Hop	0
0	like	46642
1	got	35368
2	know	33244
3	don	32809
4	just	25822
5	ain	21722
6	nigga	20765
7	love	19968
8	yeah	18635
9	let	17348
10	shit	16286

	Rock	0
0	don	52511
1	love	43442
2	know	43362
3	just	40292
4	ll	37227
5	like	35838
6	oh	35330
7	got	29389
8	ve	28977
9	time	28532
10	let	23199

	Pop	0
0	love	49089
1	don	45462
2	oh	43967
3	know	41600
4	like	37362
5	just	34913
6	ll	27505
7	baby	26319
8	got	25833
9	let	24729
10	yeah	22165

n_gram_df.head(20)

	Rock	0	Pop	1	Hip Hop	2	All	3
0	yeah yeah	6201	yeah yeah	7251	yeah yeah	4536	yeah yeah	17988
1	love love	5157	love love	7069	know know	2505	love love	14679
2	hey hey	3832	know know	4362	love love	2453	know know	10613
3	know know	3746	want want	3834	feel like	1755	want want	8711
4	want want	3131	baby baby	3002	want want	1746	hey hey	7808
5	come come	2822	want know	2552	know got	1728	feel like	7058
6	feel like	2803	feel like	2500	got got	1711	baby baby	6662
7	baby baby	2335	hey hey	2445	like like	1643	want know	6159
8	want know	2237	ooh ooh	2252	hey hey	1531	come come	5632
9	let let	1921	let let	2226	know want	1473	got got	5365
10	rock roll	1920	know want	2200	let know	1386	know want	5249
11	time time	1738	got got	1979	want know	1370	let let	5158
12	got got	1675	like like	1829	baby baby	1325	know got	4940
13	far away	1627	come come	1810	baby girl	1064	like like	4489
14	know want	1576	know got	1769	look like	1017	let know	4240
15	know going	1449	know love	1755	let let	1011	ooh ooh	3965
16	know got	1443	love like	1693	come come	1000	know love	3943
17	going going	1423	let know	1623	snoop dogg	974	time time	3559
18	close eyes	1405	feels like	1519	girl know	968	love like	3355
19	know love	1350	want love	1503	like know	943	feels like	3282

n_gram_df.head(20)

	Rock	0	Pop	1	Hip Hop	2	All	3
0	yeah yeah yeah	3027	yeah yeah yeah	3658	yeah yeah yeah	1961	yeah yeah yeah	8646
1	love love love	2080	love love love	2997	love love love	976	love love love	6033
2	hey hey hey	1906	know know know	1442	hey hey hey	744	hey hey hey	3853
3	come come come	1177	want want want	1301	know know know	724	know know know	3296
4	know know know	1130	hey hey hey	1203	like like like	507	want want want	2786
5	want want want	1045	ooh ooh ooh	1202	baby baby baby	507	baby baby baby	2548
6	baby baby baby	995	baby baby baby	1046	got got got	452	come come come	2294
7	let let let	720	let let let	782	want want want	450	ooh ooh ooh	2037
8	run run run	539	work work work	774	come come come	412	let let let	1856
9	ooh ooh ooh	537	nah nah nah	725	work work work	369	got got got	1491
10	whoa whoa whoa	500	come come come	708	girl girl girl	358	like like like	1374
11	got got got	476	like like like	584	let let let	354	work work work	1204
12	time time time	443	got got got	563	stop stop stop	318	nah nah nah	1191
13	going going going	415	way way way	530	ooh ooh ooh	298	whoa whoa whoa	1039
14	doo doo doo	403	know want know	426	whoa whoa whoa	292	way way way	1037
15	alright alright alright	399	night night night	411	nah nah nah	290	run run run	965
16	away away away	359	shake shake shake	400	white white white	282	know want know	961
17	way way way	354	dance dance dance	384	money money money	276	want know want	888
18	round round round	352	want know want	357	low low low	257	time time time	875
19	want know want	349	run run run	349	night night night	248	round round round	865

However, bigrams and trigrams looked very similar regardless of the genre with many ‘yeah, yeah, yeah’s.

3.5 **Feature Engineering.** Remembering recommendations from Dr. Bengfort and looking at a df with every single word and its frequency, we decided to pursue the ideas of (1) domain-specific stopwords list, and (2) domain-specific sentiment. It started as another ‘branch’ our notebooks but became a concerted coding effort for the better part of four days. To avoid, or at least minimize, leakage, the lists were created using 80% of the total dataset with the same distribution of the target feature, genre.

3.5.1 Stopword lists are small lists for numerous words, with little meaning. What if they are also huge lists of little used words, each lacking meaningful statistical relevance? If one gets rid of 20,000 words used only 5 times each in a corpus, they've gotten rid of 100,000 points in a tf-idf sparce matrix. Stopword lists are often applied without much thought or concern, despite having a dramatic impact on the corpus left behind and any follow-on feature engineering. Looking further, "We hence recommend better documentation, dynamically adapting stop lists during preprocessing, as well as creating tools for stop lists quality control and automatically generating stop lists."[5] The process developed leveraged sklearn **CountVectorizer** and allowed the group to breakout words for various genres with a metric for how common they were (frequency). **The first step** in the process was to capture all words in all the lyrics. This was over 135,000 (from 30.6M total). Following a recommendation from Dr. Bengfort, a stop words list was made which constituted the least used words whose frequency sum added up to 5% of the total – about 96,000 different words, each used less than 36 times. The belief was that removing these words will reduce noise for clustering types of models.

3.5.2 **The next step** resulted in total word lists by genre. We **pd.merged** those lists pairwise (Hip Hop and Rock, etc.).

```
1 hiphopandpop.groupby(['_merge']).count()
```

_merge	Word	0_x	0_y
left_only	37476	37476	0
right_only	18496	0	18496
both	26930	26930	26930

```
1 hiphopandpop.head(20)
```

	Word	0_x	0_y	_merge
0	like	46642.0	37362.0	both
1	got	35368.0	25833.0	both
2	know	33244.0	41600.0	both
3	don	32809.0	45462.0	both
4	just	25822.0	34913.0	both
5	ain	21722.0	9575.0	both
6	nigga	20765.0	733.0	both
7	love	19968.0	49089.0	both
8	yeah	18635.0	22165.0	both
9	let	17348.0	24729.0	both
10	shit	16286.0	1561.0	both

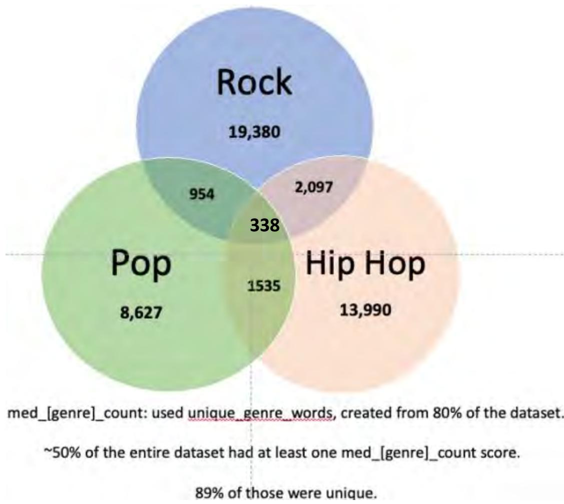
By using **indicator=True** & **groupby**, we broke out lists of words which were in one list, but not the other (*Hip Hop through Rock*, etc.). We then **pd.merged** the resulting dataframes by genre (*Hip Hop through Rock* with *Hip Hop through Pop*) and selected the words that were in both dataframes. With that, we had a list of words truly unique to each genre. The (80%) Pop/Rock/HipHop lexicon has 66,691 words.

Total Words		Words Used	Unique Words (100%)	Unique Words (80%)	Canonical Words
135,665	Hip Hop	80,152	40,652	29,843	2,054
	Pop	55,477	19,010	23,091	1,073
	Rock	69,860	30,702	13,757	1,871

3.5.3 We used the domain-specific stopwords list amended to NLTK stopwords list to make a third set of lyrics. Sml_lyrics is ~ 600,000 less words than med_lyrics, which is ~21 million less than full_lyrics. We re-ran the feature engineering steps on the sml_lyrics (counts, sentiment, affinity) and had another set of features to consider. Next we modified the NLTK stopwords process to create a count in each instance of how many of which genre_specific words were in each med_lyric and sml_lyric. Again, more features to consider.

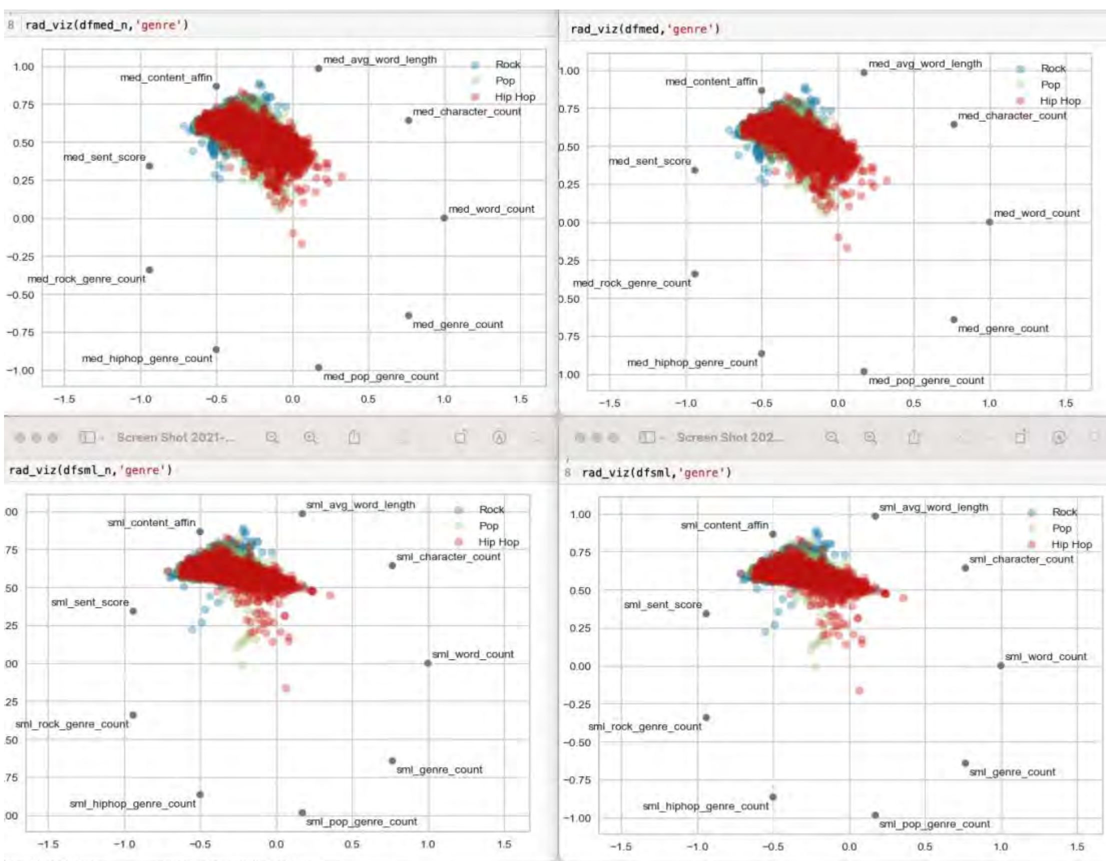
---

[5] Notham, Qin, Yurchak "*Stop Words Lists in Free Open-source Software Packages*" Proceedings of Workshop for NLP Open Source Software, pages 7-12



3.6 **EDA and Visual Analysis.** From the original data set with four features, we had grown to 29 features across three main sets. Features came from either the full_lyrics, the med_lyrics or the sml_lyrics. Getting from full to med went through genism STOPWORDS. From med to sml went through NLTK's stop_words and the genre stop words list we'd built. For full, med and sml there are word and character counts. For med and sml there are affinity and sentiment scores / labels, and then the count, by genre, of genre-specific words. Forty eight percent of the entire dataset set had at least one genre 'hit'. And 89% of those were exclusively in one genre.

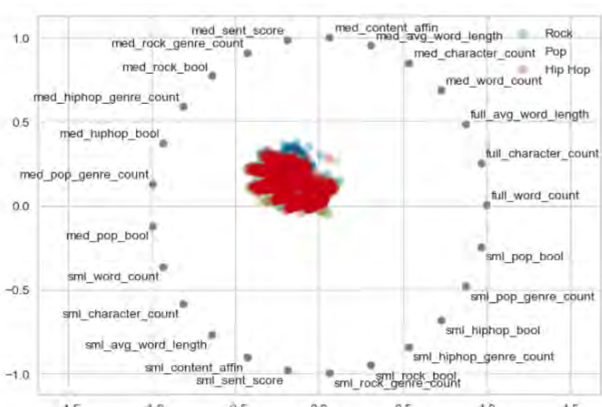
We knew we had collinearity when we plotted features from across each of the three pillars (full, med, sml). But apparently, we had it within the pillars as well. We attempted to bring out the importance of the genre_count by scaling (MinMaxScaler) those features - to no obvious effect. RadViz plots below are broken out by med & sml features, and then those same features scaled.



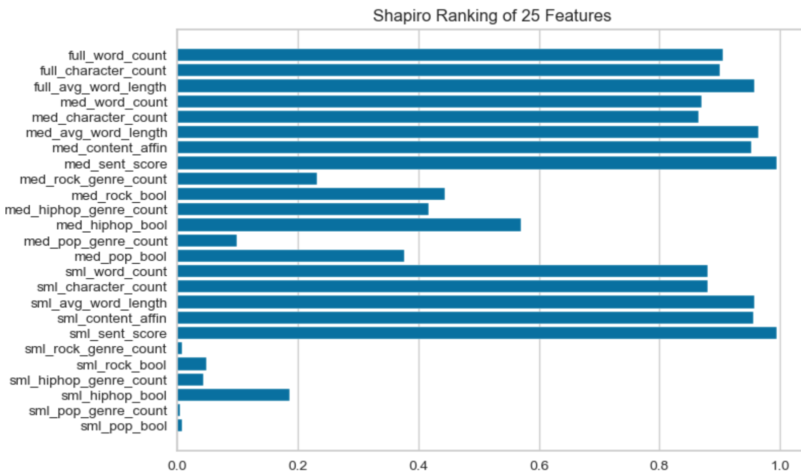
3.7 **Feature Engineering.** An early run of various ML pipelines showed that the algorithms were often confusing Rock and Hip Hop. This made it obvious the genre_counts weren't making a difference even though 70% of Hip Hop instances had a unique Hip Hop word. Also, clustering visualizations showed that scaling genre_counts didn't make the difference obvious either (all other things being equal). edecided to create a new feature, a simpler binary feature which indicated the presence of one or more domain specific words. [size]_[genre]_bool We had 35 features.

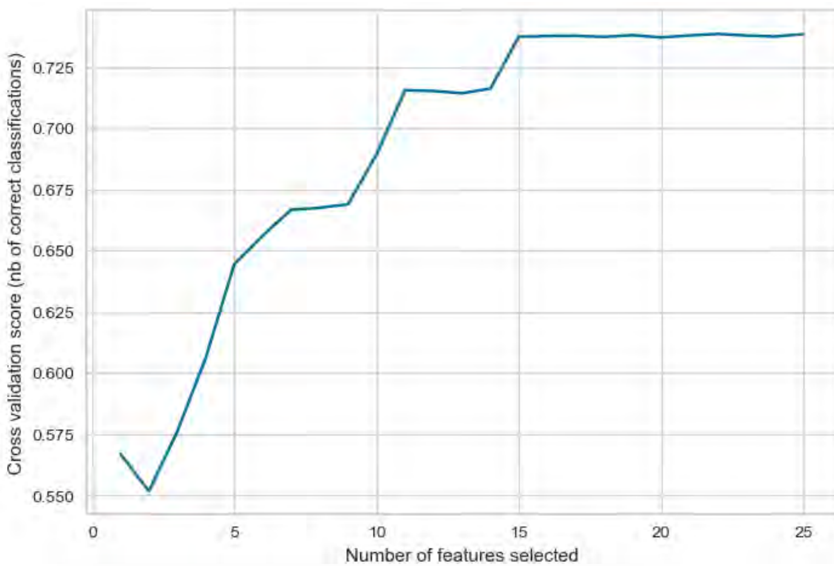
#	Column	Non-Null	dType	Notes:
0	genre	86290	Object	Target
1	song_name	86290	Object	dataset
2	lyrics	86290	Object	dataset
3	full_word_count	86290	Int64	lambda
4	full_character_count	86290	Int64	lambda
5	full_avg_word_length	86290	float64	lambda
6	med_lyrics	86290	Object	cleaned lemmatized, gensim stopwords, 39% of full_lyrics
7	med_word_count	86290	Int64	lambda
8	med_character_count	86290	Int64	lambda
9	med_avg_word_length	86290	float64	lambda
10	med_content_affin	86290	float64	AFFIN lexicon
11	med_sent_label	86290	Object	TextBlob - positive, negative, neutral
12	med_sent_score	86290	float64	TextBlob - -0.5 to +0.5
13	med_vector	86290	Object	NLTK punkt and wordnet, bag of words, in order.
14	med_rock_genre_count	86290	float64	# of unique rock words in the lyrics X .01
15	med_rock_bool	86290	Int64	0 or 1, rock word present?
16	med_hiphop_genre_count	86290	float64	# of unique hiphop words in the lyrics X 100
17	med_hiphop_bool	86290	Int64	0 or 1, hiphop word present?
18	med_pop_genre_count	86290	float64	# of unique pop words in the lyrics X 1.0
19	med_pop_bool	86290	Int64	0 or 1, pop word present?
20	med_genre_count	86290	Int64	Sum of rock, pop, and hiphop genre counts.
21	sml_lyrics	86290	Object	med_lyrics run through NLTK stop_words and genre_stopwords
22	sml_word_count	86290	Int64	lambda
23	sml_character_count	86290	Int64	lambda
24	sml_avg_word_length	86290	float64	lambda
25	sml_content_affin	86290	float64	AFFIN lexicon
26	sml_sent_label	86290	Object	TextBlob - positive, negative, neutral
27	sml_sent_score	86290	float64	TextBlob - -0.5 to +0.5
28	sml_vector	86290	Object	NLTK punkt and wordnet, bag of words, in order.
29	sml_rock_genre_count	86290	float64	# of unique rock words in the lyrics X .01
30	sml_rock_bool	86290	Int64	0 or 1, rock word present?
31	sml_hiphop_genre_count	86290	float64	# of unique hiphop words in the lyrics X 100
32	sml_hiphop_bool	86290	Int64	0 or 1, hiphop word present?
33	sml_pop_genre_count	86290	float64	# of unique pop words in the lyrics X 1.0
34	sml_pop_bool	86290	Int64	0 or 1, pop word present?
35	sml_genre_count	86290	Int64	Sum of rock, pop, and hiphop genre counts.

3.8 **Visual Steering and Feature Selection.** *"Most of these techniques are univariate, meaning that they evaluate each predictor in isolation. In this case, the existence of correlated predictors makes it possible to select important, but redundant, predictors. The obvious consequences of this issue are that too many predictors are chosen and, as a result, problems arise."*[6] An example of the *problem*, to the left. In order to move through the visualizers faster, we created a numeric-only dataset and toured the various tools to conduct feature selection. As this was a numeric input with a classification output, attempting feature selection one at a time fit within an Analysis of Variance (ANOVA) worked best. Sklearn, **f_classif()** does not have a visualizer. Run multiple times selecting an expanding group of features (k=3,5,7,10, 12, 15, 20, 25) across three different models., documenting F1 score and **selector.support_** each time created our own

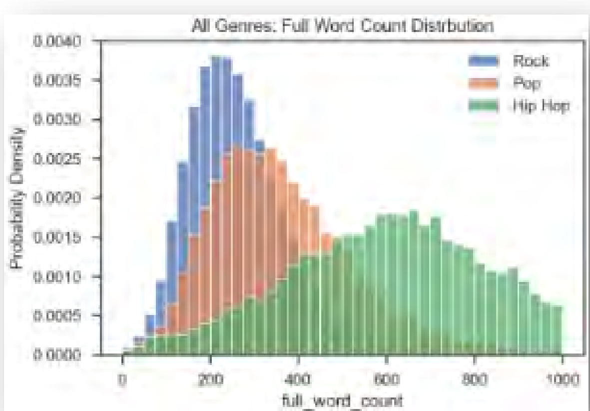


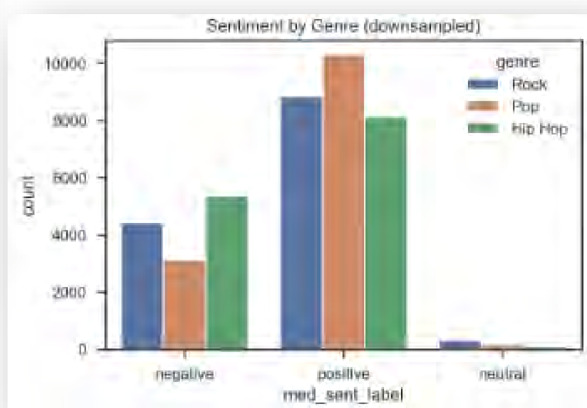
[6] Kuhn, Johnson "Applied Predictive Modeling: 2nd ed, 2018. Springer Publishing

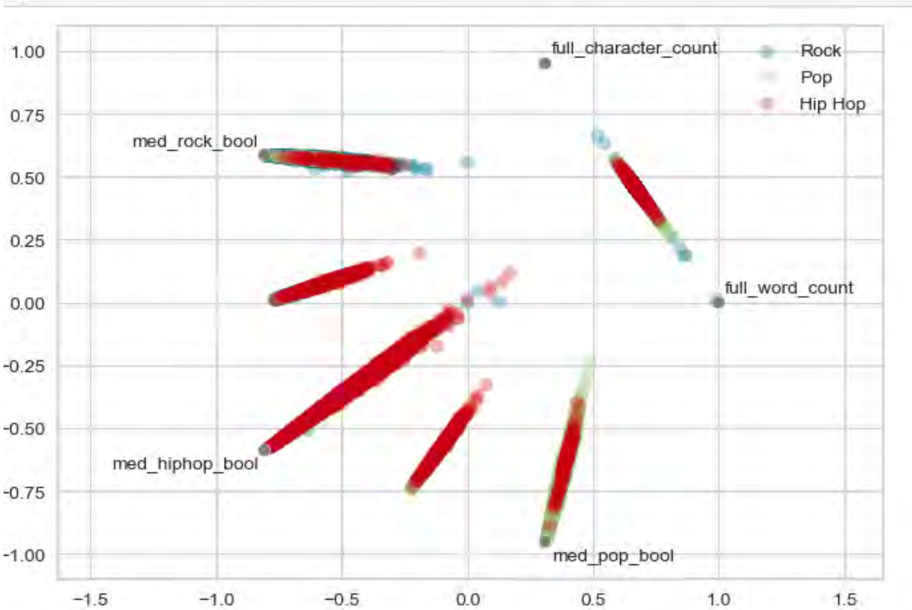




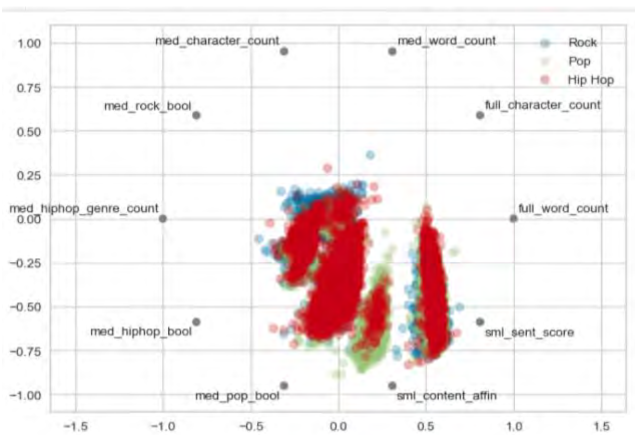
visualizer. ANOVA recommended various word counts and the [size]_[genre]_bool features. The 'knee in the curve' for best F1 was ~ 10 features. ANOVA did not select affin or sentiment. Wetried Rank1D and Rank2D next. These are best with regressions, and not classifiers, but they are easy and still useful for getting a different look at the same data. Rank1D selected affin and sentiment and didn't select [size]_[genre]_bool. Rank2D weakly highlighted the power of cross-referencing word counts, [size]_[genre]_bool and sentiment and/or affin. We next looked at Recursive Feature Elimination and Cross Validation (RFECV) to see if I could break the tie. Again, ten features were about the right number. It was harder to prioritize features using RFECV. It would just report the 'top 18'. However word counts, sentiment/affin and [size]_[genre]_bool were in the top 18. EDA showed why some things were impactful. Hip Hop was just much more verbose. Sentiment was not as stark, but Hip Hop sentiment was negative 42% of the time, whereas Pop was negative 24%.



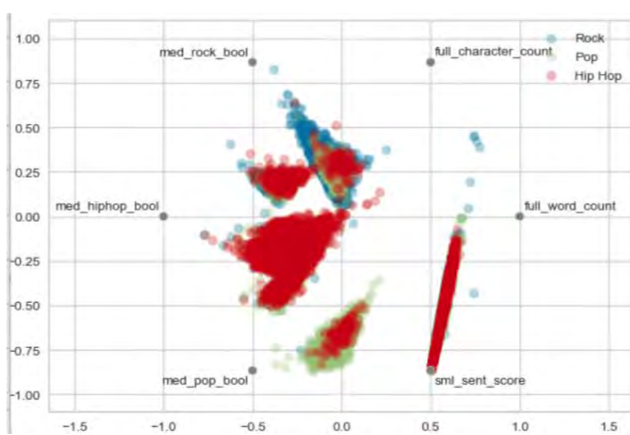




Perhaps because of our own biases, we explored the recommended features from ANOVA first. The RadViz was different, exciting. The breakouts between Pop, Rock and Hip Hop were right there, pointed at the target. There was some red everywhere, but that could have just been a visualization issue. Red shows up more when Alpha=.03.
But, even if that were true I knew the lyrics did not conform to such tight buckets. For every



Snoop Dog making classic Hip Hop, there is a Linkin Park performing at the intersection of Rock and Hip Hop. And don't get us started about *Old Town Road*. This was too artificial. We put in sentiment and affin next. And then we tried all of the features, in all of the ways. Sentiment and affin 'mattered'. But they also muddied the picture. Without them, the picture was too clean unless redundant word / character counts were added.
Next we tried just affin, and then just sentiment. We chose sml vice med as the range and standard deviations were broader. These visualizations seemed the best fit (most defined groups) yet.



RFECV broke out affin above sentiment. In an unscientific look at RadViz, the sml_content_affin groups look more defined. And because sent_label encodes the sentiment information in a more definitive way (0.0001 and a 0.98 sent_scores are both 'positive') we felt there was a chance that clustering algorithms would pick up the Hip Hop / Pop sentiment divide better. We next picked the full word/character counts as the differences between genres was larger when compared to med and sml. Finally, the three med_[genre]_bool features rounded out the numeric feature selection.

3.9 **Modeling.** There were four types of features selected: objects (sml_vector); categorical (sml_sent_label (positive, negative, neutral); numbers with outliers (word and character counts); and numbers without outliers (genre Boolean feature and sml_content_affin). Each type needed to be scaled or otherwise prepared for modeling in different ways. A column transformer prepared the data which was then fed into a series of models for preliminary evaluation. Code block below.

```
target_mnb = dfd.genre
features_mnb = dfd[['full_word_count','full_character_count',
                                   'med_rock_bool','med_hiphop_bool','med_pop_bool',
                                   'sml_word_count','sml_character_count',
                                    'sml_sent_label','sml_content_affin','sml_vector']].copy()
```

```
# defining the numerical, categorical and textual features
numerical = ['full_word_count','full_character_count','sml_word_count','sml_character_count']
negative_values = ['sml_content_affin','med_rock_bool','med_hiphop_bool','med_pop_bool']
categorical = ['sml_sent_label']
textual = ['sml_vector']
```

```
ct = ColumnTransformer(
    [('num', RobustScaler(), numerical),
     ('neg_values', MinMaxScaler(feature_range=(0,2)), negative_values),
     ('cat', OneHotEncoder(),categorical),
     ('tfidf', TfidfVectorizer(max_features = 6000,
                               stop_words = 'english',
                               ngram_range=(1,1)), 'sml_vector')
    ], n_jobs=3, verbose=True)
```

```
ColumnTransformer(n_jobs=3,
                  transformers=[('num', RobustScaler(),
                                 ['full_word_count', 'full_character_count',
                                  'sml_word_count', 'sml_character_count']),
                                ('neg_values',
                                 MinMaxScaler(feature_range=(0, 2)),
                                 ['sml_content_affin', 'med_rock_bool',
                                  'med_hiphop_bool', 'med_pop_bool']),
                                ('cat', OneHotEncoder(), ['sml_sent_label']),
                                ('tfidf',
                                 TfidfVectorizer(max_features=6000,
                                                 stop_words='english'),
                                 'sml_vector')],
                  verbose=True)
```

```
# Creating the feature matrix
X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)
print(f'Shape of Term Frequency Matrix of train: {X_train.shape}')
print(f'Shape of Term Frequency Matrix of test: {X_test.shape}')
```

```
Shape of Term Frequency Matrix of train: (32544, 6011)
Shape of Term Frequency Matrix of test: (8136, 6011)
```

```
# ExtraTrees classifier
etc = ExtraTreesClassifier(n_estimators=165, criterion='gini',max_depth=350, n_jobs=-1)
# Training the model
etc.fit(X_train, y_train)

#Predict the Test using RandomForest classifier
y_pred_etc = etc.predict(X_test)
print('Accuracy on x_train is',etc.score(X_train, y_train))
print('Accuracy on x_test is',etc.score(X_test, y_test))
```

The first look at 15 different models is below. The green highlights performance in the top 20%, red shows poor performance. As expected, the models were more successful identifying the genre using lyrics when the songs were from the Hip Hop genre averaging an F1 score of .851.

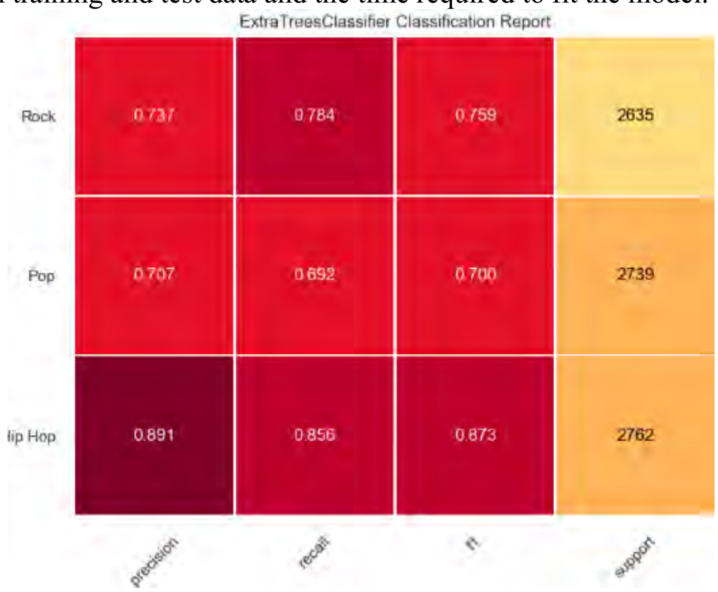
	Rock	Pop	Hip Hop	F1
LinearSVC,	0.746	0.646	0.858	0.750
SVC	0.674	0.672	0.822	0.723
BaggingClassifier,	0.708	0.669	0.856	0.744
ExtraTreesClassifier,	0.75	0.694	0.867	0.770
RandomForestClassifier,	0.744	0.666	0.859	0.756
DecisionTreeClassifier	0.669	0.615	0.82	0.701
AdaBoostClassifier	0.738	0.648	0.863	0.750
KNeighborsClassifier	0.659	0.526	0.827	0.704
LogisticRegressionCV,	0.747	0.694	0.87	0.770
LogisticRegression,	0.745	0.692	0.874	0.770
SDGClassifier	0.749	0.688	0.866	0.768
MultinomialNB	0.735	0.689	0.853	0.759
GaussianNB				
BernoulliNB	0.733	0.61	0.807	0.717
MLPClassifier	0.7	0.646	0.851	0.732
GradientBoostingClassifier	0.745	0.685	0.869	0.766
Avg all models	*0.723*	*0.663*	*0.851*	

3.10 **Hyperparameter Tuning.** The group selected four models to tune. Two were chosen based exclusively preliminary performance (ExtraTreesClassifier, LogisticRegressionCV). Two were chosen simply to tune different families of models (MultinomialNB, MLPClassifier).

3.10.1 **ExtraTreesClassifier.** This ensemble method performed well from the start. The out-of-the-box parameter settings left little to be improved upon. Only adding additional estimators and constraining the max depth forced a slight improvement. The table below shows which parameters were explored, which were default (in red) and which were selected as most effective (highlighted in yellow).

n_estimators	10	50	100	125	150	175	200
criterion	gini	entropy					
max_depth	100	250	500	750	1000	none	
bootstrap	TRUE	FALSE					
oob_score	TRUE	FALSE					
warm_start	TRUE	FALSE					
Parameter1	Grid Serached	Default	Selected				

The confusion matrix below shows the precision recall, F1 and test set for each of the genres using ExtraTreesClassifier and optimum settings. The print statement below the confusion matrix highlights the difference between training and test data and the time required to fit the model.



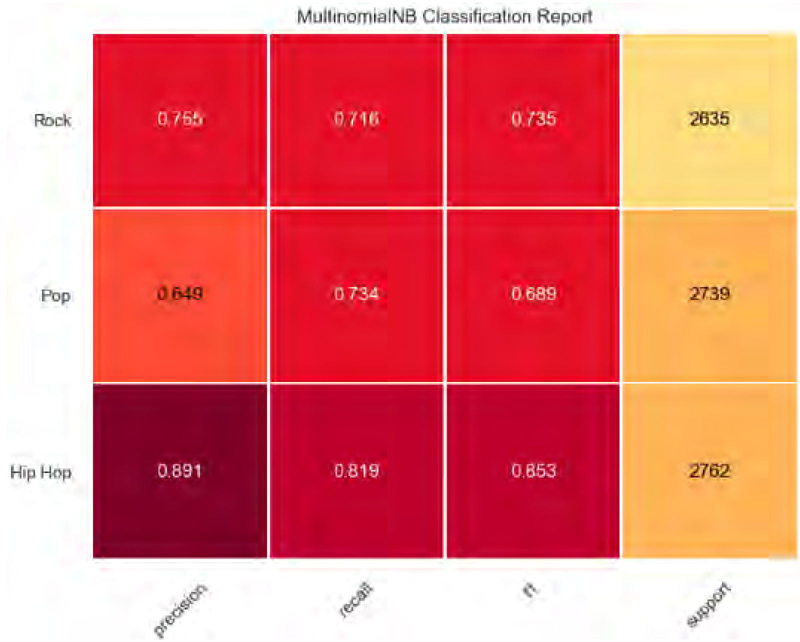

```
Accuracy on x_train is 0.9999078171091446
Accuracy on x_test is 0.7764257620452311
CPU times: user 2min 43s, sys: 967 ms, total: 2min 44s
Wall time: 22.7 s
```

Hyperparameter tuning did not accomplish much with this classifier.

	Rock	Pop	Hip Hop	F1
First Pass	0.75	0.694	0.867	0.770
Second Pass	0.759	0.7	0.873	0.777
Delta	0.009	0.006	0.006	0.007

3.10.2 MultinomialNBClassifier. This probabilistic classifier had few parameters to tune, the parameters were set correctly from the beginning, and was very, very fast.

Alpha	0	0.5	1	1.5	2	3
fit_prior	TRUE	FALSE				

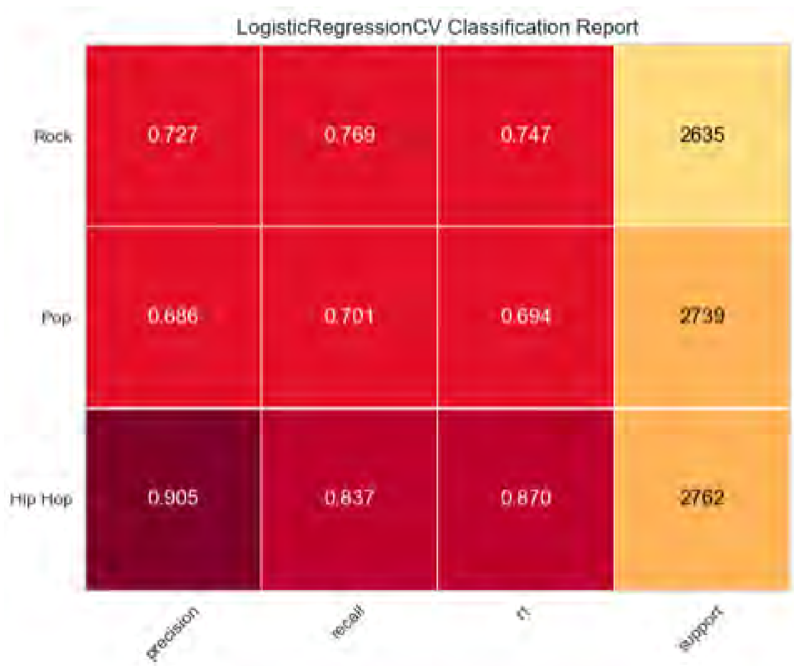



```
Accuracy on x_train is 0.7745206489675516
Accuracy on x_test is 0.7568829891838741
CPU times: user 33 ms, sys: 2.66 ms, total: 35.7 ms
Wall time: 33.7 ms
```

	Rock	Pop	Hip Hop	F1
First Pass	0.735	0.689	0.853	0.759
Second Pass	0.735	0.689	0.853	0.759
Delta	0	0	0	0

3.10.3 LogisticRegressionCV. The parameters for this model could not be improved upon.

Cs	1	5	10	25		
CV	1	5	12			
Solver	lbfgs	newton-cg	liblinear	sag	saga	saga
Penalty	l2	l2	l1, l2	l2	elasticnet	l1
multi_class	auto	ovr	multinomial			



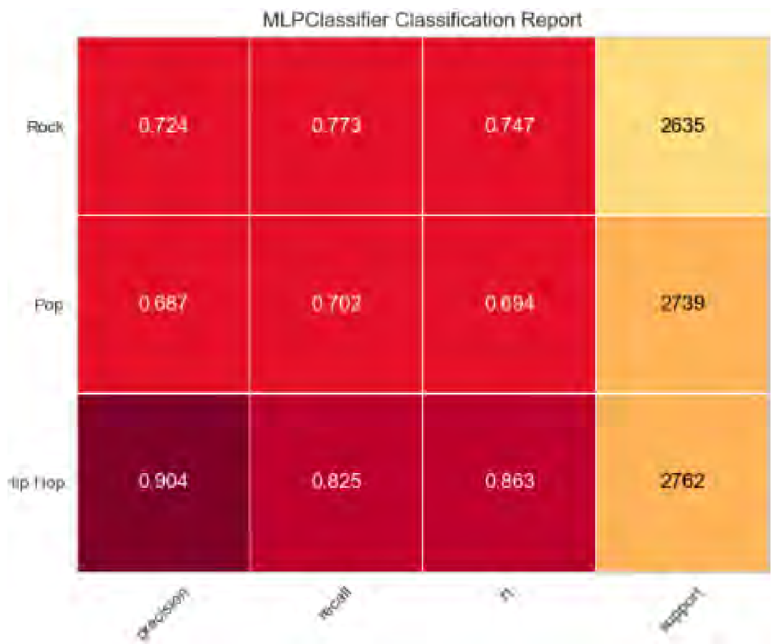

```
Accuracy on x_train is 0.8054633726647001
Accuracy on x_test is 0.7689282202556539
CPU times: user 1min 25s, sys: 2min, total: 3min 25s
Wall time: 1min 6s
```

	Rock	Pop	Hip Hop	F1
First Pass	0.747	0.694	0.87	0.770
Second Pass	0.747	0.694	0.87	0.770
Delta	0	0	0	0

3.10.4 MLPClassifier.

This multi-layer perceptron classifier had myriad tunable parameters and was fascinating to work with.

hidden_layer_sizes	1	3	5	50	100	150	200	300
solver	lbfgs	sgd	adam					
activation	relu	logistic	tanh					
max_fun	15000	17000						
alpha	0.00001	0.0001	0.001	0.1	1	5		

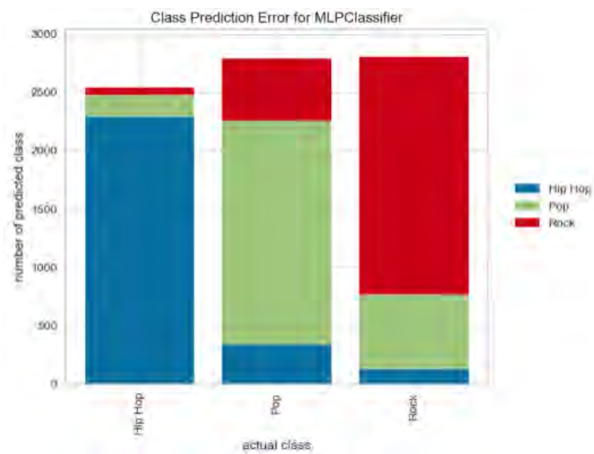



```
Accuracy on x_train is 0.7820796460176991
Accuracy on x_test is 0.7672074729596854
CPU times: user 12min 4s, sys: 21min 47s, total: 33min 51s
Wall time: 7min 13s
```

	Rock	Pop	Hip Hop	F1
First Pass	0.7	0.646	0.851	0.732
Second Pass	0.746	0.696	0.863	0.768
Delta	0.046	0.05	0.012	0.036

4 CONCLUSION.

The data science pipeline and this analysis supports the hypothesis that a song's genre can be identified using the lyrics and machine learning algorithms. Further, in that same pipeline, lyrics of Hip Hop songs make them more identifiable. Creation of a genre-specific stopwords list reduced the overall size of the corpus required to be put into machine learning algorithms improving processing time, slightly. A genre-specific lexicon of words was an important feature used by the algorithms to classify the labeled training set. Even with these additions to Python's NLP tools, classifying Pop music, as compared to Hip Hop or Rock, was difficult and severely degraded the performance overall.



5 NEXT STEPS.

Performing this type of analysis more accurately, faster, across more genres in a repeatable way would require many improvements to the process used here. More data, with more classes would add relevance to this set of genres. Expanding beyond lyrics to sound would bring a dramatic improvement in performance as the combination of lyrics and sound is at the key to what a musical genre uses to describe itself. A function to identify choruses (repeated lines of lyrics) would allow running sentiment/ objectivity lexicons, term frequency (inverse document frequency) or n-gram analysis on the 'meat' of the lyrics, the verse. That would perhaps improve the performance of those NLP tools. A model based on a Feature Union would enable weighting the various intertwined pipelines differently. This would add a new, perhaps critical, dimension to hyperparameter tuning.