Project 2: Logistic Regression and Friends

Miguel Cordova, Rahul Payeli, Gabriel Urbaitis

In music streaming, the challenge of accurately classifying songs into genres remains a pivotal aspect of enhancing user experience. This paper delves into machine learning techniques to tackle this challenge, focusing on logistic regression and gradient descent. We explore the extraction of audio features such as Mel-Frequency Cepstral Coefficients (MFCC) and Chroma features and their utility in genre classification. Furthermore, we compare the performance of our logistic regression model with other classical classifiers like Support Vector Machines (SVM) and Random Forests. Our findings highlight the potential of machine learning in revolutionizing music discovery and pave the way for further research in this domain.

I. Introduction

The landscape of music streaming is constantly evolving, with the quest to enhance user experience at its core. A crucial aspect of this experience is the ability to classify songs into genres, thereby revolutionizing music discovery accurately. As the complexity of music and technology advances, traditional methods of genre classification still need to be improved. In This project, we leverage machine learning to address this challenge, focusing on logistic regression and gradient descent.

Given audio data's rich and high-dimensional nature, logistic regression emerges as a suitable choice for its ability to handle multi-class classification tasks. Additionally, we explore feature extraction techniques such as Mel-Frequency Cepstral Coefficients (MFCC) and Chroma features, which are instrumental in capturing the unique signatures of different music genres. The project also involves a comparative analysis with other classical machine learning models like Support Vector Machines (SVM) and Random Forests, further enriching our understanding of the data and the model's performance.

This journey into the heart of music and machine learning is not just about technical prowess; it's about igniting a passion for data and unlocking new possibilities in music discovery. As we embark on this adventure, we aim to showcase our skills and creativity, all while contributing to the ever-evolving world of music streaming. The stage is set, and the data awaits – let the symphony of algorithms begin!

II. Design and Implementation

In alignment with the project guidelines, we meticulously employed a suite of feature extraction techniques to extract the essence of each audio file. Our approach encompassed the use of the Short-Time Fourier Transform (STFT), Mel-Frequency Cepstral Coefficients (MFCC), Chroma Features, and Spectral Contrast, each selected for its unique ability to capture different facets of the musical landscape as we saw we would need them because each genres capacity for variation in waveform. Following the extraction process, we harnessed the power of Principal Component Analysis (PCA) to reduce the dimensionality of our dataset adeptly, ensuring a more manageable and efficient analysis well also standardizing the data to allow for more in-range results.

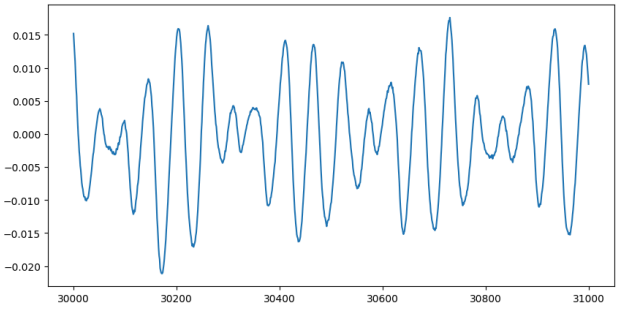
With our feature-rich dataset, we focused on the core of our classification task. We developed a logistic regression model from the ground up, tailored to discern each audio file's genre accurately. By computing the probabilities associated with each genre, our model offers a nuanced and sophisticated understanding of the musical genres represented in our dataset. Through this comprehensive approach, we strive to classify music precisely and illuminate

the intricate patterns defining the auditory experience.

A. *Extracting Data*

In our music classification project, we analyzed audio data for different genres to understand their unique waveform structures. For example, blues showed a smooth and mellow waveform, while classical exhibited complex patterns with varying amplitudes and frequencies. Country music displayed a mix of smooth and sharp transitions, indicating a blend of acoustic and electric instruments. Disco had a consistent beat and rhythmic pattern, reflecting its danceable nature. Hip-hop was characterized by sharp peaks representing its beats and rhythms, while jazz showed a mix of smooth and sharp transitions, highlighting its improvisational nature. Metal had intense and aggressive patterns, pop featured catchy and melodic structures, reggae had a relaxed rhythm with off-beat accents, and rock showed powerful and energetic structures.

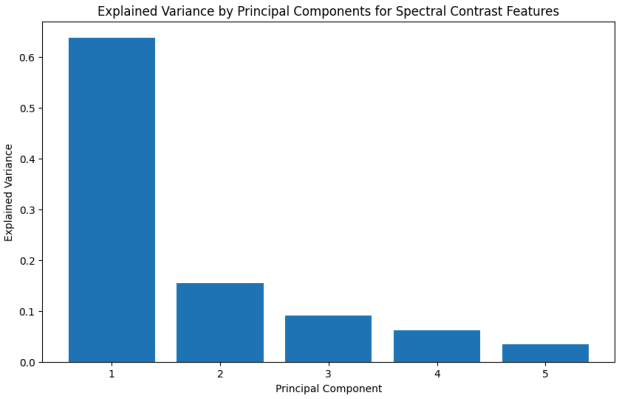
Based on these analyses, we decided to use specific audio features that capture the essence of each genre. We plan to extract Mel-Frequency Cepstral Coefficients (MFCCs) to capture timbral and spectral qualities, Chroma Features for musical pitches, Spectral Features for texture and frequency content, Rhythm Features for tempo and beat onset strength, and Zero-Crossing Rate for the rate of sign changes in the waveform.



We opted to take the mean of the large vectors and matrices for feature reduction, as opposed to applying Principal Component Analysis (PCA). Our rationale was that PCA might still leave us with an excessive number of features, whereas calculating the mean would provide us with a singular vector, significantly reducing our dimensionality. However, we acknowledge that there were potential improvements we could have explored. For instance, applying PCA twice, implementing additional noise reduction techniques, or extracting other statistical measures like the range or median might have enhanced our results further.

B. *PCA Standardization*

We decided to standardize the data before applying PCA to three distinct features: chroma, Mel-Frequency Cepstral Coefficients, and spectral contrast. Our rationale for this approach was to reduce the dimensionality of our feature set while retaining the most significant components. We established a threshold, deciding to keep only those components that explained more than 5% of the variance in the data. This decision was based on the observation that components contributing less than this threshold had a negligible impact on our analysis.



Looking forward, we are contemplating potential improvements to our methodology. One avenue for exploration is the possibility of applying PCA to all three features after combining them, which might reveal new insights or patterns. Alternatively, we are considering applying PCA to all features simultaneously, which could provide a more holistic view of the data's variance. These adjustments could enhance the effectiveness of our feature reduction and, subsequently, the accuracy of our music genre classification.

C. *Softmax*

In logistic regression, the softmax function is used when dealing with multiclass classification problems, where the target variable has more than two possible classes or categories. We chose this over sigmoid because softmax deals with multiclass and is numerically more stable than sigmoid.

We also added a technique of subtracting the max value from each element given a row, which can potentially avoid the numerical overflow.

```
def softmax(z):
  """
  This function performs softmax on a given matrix z.

  Parameters:
  z (numpy array): This is a dot product of X_samples and weight

  Returns:
  predicted probabilites of the given input matrix z.
  """
  #This max value substraction is useful to prevent numerical overflow.
  max_z = np.max(z, axis=1, keepdims=True)
  exp_z = np.exp(z - max_z)

  exp_sum = np.sum(exp_z, axis=1, keepdims=True)
  sm = exp_z / exp_sum
  return sm
```

D. calError

The difference between predicted probabilities and one hot encoded matrix is calculated in this function. (Error)

```
def calError(weight, X_samples, Y_target):
  """
  This function is used to calculate the error.

  Parameters:
  weight (numpy array): weight matrix.
  X_samples (numpy array): data samples.
  Y_target (numpy array): traget values.

  Returns:
  returns the difference between the oneHotencoded matrix and predicted probabilities i.e., error
  """
  z = np.dot(X_samples, weight)
  pred_prob = softmaxNormalized(z) #probability prediction using the softmax function
  oneHotCoded = pd.get_dummies(Y_target) #one Hot encoding
  diff = np.subtract(oneHotCoded, pred_prob) #error calculation
  return diff
```

E. Gradient Ascent

The gradient ascent is used to find the local maximum of a function. The Ascent function implements the gradient ascent algorithm from scratch. It fetches the error or difference between one-hot encoding and predicted probabilities, which is used in calculating the gradient. This gradient ascent is used in the weight update.

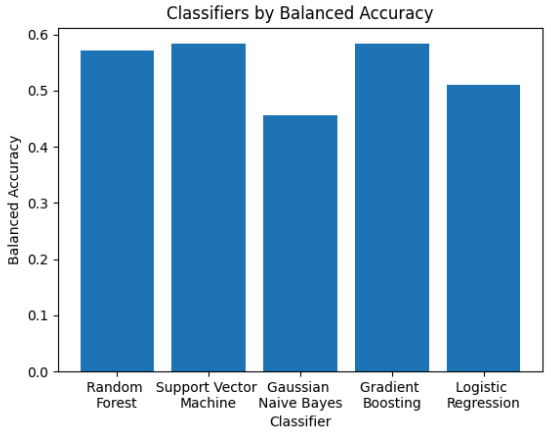
This Ascent function handles the learning rate(mu), and the convergence criteria(epsilon) in the right way, to get the correct weights for prediction.

The learning rate (**mu**) is used to determine the step size at which the weights are updated in each iteration. A smaller learning rate leads to slower convergence but can help in finding a more precise optimum, while a larger learning rate may lead to faster convergence but might skip the optimum. Here, we are using **0.00151 for mu**, after fine-tuning the hypermeters.

The convergence criteria (**epsilon**) is used to determine when to stop the weight update process. In each iteration, the change in weights is calculated, and if the change falls below the specified epsilon threshold, the algorithm considers it as convergence and stops the update process. This ensures that the algorithm does not continue indefinitely and terminates when the weights stabilize. Here, we are using **0.00151 for mu**, after fine-tuning the hypermeters.

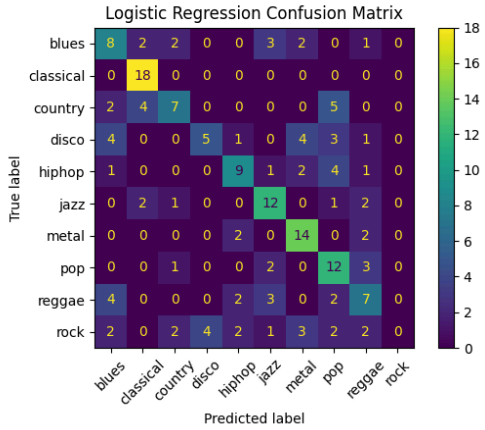
```
def Ascent(X_samples, Y_target, weight, mu, lmb, epsilon = 1e-1):
  """
  This function does the gradient ascent, and weight update calcualtion.

  Parameters:
  weight (numpy array): weight matrix.
  X_samples (numpy array): data samples.
  Y_target (numpy array): traget values.
  mu (float): hyperparameter used as the step size
  lmb (float): hyperparameter used in L2 regularization
  epsilon (float): hyperparameter used in stopping the weight update.

  Returns:
  returns the updated weight.
  """
  prevWeight = weight

  while True:
    diff = (calError(weight, X_samples, Y_target)) #gets error calculated
    gradient = np.dot(X_samples.T, diff) # calculating gradient
    weight = weight + mu * (gradient - lmb * weight) # weight update
    weightChange = np.abs(prevWeight - weight).sum() # weight change calculation
    print('weightChange & epsilon:',weightChange,' ',epsilon)
    if weightChange < epsilon: #checks if the weight change is less then epsilon(weight converge check)
      break
    prevWeight = weight
  return weight
```

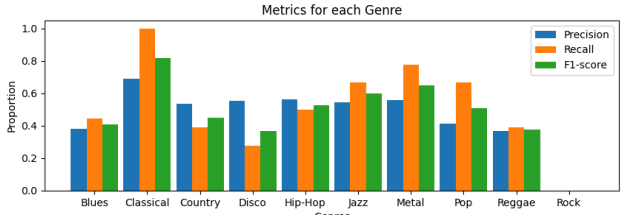
F. Analysis



Balanced Accuracy for Random Forests was .572222,
for Support Vector Machines (SVM) .583333, for Gaussian Naive Bayes .455556, for Gradient Boosting .583333 and for Logistic Regression .511111. Gradient Boosting and SVM had the highest accuracy. Both can capture complex, non-linear relationships between features and target classes, and this may explain why they were more

accurate than Logistic Regression, as Logistic Regression assumes a linear relationship between features and log odds of the target. Random Forests were next highest in accuracy and close to the Gradient Boosting's shared best accuracy. This may be because both use forests of decision trees and averaging predictions which make them robust to noisy data. Gaussian Naive Bayes assumes that features are conditionally independent given the class label, and as the assumption does not hold given our genres, it may explain why it was the lowest of all the classifiers compared, as the other classifiers do not make such a strong independence assumption. Overall, the differences between the classifiers are relatively small, so their changes may be explained more by hyperparameter tuning, which may be an avenue for further improvement.





Our Logistic Regression model succeeded most in classifying classical music, classifying all 18 samples in the test set correctly. Its precision and F1-Score were also the highest of any genre suggesting that not only was it easy to identify when given a sample, but also an unlikely target for a sample to be misclassified as. By contrast, Rock, which didn't have any samples in the test set so it couldn't be misclassified, had at least one misclassified sample from every other genre except classical. No other genre had so many other genres misclassified to it as the target.

A possible explanation for the contrast is that Classical tends to have a more standardized structure and composition, whereas Rock can vary widely in terms of instrumentation, vocal styles, rhythms, and song structures. The variability in Rock music can make it harder for the classifier to identify consistent patterns and characteristics associated with it.

The next highest Recall to Classical that stood out was Metal. This was likely easier to classify because Metal music often exhibits distinct creatures like aggressive vocals, distorted guitars, fast tempos, and complex rhythms.

The Lowest, non-Rock Recall was Disco. Disco is likely difficult to classify because it incorporates elements from genres like funk, soul, pop, and electronic music, so again, classifiers will have a hard time identifying consistent patterns associated with it.

In the middle of the spectrum of Recalls, there is more variance, so it is better to look at what were common misclassifications to understand the middle of the pack. The most misclassified was Country as Pop at 5, followed by Disco as Blues or Metal, Hip-hop as Pop, and Reggae as Blues at 4.

Country and Pop share elements such as catchy melodies, upbeat rhythms, and polished production. Hip-hop and Pop also have catchy melodies, and they also share rhythmic beats. Reggae and Blues share slow to moderate tempos and emphasize repetitive rhythmic patterns. As discussed earlier, Disco's many influences can make it somewhat easy to misclassify.

In conclusion, accurately classifying music genres presents a multifaceted challenge due to the diverse range of musical styles, influences, and subjective interpretations inherent in genre classification. While some genres may exhibit distinct characteristics that lend themselves to more accurate classification, others, such as disco, may present complexities that lead to misclassification errors, particularly when similarities with other genres exist.

A. *Accuracy*

Measure the proportion of correctly predicted labels out of the total predictions made.

Our model achieved a balanced accuracy of 51.11% on the test dataset (Y_test in the code). The model can predict 51.11% of the given data correctly, which means it is learning the significant patterns from the training data.

Initially, the model was trained with data without PCA, which gave an accuracy of 28.6%, but post-PCA (dimensionality reduction), the accuracy jumped to 51.11%. This suggests that PCA played an important role in converging the weights correctly, leading

B. *Time to converge*

This describes the time taken by the algorithm to stop the weight update process, after reaching the optimal weights.

The algorithm took 733.7793726921082 seconds to converge with **no. of samples** of data at 700, no. of **features** at 16, **mu** at 0.00151, **lmb** at 0.0151, and **epsilon** at le-2.

The time of convergence varies based on the three hyperparameters. When mu & lmb are increased, the time to converge also increases. On the other hand, as the epsilon increases the time to convergence decreases.

At epsilon = le-2, mu=0.00151, and lmb=0.0151 the Time to convergence = 733.7793726921082

At epsilon = le-1, mu=0.00151, lmb=0.0151 the Time to convergence = 318.79377269744873

C. *Prediction Time*

This tells about the time taken by the algorithm to predict the probability of the test sample.

Our model takes a time of 0.00049614906311 seconds to predict the probabilities using updated weights.

D. *Effects of Learning Rate(mu) on Performance*

The learning rate (mu) is a hyperparameter that controls the step size at which the weights are updated during the gradient ascent algorithm. We experimented with different values of the learning rate to observe its effects on model performance. Here are our findings:

- With a learning rate of 0.00151, the model converged after 341.7869915962219 seconds, achieving an accuracy of 51.30%.
- When we increased the learning rate to 0.00152222, the model converged in less time after 256.51291680336 seconds, achieving an accuracy of 51.30%.
- When we decreased the learning rate to 0.00150 the model took a little longer to converge. It took 344.96619606018 seconds. The accuracy remained like the initial learning rate, 51.30%.

E. *Results*

Upon submission, our developed model achieved a balanced accuracy/Kaggle score of 57.000%. This metric represents the average recall from each class.

While a score of 0.5111 suggests reasonable success in instance identification, there is still room for improvement to achieve a higher score.