MUSIC SIGNAL ANALYSIS: REGRESSION ANALYSIS

V. N. Aditya Datta Chivukula and Sri Keshava Reddy Adupala

Department of Computer science and Engineering, International Institute of Information Technology, Bhubaneswar, Odisha, India

ABSTRACT

Machine learning techniques have become a vital part of every ongoing research in technical areas. In recent times the world has witnessed many beautiful applications of machine learning in a practical sense which amaze us in every aspect. This paper is all about whether we should always rely on deep learning techniques or is it really possible to overcome the performance of simple deep learning algorithms by simple statistical machine learning algorithms by understanding the application and processing the data so that it can help in increasing the performance of the algorithm by a notable amount. The paper mentions the importance of data pre-processing than that of the selection of the algorithm. It discusses the functions involving trigonometric, logarithmic, and exponential terms and also talks about functions that are purely trigonometric. Finally, we discuss regression analysis on music signals.

KEYWORDS

Machine Learning, Regression, Music Signal Analysis.

1. INTRODUCTION

Regression analysis gained its importance when several statisticians found out its applications in the real-world such as predicting the price of land in a certain city, estimating the complex polynomials through working on the dataset provided, estimating whether a given medicine will work on a large amount of people, etc. [1] also gained its profound importance during the past decade with its description of solving various statistical models. [2] also came into the picture showing its influence over dealing with trigonometric functions, but, there are some areas where we need to understand the importance and need for a perfect combination of above-mentioned approaches in a simple way to enhance the accuracy of results and to understand the true efficiency of regression analysis in many other fields which recently growing with respect to the growing demand for new applications in research. Some primary variations of regression are [3-5], etc. These algorithms have their own importance individually and are application-specific. Therefore, the practical realization of technical research applications needs their respective algorithms or approaches which can better the efficiency and accuracy of the applications with the least error possible.

1.1. Motivation

This paper discusses about trigonometric regression and polynomial regression on hypothesis involving logarithmic or exponential terms to establish the importance of adding features to the dataset for better results. Thus, the paper also provides the contrast between the performance delivered by the above-mentioned methods and simple neural networks. Hence, by establishing

David C. Wyld et al. (Eds): MLIOB, SIPO, NET, DNLP, SOEA, AISCA - 2021
Pp.65-75, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111205

the context, music signal analysis is performed considering the same idea. The idea of the paper is that a proper data pre-processing step can highly reduce the error and allows us to solve problems with much more light-weight and basic methods.

2. REGRESSION ANALYSIS OF THE TRIGONOMETRIC FUNCTION

In this section we will consider a trigonometric function as shown in equation 1. To take a completely random function, we considered generating a random function. To generate a random trigonometric function, we have used the python code as provided in listing1. In the code, there is a feature list containing all features of interest. There is a single 'For' loop ranging from 0 to length of feature list. An individual is allowed to choose a range which is equal to the number of terms that are required in the end polynomial. For each iteration of the loop, we randomly select coefficient for each term and the term itself from the feature list. Then, we multiply the coefficient and store the resulting string in a list known as function. We continue the same until the loop is completed. Hence, we end up having a list of terms as strings. Finally, we join all the strings using 'join' function which results in a random trigonometric polynomial in string datatype. It should be noted that range of loop is the number of terms one desires in the end function. One other point is that, feature 'x' is not considered while generating the function as the interest of this section was to discuss a pure trigonometric function. In this section

Listing 1: Python Code for generating function with only trigonometric terms

```
feature = ['x','np.sin(x)','np.cos(x)', 'np.sin(x)*np.cos(x)']
function = []
for i in range(len(feature)):
  coef = str(np.random.choice(np.ara-
                    -nge(100)))
  term = coef + '*' + np.random.cho-
       -ice(feature[1:])
  function.append(term)
function = '+'.join(function)
function = 'y = ' + function
```

Equation (1) is the function taken to explain the importance of trigonometric features in regression analysis for this section.

$$95*np.sin(x)*np.cos(x)+37*np.sin(x)+90*np.sin(x)*np.cos(x)+45*np.sin(x)*np.cos(x) \quad (1)$$

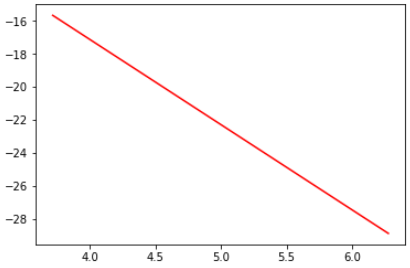

Figure 1 : plot showing predictions on y-axis with inputs on x-axis for simple linear regression

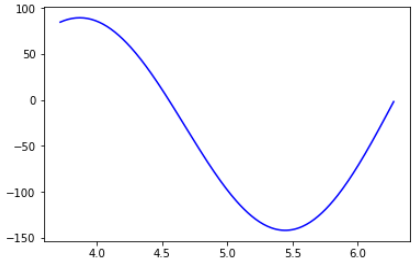

Figure 2: plot depicting desired outputs for the inputs.

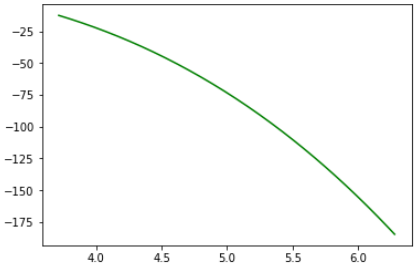

Figure 3 : plot with predictions on y-axis and inputs on x-axis for polynomial regression

If we carefully observe there are no terms with ‘x’ raised to a certain power. When we apply linear regression analysis on the dataset with input as ‘x’, where ‘x’ belongs to the range $[-\pi, \pi]$ in steps of 0.01, and output ‘y’ calculated according to the equation (1) for thousand samples, we get the graph shown in figure 1 which depicts the performance of the linear regressor on the test set, whereas the expected performance or the desired performance is as shown in figure 2. Hence, we can decide that the linear regressor performed poorly as expected. Now, if we use a polynomial regressor and consider the hypothesis degree to be 2 and train on the same training data and test it, we obtain performance as shown in figure 3. It is expected that the polynomial regressor cannot predict the trigonometric terms as there is no feature which is trigonometric in nature. Now, one can always think about using a simple neural network [6], but that also would not work as the training set is too low for the neural network to generalize the trigonometric hypothesis and training the network excessively for a greater number of epochs would result in overfitting of data and also does not assure accuracy. We can also try with [7] but, we should not forget the fact that LSTM networks require

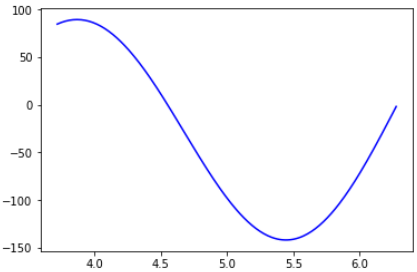

Figure 4: plot with inputs on x-axis and predictions by simple linear regression after adding trigonometric features to the dataset on y-axis.

a high amount of data and moreover are computationally expensive as compared to the simple neural networks and regression analysis discussed above. Now, if we closely look at the situation and introduce the trigonometric terms in the hypothesis considered in the case of simple linear

regression as redefined according to equation (2) and train on the dataset with a new hypothesis and apply linear regression analysis then we can observe the performance as shown in figure 4. Thus, by looking at figure 2 and figure 4, we can understand the importance of trigonometric features in linear regression provided the dataset has a trigonometric relationship. We can even look at table 1 to just checkup on the errors obtained with each regression approach discussed. Generally, trigonometric regression analysis need can be observed in the fields like signal processing and wave analysis. We are going to continue this idea as polynomial trigonometric regression in section 3 which actually makes us think to consider adding trigonometric features as a primary data preprocessing step whenever we encounter with regression analysis problems.

Table 1 Error table for pure trigonometric function by different algorithmic approaches.

ALGORITHM	ABSOLUTE ERROR
Proposed Approach	6.610267888618182e-12
Linear Regression	18573.351509906905
Polynomial Regression	15689.82990204867

3. REGRESSION ANALYSIS OF POLYNOMIAL WITH TRIGONOMETRIC FEATURES

In section 2 we have discussed function having only trigonometric terms without the mixture of linear or quadratic terms in ‘x’, where ‘x’ is the input value. Consider a function as described by equation 2 in which we observe terms such as ‘x*cos(x)’ and so on, which is difficult for simple neural networks and even the simple statistical regression algorithms like linear regression and polynomial regression to learn on minimal data.

Equation 2 is generated using the code provided by listing 2. To briefly explain the algorithm, in first loop the degree of the polynomial is kept as range and all orders of input feature 'x' are included in the features list. Then, every term in the 'terms' list is included in the features list. Now, when the 'features' list is ready, a 'function' is defined, in which, an empty list 'T' is considered and the number of terms in the generated polynomial is decided at random by keeping a maximum upper-limit. Now, a loop is considered keeping number of terms as range and for each iteration a term is appended to list 'T' by generating the term with a randomly selected number of features. Finally, polynomial is created by joining the terms stored in list 'T'.

Listing 2: Python code to generate a random mixed polynomial

```
x = np.pi # buffer value
functions = []
terms = ['np.cos(x)','np.sin(x)','np.tan(x)', 'np.log(x)','np.exp(x)']
features = []
for i in range(2):
    features.append("x**"+str(i+1))
for i in terms:
   features.append(i)
# generating function
def function():
   T = []
   number_terms = np.random.cho-
                -ice(np.arange(10))+1
   for i in range(number_terms):
```

```
    num_features = np.random.cho-
            -ice(len(features))+1
    l = []
    for j in range(num_features):
       l.append(features[np.random.cho-
       -ice(np.arange(len(features)))])
    t = '*'.join(l)
    T.append(t)
  func = '+'.join(T)
  func = 'y = ' + func
  return func
```

Hence, here too we can add the additional features which include trigonometric, logarithmic and exponential features in 'x' and also consider all permutations possible once the individual estimates the degree of polynomial the learning hypothesis would belong to in the same way as we do in case of normal polynomial regression. If we carefully observe figure 5 which depicts the predictions by support vector regression trained on dataset with inputs ranging from $-\pi$ to π and outputs calculated according to equation 2, we see that the expected plot as in Figure 6 is completely different from what has been predicted which leads to high absolute error on test set. When we apply polynomial regression analysis keeping the degree as 2, then also we can see that the plot by polynomial regression as depicted in figure 7 is mostly off in predicting the desired outputs as shown in figure 6.

$$Y=[e^{x}*\cos(x)*\tan^{2}(x)]+[x^{3}*\sin(x)]+[x^{3}*\tan(x)*\sin(x)*\log(x)]+[x^{2}]+[x^{3}*\cos(x)*\tan(x)*e^{x}*\log(x)] +[e^{x}*\tan(x)*x^{4}] \quad (2)$$

Hence, if we are able to actually consider the list of additional features which are all possible permutations of 'x' with trigonometric, logarithmic and exponential functions acting upon it and then apply linear regression analysis, we observe the desired plot as in figure 8 which is almost similar to actual relationship showcased in equation 2. If we compare figure 6 and figure 8, we can understand that the simple addition of all combination of functional features can affect the performance of an algorithm by a great extent. Table 2 depicts the errors obtained by discussed algorithms.

Table 2 Error table for polynomial with complex terms by different algorithmic approaches

Algorithm	Absolute Error
proposed approach	27.97901221743491
Support Vector Regression	14177902477532.947
Polynomial Regression	15.715957e+12

If one thinks that the number of permutations is increasing with degree of the hypothesis then he can apply dimensionality reduction techniques such as principal component analysis and thereby decreasing the computational time taken. This approach is only successful when the input is related to output with assumed combinations of features, in this case which are trigonometric, logarithmic and exponential. We can also analyze data in preprocessing stage to identify more complex functions as features in 'x' depending upon the dataset.

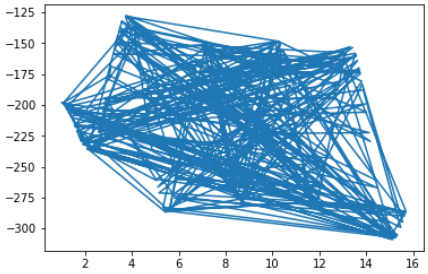

Figure 5: plot depicting predictions on y-axis and input value on x-axis by support vector regression

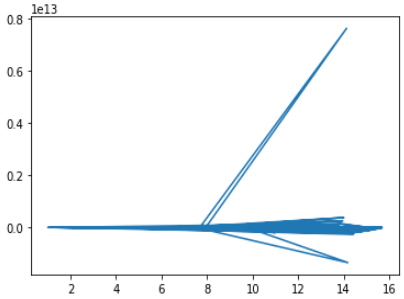

Figure 6: plot depicting expected outputs on y-axis for inputs on x-axis

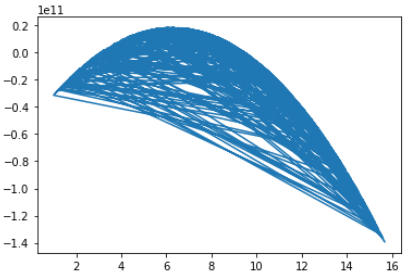

Figure 7: plot depicting predictions on y-axis for inputs on x-axis by polynomial regression

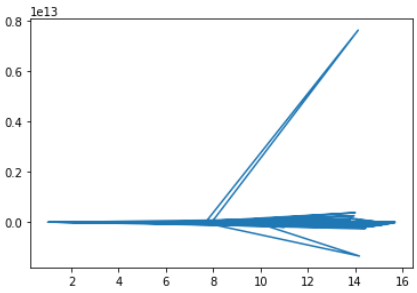

Figure 8: plot depicting predictions on y-axis for inputs on x-axis by linear regression after addition of features discussed in section 3

4. MUSIC SIGNAL ANALYSIS

Music signal is one of the complicated signals out there and definitely making an machine learning algorithm to learn from it and make it figure out parameters such as amplitude,

frequency and phase is a difficult task as the superposition of several sinusoidal waves change after very short amount of time over complete time interval, but, if we assume that there are only a constant number of waves superposed over each short time frame and consider a superposition as shown in equation 3, then we can optimize the parameters using many optimization algorithms out there. In this case we have taken gradient descent algorithm to optimize which is simple to understand and apply. Here, we considered a random background music track [8] for explanatory purpose and considered first 800,000 samples of the audio amplitudes from left channel, then, we have further divided the entire training set into 800 segments with each containing 1000 samples. These 1000 samples are trained thereby, optimizing the parameters in hypothesis which are amplitude, frequency and phase of each of the constant number of waves considered, here we assumed the constant value to be 20 for explanatory purpose. This summarizes the problem as to optimize the parameters frequency, amplitude and phase of each of the 20 waves in that particular time frame of 1000 samples using gradient descent assuming the step size as 1 and considering squared error as loss function.

$$y = \sum_{i=0}^{20} a\_i * sin(2\pi(f\_i * x + phase\_i)) \quad (3)$$

a_i – amplitude parameter of i^{th} wave
f_i - frequency parameter of i^{th} wave
phase_i – phase parameter of i^{th} wave

One can always experiment upon different optimizing algorithms and consider different values for the hyperparameters mentioned according to the audio data they have. We have also normalized the time frame values which act as input by dividing each value on time axis with 44100 and then subtracting the mean from the input array and finally dividing it with the standard deviation. Two approaches have been followed to actually perform regression analysis as described above. The first approach is simple way of optimizing all the parameters of a particular time frame simultaneously at each step of gradient descent [9], but, this method forces the waves to learn independently of each other which results in same optimized parameters for each wave, that is, for example if frequency is 1, amplitude is 1 and phase is 0 for the first wave in the hypothesis after optimizing, then, the each of the remaining 19 waves of that time frame will also have the same values for frequency, amplitude and phase respectively. From first approach one can easily understand that the conventional form of regression analysis cannot be performed for music signal and hence, we have considered a second approach which is to optimize the second wave with respect to first, third with respect to second and first, and so on, similar to cost functions described by Algorithm 1.

As shown in the algorithm 1 we can update array ‘h’ which stores the superposition values of all ‘i’ number of waves when optimizing ‘i+1’ wave’s parameters, so that, the superposition value can be added to redefine cost function for each wave pertaining to the same time frame and thereby, optimizing the parameters of each wave with respect to the values obtained by the superposition of previous waves. The figure 9 represents the plot between the desired amplitudes and the time, and, figure 10 shows the plot obtained by the hypothesis considered, which is the superposition of 20 sine waves, but, the plot as in figure 10 is obtained by calculating amplitudes according to equation 4, where we did not consider amplitude parameter of each sine wave of that time frame as they were not even close to the desired values and scaling up the error by large extent which can be observed in figure 11. This is a drawback currently but, if followed a different technique of optimization for amplitude parameter, then definitely we can make this approach work.

$$y = \sum_{i=0}^{20} sin(2\pi(f\_i * x + phase\_i)) \quad (4)$$

One important thing is that, for optimizing amplitude or frequency or phase we considered the gradients as follows:

$$Ga = step * (h + a\_i * sin(2\pi x) - y) * (sin(2\pi x)) \quad (5)$$
$$Gf = step * (h + sin(2\pi * f\_i * x) - y) * (2\pi x * cos(2\pi * f\_i * x)) \quad (6)$$
$$Gp = step * (h + sin(2\pi x + 2\pi p\_i) - y) * (2\pi * cos(2\pi x + 2\pi p\_i)) \quad (7)$$

Ga – amplitude gradient, Gf – frequency gradient, Gp – phase gradient

---

Algorithm 1: Optimization

Input: data x, size n; amplitudes y, size n; step s; starting index of time frame start
Input: parameters parameters, size (20, 3)

Initialize h = array(zeros(1000))

for k=0 to 19 do
for j=0 to 9 do
for i=start to start+1000 do
Assign Ga=step*(h_i+ parameter s_{k0}*(sin(2πx_i)-y_i)*(sin(2π*x_i))
Assign
Gf = step*(h_i+sin(2π*$parameters_{k1}$*x_i)-y_i)*(2π*$x_i$cos(2π*$parameters_{k1}$*x_1))
Assign
Gp = step*(h_i+sin(2π*x_i+2π*$parameters_{k2}$)-y_i)*(2πcos(2πi*x_i+2π*$parameters_{k2}$))
Assign $parameters_{k\_0}$ = $parameters_{k\_0}$-Ga
\STATE Assign $parameters_{k\_1}$ = $parameters_{k\_1}$-Gf
\STATE Assign $parameters_{k\_2}$ = $parameters_{k\_2}$-Gp
\ENDFOR
\ENDFOR
for v=start to start+999 do
Assign w = vmod1000
Assign
h_w = h_w + ($parameters_{k0}$ * np.sin(2 * np.π * ($parameters_{k1}$ * x_v + $parameters_{k2}$)))
ENDFOR
ENDFOR

Where, we only consider the effect of the parameter for which we compute the gradient. For example, while computing the gradient for amplitude parameter we make f_i as 1 and p_i as 0 and hence, we are optimizing only amplitude with respect to the samples, which is to try fit amplitude parameter for that wave for that time frame completely. Similar pattern can be observed for frequency where a_i is made 1 and p_i as 0 and in case of phase gradient a_i is 1 and f_i is also one. This can be understood as independent parameter training for which we got the results as shown in figure 10. We have also considered dependent parameter training where we try to optimize one with respect to other, for which the gradients are as follows:

$$Ga = step * (h + a\_i * sin(2\pi * f\_i * x) - y) * (sin(2\pi * f\_i * x)) \quad (8)$$
$$Gf = step * (h + sin(2\pi * f\_i * x) - y) * (2\pi x * cos(2\pi * f\_i * x)) \quad (9)$$
$$Gp = step * (h + a\_i * sin(2\pi * f\_i * x + 2\pi p\_i) - y) * (2\pi * a\_i * cos(2\pi * f\_i * x + 2\pi p\_i)) \quad (10)$$

Ga – amplitude gradient, Gf – frequency gradient, Gp – phase gradient

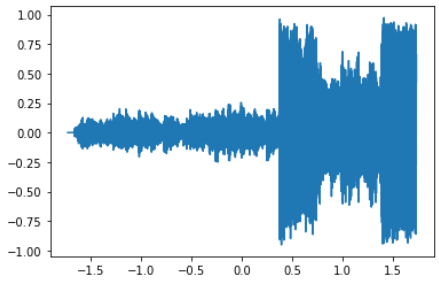

Figure 9: plot original audio data with desired amplitudes on y-axis and time period on x-axis.

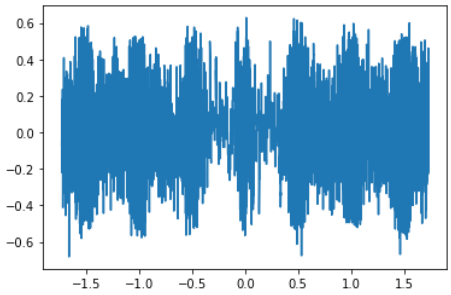

Figure 10: plot with predicted amplitudes on y-axis and time period on x-axis by following independent parameter training excluding amplitude parameter.

where, frequency is computed independently and amplitude is computed with respect to frequency parameter and finally phase parameter is computed with respect to frequency and amplitude parameter. For, dependent parameter training we observed a higher loss and hence, currently independent parameter training is better. On an important note, as we have not predicted the amplitude parameter for 20 waves of each time frame properly, we have divided the final value by 20 which should be the mean amplitude at that particular instant. The plot observed for dependent parameter training can be observed in figure 12 and we can also observe figure 11 in which we calculated amplitudes considering amplitude parameter for each of 20 waves in that time frame and clearly decide why we did not consider amplitude parameter.

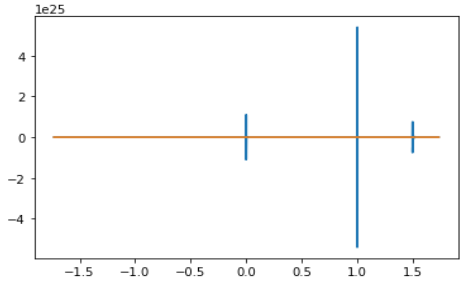

Figure 11: plot with predicted amplitudes on y-axis and time period on x-axis by following independent parameter training including amplitude parameter where horizontal plot represents original signal.

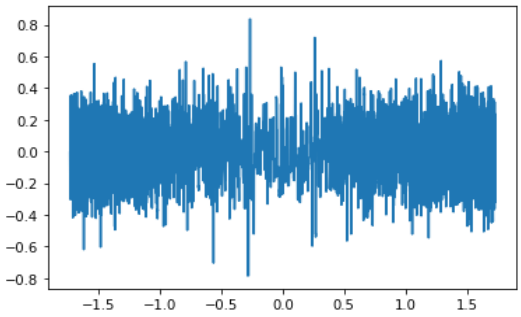

Figure 12: plot with predicted amplitudes on y-axis and time period on x-axis by following dependent parameter training.

5. CONCLUSION

Regression algorithm is the most fundamental and important algorithm which can be powerful when hypothesis, optimization and features are selected properly. It has the potential to even perform better than the current advanced machine learning techniques. With this theory we try to propose that, as algorithm selection is important for an application, similarly, data pre-processing and hypothesis reformulation is also that important. We need to focus on formulating the underlying functions in pre-processing stage itself so that even on less amount of data, the algorithm can perform much more efficiently and we can eliminate the risks such as underfitting or overfitting. This also specifies that we need to conduct more experiments with each algorithm by reformulating some of its parts on the dataset, so that, we can understand some of the relationships in the dataset and even have a combination of different machine learning algorithms acting on same dataset which may be much more efficient, and also understand the power of interdisciplinary algorithms. This also sheds light on the fact that adding features by exploring dataset can boost algorithm's performance and efficiency.

ACKNOWLEDGEMENTS

The authors would like to thank Abhiram Reddy for his participation, assistance and his valuable time. We would like to thank our guide, Mr. Rupaj Kumar Nayak for his guidance for this work.

REFERENCES

[1] Yao, F., Müller, H. G., & Wang, J. L. (2005). Functional linear regression analysis for longitudinal data. The Annals of Statistics, 2873-2903.
[2] Eubank, R. L., & Speckman, P. (1990). Curve fitting by polynomial-trigonometric regression. Biometrika, 77(1), 1-9.
[3] Zou, K. H., Tuncali, K., & Silverman, S. G. (2003). Correlation and simple linear regression. Radiology, 227(3), 617-628.
[4] Ostertagová, E. (2012). Modelling using polynomial regression. Procedia Engineering, 48, 500-506.
[5] Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. R news, 2(3), 18-22.
[6] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press.
[7] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
[8] DJ Sakthi. (2020, April 21). Charlie Bgm Mix[Video]. YouTube. https://www.youtube.com/watch?v=nopQ6TT_pGo
[9] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

AUTHORS

V. N. Aditya Datta Chivukula is currently an undergraduate student in the Department of Computer Science and Engineering at International Institute of Information Technology, India. His area of interests are in Machine learning, Deep learning and Natural Language Processing.

Sri Keshava Reddy Adupala is currently an undergraduate student in the Department of Computer Science and Engineering at International Institute of Information Technology, India. His area of interests are in Data Analytics, Data Visualization and Machine Learning.