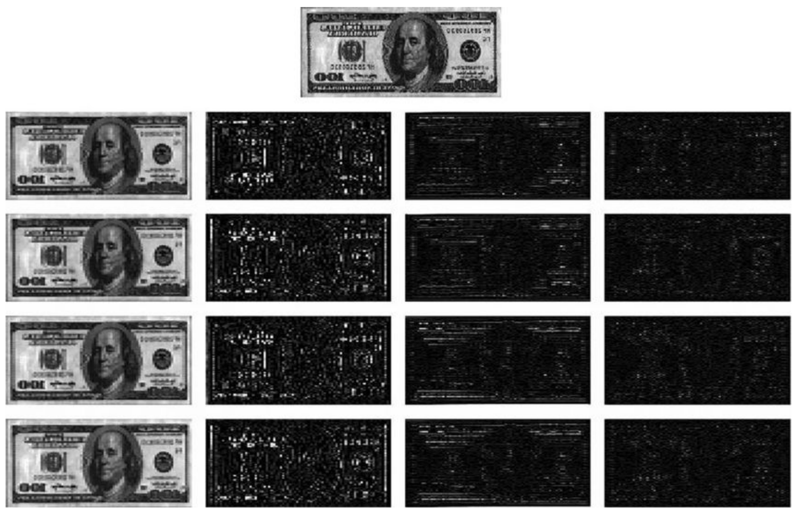
Enrique Posada Lozano
A01700711

Banknote Authenticity

Introduction:

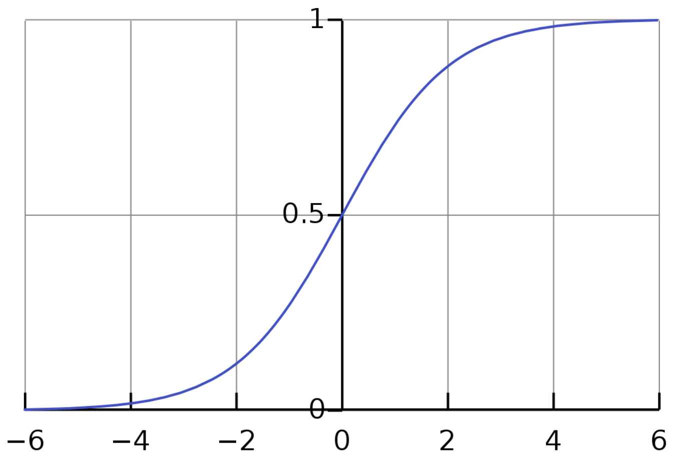
As we all surely know, money moves the world. We buy things, we sell things (or services) in an endless cycle of consumerism. Because of this, ever since the beginning of using banknotes, people have tried to create fake money (counterfeit). Banks have implemented and tried various techniques in order to identify and distinguish counterfeit banknotes. A counterfeit banknote is a replica of a real banknote that might seem real until verified carefully. A model is proposed to “solve” this problem by receiving parameters from a banknote and therefore determining whether the banknote is real or not. For this, the dataset analyzed and used to train the model comes from the UCI Machine Learning repository. The dataset was obtained by using a machine that obtains data of banknotes wavelet transformed image. The attributes within the dataset are variance of wavelet transformed image, the skewness of the wavelet transformed image, curtosis of the wavelet transformed image, entropy of the image and finally a class (indicating whether a banknote is authentic or not). To look more into this, variance is the value determining how each pixel varies from from its neighboring pixels, skewness gets a value saying how symmetrical is the banknote, curtosis is a value that measure whether the data is heavy-tailed or light-tailed and entropy is a value that determines the quantity of information that must be coded for (done by a compression algorithm).



Logistic Regression:

A logistic regression model is proposed to classify information of banknotes in whether they are authentic or they’re not, with a zero indicating a banknote is counterfeit and one indicating the banknote is authentic. Initial values for the parameter coefficients are insitalized. Values are predicted by following the formula : `yHat = 1.0 / (1.0 + e ^`

`(-(b0+b1*x1...+bn*xn)))`. As previously mentioned, the dataset contains the attributes Variance, Skewness, Entropy, Curtosis and Class from which the features (parameters) are Variance, Skewness, Entropy and Curtosis in order to predict the Class of a banknote.
Throughout the process, Gradient Descent is applied in order to diminish error and increase accuracy. The following formula was programmed in order to update the coefficient values and obtain more precise predictions: `newParams = params - learning_rate * (1/len(features) * acc)` where acc is the dot product of the error and the features of the model. The error is obtained by calculating the difference between the calculated values (known as the hypothesis) and the expected values. In order to calculate the hypothesis, an activation function is needed and the one chosen for this implementation was the sigmoid function which is shown below only for general knowledge purposes.



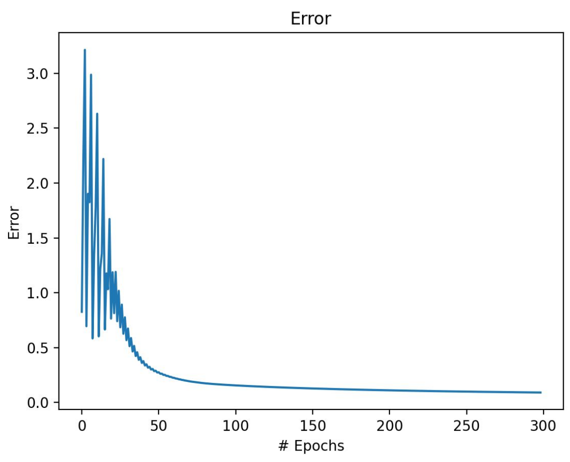
Sigmoid Activation function

As for model performance, the loss function to measure the performance of the classification model is Cross-Entropy, which consists of the following formula : `-(ylog(p)+(1-y)log(1-p))` and log here is a natural logarithm, and at the end all calculated values are added up. After applying the loss function, the mean squared error is calculating by having the calculated total error divided by length of the the dataset.

Model Validation and Results:

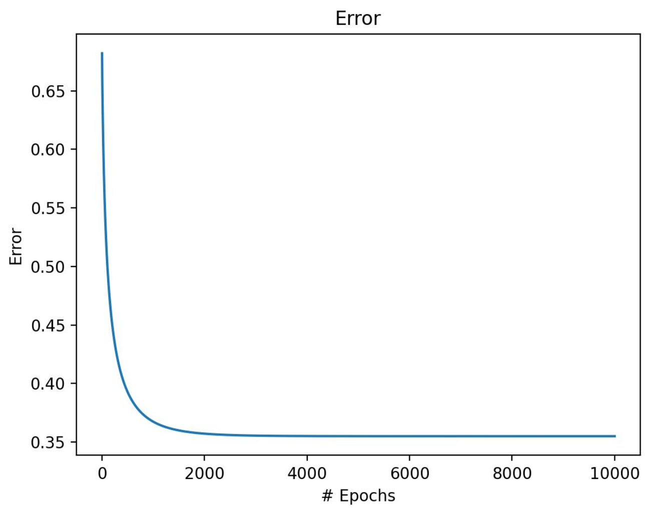
In order to make predictions, the model is trained by splitting data into a training subset consisting of 25% of the data and a test subset with the rest of the data. Each time the code is executed, data is randomly split between the two subsets. Then, it was originally intentioned to train the model with a learning rate of 0.01, however it was scaled up to 0.3 in order to reduce error in much faster manner. As far as results will show, the learning rate had to be pumped all the way up to 3.8, but was chosen to remain at 3.5, a big learning rate that provides quick learning for the model. Using a learning rate bigger than 4 would start bring divergence from the data instead of convergence, providing a result like the following graph demonstrates.

Enrique Posada Lozano
A01700711



Error with a learning rate of higher than before creates the following error, causing a convergence until at least 50 epochs have passed.

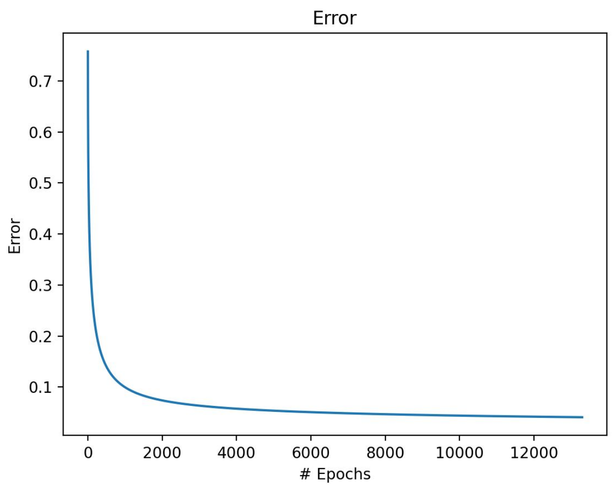
The training of the model can conclude in only one of three ways : either by reaching a given number of epochs (in my case, 1000 initially and now 10000 epochs), having no change between the previous coefficients and the new calculated coefficients, or having the error at a minimum (initially 0.1, now 0.05). One of the main problems initially with the model was not adding the bias. After having the model work, the minimum error reached no matter how many epochs was around 0.35, thus a column for the bias was added as well a new coefficient for the bias.

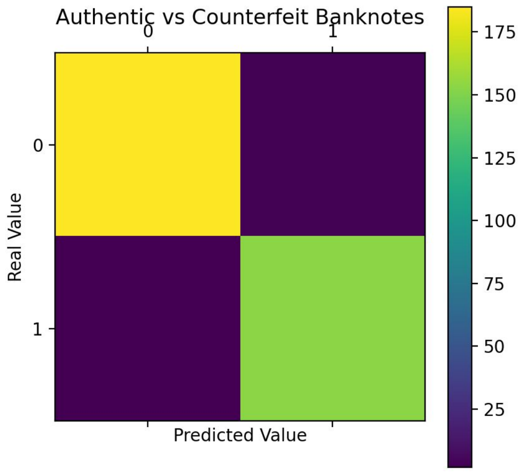


Error before insertion of Bias

Adding the Bias adjusted data significantly, dropping the error up to 0.04 and having an accuracy around 97%. In order to compare the results of the model, *scikit* linear model library was implemented in order to calculate Linear regression, and have an optimized and trained model show the accuracy it had, which goes up to a minimum of 98% approximately.
Having now a functional model that does provide results given random data to train and test, there was a second problem, the performance of the algorithms. In order to work with the dataset, the program relies heavily on pandas and numpy, and so both libraries rely with their own operations for better optimization. Since Dataframes were being used and the computations were being done on each individual instance of the dataframe, having to

search and access data, much worse being calculated inside a nested loop made things real slow, therefore numpy comes to a good use because it can make quick calculations on pandas dataframes. Therefore all code had to be redone and adapted for numpy, making the program at least 10x faster than before.





As one can see, using a learning rate of 3.5, the optimized version of the algorithm provided an error of 0.04 in a minute doing 13298 epochs , compared to 8 minutes for just 1000 epochs. An error of 0.04 was chosen as a minimum because data then tends to reach a global minimum around 0.035, and going further just seems to give the model overfitting, missing a little of accuracy in the test data. Besides, going below this number doesn't seem to greatly improve performance since accuracy will at much remain at around 98% for the test data. The best result obtained from the model was by having an accuracy of 98% on training data, but having a 99.32% of accuracy in the test data, misclassifying only 7 predictions out of 1029 instances. one can see the amount of elements that matched the correct output in the Confusion Matrix provided above, and had only 5 misclassifications of over 343 instances.

Conclusión:

As a way to close this report, I can only conclude on the value that optimization has on computation and how Bias can quickly change the outcome. If one seems to get stuck on on not reaching a global minimum, it's probably because of the bias. Then, another thing to recall is the fact that to always seek optimization, because things seem simple running once or twice, but training a model provides a way of start digging into speeding up a model's performance and coding overly.

References:

http://archive.ics.uci.edu/ml/datasets/banknote+authentication
https://github.com/QuiquePosada/BanknoteAuthentication_LogisticRegression_ML
https://en.wikipedia.org/wiki/Feature_scaling

Enrique Posada Lozano
A01700711

https://www.semanticscholar.org/paper/Analysis-of-Banknote-Authentication-System-using-Shahani-Jagiasi/6cdb81495a825d102a665493e595d8c06883b07e
https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html