**Explainability**

**How I approached this:**

1. This task was very straightforward when we talk about the model’s implementation here.
2. First was understating the data, so after importing the files, I tried to understand how the data looked like, if there was any empty field or not.
   1. The data looked clean and there was not any of the field which had any of the null values in the train data.
3. Then after importing the files, I then applied the following functions.

**def DT\_explain(train\_file,test\_file):**

1. After segregating the descriptive features and target feature, I went ahead with training the data first with the default parameters and with normalization, without normalization and with standard scalar.
2. I also used the model in the same function to print not only accuracy but precision, recall and f1 score for both the train and test data and see how the results were.
3. I then used the plot\_tree function to show the data directly but the result for the image had text overwritten in it and it was difficult to read.
4. So, instead of using only plot\_tree alone, I used the function with the mathplot lib to plot the image and after some tweaking got a good result.
5. This function was straightforward, and I didn’t encounter any issue other than finding the correct parameters for the image to be clear and crisp.

**def LR\_explain(train\_file,test\_file):**

1. I had worked with this a lot, so understanding this was not an issue.
2. I followed the same principle of the segregating the descriptive features and target feature described above.
3. I chose to go ahead with the default parameters but got error which was because the number of maximun\_iterations were low. Now because of this, I chose the max\_iterations as 1000 and it was working perfectly.
4. In this function the process involved same for fitting, the model, and predicting the result as dt\_explain. The only change was changing the classifier and printing the weights of the feature which helped me with the model explainability. Which we’ll discuss later.

**def MLP\_explain(train\_file, test\_file):**

1. This one was a bit complicated to implement.
2. I first went ahead with changing some of the default parameters like choosing one input layer, 1 hidden layer and 1 output layer, with input layer having 4 neurons because of 4 input features and then 2 neurons for the 1st hidden layer and then one single neuron for the outpout layer. I also changed the optimizer to the lbfgs, as this works really well for the MLP classifier, I also changed the random state to 1 rather than none and decreased the learning rate to 1-e5 from 1-e4(default value) becuase I wanted the model to update the weights slowly rather than quickly, which may lead to weights not being optimized. I kept the max\_iterations as default value which is 200 but encountered the same problem of max\_iterations convergence like I experienced with the LR\_explain. So, I used max\_iterations as 1000.
   1. Max\_iterations are nothing but how many times the model should try to update weights before completing the training.
3. Then the complication came with seeing the weights. As I know that the explainability decreased with the complicated model, but I tried my best to understand this by printing out all the weights in each of the neuron at all the layers.
   1. But this was a bit hard to gather the explainability, so I then printed the name the features along with the name of the weights for the features that is being sent to the 1st hidden layer. This helped me gain more clarity on the weights.

**Metrics Used for the calculations and what they mean:**

• **Accuracy:** Represents the proportion of correctly classified instances, offering a straightforward measure of the model’s overall performance.

• **F1-Score:** Provides a balance between precision and recall, making it especially useful for imbalanced datasets by minimizing both false positives and false negatives.

• **Precision:** Measures the proportion of correctly predicted positive instances among all predicted positives, reflecting the reliability of positive classifications.

• **Recall:** Evaluates the proportion of actual positive instances correctly identified by the model, emphasizing its ability to capture all relevant cases.

**Detailed Analysis and Results:**

**Logistic Regression:**

**Figure 1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Logistic Regression Model and Variation | Accuracy | Precision | Recall | F1 Score |
| LR with Basic Normalization across rows (Test Data) | 0.789 | 0.818 | 0.767 | 0.792 |
| LR without Normalization(Test Data) | **0.970** | **0.967** | 0.975 | **0.9715** |
| LR with Standard Scalar(Test Data) | **0.970** | **0.967** | 0.975 | **0.9715** |
| LR with Basic Normalization across rows(Train Data) | 0.816 | 0.833 | 0.794 | 0.813 |
| LR without Normalization(Train Data) | 0.965 | 0.958 | 0.972 | 0.965 |
| LR with Standard Scalar(Train Data) | 0.966 | 0.956 | 0.978 | 0.967 |

Figure 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | LR with Standard Scaler | LR without Normalization | LR with Basic Normalization across rows |
| Salary | 0.9003 | 0.0000 | 0.9820 |
| Age | 0.0802 | 0.0025 | 0.0350 |
| Credit Score | 7.1995 | 0.0858 | 1.1641 |
| Debt | -1.7650 | 0.0001 | 7.4074 |

**For Test Data(Refer Figure 1 and 2):**

The results for the Logistic Regression model are straight forward. The way I tested for the Logistic Regression model was with three variations, one by using normalize function, which normalizes the data along the row. This is good, but with this I believed that the result for the model won’t be good as the row data for each example would be different and this won’t be a great way for the model to be trained on, and not to mention this will be different for the test data as well. This assumption was proved correct when I saw the accuracy, precision, recall and F1 score for this was very low. Then, I tried without normalization and saw that the model performed amazingly well, when comparing this with the normalized data. I then chose to use Standard Scalar, which proved to be better than the normalized data, as the data was normalized across the mean of the train data and by a variation of 1. The same transformation was applied to the test data as well. This resulted in the consistent normalization across all the train data and test data. The result for this though was exactly equal to the model trained and tested on non-normalized data.

Talking about the explainability, I can see why the model for normalization for rows (basic normalized function) failed to produce good result. The weights it considered for salary and age was like the weights of normalization using the standard scalar (best performing normalization), but it failed in giving the higher weight to the credit score and then failed miserably when it gave very high positive value to the debt. This led to debt being the most important feature for its prediction which led to it producing bad results. When talking about the LR model where I didn’t use any type of normalization, the weights was given very low to the debt, but the credit score was given a very high value which would’ve led to great result, but one thing to notice is it gave 0 value to the salary. **Based on this we can infer that the credit score is the column which holds the most importance in predicting the approval or denial of the loan.**

**For Train Data (Refer Figure 1 and Figure 2):**

The result for train data, is though not advised to generate the inference for the model, as it will produce good result on the data that it has already seen before. This is proved by the model trained on the basic normalization; the model showed jump across all the metrics. This however cannot be said about the model that was not normalized or normalized using standard scalar. With this the model showed similar but low accuracy but had a bit better recall. Now as the model was able to generalize well which can be seen with the result from the test data it did so on the train data as well. But like I said, the metrics from the train data makes no sense when evaluating the model.

**MLP Classifier:**

**Results:**

|  |  |  |
| --- | --- | --- |
| **Metric** | **Test Data** | **Train Data** |
| Accuracy | 0.973 | **0.975** |
| F1 Score | 0.974 | **0.975** |
| Precision | 0.966 | 0.959 |
| Recall | 0.982 | **0.992** |

**Input Layer features weights**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Neuron 1** | **Neuron 2** | **Neuron 3** | **Neuron 4** |
| Salary | 0.044163 | 0.002078 | -0.716486 | 0.956755 |
| Age | 0.167402 | -0.310329 | -0.309071 | -0.248925 |
| Credit Score | -10.561447 | -3.556751 | -5.562974 | 4.490124 |
| Debt | -0.002942 | 0.874210 | 5.611153 | -0.604041 |
| Bias | 1.996679 | 4.511827 | -2.494882 | -1.679687 |

**Layer 2 Weights (4x2) – 4 values(from input layer neurons) converging to 2 neurons of hidden layer**

|  |  |
| --- | --- |
| **Neuron 1** | **Neuron 2** |
| -2.109667 | 2.757478 |
| -0.125402 | 0.258952 |
| -3.411860 | 1.981589 |
| -0.999119 | -1.149273 |
| -3.040283(Bias) | 1.749781(Bias) |

**Output Layer Weights (2x1)- 2 Values (from hidden layer neuron) converging to one neuron to produce the result**

|  |
| --- |
| **Neuron 1(Output Layer)** |
| 1.441799 |
| -4.501193 |
| 12.422193(Bias) |

The results for the MLP Classifier are quite interesting, but understanding the model explainability was a bit tricky compared to Logistic Regression. The way I tested the MLP model was by using a simple architecture with 4 input neurons, a hidden layer with 2 neurons, and an output layer with 1 neuron. I trained the model using standardized data to ensure that both the train and test data were scaled consistently. The evaluation metrics—accuracy, precision, recall, and F1 score—were all quite high, which shows that the model is performing well. But when it comes to explaining why the model is making certain predictions, things get a bit complicated.

Looking at the input layer, it was somewhat easier to interpret the feature importance. When I analyzed the first layer’s weights, I could see how features like salary, age, credit score, and debt were contributing to the model’s decision-making. **For example, credit score had a high weight, which makes sense because it plays a big role in loan approvals. Debt, on the other hand, had a strong weight as well, which also aligns with what we expect in real-life scenarios. So, at this stage, I could infer that the model was learning in a way that made sense.**

However, things got much more complicated when I looked at the hidden layers. The weight values in the hidden layers didn’t directly correspond to the original features anymore, which made it tough to figure out how much influence each feature had on the final decision. This is because the weights in hidden layers are not directly tied to the input features but rather represent complex interactions between the neurons. The hidden layers essentially transform the input into more abstract representations, which makes it difficult to explain why the model is making specific predictions. Some of the hidden layer weights seemed a bit non-intuitive, which further complicates the explainability.

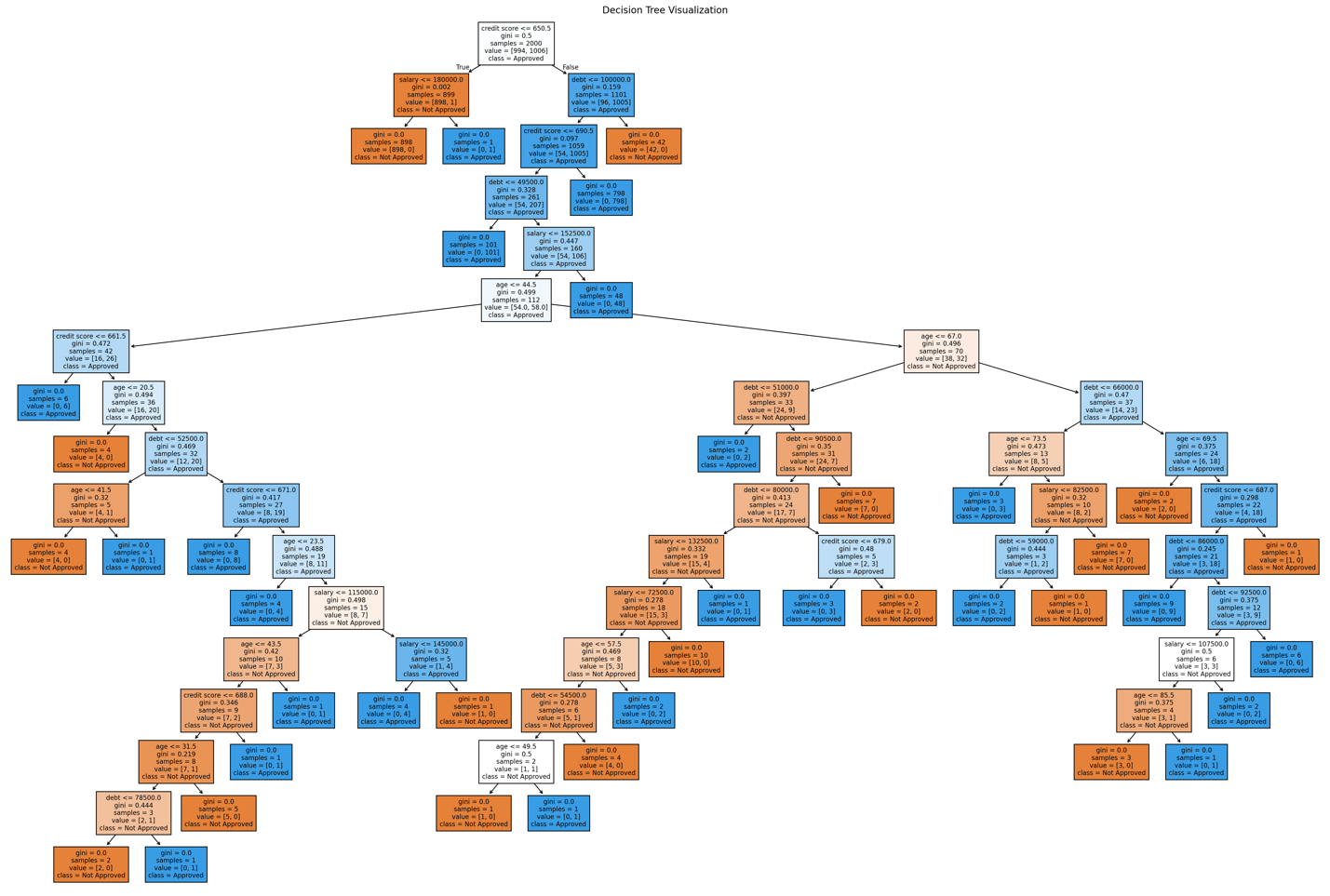
Even though explaining the hidden layers was tough, I was still able to get some insights from the final output layer weights. These weights showed which hidden features were playing a major role in the final decision. But again, the challenge was in mapping these insights back to the original input features, since the hidden layers distort those relationships in a way that’s hard to interpret.

One thing I noticed is that while the model is producing really good results, explainability is still a major challenge. Understanding how the features affect the output at the input layer level is manageable, but once we get deeper into the network, it becomes difficult to track how the weights converge and interact. This makes it harder to trust the model completely, even though the results look good.

**Decision Tree:**

**Results:**

|  |  |  |
| --- | --- | --- |
| **Metric** | **Test Data** | **Train Data** |
| Accuracy | 0.970 | 1.0 |
| F1 Score | 0.971 | 1.0 |
| Precision | 0.966 | 1.0 |
| Recall | 0.977 | 1.0 |



The explainability comes easiest with the Decision Tree out of all the models in this. With decision Tree we can visualize the path it would’ve too to make the decision it made (as can be seen from the image above). The division of the nodes into true of false based on a condition can help us guide us on the direction to decide. Here the different features are divided into forming the decision tree, that later splits into the two nodes into a left node and right node based on some condition. **But the feature which causes the first split provided the highest information gain and is the reason why it’s chosen as the root node. In this model, the most important feature is the credit score.**

One thing to note that out of all the three model only decision tree was the one which got 1.0 result across all the metrics with the train data. This was because the decision tree grew whole and saturated all the condition based on the train data and thus gave us metrics of 1. It though was able to generalize well with the test data as well, but according to me, if the data would’ve been bigger, it would’ve costed us computation and probably would’ve led to overfitting as well.

All in all, explainability was greatest with the decision tree with the visualization and then understanding how the model made the decision it made.

**Conclusion:**

As expected, out of all the three models the decision tree gives the highest explainability. With the image of the formation of the tree, the visual aid helps to understand the path the tree took in taking its decision. Somewhat bit complex with respect to decision tree but still very easy to understand model was Logistic Regression Model. The weights helped us understand which feature was the most prominent feature in its decision making. In contrast, when we talk about the MLP Classifier, this becomes a challenge. First, we can try to understand somethings from the initial layer, but this becomes way more challenging when we talk about the hidden layers that are inside the MLP Classifiers. Now in this example we had 4 neurons for the input layer, 2 for the hidden layer and 1 for the output layer. Now even with a model this simple, it became difficult for us to understand, and this amplifies by a lot when we talk about the Al models used in real life. These models hav a lot of neurons per layer which are interconnected by neurons from hundreds of neurons from previous layer. This is where I think the explainability becomes next to impossible. Theoretically you can do calculation of each weight, assign blame to each neuron for the result, but it’s very complex.