Chapter 5

Credit Case Study

5.1 Problem Statement

Companies which operate in the credit environment want to see returns on their investments. Before a loan is granted to a prospective client the appropriate risk analysis needs to be performed. In many instances this is a manual process where a credit employee has to assess a clients credit portfolio and ascertain if they are eligible for a loan and if so, on what terms. As most manual processes it can be slow and prone to human-error. By adopting machine-learning we are able to eliminate the human risk factor and significantly increase the speed. Linear modelling techniques such as *Logistic Regression* are the preferred choice due to them being inherently interpretable. Interpretability is a strict requirement due to the need to understand why the model is making the decisions that it does. Linear models are however very limited in what they can do and by using Neural Networks or other nonlinear models we may be able to significantly increase the accuracy at identifying risk, however such techniques are generally seen as *black-boxes*. In this chapter we use SHAP in order to gain some interpretability into credit risk Neural Networks so that they could be considered as possible solutions.

5.2 The Data set

Up until this point we have used our tools on *toy* examples where the data has been engineered to work well with machine-learning. In this chapter we will be using a proprietary credit risk dataset which has been provided to us by *Praelexis* which aims to simulate actual features used by credit companies to calculate potential client risk. Our dataset consists of a total of 200000 client samples where 28768 defaulted on their loan and the remaining 171232 did not. As we can see we have a heavy class imbalance in the data with a ratio of roughly 85 : 15. In practice this ratio of imbalance can be a lot higher [54]. The trained model may be skewed to more likely predict the most present class. How we solve this issue is discussed in Section 5.6.4.

5.3 Aim

Our aim is to see if we can provide enough interpretability into a credit Neural Network so that it may be considered as a possible alternative to linear models by credit companies looking to adopt machine-learning. In order to achieve this we train a *Logistic Regression* model which is inherently interpretable and a generic feedfoward Neural Network and compare their interpretability. The goal of these models is to predict a probability that a client will default on a loan given their input features. As we have seen in Chapter 4, SHAP is superior to LIME in every way but speed and therefore we shall use SHAP as our preferred tool for interpretation.

5.4 Weights of evidence

The weights-of-evidence (WOE) transformation [55] is a nonlinear transformation of the original variables. It is used to measure the strength of each attribute at separating good and bad loans. It can be seen as the measure of the difference in the proportioning of good and bad within each attribute (i.e., the odds of a person of that attribute being good or bad) [55]. Continuous variables are first converted to categorical variables through a process called *binning*. Using the information from the training data we calculate the WOE value for each bin. Each client is

either considered to be good (0) or bad (1). The WOE can be calculated as,

$$\text{WOE} = \left[\ln\left(\frac{\text{Distr Good}}{\text{Distr Bad}}\right)\right] \tag{5.1}$$

The steps of calculating WOE can be therefore be summarized as follows:

1. If the variable is continuous separate the data according to the distribution into multiple parts known as *bins*.
2. Calculate the number of good and bad loans in each bin.
3. Calculate the % of good and bad loans in each bin.
4. Calculate the WOE of each bin by using (5.1).
5. Replace the raw variable values in the input with the WOE value of the bin that it belongs to.

We can see an example of creating WOE bins for a variable in Table 5.1. The *boundaries* are the bin ranges and can either be manually or automatically chosen. The *Distr good* and *Distr bad* is extracted from the training dataset and by using (5.1) we are able calculate the WOE value for each bin. For example suppose that our input variable has a value of 1.5. Therefore it falls into the $(1, 2]$ bin and the value 1.5 will be replaced with the WOE value -0.07. In Figure 5.1 we can see an example of WOE binning for a binary classifier. The axis are the values of the two features. The red star and blue dot represent the 2 classes. The shaded parts are the values which will be assigned to the 2 classes after the WOE transformations. This illustrates how the WOE transformation allows us to construct a nonlinear decision boundary.

5.4.1 Benefits of WOE

- Missing values are separated into their own bins and is therefore easier to handle.
- Provides a linear relationship with log odds.

- It can treat outliers. Extreme values which fall outside the average range are given their own bin.
- We are able to determine the stability of our model by comparing the WOE values of the training and unseen data.
- Converts continuous variables into categorical variables so that there is no distinction between them.

For our dataset we have chosen to use *automatic binning* because we want the preprocessing steps to be as automated as possible. This is to prevent our domain knowledge from affecting the produced explanations.

	Bin	Boundaries	Count good	Distr good	Count bad	Distr bad	WoE
	1	$(-\infty, 1]$	1760	0.0973	798	0.2033	−0.37
	2	$(1, 2]$	5238	0.2896	1223	0.3116	−0.07
	3	$(2, 3]$	7881	0.4357	1034	0.2634	0.50
	4	$(3, \infty)$	3210	0.1775	870	0.2217	−0.22
Total			18089	1.0	3925	1.0	

Table 5.1: WOE example

5.5 Metrics

Before we discuss our models architecture we introduce the metrics with which we use to measure the performance of the model. Firstly we introduce some terminology used within the metrics definitions. To provide some context we will be using our problem of predicting whether a client will default on their loan or not. We refer to clients which will default on their loan as *defaulting clients* and those which will not default as *regular clients*. Defaulting clients that have been correctly identified by the model are referred to as *True Positives (TP)*, where as regular clients which are incorrectly identified as defaulting clients are referred to as *False Positives (FP)*. Regular Clients which are correctly identified are referred to as *True Negatives (TN)*, where as defaulting clients which are incorrectly identified as regular clients are referred to as *False Negatives (FN)*.

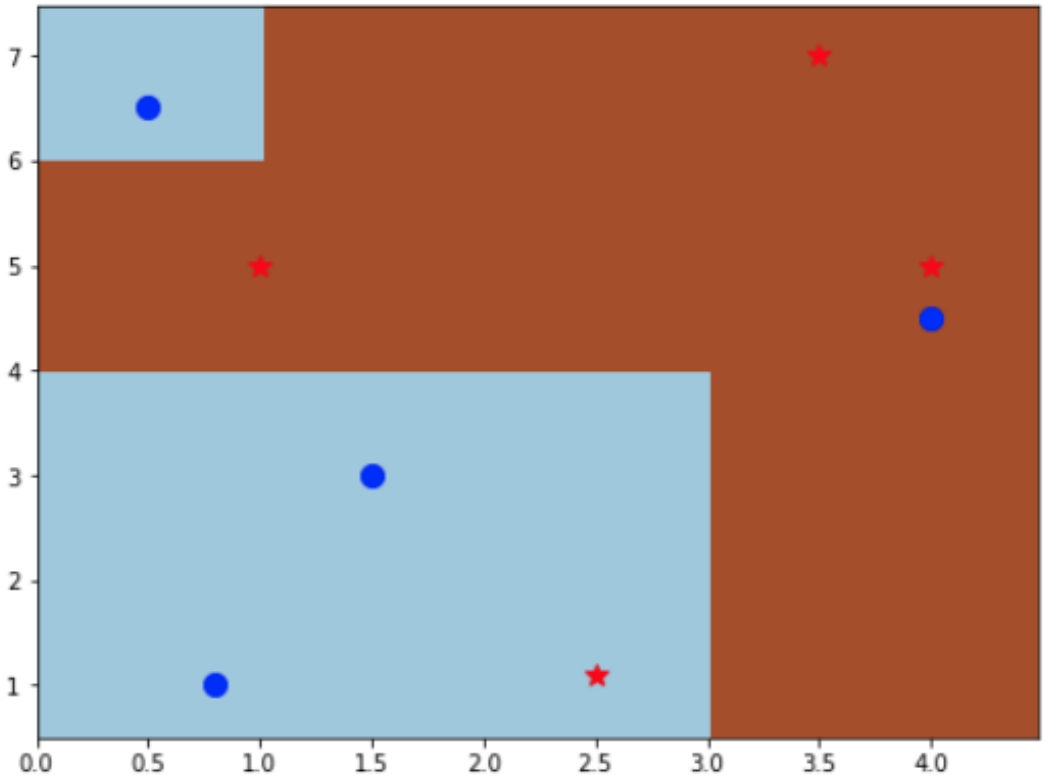


Figure 5.1: The decision boundary constructed by weights of evidence binning.

5.5.1 Accuracy

In most examples the common metric that is used to measure how well a model performs is the *accuracy* measure, however it does not work as well if the dataset has a class imbalance. Suppose in the previous example of identifying fraudulent claims that the training dataset consisted of 95% legitimate claims and 5% fraudulent claims. If we engineer the model to simply predict every claim to be legitimate then we will achieve a 95% accuracy, however the model would never be able to identify fraudulent claims and therefore does not solve our problem. Although a high accuracy does not necessarily mean a model is performing well, a low accuracy can be indicative of a poor performing model and therefore has relevance. Using the previously defined terms we can define accuracy as,

$$\text{Accuracy} = \frac{TN + TP}{FN + FP + TN + TP} \tag{5.2}$$

5.5.2 Precision

An important metric is called *precision* which is the proportion of positive identifications that were actually correct. In the previous example of identifying fraudulent claims, precision would be the percentage of *actual* fraudsters in the group of people that the model has predicted to be fraudsters. Therefore using this measure we are able to determine the ratio of how many of our regular clients we have falsely predicted to be a fraudster. We define precision as,

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

5.5.3 Recall

Recall is similar to precision in that it is the proportion of actual positives that were identified correctly. In the example, recall would be the percentage of fraudsters the model was able to find in the entire *pool* of fraudsters. Therefore we define recall as,

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.4}$$

5.5.4 F1 Score

The *F1 Score* is the harmonic mean between the precision and the recall. It is usually used in conjunction with accuracy as a single measure of a models performance. If we are looking to create a model balanced between recall and precision then the F1 score is the metric to use. The F1 Score ranges from 0 to 1. 1 indicates perfect precision and recall where as 0 is when either the precision or recall is 0.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{5.5}$$

5.5.5 Trade offs

Although precision and recall both seem like metrics that we should aim to maximize in our model, in practice it is usually not possible. We have to select the metric which is most valuable to us based on the problem and use that as the main performance measure while keeping the others in mind. In the case of our model

which detects defaulting clients we have to weigh whether marking regular clients as defaulting clients (Precision) or identifying less actual defaulting clients overall (Recall) is more detrimental to our business. A possible solution is assigning a cost to each error made in our predictions in order make more informed decisions.

5.6 Preprocessing

5.6.1 Feature Standardization

Before we begin training our model we first have to *standardize* the features. This entails subtracting the mean and scaling to unit variance so that the standardized data has a mean(μ) of 0 and a standard deviation(σ) of 1,

$$X_{standardized} = \frac{X - \mu}{\sigma} \tag{5.6}$$

This is needed so that features have a variance within the same magnitude, because if one feature has a variance magnitudes larger than others it might dominate the objective function and in turn make the estimator rely too much on it and unable to learn from other features.

Effect on feature attributions

Standardizing the values changes their meaning to how far they are from the mean in terms of standard deviations. This means it is possible for strictly positive features to take on negative values when standardized. We have to account for this when observing the explanations.

Weights of evidence

For the WOE models the distributions of good and bad are used, which is already a form of standardization. Therefore we only perform explicit feature standardization for the raw input models.

5.6.2 Splitting the dataset

Before we start using the data we first have to split the dataset into 3 isolated sets namely *training set*, *validation set*, and *test* set. The test and validation set will each encompass 20% of the total data and the training set the remaining 60%. The training set is what we use for training the model, the validation set is used to fine-tune any parameters specific to the model such as the regularizer coefficient which we will discuss in Section 5.6.3, and the test set is only used at the end in order to evaluate the models performance on unseen data. Since we have a heavy class imbalance in the dataset, we use a *stratified* split which allows each of the 3 sets to maintain the ratio of class imbalance present in the overall dataset.

5.6.3 Feature Selection

Since there are 33 features present in the dataset, we look to reduce the number of features by carefully removing those which are better explained by other features or add nothing of value to the model. For the model which uses the raw features we will make use of a simple linear classifier with an L1 penalizer for feature selection. Since by default L1 regularization is able set weight coefficients to 0 we are to able remove weights which have little or no contribution to the model. For the WOE model, we will be using Information Values extracted from the WOE for feature selection.

Benefits of feature selection

- Decreased training time.
- More stability if correlated variables are removed.
- Improve the classification scheme by removing low contributing features.
- Explanations provided by SHAP are easier interpreted due to less features needing to be explained.

Information value

The Information Value (IV) [55] of a variable can be considered as the total predictive strength of the variable and is an indication of that variable's ability to separate between good and bad loans. It can be seen as the symmetric alternative to the *Kullback-Leibler (KL) divergence* [56]. The IV can be calculated using the following formula,

$$\text{IV} = \sum_{i=1}^{n} (\text{Distr Good}_i - \text{Distr Bad}_i) \cdot \text{WOE}_i \tag{5.7}$$

substituting the equation for WOE from (5.1),

$$\text{IV} = \sum_{i=1}^{n} (\text{Distr Good}_i - \text{Distr Bad}_i) \cdot \ln\left(\frac{\text{Distr Good}}{\text{Distr Bad}}\right)_i \tag{5.8}$$

where the sum is taken over all bins n. A good indication of how the IV of a variable relates to their predictive strength is shown in Table 5.2. By calculating

IV range	Interpretation
< 0.02	Not predictive
$[0.02, 0.1)$	Weak predictive
$[0.1, 0.3)$	Medium predictive
$[0.3, 0.5)$	Strong predictive
> 0.5	Suspicious

Table 5.2: Interpretation of IV values.

the IV of each of the features we are able to keep features within a certain threshold and remove the rest. For our case we have chosen the minimum threshold as 0.02 and maximum threshold as 0.6. Looking at Figure 5.2 we can see the IV of all the features and which features have been chosen to be excluded and included according to our thresholds. We have decreased the number of features from 33 to 22.

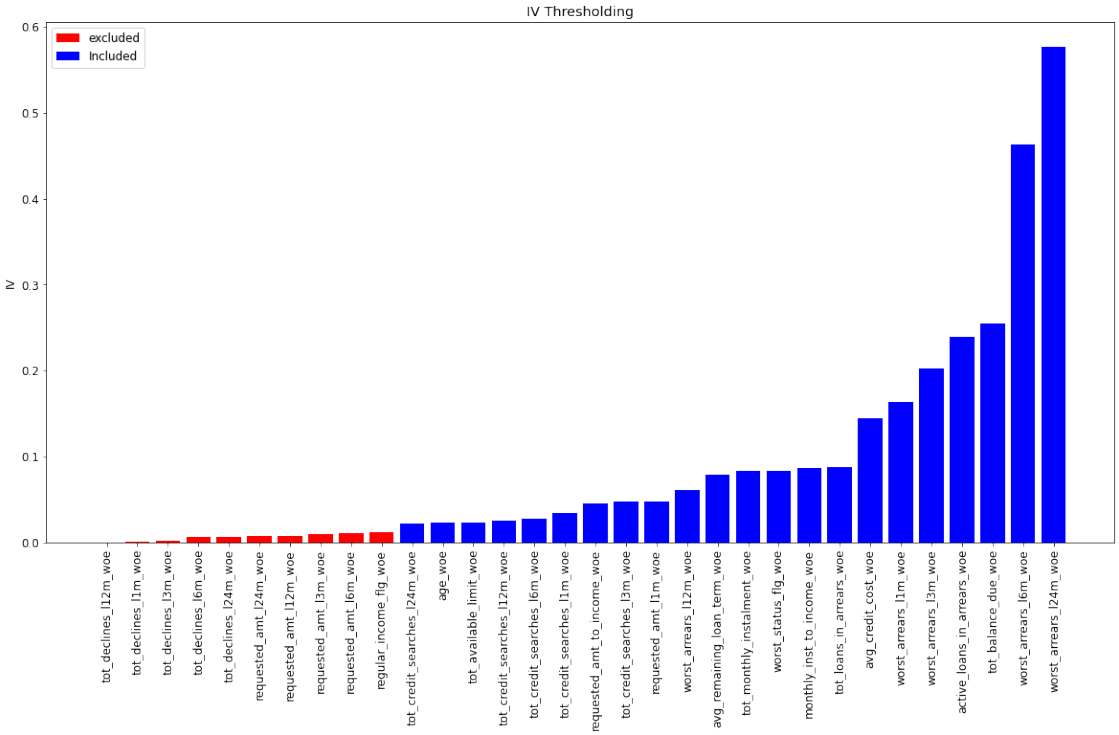


Figure 5.2: Information Value Thresholding on input features.

L1 Regularizer

For the raw features we use a model trained with a L1 regularizer which is referred to as a *Lasso* model for feature selection. L1 regularization is able to assign higher weighting to features which play a larger role in classification and in turn can set those that do the least to 0. Given a linear model,

$$Y = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_0, \tag{5.9}$$

where Y is the label, x_i are the features and β_i are the weight coefficients. The cost function can be written as,

$$\sum_{i=1}^{m} (Y_i - \sum_{j=1}^{n} \beta_j x_{ij})^2, \tag{5.10}$$

where m is all the input samples in the training set and n is the number of features in a sample. We add a regularization term with the L1 penalizer,

$$\sum_{i=1}^{m}(Y_i - \sum_{j=1}^{n}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{n}|\beta_j|, \tag{5.11}$$

where λ is a coefficient that we can tune to adjust how aggressively we constrain the weights. Figure 5.3 demonstrates the constraint region built by the L1 regularizer for a model which contains two weight coefficients β_1 and β_2. $\hat{\beta}$ is the unconstrained least squares estimate. The red ellipses are the contours of the least squares error function. The blue area represents the feasible region $|\beta_1| + |\beta_2| \leq t$ of the constraints introduced by the penalty where t is the coefficient of the regularizer. The possible values are those where the contour and diamond meet and as we can see at 4 points of the diamond one of the weight coefficients are 0. This extends to higher dimensions where there are more possibilities of weights being 0. We are looking for the intersection of the red ellipses and the blue region as the objective is to minimize the error function while maintaining the feasibility. Let p be the number of features and therefore the dimensionality, in this case we have $p = 2$. When $p > 2$ the diamond becomes a rhomboid and has many corners flat edges and faces which have the opportunity to become 0.

Selecting the regularization coefficient

As discussed previously the higher we set the value of the regularization term the stricter we constrain the weight coefficients. Therefore the larger λ is the stricter the feature selection is as more coefficients will be set to 0. In order to select a λ that provides us with the best features we need to tune this as a hyper parameter by using our validation set. We need to make sure that we do not throw away any features that have information that cannot be explained by the selected features. Therefore we tune λ by making it stricter and monitoring how it affects the performance of the model. In Figure 5.4 we have plotted the results of the *accuracy*, *recall*, and *precision* metrics against a stronger penalty. Note that the L1 coefficient in this case is inversely proportional to how strict the regularizer is. As we can see both the recall and precision take quite a significant drop once λ reaches

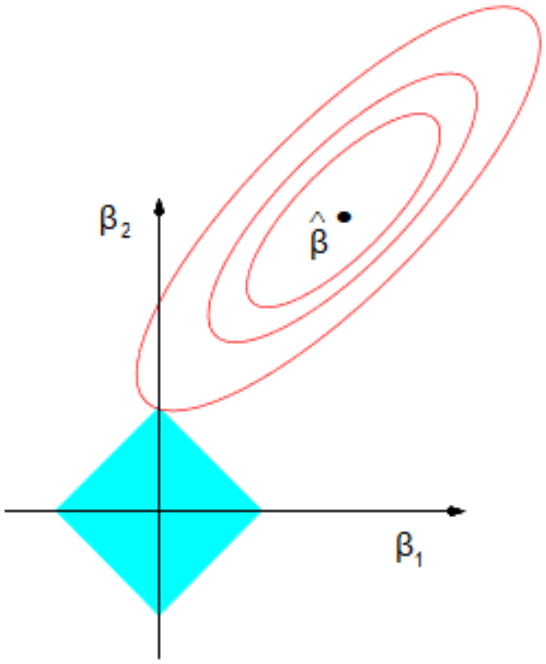


Figure 5.3: Boundary of lasso model. **Image from** [57].

0.0001. Therefore 0.001 is the smallest that we can set λ before we start noticing a drop in the performance of the model. Setting λ to 0.001 reduces the number of features from 33 to 17. This is a big reduction while hardly losing any strength within the model, this may not be the most stable approach if we are looking to productionize this model but it allows the interpretation to be simpler to showcase due to the reduced number of attributions. These features are described in Section 5.9.

5.6.4 Class imbalance

Class imbalance has a significant effect on conventional classification techniques because they assume a balance of classes [58]. In credit companies it is expected that there is a noticeable class imbalance within their data because they can not afford to have many bad loans [59][54]. This results in there being far more loans which could be considered *good*. This makes it difficult for many machine-learning techniques to learn the boundary between a good and bad loan because it does

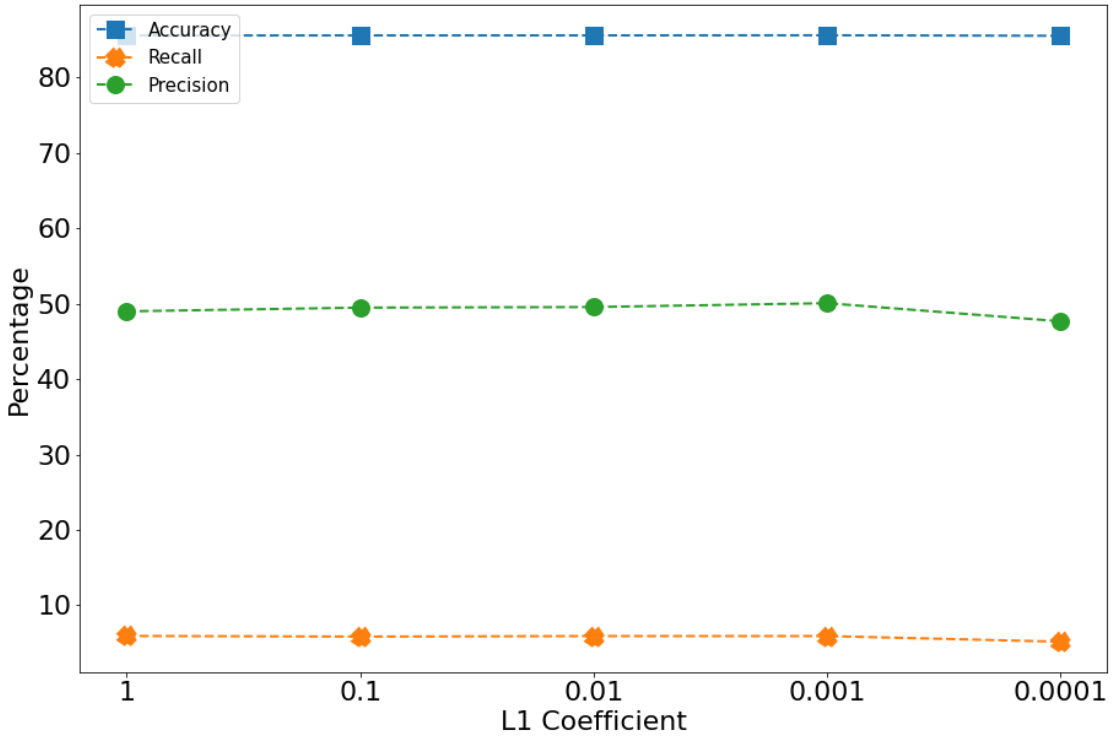


Figure 5.4: Choosing the L1 coefficient.

not have enough bad loans to properly identify them. Selectively sampling from the data in order to create a balanced dataset [45] is a possible solution however this results in a far smaller training set and thus has an overall negative impact if we have a small sample size. It can be shown that by providing misclassification costs to our model provides an increase in the performance of classifiers [60]. If we suppose that misclassifying a client that belongs to the *default* as the *not default* class is r times as serious as the reverse. We can provide a weighting which penalizes such misclassifications r times as heavily as the reverse. In order to minimise the overall weighted misclassification rate we have to minimise,

$$\text{Assign to class 1 if } p(1|x) > (1+r)^{-1} \text{and to class 0 otherwise} \tag{5.12}$$

We can incorporate these misclassification costs into our model by providing a weighting of classes which penalizes the model more for predicting the less present class incorrectly which is the *default* class in our case. By using these bias class

weights we may be able to identify more clients which may default, however we lose some accuracy in predicting non-defaulting clients.

$$\text{Class weights} = \frac{|X|}{|y| \cdot [y_1, ..., y_n]} \tag{5.13}$$

Where $|X|$ is the number of samples, $|y|$ is the number of classes, y_i is the number of labels of class i. By using the values in our dataset we get,

$$\begin{aligned} \text{Class weights} &= \frac{120000}{2 \cdot [102739, 17261]} \\ &= [0.584, 3.476] \end{aligned} \tag{5.14}$$

by using these class weights the model is penalized roughly 5.95 times more for incorrectly classifying *defaulting* clients as *not defaulted*. This causes the weights to be optimized to predict the default class more. For the raw input models we will use versions with and without these biased class weights and compare their results and how their interpretations differ.

5.7 Models

5.7.1 Logistic Regression

Logistic Regression is a widely used technique. Even some Deep Neural Networks use it within their output layer. It is known to work well for binary classification problems and provides a soft prediction.

Soft Predictions

A soft prediction refers to a prediction which gives the probability that a input belongs to a specific class, where as a hard prediction simply assigns a 0 or 1 regardless if the prediction was on the boundary between the two classes. In Figure 5.5 we can see the difference between the thresholding mechanisms used in hard and soft predictions, the x-axis represents the output before we threshold and the y-axis is the value after it goes through the activation function. When using a hard threshold the values are either set to 0 or 1. For soft threshold the value is set as

a probability between 0 and 1, this allows us to see the confidence of the model in the prediction. For example in Figure 5.5 if the value is 0, hard thresholding will force it to 1, however soft thresholding would set it to 0.5 which indicates that it could belong to either class.

Making a prediction using Logistic Regression

Given a general linear model,

$$y = \sum_{i=1}^{n} w_i \cdot x_i + w_0, \tag{5.15}$$

where n is the number of features in the input. We introduce the logistic function, or more commonly referred to as *sigmoid* [61],

$$\text{Sigmoid}(z) = \frac{1}{1 + e^{-z}} \tag{5.16}$$

Substituting in (5.15), this can be rewritten as,

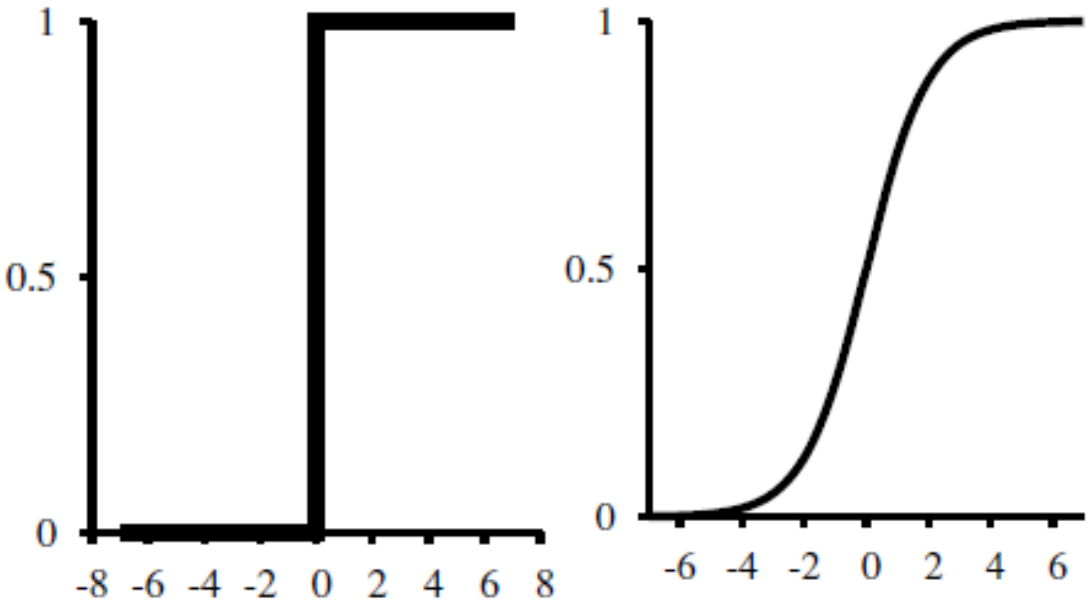
$$\text{Sigmoid}(w \cdot x + w_0) = \frac{1}{1 + e^{-(w \cdot x + w_0)}} \tag{5.17}$$

Once the model is trained, we substitute the input vector into (5.17) in order to make a prediction.

Cross entropy cost function

The cost function can be seen as a measurement of how incorrect the model is at estimating the relationship between the input X and their corresponding labels y. It can also be seen as the distance between the predicted labels and the actual labels. Let the sigmoid function of the model be $h_w(x)$ and the cost function be $J(w)$ then,

$$J(w) = \begin{cases} -\log(h_w(x)) & \text{if } y = 1 \\ -\log(1 - h_w(x)) & \text{if } y = 0 \end{cases}$$



(a) Hard Threshold

(b) Soft Threshold

Figure 5.5: Comparison of hard and soft thresholds. **Image from**[61].

can be condensed as,

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log(h_w(x^{(i)})) + (1-y^{(i)})log(1-h_w(x^{(i)}))\right] \tag{5.18}$$

Now that we have the cost function, we need to somehow decrease it until the weights converge.

Gradient Descent

In order to find the optimal parameters for the weights we have to minimize the cost function using gradient descent. Let y be the true labels and $h_w(x)$ be the predicted labels. When using gradient descent, we set the weights to some initial value either randomly or by some initialization algorithm. We can update the weights by subtracting the derivative of the cost function,

$$w_j \longleftarrow w_j - \alpha \cdot \nabla_w J(w) \tag{5.19}$$

where α is referred to as the *learning rate* which is how large the steps we take when updating the weights are. Since h_w is the sigmoid function we can simplify this to,

$$w_j \longleftarrow w_j - \alpha \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{5.20}$$

This continues until the weights have converged. In Figure 5.6 we can see how the weights update during gradient descent. We start at some initial weights and after each weight update iteration we try to reach the minimum of the cost function. When we reach the minimum, the weights are considered to have converged and the training is completed as we have found the optimal parameters. Note that most if not all of the popular optimization procedures in Neural Networks is based on this simple idea, because the gradient can be efficiently calculated using *back-propagation*. It should also be noted that numerous important modifications have been made for the sake of efficiency and robustness.

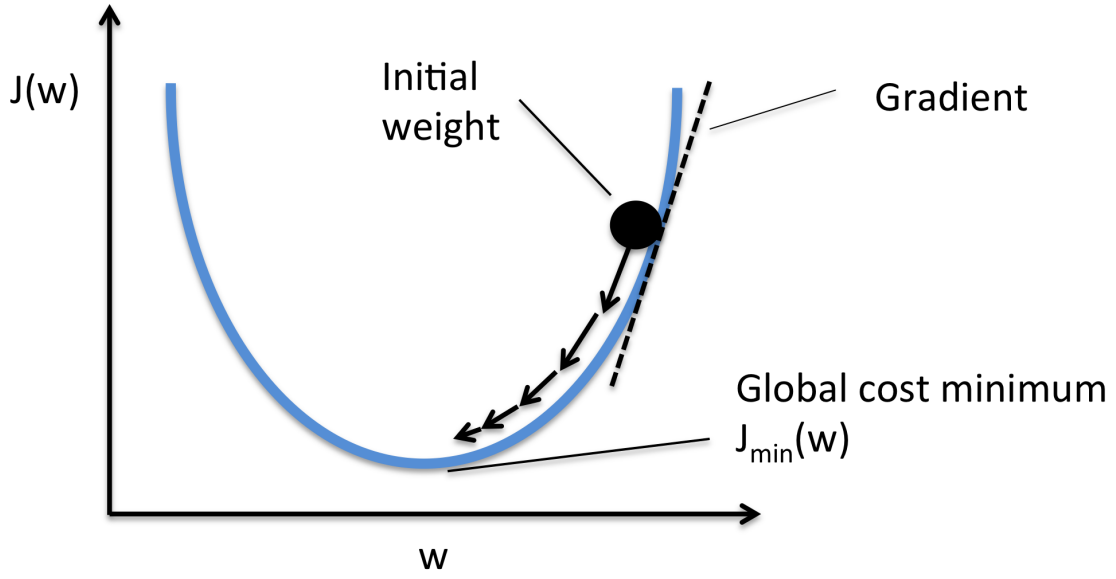


Figure 5.6: How Gradient Descent is performed. **Image from** [5]

Interpreting Logistic Regression

Since the outcome of Logistic Regression is a probability between 0 and 1, we can not simply look at the weight coefficients as a direct interpretation. The weighted sum in the linear equation (5.15) is transformed into a probability by the sigmoid function (5.17). Therefore the weights do not affect the probability linearly. Introducing the log odds function,

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = w_0 + w_1x_1 + ... + w_nx_n \tag{5.21}$$

We can adjust this equation in order to determine how the prediction changes when one feature x_j is changed by 1 unit [62]. Applying the exp function to both sides we end up with,

$$\frac{P(y=1)}{1-P(y=1)} = odds = \exp(w_0 + w_1x_1 + ... + w_nx_n) \tag{5.22}$$

Comparing the ratio of two predictions when increasing a single feature by 1,

$$\frac{odds_{x_j+1}}{odds} = \frac{\exp^(w_0 + w_1x_1 + ... + w_j(x_j+1) + ...w_nx_n)}{exp^(w_0 + w_1x_1 + ... + w_jx_j + ...w_nx_n)} \tag{5.23}$$

Then applying the following exponential rule,

$$\frac{\exp(a)}{\exp(b)} = \exp(a-b) \tag{5.24}$$

and removing many terms,

$$\frac{odds_{x_j+1}}{odds} = \exp(w_j(x_j+1) - w_jx_j) = \exp(w_j) \tag{5.25}$$

The *Odds Ratio (OR)* can be seen as a measurement of the strength between 2 events. Given 2 events A and B, we can define the OR as the ratio of the odds of A occurring in the presence of B compared to the odds of A occurring in the absence of B [63].

- OR=1 The events A and B are independent.
- OR> 1 If B is present then A has higher odds of occurring .

- OR<1 If B is absent then A has higher odds of occurring.

Therefore according to (5.25) a change in "a feature by one unit changes the odds ratio (multiplicative) by a factor of" [62] $\exp(w_j)$. Another interpretation is that by changing a feature's value "by one unit increases the log odds ratio by the value of the corresponding weight" [62]. Interpretation is different depending on the feature type [62]:

- *Numerical features*: If you increase the value of feature x_j by one unit, the estimated odds change by a factor of $\exp(w_j)$ [62].
- *Binary categorical features*: We refer to one of the categories as the *reference* category. Changing the feature x_j from the reference category to the other category in turn changes the estimated odds by a factor of $\exp(w_j))$ [62].
- *Categorical feature with more than two categories*: When dealing with multiple categories one-hot-encoding is commonly used. For a feature which has N categories we would need N-1 columns in our one-hot-encoder. The N-th category is considered the reference category. The interpretation for each category then is equivalent to the interpretation of binary features [62].

In Table 5.5 we can see an example of a Logistic Regression classifier trained to predict the probability of cervical cancer given certain risk factors [62]. Looking at the *Num. of diagnosed STDs* feature which is numerical, increasing the number of STDs by 1 would in turn increase the odds of cancer vs. no Cancer by a factor of 2.26. Looking at the feature *Hormonal contraceptives y/n* which is a binary categorical feature indicates that for women who use hormonal contraceptives, the odds for cancer vs. no cancer are by a factor of 0.89 lower, compared to women without hormonal contraceptive. It is important to note that these interpretations are only true if every other feature stays the same.

5.7.2 Feedforward Neural Network

We will be training a generic feedforward Neural Network with a single hidden layer. The architecture can be seen in Figure 5.7. Every layer in the Neural Network is *Fully Connected* which means that every incoming node is connected

	Weight	Odds ratio	Std. Error
Intercept	−2.91	0.05	0.32
Hormonal contraceptives y/n	−0.12	0.89	0.30
Smokes y/n	0.26	1.29	0.37
Num. of pregnancies	0.04	1.04	0.10
Num. of diagnosed STDs	0.82	2.26	0.33
Intrauterine device y/n	0.62	1.85	0.40

Table 5.3: "The results of fitting a logistic regression model on the cervical cancer dataset. Shown are the features used in the model, their estimated weights and corresponding odds ratios, and the standard errors of the estimated weights" [62]. **Table from** [62].

to every outgoing node. The input layer is simply the input features. The *hidden layer* contains 9 nodes for the raw features and 11 nodes for the WOE network. For both networks the output layer is a single node that uses the *sigmoid* activation function which is the probability that the client will default. The Neural Network only has a single hidden layer and is considered very small when compared to modern networks.

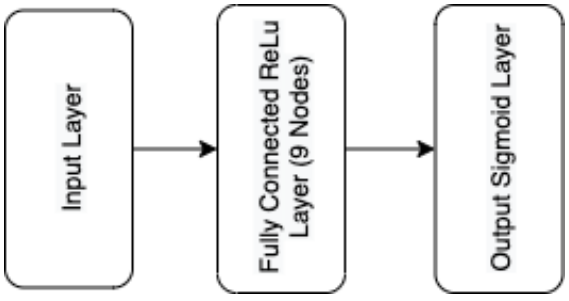


Figure 5.7: Architecture of credit Neural Network.

5.7.3 Trainable weights

In order to show that a Neural Network is theoretically more powerful than Logistic Regression, we have to compare their *trainable weights*. Trainable weights are the weights which models will estimate in order to make a prediction. Theoretically the more weights we can estimate, the more accurate the predictions will be. However

it is important that we account for *overfitting* which is when the model corresponds too closely to the training data and thus fails to predict unseen data reliably. In practice there are other factors such as the networks architecture and preprocessing techniques which have an effect. We can calculate the trainable weights with,

$$\text{LR} = \text{Input weights} + \text{Intercept}$$
$$\text{NN} = \text{Input weights} \cdot \text{Hidden weights} + \text{Hidden weights} \cdot \text{Output weights}$$

We can see the comparison of the trainable weights between the models in Figure 5.4.

Model	**Input**	**Hidden**	**Output**	**Intercept**	**Trainable Weights**
Logistic Regression	17	0	1	✓	18
Logistic Regression WOE	22	0	1	✓	23
Neural Network	17	9	1	×	162
Neural Network WOE	22	11	1	×	253

Table 5.4: Comparison of Trainable Weights between the different model architectures

5.8 Results

In this Chapter we will compare the following models:

- Logistic Regression with raw inputs.
- Logistic regression with raw inputs and altered class weights.
- Logistic Regression with WOE inputs.
- Neural Network with raw inputs.
- Neural Network with raw inputs and altered class weights.
- Neural Network with WOE inputs.

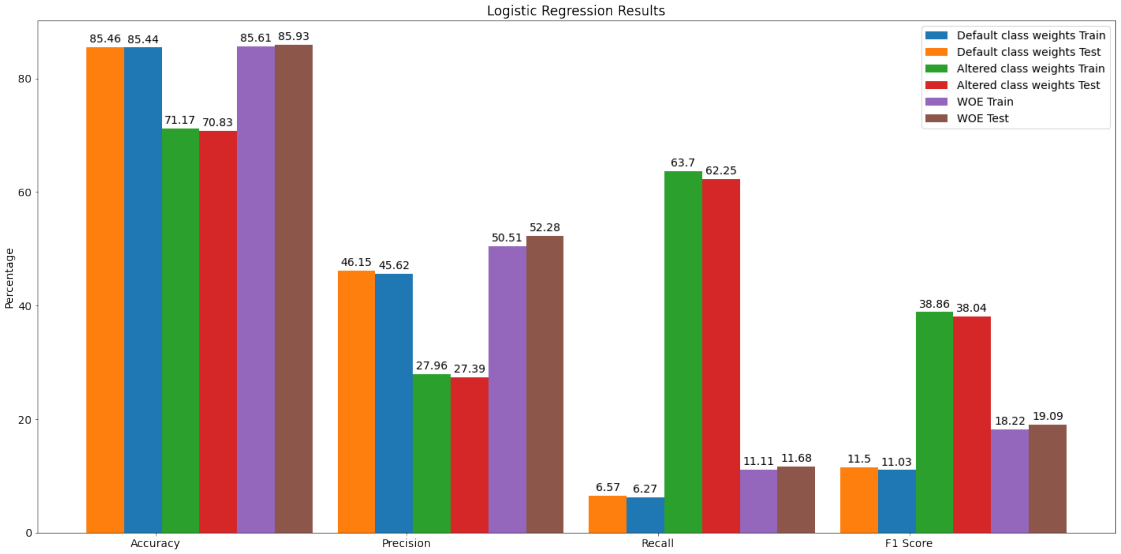


Figure 5.8: Results of the Logistic Regression classifier.

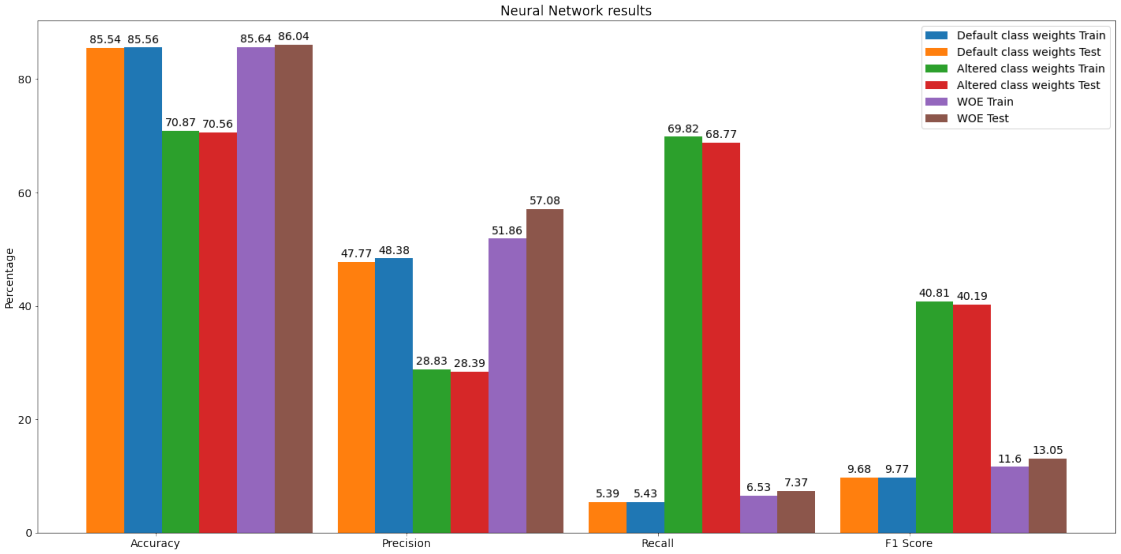


Figure 5.9: Results of the Neural Network.

5.8.1 Comparison

Comparing the results from the Logistic Regression classifier in Figure 5.8 and Neural Network classifier in Figure 5.9 their *accuracy* is mostly the same with the only real difference being a slight increase in *precision* and *recall* in the Neural Network which also in turn results in a difference in *F1 scores*. From this it may seem that there is no reason to use a Neural Network classifier, however there are plenty of optimizations and various other network compositions that may provide better results. Our purpose is to provide interpretability and therefore we are not concerned with building the most robust network possible.

5.8.2 Normal vs Bias class weights

In both models when using the bias class weights discussed in Section 5.6.4 we see on average a 15% drop in accuracy which is quite significant, however an enormous increase in *recall* which means the bias versions are better at identifying clients which *defaulted*. For credit companies where the purpose is to identify clients which may default on their loans, the bias weights provide better results. In Section 5.10 we compare both the normal and bias class weights models and observe the explanations produced by SHAP in order to compare the difference in feature attributions.

5.8.3 Raw Inputs vs WOE

When looking at the comparisons from the raw input features to the WOE transformed features in both the Logistic Regression results in Figure 5.8 and the Neural Network results in Figure 5.9 there does not some to be a significant difference. The only noticeable difference is a slight increase in precision. WOE transformation are the standard when working in the credit industry and from these results it may seem like there isn't much merit in it, but we will see in the *explanations* in Section 5.10 that there is a significant difference.

5.9 Feature descriptions

In order to make sense of the feature attributions we first have to describe what each feature means. From our credit domain knowledge we are able to specify which features magnitudes increase proportionally to the risk that the client will default.

5.9.1 Not Default

The following features decrease the risk of the client defaulting,

age (For clients who are middle-aged)
: The age of the client.

regular_income_flg
: A flag which indicates whether the client has a regular income or not.

tot_available_limit
: Total monthly limit available to client.

5.9.2 Default

The following features increase the risk of the client defaulting,

age (For clients who are very young or old)
: The age of the client.

requested_amt_l1m
: Requested amount over a month.

requested_amt_l24m
: Requested amount over 24 months.

requested_amt_to_income
: Ratio of requested amount to monthly income.

avg_credit_cost
: Average cost of credit.

avg_remaining_loan_term

The average amount of months left on the clients loans.

tot_monthly_instalment

Monthly installment paid towards the loan.

monthly_inst_to_income

Ratio of income to monthly installment.

tot_credit_searches_l1m

Total credit searches in the last month.

tot_credit_searches_l3m

Total credit searches in the last 3 months.

tot_credit_searches_l6m

Total credit searches in the last 6 months.

tot_credit_searches_l12m

Total credit searches in the last 12 months.

tot_credit_searches_l24m

Total credit searches in the last 24 months.

worst_status_flg

Flag of whether this loan is the most in arrears.

tot_declines_l12m

Total credit declines in the last 12 months.

active_loans_in_arrears

Active loans that are in arrears.

tot_loans_in_arrears

Total loans that are in arrears.

worst_arrears_l1m

Worst arrears in the past month.

worst_arrears_l3m
Worst arrears in the past 3 months.

worst_arrears_l6m
Worst arrears in the past 6 months.

worst_arrears_l12m
Worst arrears in the past 12 months.

worst_arrears_l24m
Worst arrears in the past 24 months.

tot_balance_due
Total balance due on the loan.

5.10 Interpretation

In this section we observe how the inherent interpretability of Logistic Regression compares to the explanations provided by SHAP for the Neural Network. We provide interpretation for both the normal and bias class weights models as well as a comparison between the attributions generated by the WOE features compared to the raw features.

5.10.1 Logistic Regression

As we have discussed in Section 5.7.1 for Logistic Regression the weight coefficients are not enough to provide interpretability. We have to extract the weight coefficients w from the linear equation (5.15) and also calculate their odds ratio (which is described in Section 5.7.1) with (5.25). We have tabulated the weight coefficients as well as their respective odds ratios. We have also graphed the odds ratio for extra clarity. Table 5.5 and Figure 5.10 is for the raw input model. Table 5.6 and Figure 5.11 is for the raw input model with altered class weights. Lastly Table 5.7 and Figure 5.12 is for the WOE model.

WOE weight coefficients

From Table 5.7 we can see that all of the weight coefficients are positive for the WOE classifier. It may seem like every variable positively contributes towards the *default* class. This is however not the case as we do not know the range of the magnitudes of the variables by just looking at this table. If the woe value of a feature in a specific bin is negative then even if the weight coefficient is positive it will provide a negative attribution. Therefore Table 5.7 can not be considered sufficient information as we would need to view the range of the feature values to discern whether a feature decreases or increases the probability of *defaulting*.

Feature	**Weight**	**Odds ratio**
requested_amt_l24m	0.0028	1.002804
tot_declines_l12m	0.0098	1.009848
requested_amt_to_income	0.0109	1.010960
tot_credit_searches_l12m	−0.0207	0.979513
worst_arrears_l24m	0.0233	1.023574
tot_available_limit	−0.0305	0.969960
tot_credit_searches_l3m	0.0335	1.034067
age	−0.0474	0.953706
tot_loans_in_arrears	0.0565	1.058127
regular_income_flg	−0.0645	0.937536
tot_monthly_installment	−0.0659	0.936224
worst_status_flg	0.0676	1.069937
avg_credit_cost	0.0781	1.081231
monthly_inst_to_income	0.0832	1.086759
avg_remaining_loan_term	0.0846	1.088282
worst_arrears_l6m	0.1027	1.108159
active_loans_in_arrears	−0.2181	0.804045

Table 5.5: Weight coefficients for Logistic Regression.

5.10.2 Neural Network

For the Neural Network we will be using *Deep SHAP* as it leverages the composition of Neural Networks to significantly increase the speed of SHAP. We will be looking at how SHAP explains individual predictions as well as the explanation for the entire model.

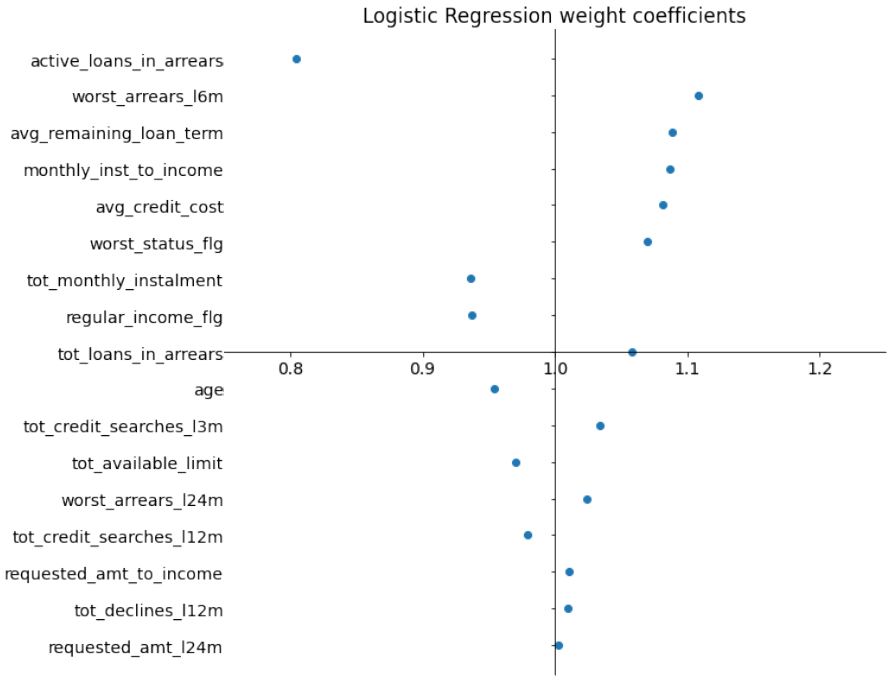


Figure 5.10: Odds ratios for Logistic Regression.

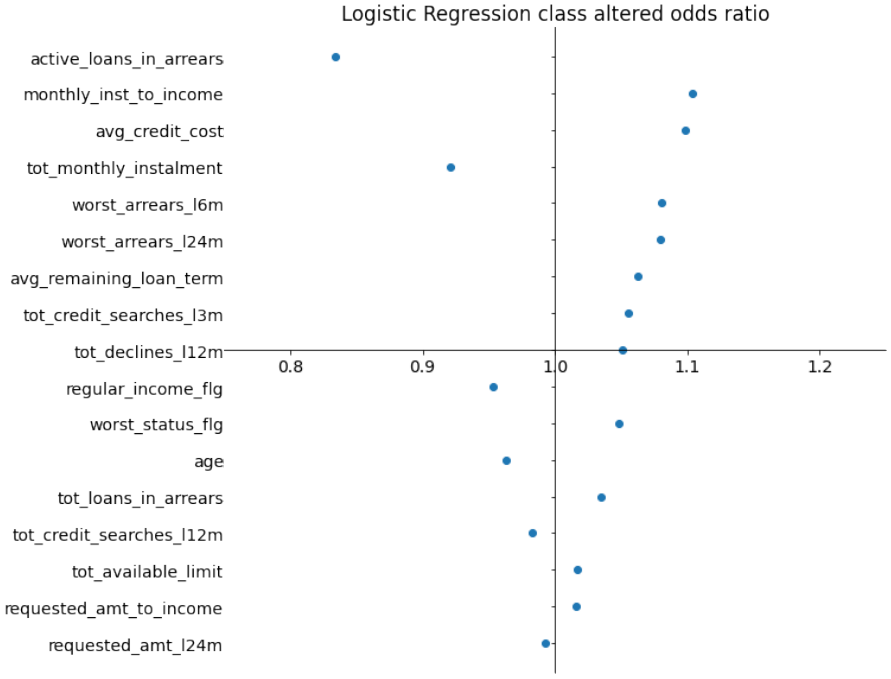


Figure 5.11: Odds ratios for Logistic Regression with altered class weights.

Feature	Weight	Odds ratio
requested_amt_l24m	−0.0076	0.992429
tot_declines_l12m	0.0497	1.050956
requested_amt_to_income	0.0161	1.016230
tot_credit_searches_l12m	−0.0172	0.982947
worst_arrears_l24m	0.0769	1.079934
tot_available_limit	0.0171	1.017247
tot_credit_searches_l3m	0.0539	1.055379
age	−0.0375	0.963194
tot_loans_in_arrears	0.0339	1.034481
regular_income_flg	−0.0481	0.953038
tot_monthly_installment	−0.0819	0.921364
worst_status_flg	0.0475	1.048646
avg_credit_cost	0.0941	1.098670
monthly_inst_to_income	0.0989	1.103956
avg_remaining_loan_term	0.0609	1.062793
worst_arrears_l6m	0.0773	1.080366
active_loans_in_arrears	−0.1814	0.834102

Table 5.6: Weight coefficients for Logistic Regression with altered class weights.

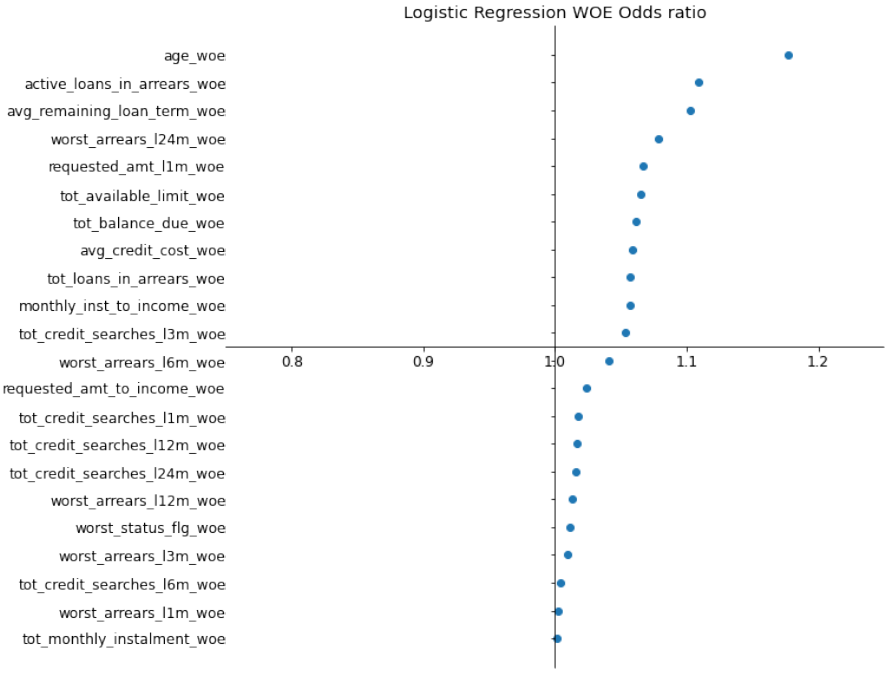


Figure 5.12: Odds ratios for WOE Logistic Regression.

Feature	Weight	Odds ratio
tot_monthly_instalment_woe	1.001401	0.0014
worst_arrears_l1m_woe	1.002102	0.0021
tot_credit_searches_l6m_woe	1.004108	0.0041
worst_arrears_l3m_woe	1.009041	0.0090
worst_status_flg_woe	1.010859	0.0108
worst_arrears_l12m_woe	1.012984	0.0129
tot_credit_searches_l24m_woe	1.015621	0.0155
tot_credit_searches_l12m_woe	1.016332	0.0162
tot_credit_searches_l1m_woe	1.017145	0.0170
requested_amt_to_income_woe	1.024188	0.0239
worst_arrears_l6m_woe	1.040603	0.0398
tot_credit_searches_l3m_woe	1.053376	0.0520
monthly_inst_to_income_woe	1.056541	0.0550
tot_loans_in_arrears_woe	1.057280	0.0557
avg_credit_cost_woe	1.059079	0.0574
tot_balance_due_woe	1.061412	0.0596
tot_available_limit_woe	1.064814	0.0628
requested_amt_l1m_woe	1.067159	0.0650
worst_arrears_l24m_woe	1.078100	0.0752
avg_remaining_loan_term_woe	1.102963	0.0980
active_loans_in_arrears_woe	1.109268	0.1037
age_woe	1.176919	0.1629

Table 5.7: Weight coefficients for WOE Logistic Regression.

Model explanation

We start by explaining on a global level the attributions the model assigns to each input feature and compare it to the Logistic Regression explanations. Figure 5.13 is the explanation for the normal network, Figure 5.14 is for the bias class weights network, and Figure 5.15 is the WOE network. Rather than just giving a single value for attributions SHAP is able to plot over multiple instances in the dataset and showcase how each feature attributes to different instances. The y-axes are the feature names and the x-axes are their corresponding SHAP values, larger values impact the model more. Positive values attribute towards the *default* class and negative values to the *not default* class. The hue of a point ranges from blue to red where the redder a point the larger the magnitude of that feature was in that particular instance.

Explanation Using the information from Section 5.9 we know that the larger the magnitude of the *Age* feature the more it contributes against the client defaulting. In Figure 5.13 and 5.14 we can see that the larger the magnitude of age the SHAP value falls more into the negative region, this indicates that the older the client is, the less likely they are to *default*. This coincides with our knowledge. On the inverse we know that the *avg_credit_cost* feature contributes towards the client *defaulting*, this is also shown in Figure 5.13 and 5.14 where the larger the magnitude of *avg_credit_cost* is, the larger it's SHAP value is and thus the higher contribution it had towards predicting that the client would default.

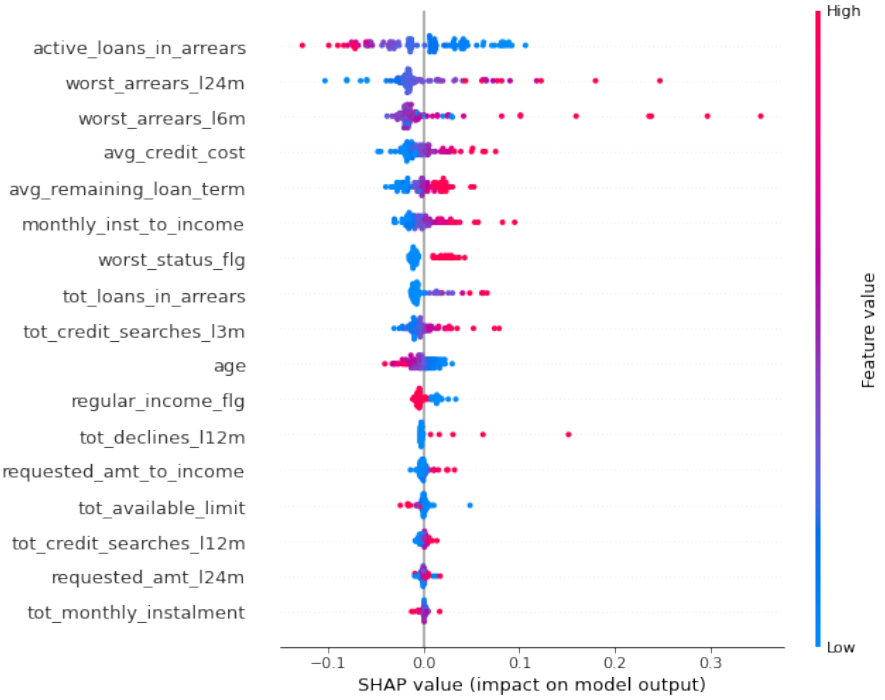


Figure 5.13: Deep SHAP interpretation for Neural Network.

Individual predictions

An advantage of using a tool such as SHAP is that we are able to obtain explanations for individual predictions rather than just for the entire model. This is useful for credit employees as this allows them to obtain explanations for an individual

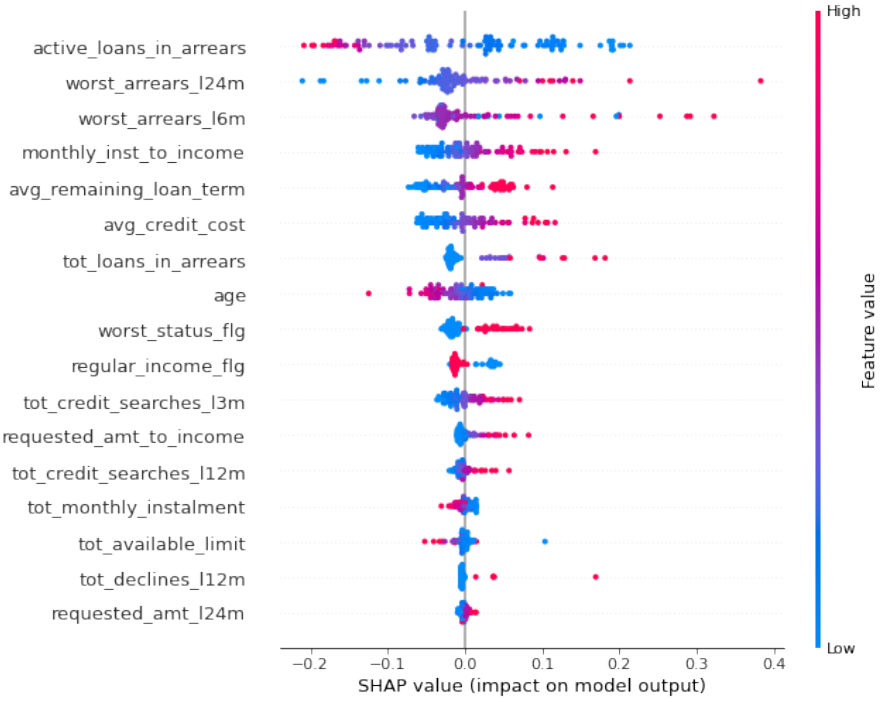


Figure 5.14: Deep SHAP interpretation for class weight bias Neural Network.

client. We have chosen a single client which has *defaulted* and in Figure 5.16, 5.17 and 5.18 we can see a plot of the explanation for it's prediction. Figure 5.16 is the *normal* class weights and the output of the model is that it is 27% certain that the client will *default on their loan*. Figure 5.17 is the explanation for the *bias* class weights model and it is 71% certain that the client will *default*. Figure 5.18 is the explanation for the WOE Neural Network and it is 37% that the client will *default*. The y-axes are the input features and the x-axes is the model output value. The bottom of the line is the average model output value, which is the models default output value given that there is no information about the input features. As the line reaches the top, each feature either adds or subtracts from the models predicted value, therefore features that subtract from the value are negative attributions and those that add are positive attributions. The numbers in brackets are the values of that feature of this particular client standardized.

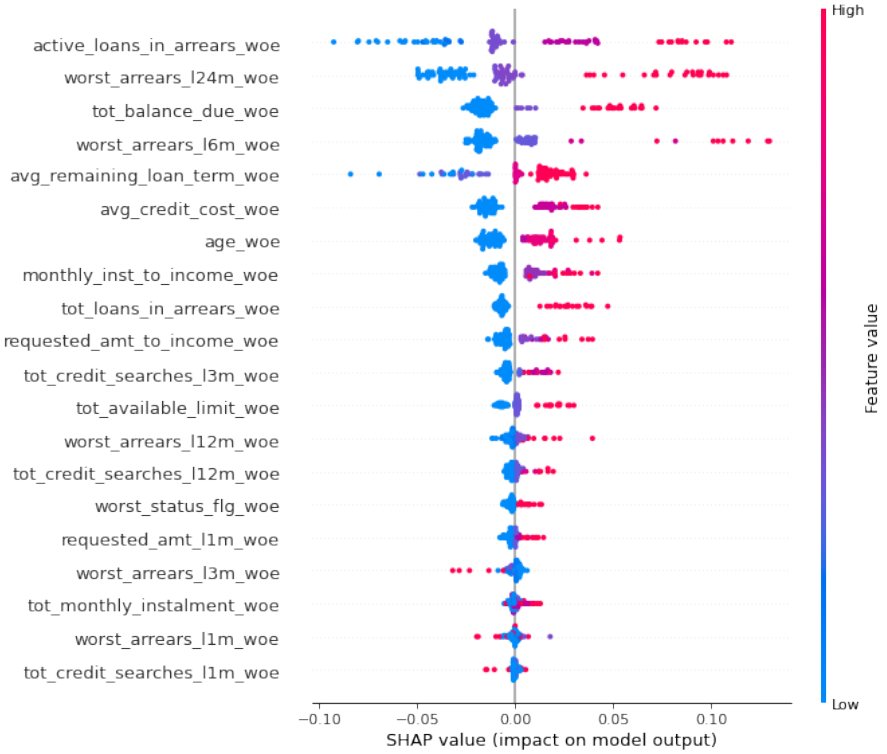


Figure 5.15: Deep SHAP interpretation for WOE Neural Network.

5.10.3 Comparison

Comparing the weight coefficients obtained from Logistic Regression with the SHAP explanations we can see that although the SHAP values have a different meaning than the raw weight coefficients they are comparable. They both provide a relative value of how much that particular feature would affect the prediction of the model. The SHAP explanations are far more descriptive as they are able to provide explanations over multiple instances, which shows consistency and we are also able to provide explanations for individual instances. Since the SHAP values are consistent with what we expect we can conclude that we have successfully provided explanations into this previously considered *black-box* credit risk Neural Network. It is important to note that there is no known way to quantitatively measure the difference between the Logistic regression weight coefficients and the SHAP values produced from the Neural Network. If further research is to be done it is important to further explore possible methods that can provide quantitative comparisons.

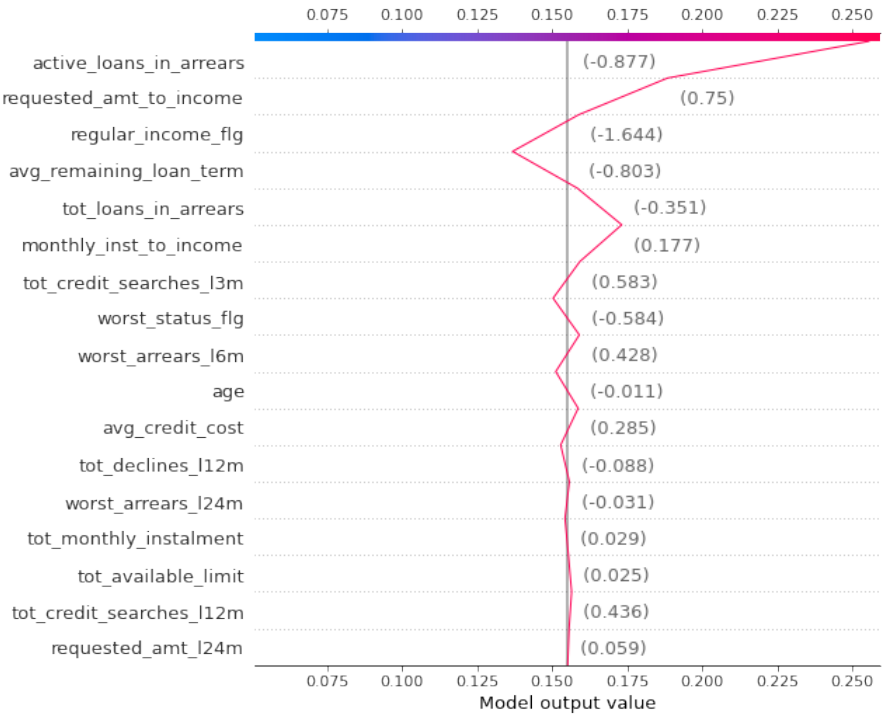


Figure 5.16: Single prediction Deep SHAP interpretation for Neural Network.

5.10.4 Exposing a problem with the raw input models

If we refer back to Section 5.9 we can see that the *active_loans_in_arrears* feature is the number of active loans that the client currently has in arrears. An increase in its feature value would in turn cause the risk of the client to increase. As we can see for the WOE model in Figure 5.15 the larger the magnitude of *active_loans_in_arrears* the higher the contribution towards the *Default* class as expected. However looking at the SHAP explanations for the raw input models in Figure 5.13 and 5.14 the larger *active_loans_in_arrears* is the less likely the client is to *default* . This means that there is an obvious flaw present with the raw input models with regards to this feature. This is not obvious from just observing the Logistic Regression models since it hard to determine how the feature reacts over multiple instances and different magnitudes by only looking at the features overall contribution. Even though the performance of the two models are relatively the same by viewing the SHAP explanations produced, we were able to identify this problem.

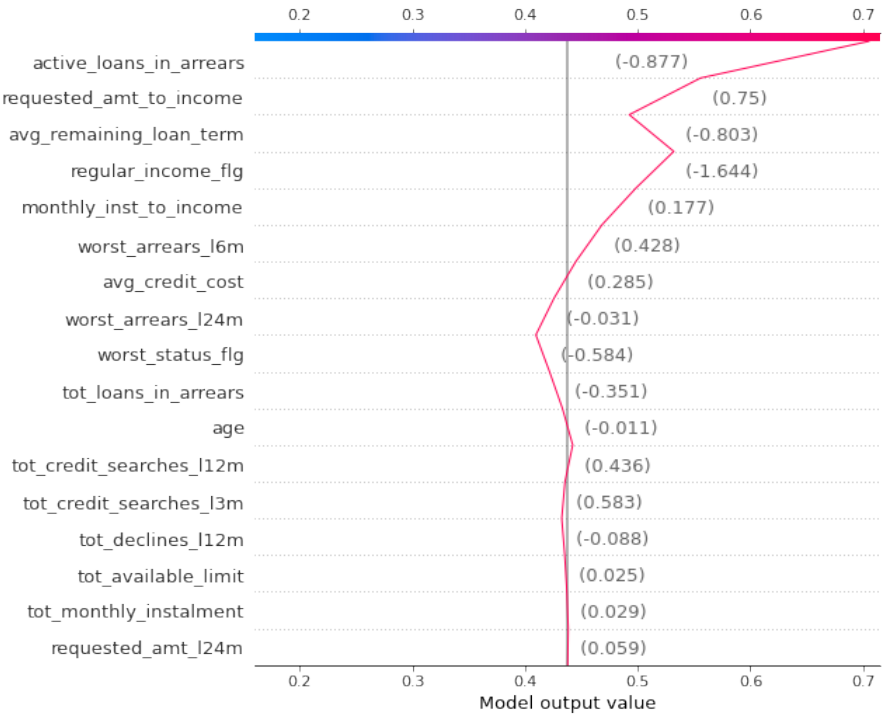


Figure 5.17: Single prediction Deep SHAP interpretation for class weight bias Neural Network.

5.10.5 Monitoring changes within the architecture

Figure 5.19 is the SHAP explanation of the WOE Neural Network with the hidden layer removed form the network itself. When comparing this to Figure 5.15 there is a noticeable difference as the *worst_arrears_l24m_woe* has the largest contribution. The explanation is expected to change but SHAP can provide some insight to what the changes are. When training a neural network one of the hardest problems is figuring out what the architecture should be. An example would be for a simple neural network how do we decide how many hidden layers? What about how many neurons in each layer? Usually we would just use our metrics and experiment with the architecture. However metrics don't really tell the whole story and with SHAP we can observe how the architecture affects the variables attributions themselves. Thus by using SHAP we are able to determine how a change in the networks architecture affects the feature attributions.

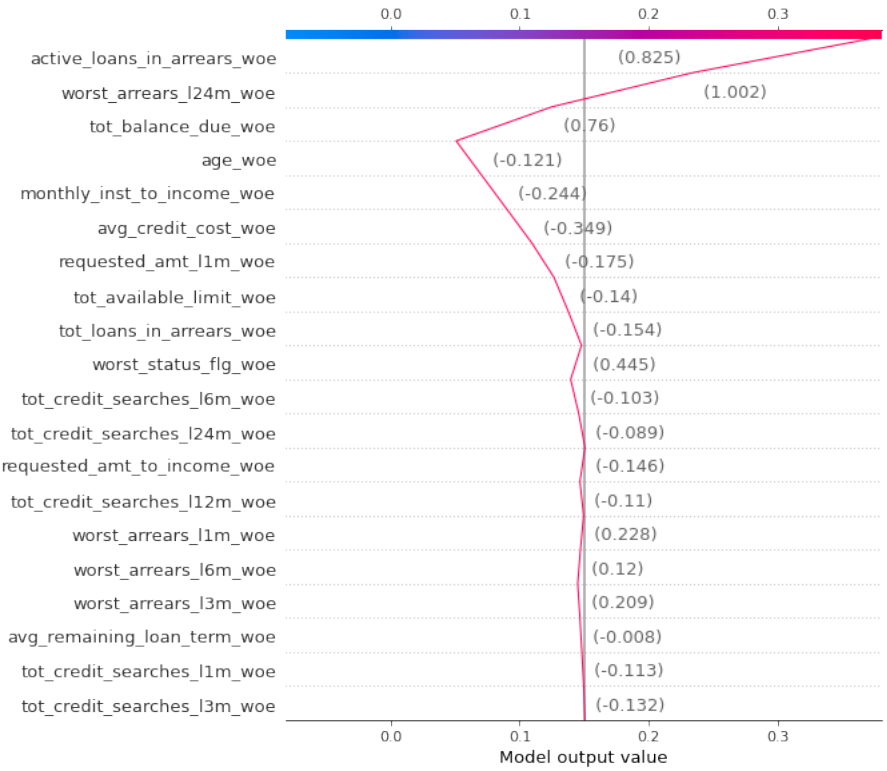


Figure 5.18: Single prediction Deep SHAP interpretation for WOE Neural Network.

5.10.6 Attributions between normal and bias class weights

Comparing the feature attributions between Figure 5.13 and 5.14, there are a few key differences. The first is that the order of the features are different which means the features which contribute the most differ. Looking at *monthly_inst_to_income* and *total_loan_in_arrears* in Figure 5.14, it can be seen that they have stronger attributions towards the *default* class in the weighted model. Another notable difference is the *worst_arrears_l24m_feature*, in Figure 5.14 the maximum value possible SHAP value is larger which means that this feature can have a larger effect on the models outcome. From this we can see that it seems that by using the misclassification costs introduced in Section 5.6.4 some of the features were given more predictive power towards the *default* class.

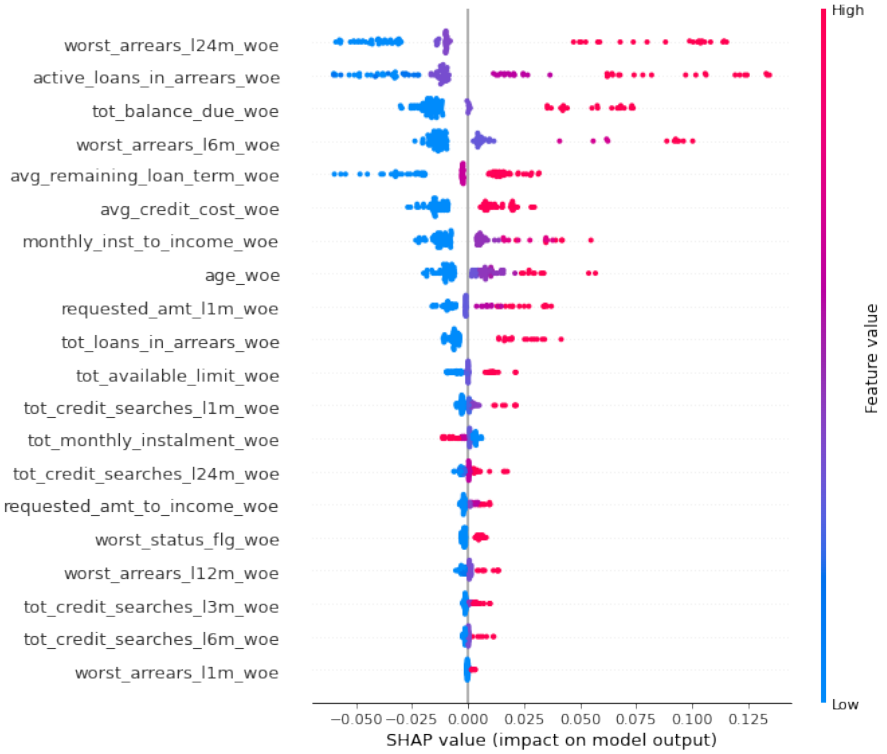


Figure 5.19: Neural Network WOE with just the input layer and output sigmoid layer.

5.10.7 Prediction strength relative to Feature magnitude

With SHAP we are able to sample the explanations over multiple instances. If the chosen background instances are diverse, we are able to view how the feature attributions react over varying magnitudes. For example if we look at the odds ratio for *worst_status_flg* in Table 5.5 we can see that the feature has an *odds ratio* of 1.069937 which by using our domain knowledge we know that this is a binary categorical variable which can only take on a value of 0 or 1. However if we did not have domain knowledge about this variable it could possibly be mistaken as a *continuous variable* which makes it seem as though it could have a much larger contribution than it actually does. Now if we compare to this to the SHAP explanation in Figure 5.13 we can see how the features contributions reacts at different values. From this we can see that regardless of it's magnitude the feature relatively has the same contribution if it is high and a small contribution when it

is low. It would require more effort and observing the original feature to discern this from just the Logistic Regression explanation. This could possibly be used for *Feature Selection*. By observing how the contribution of a feature changes over varying magnitudes we can choose to manually remove features which regardless of their intensity seem to add little value. If we once again look at Figure 5.13, the feature *tot_monthly_instalment* seems have a very low SHAP Value even when it's feature value is high and could possibly be considered for removal.