Imperial College London

COURSEWORK 2

NEURAL NETWORKS

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Introduction to Machine Learning

Authors:

1. Dhruva Gowda Storz *(01807283)*
2. Alexander F. Spies *(01055106)*
3. Alex Gaskell *(01813313)*
4. Se Hyung Park *(01602366)*

Date: November 25, 2019

Table of Contents

List of Figures

List of Tables

1 Overview

We present our implementations of claim classifiers built using Keras and Tensorflow (1; 2) for the purpose of pricing automotive insurance, with training performed on a historical insurance dataset. In Part 2, we create a binary classifier to predict whether a client is likely to make a claim, and tune the classifier's model after addressing the imbalanced nature of the dataset. In Part 3 we develop a pricing model based on the claim prediction model from Part 2 coupled with a tuned severity model. This model is then tested and tuned based on its performance in a competitive insurance market populated with competing AI insurance policy offerers.

2 Insurance Claim Classification

2.1 Implementation

Data Exploration

Figure 4 shows the correlation matrix for the `part2_data.csv` (based on (3)), in which variable pairs with strong positive correlations are shown in dark red and pairs with strong negative correlations shown in dark blue. This figure highlights that there are few strong correlations between the features, as illustrated by the weakness of colour for the first two columns in the figure. This in turn means that even a sophisticated model will struggle to accurately predict both class labels based on the features provided.

Futhermore, Figure 5 shows a representation of the dataset after undertaking dimensionality reduction using principal component analysis (PCA) with $n = 2$. Here we see that projecting the data in the two leading principal components (those with the greatest variance, i.e. most information about the data) results in largely overlapping distributions. From this we can deduce that the predictive power of the features in lieu of their labels, beyond the linear combination given by the PCA basis transformation, is subtle at best. In addition, we ran ordinary least squares regression on the dataset to further gauge the explanatory power of the features. Consistent with our findings from the PCA, very few variables provided statistically significant contributions. Consequently, we should expect linear models to perform relatively well on the dataset, and that adding additional complexity in the form of neural networks is likely to yield only marginal improvements which themselves will require the successful identification of necessarily sub-leading non-linear relationships in the data (these ideas are developed further in subsection 3.4).

Indeed, while trialling our initial neural network models on the dataset we found three features, `vh_cyl, vh_din, vh_sale_begin`, with negligible explanatory power, and consequently chose to omit these from the input data. The remaining variables chosen to be included in the binary classification task were `drv_age1` (driver's age), `vh_age` (vehicle's age), `pol_bonus` (bonus coefficient), `vh_sale_end` (years for which the vehicle has been out of production), `vh_value` (vehicle's value), `vh_speed` (vehicle's maximum speed).

Data Preprocessing

Normalization is performed by centering the distribution of every feature, and rescaling these to have unit variance. For this purpose, scikit-learn's `StandardScaler` was used (4). The corresponding transformation of a feature vector from the initial dataset $\boldsymbol{X}$ to the normalized dataset $\boldsymbol{z}$ is given by

$$\boldsymbol{z} = \frac{\boldsymbol{X} - m}{s}, \tag{1}$$

where $m = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean and $s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_k - m)^2$ is the sample variance. Feature normalization is important for effectively training neural networks as, by removing the scale dependence of initial weights, it can make training faster as well as reducing the likelihood of getting stuck in local optima. Additionally, we ensure that feature normalization is carried out after splitting the dataset (for training and testing) to avoid "training" our scaler on the testing data; as such, the means and variances computed for the training data are used to rescale the test data [1].

It is worth noting that the provided insurance dataset is significantly unbalanced, with examples of contracts on which claims were made being underrepresented by 9:1. This imbalance needed to be addressed as neural network classifiers typically overfit on the over-represented class when trained on imbalanced data. One reason for this follows from the binary cross-entropy for data with labels y and predicted labels $\hat{y}$ which was used a loss function. From the functional form,

$$-\sum_{i}^{N} y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}), \tag{2}$$

we see that the error from the majority class contributes significantly more toward the loss value than the minority class. As a result, this loss function is biased towards the majority class and fails to adequately capture the prediction errors over all classes.

In order to obtain a balanced dataset for training, naive random oversampling and undersampling methods were initially implemented, but more sophisticated methods presented in (5) were found to offer superior performance. In particular, we utilised the Synthetic Minority Oversampling Technique (SMOTE) (6), which oversamples the minority class by randomly picking a point from the minority class and computing the k-nearest neighbors for this point, and interpolating between these. Additionally, we also implemented a variant of SMOTE which utilizes Tomek Links (7). These essentially function to ensure that the nearest neighbors used for interpolation are only of the same class, thus theoretically providing more legitimate data (than vanilla SMOTE) for the classifier to train on.

In order to ensure that the scaler is trained on purer data, and to reduce the effect of outliers (i.e. large variations in respective feature dimensions), we perform feature normalization before utilizing these oversampling techniques.

Neural Network Architecture

Figure 1 shows the architecture of the network used for the binary classification task. Hidden layers were dense layers with ReLu activation functions, and the output layer was a single neuron with a sigmoid activation function, producing a single output between 0 and 1, interpretable as the confidence in predicting class 1[2]. More details of the architecture and training parameters are given in Table 1.

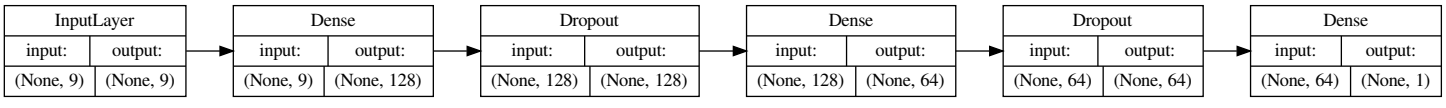


Figure 1: Neural Network Architecture

[1] When other scalers were used the same ordering of procedures was respected, however, the relevant "learned" parameters were not necessarily the same.

[2] As we perform a cutoff at 0.5 for classification, the lack of probability calibration at this stage is not problematic.

Training Epochs	Batch Size	Activation Function	Dropout	Learning Rate	L2 Regularization
10	64	ReLu	0	0.01	0

Table 1: Initial Network Hyperparameters

Owing to the complexity of the dataset, we used relatively large layer sizes in order to capture any interesting interactions between features and the target. The initial model did not significantly overfit the data, and achieved a relatively high (compared to more sophisticated models) AUC-ROC of 0.60. However, this was improved after doing a hyperparameter search which applied appropriate dropout and regularization values, which increased the AUC-ROC to 0.62. The purpose of regularization is to penalise large weights which has the effect of pushing weights toward zero, hence preventing overfitting and leading to more reliable learning.

2.2 Architecture Evaluation

Metrics

The classification accuracy with an imbalanced dataset can be misleading. For instance, with the dataset analysed, a classifier predicting only the majority class will still have an classification accuracy of above 90%.

To appropriately evaluate the performance, we first compute a confusion matrix where the columns represent predicted labels and rows represent ground truth labels. To account for the imbalances, we normalized the confusion matrix by row, i.e., by class, from which the recall and precision were derived.

Recall is defined as

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}, \tag{3}$$

capturing how well our model predicts the class compared to the true class of the instances. In particular, recall for the minority class was of particular interest to observe how well those who have made claims were being classified.

Precision is defined as

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}, \tag{4}$$

and provides a measure of how likely it is that an instance was from a particular class after the model predicted it was from that class. In particular, the precision for the majority class was of particular interest to observe how many of the minority class were being captured when predicted not to have made a claim.

In addition to the above, another key metric for evaluation was the AUC-ROC curve. The receiver operating characteristic (ROC) curve is a plot of the true positive rate against the false positive rate, defined as:

$$\text{FPR} = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}}, \tag{5}$$

The true positive rate and false positive rate are traced out for every decision boundary, thereby the ROC curve is invariant to class ratios and the decision boundary. The area under the curve (AUC) is a measure of separability between the the classes and represents how well the model is able to distinguish between the the classes. A model with AUC equal to one is able to perfectly distinguish the classes whereas an AUC of 0.5 implies that the model is unable to separate the classes.

Initial Performance

Due to the difficulty of classifying the provided dataset, as elucidated in Section 2.1.1, we strived to obtain a good initial architecture for the model that would be able to capture most of the inherent correlations between features and the target. The results we obtained were are provided in Table 2.

	Normalized Precision	Normalized Recall	F1-Score
Class 0	0.59	0.58	0.58
Class 1	0.59	0.60	0.54

Accuracy	0.58
ROC-AUC	0.60

Table 2: Performance of the initial model

Final Model Performance

After hyperparameter tuning (see subsection 2.3), our final model had an AUC-ROC of 0.62, as shown in Table 3.

	Normalized Precision	Normalized Recall	F1-Score
Class 0	0.60	0.49	0.58
Class 1	0.57	0.67	0.54

Accuracy	0.51
AUC-ROC	0.62

Table 3: Performance of the final model

This is a marginal increase from the initial value of 0.60, but for difficult to classify real world data, this incremental increase can affect overall profits of the insurance company significantly. The difference in ROC curves can be seen in Figure 2. It is interesting to note that the accuracy went down after tuning, however, as mentioned previously, accuracy is not the best metric for evaluating model performance, and AUC-ROC serves as better estimate given the imbalanced nature of the data. Figure 2 also shows us that optimizing the hyperparameters makes the graph more symmetrical, and increases the area under the curve, which is an indication of better model performance.

2.3 Hyperparameter Tuning

The key hyperparameters we were looking to optimise were the number of epochs, batch size, activation function, dropout rate, learning rate, and l2 regularization value. We optimised to the AUC-ROC as this provides a gauge of model performance which is independent of the decision boundary. The values used in the search are shown in Table 4.

Epoch Size	Batch Size	Activation Function	Dropout	Learning Rate	L2 Regularization
5	32	ReLu	0	0.01	0
10	64	linear	0.1	0.001	0.01
20	128		0.2		0.001
			0.3		

Table 4: Configurations of hyperparameters used in the hyperparameter search.

The hyperparameter search was done over a search space comprising all combinations of the values shown in Table 4. This resulted in a search space of 432 distinct combinations. The search was done on all combinations at once as optimizing one or two at a time may be faster, but might result in overlooking possible beneficial interactions between hyperparameter values. Therefore, searching all combinations is the most thorough approach, and is fast when using a GPU.

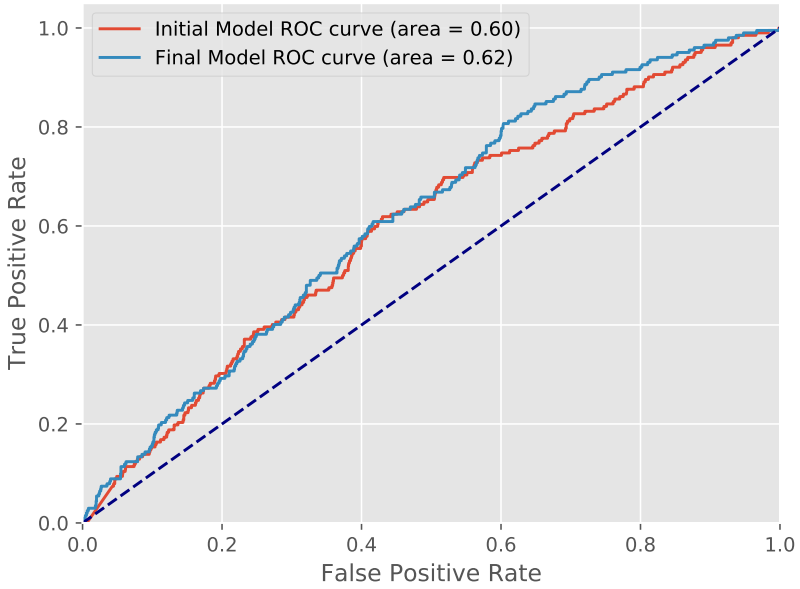


Figure 2: Part 2 ROC curves

To carry out the hyperparameter search, we first created a test set comprising 20% of the data, which was set aside for the final evaluation, with the remaining 80% used for training and validation. Additionally, the same random seed was used for the initial splitting of the data in every run. This was done to ensure that any changes to model performance were only due to changes to the hyperparameters, so we could accurately explore their effects on the model. We then performed a 5-fold cross-validation on each combination of the the hyperparameters for evaluation purposes. When performing cross-validation, we ensured that oversampling was applied before splitting into training and validation datasets to prevent the synthetically generated samples also being present in the validation set, whose information would partially be captured by the synthetic data in the training set, and thereby make validation measurements less valid. Just like the intial model, SMOTE and the removal of Tomek Links (see Data Preprocessing in section 2.1) were used to oversample the datasets.

The model was then built, compiled and fit to the cross validation datasets, and the average AUC-ROC for all cross validation runs was obtained. AUC-ROC was used as a metric for optimization since the test dataset is unbalanced. Therefore the magnitude of the accuracy measure is influenced highly by the dominating class. AUC-ROC however, gives a much better indication of performance between both classes in the unbalanced test set. After the hyperparameter search function has looped over every possible combination of parameters, it returns the combination which achieved the maximum AUC-ROC in the form of a tuple.

The search process returned the following optimal parameters, which were implemented in the final model.

Epoch Size	Batch Size	Activation Function	Dropout	Learning Rate	L2 Regularization
20	32	ReLu	0.1	0.001	0.01

3 Competitive Insurance Pricing Market

3.1 Model selection

Our initial work used the optimal model found in Part 2 (adapted for additional features). However, we found that, because the dataset for part 3 has more features than part 2, our model was not optimal and a more complex model performed better (our model architectures for the non-linear and linear models are outlined in Figure 6). We optimised our model to maximise the area under the AUC-ROC curve, and we found that by increasing the complexity of the model it was better able to identify low risk customers who were unlikely to make a claim (i.e. precision on class 0). This was consistent with our pricing startegy, as discussed below. As in part 2, we employed SMOTE to address class imbalance issues.

Additionally, whilst retaining the sample preprocessing pipeline used in Part 2, we chose to use the `RobustScaler` from scikit-learn in place of the `StandardScaler` owing to the existence of more anomolous datapoints in some of the features introduced in Part 3.

3.2 Calibration

Both models were calibrated using the scikit-learn `CalibratedClassifier class`, requiring the use of the keras `KerasClassifier` wrapper and our own wrapper class to match the method provided. The corresponding ROC curves are shown in Figure 3, which shows that the calibration of probabilities outputted by our classifier made little difference to the ROC curves. However, calibration had a positive impact on our performance in the AI market; with uncalibrated probabilities, there is no consistent connection between the model output and the risk factor of an individual. It simply states that if person A is has a higher model output than person B, A is riskier than B, but with no quantifiable measure of ”how much riskier”. The calibrated model outputs correspond more closely to a frequency model, therefore it is no surprise that the calibrated model performs better in the AI insurance market.

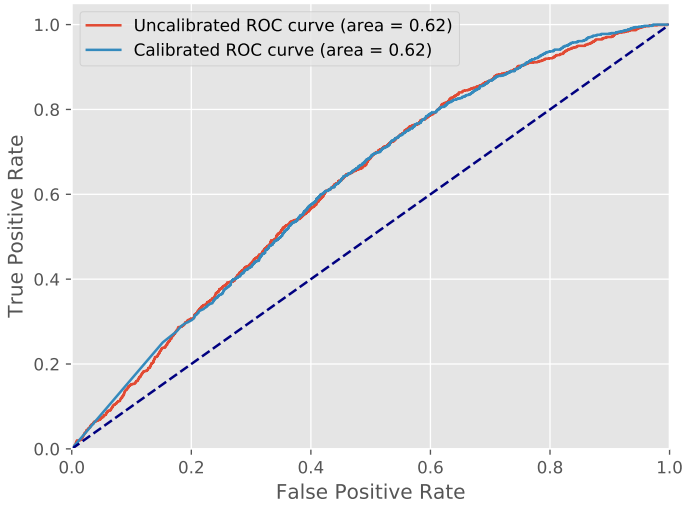


Figure 3: ROC curves showing the effect of calibrating the non-linear classifier.

3.3 Pricing Strategy

Our pricing strategy was to identity and win contracts for the low risk customers. Our hypothesis was that by winning these customers' contracts we could minimize our claim-per-customer, and therefore could translate much of the revenue from selling contracts into profit. The price offered was the product of the frequency model (our calibrated base classifier) and the severity model (which was just the mean claim as written in the skeleton code) plus a margin. By setting a relatively high margin we were able to make good profits on the low risk customers and made us too expensive for the high risk customers so we did not win their contracts.

The successful execution of our strategy manifests through the performance of our insurance pricing agent. The mean price won was the lowest in the market (£44) (as we were aiming for the low-risk customers therefore had to offer cheap premiums to win them) and the mean loss incurred was also the lowest in the market at £35 (excluding agents that did not receive any claims). Additionally, our mean price offered was the highest in the market at £153. These imply all three of the pillars of our pricing strategy were successfully executed as we won the contracts of low risk customers, hence did not receive large claims and also did not win the contracts of high risk customers. This is how we managed to achieve a 3rd place ranking in the AI market.

3.4 Linear Model Comparison

Our linear model used a logisitic regression classifier (implemented as a one-layer neural network with sigmoid activation function utilising our existing code base (i.e. a single-layer perceptron)). Our linear model also performed well, making around half of the profit of our non-linear agent.

The base classifier in the linear model performs similarly well to our non-linear base classifier. The PCA analysis in section 2.1 offers some insight as to why this is (as the part 2 data is a subset of part 3 data the results from running PCA on part 3 data do not differ significantly to the part 2 data). As noted in the earlier discussion of fig:pca in which we performed PCA the dataset, we have observed that the linear combinations of features account for the majority of correlation between labels and sets of features. As such, it is not surprising that the linear model performs relatively well. Indeed, we observed that many "semi-complex" network architectures actually performed worse than the single-layer perceptron, as their additional complexity was insufficient to capture any more nuanced and obscure trends in the data, whilst also hampering their ability to concisely represent the linear behaviour which predominantly determines the correlation in the data.

To make matters worse, this means that we are faced with an optimization problem in which improving beyond the linear model requires a complicated network, which when coupled with the relatively small amount of data and dominance of the linear correlations makes it very difficult to converge on a superior optimum. This was indeed what we observed during the training and tuning of our final network architecture.

3.5 LabTS Complications

We faced numerous substantial difficulties getting our implementations (for part 2 as well as part 3) to pass the LabTS tests. One such difficulty was converting categorical features to dummies so they could be used in the network. However, these ran into difficulties with LabTS and so (after spending a substantial amount of time identifying the issue) we decided to discard all categorical features. This harmed the performance of our model as it meant we discarded many useful features. This was one of the many complications we faced with LabTS, and cumulatively these took a substantial amount of time to overcome.

Our code never passed the tests on LabTS. However, it did pass the tests when we downloaded them and we gather from Piazza that this means our models should pass the tests upon assessment.

References

[1] F. Chollet **et al.**, "Keras," https://keras.io, 2015. pages 1

[2] M. Abadi, A. Agarwal, P. Barham **et al.**, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/ pages 1

[3] M. Waskom, "Plotting a diagonal correlation matrix." [Online]. Available: https://seaborn.pydata.org/examples/many_pairwise_correlations.html pages 1

[4] F. Pedregosa, G. Varoquaux, Gramfort **et al.**, "Scikit-learn: Machine learning in Python," **Journal of Machine Learning Research**, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/ pages 1

[5] G. Kovács, "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets," **Applied Soft Computing**, vol. 83, p. 105662, 2019. [Online]. Available: https://doi.org/10.1016/j.asoc.2019.105662 pages 2

[6] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," **CoRR**, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813 pages 2

[7] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," **SIGKDD Explor. Newsl.**, vol. 6, no. 1, pp. 20–29, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1007730.1007735 pages 2

Appendices

A Supplemental Figures

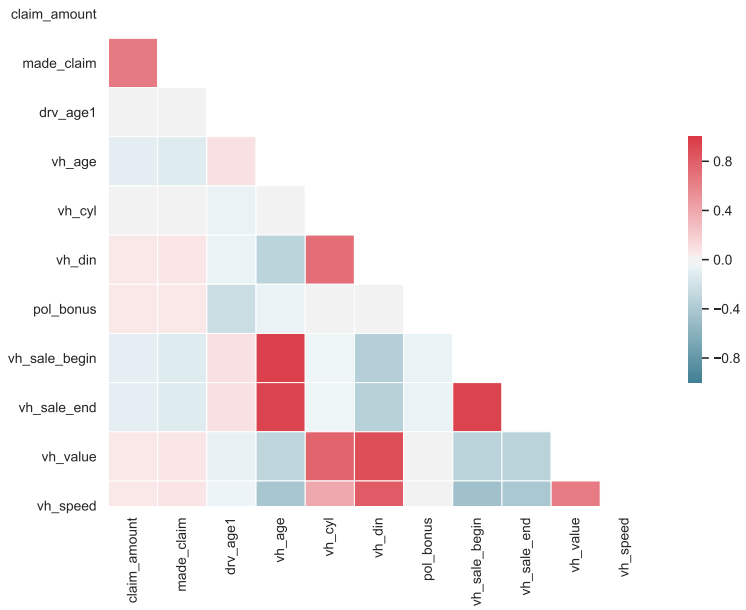


Figure 4: Plot of the correlation matrix for `part2_data.csv`. Dark red indicates a strong positive pairwise correlation between variables whereas dark blue indicates a strong negative correlation.

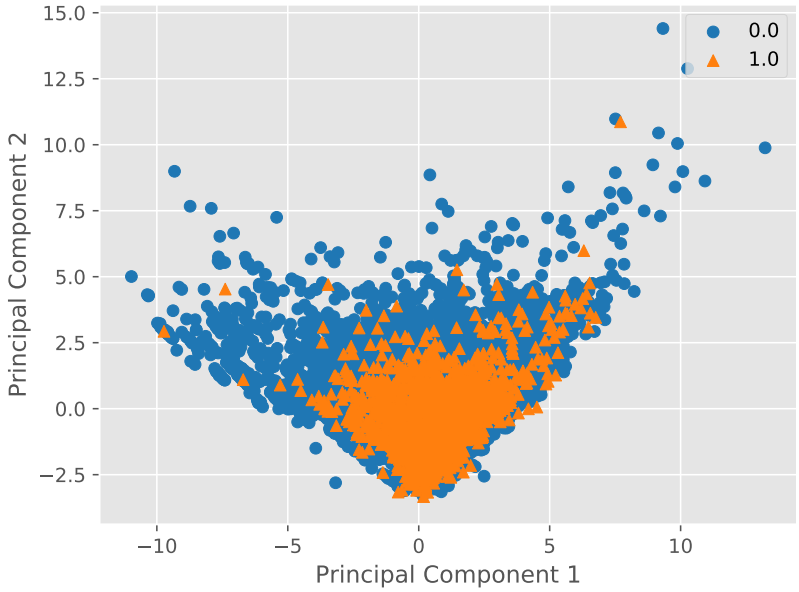
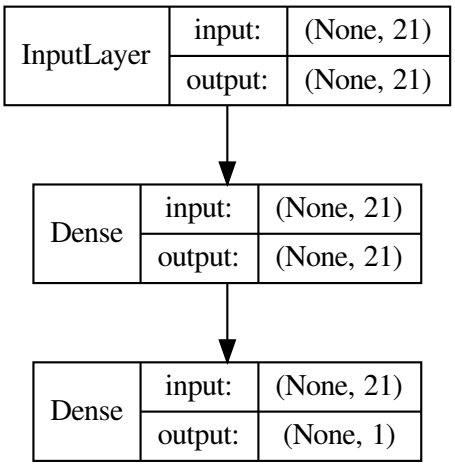
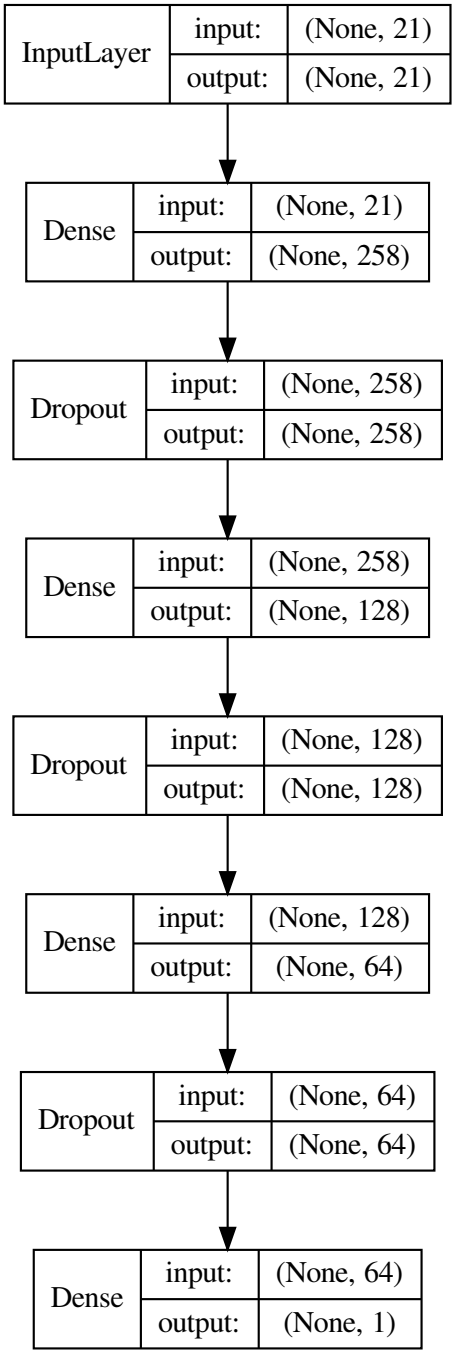


Figure 5: Plot of `part2_data.csv` post dimension reduction with Principal Component Analysis ($n = 2$).



(a) Part 3 Linear model - Single Layer Perceptron



(b) Part 3 Non Linear model.

Figure 6: Model Architectures

.