Deep Learning for Artificial Intelligence Engineering

Assignment 1

Dominic Goj (01626443)
Kirill Smirnov (11941709)

20.11.2025

1 Regression Tasks

1. Task (a)
   Figure 1 shows the distribution of median income (MedInc in $10,000 USD), population size (block group), house age (in years), average occupancy, average rooms and bedrooms of the houses as well as their spatial location in the United States (latitude and longitude) categorized in 50 bins. As the numerical values between the features differ by multiple magnitudes, the data has to be normalized. Looking at the geographical location (longitude and latitude), we can observe the existence of two data "blocks", where most of the houses are located. These blocks probably refer to the West and East Coast of the US. Also we see that most of the houses have residents with a median household income between $20,000 and $60,000 USD.
   For normalization of the data, the Scaler is fitted to the training data by calculating the mean and standard deviation of all numeric values of one feature. Then, the training data is transformed feature-wise - centered around zero and scaled - by using the mean and standard deviations calculated in the prior step. It is important that the validation and test set are not used for fitting but are transformed only with the values calculated from the training data set. Otherwise, our model would be influenced by information from the validation and test dataset. Histograms with normalized values are shown in figure 2.

   The original input dataset *X_train* consists of a 8-dimensional vector (features) with 20,640 individual rows (houses). This dataset was split into a training set of size 14,860, a validation set of size 3,716 and a test set of size 2,064. The output *y_train* is a 1-dimensional vector (price feature).

2. Task (b): 10 different feedforward neural network structures were developed and are shown in table 1. The number of hidden layers varied between 2 and 11. Each network consisted of 8 input units (8 features) and 1 output unit (house price). Variations of the learning rate are given per network structure as an example in table 1. The number of total training epochs was also altered, however, we decided to show only data for models which were trained for 50 epochs together with the best as well as the final training and validation loss. In total, 80 different combinations of model architecture, learning rate and total epochs were tested. We can observe that increasing the number of hidden layers in the network did not necessarily decrease the loss with respect to the validation set. However, networks with only a few hidden layers showed higher losses compared to middle-sized ones. For example, with the model architecture 8-16-8-1 (lr=0.01) the lowest validation loss of 0.344 after 46 epochs could be achieved while for the 8-16-32-16-8-1 structure the lowest validation loss of 0.320 after 41 epochs could be achieved (lr=0.01). One very important factor for faster convergence was the learning rate where losses achieved with learning rates of 0.005 or 0.001 were relatively higher compared to a learning rate of 0.01, in most of the cases, for the same amount of training epochs. Setting the learning rate higher to 0.1 decreased the convergence of the validation loss curve with the training loss curve, as will be shown, and led to strong fluctuations of the validation loss curve.

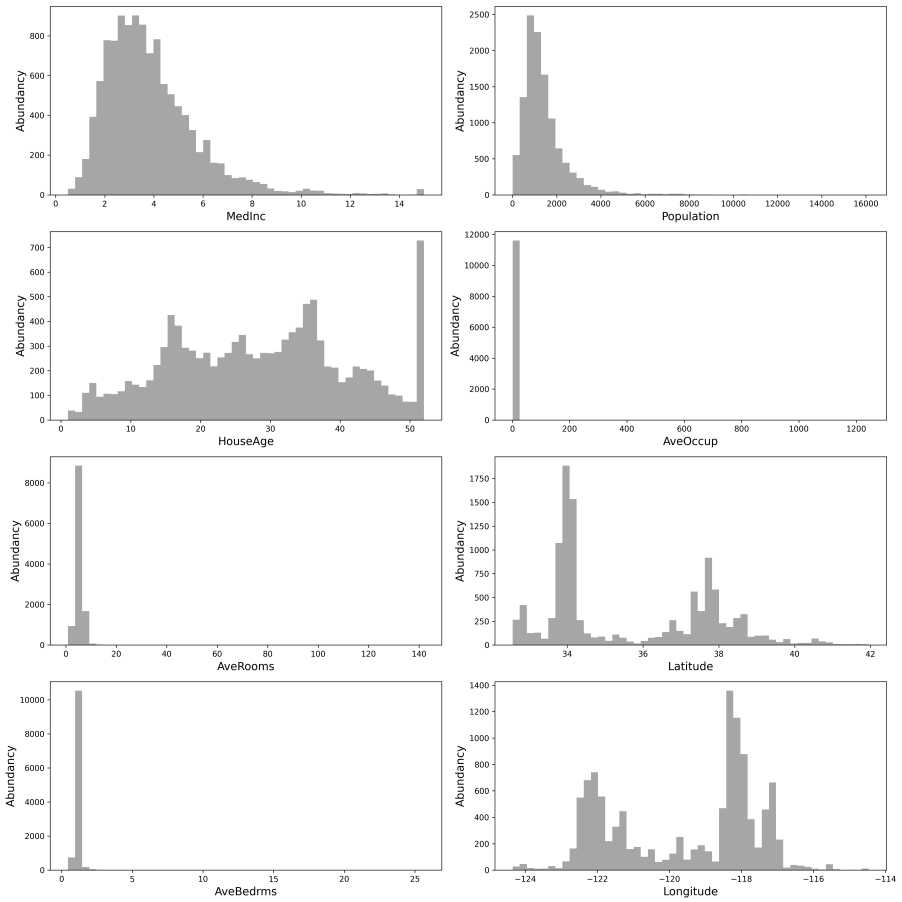

Figure 1: Histogram of california housing dataset features. MedInc = median household income (in $10k USD), AveOccup = average occupancy, AveRooms = average number of rooms, AveBedrms = average number of bedrooms.

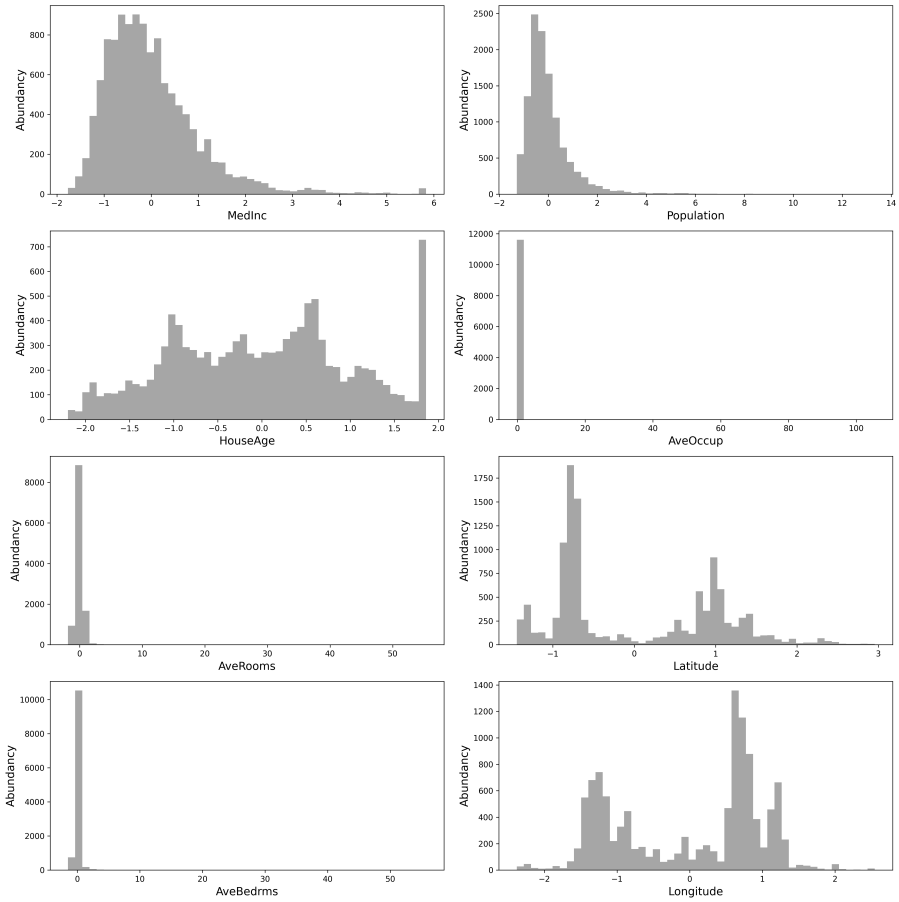

Figure 2: Histogram of california housing dataset features with fit-transformed values. For axes label explanation, refer to figure 1

.

Table 1: Comparison of Neural Network Structures and Training Results. The table summarizes 20 neural network configurations with varying numbers of hidden layers (**HL**) and hidden units per layer (**HU**). For each configuration, training and validation losses are reported for two learning rates (0.01 and 0.005). **Best Train/Val Loss** and **Epoch** indicate the minimum observed loss and the epoch at which it occurred, while **Final Train/Val Loss** refer to the losses after the final 50^{th} epoch.

Hidden Layers	**Structure**	**Learning Rate**	**Best Train Loss**	**Epoch**	**Best Val Loss**	**Epoch**	**Final Train Loss**	**Final Val Loss**
3	[16, 16, 8]	0.01	0.3044	50	0.3392	50	0.3044	0.3392
3	[16, 16, 8]	0.005	0.3285	49	0.3536	49	0.3285	0.3712
4	[16, 32, 16, 8]	0.01	0.2847	50	0.320	41	0.2847	0.3508
4	[16, 32, 16, 8]	0.005	0.3127	50	0.3461	46	0.3127	0.3463
5	[16, 32, 32, 16, 8]	0.01	0.2798	49	0.3154	47	0.2817	0.3294
5	[16, 32, 32, 16, 8]	0.005	0.3022	50	0.3392	48	0.3022	0.3573
6	[16, 32, 64, 32, 16, 8]	0.01	0.2814	50	0.3174	46	0.2814	0.3562
6	[16, 32, 64, 32, 16, 8]	0.005	0.2988	50	0.3388	42	0.2988	0.3866
7	[16, 32, 64, 64, 32, 16, 8]	0.01	0.2803	50	0.3259	49	0.2803	0.4135
7	[16, 32, 64, 64, 32, 16, 8]	0.005	0.2984	50	0.3391	44	0.2984	0.3453
8	[16, 32, 64, 128, 64, 32, 16, 8]	0.01	0.2901	50	0.3299	42	0.2901	0.3912
8	[16, 32, 64, 128, 64, 32, 16, 8]	0.005	0.3035	50	0.3442	43	0.3035	0.3444
9	[16, 32, 64, 128, 128, 64, 32, 16, 8]	0.01	0.4084	50	0.4447	50	0.4084	0.4447
9	[16, 32, 64, 128, 128, 64, 32, 16, 8]	0.005	1.3268	18	1.3102	4	1.3313	1.3583
10	[16, 32, 64, 128, 256, 128, 64, 32, 16, 8]	0.01	0.3443	50	0.3657	50	0.3443	0.3657
10	[16, 32, 64, 128, 256, 128, 64, 32, 16, 8]	0.005	1.3264	30	1.3102	36	1.3319	1.3383
11	[16, 32, 64, 128, 256, 256, 128, 64, 32, 16, 8]	0.01	0.3364	50	0.3614	50	0.3364	0.3614
11	[16, 32, 64, 128, 256, 256, 128, 64, 32, 16, 8]	0.005	1.3276	14	1.3110	47	1.3309	1.3230
2	[16, 8]	0.01	0.3120	50	0.3441	46	0.3120	0.3522
2	[16, 8]	0.005	0.3363	50	0.3626	45	0.3363	0.3789

3. Task (c)
   We decided to use the model architecture of 8 input neurons, hidden layers 16, 32, 16, 8 (4 hidden layers) and the output layer with one neuron (referred to as model 1). With this architecture, we could achieve fairly low losses while still keeping the model simple. The respective loss curves are shown in figure 3. The chosen model architecture achieved the lowest validation loss of **0.320** after 41 epochs. The learning rate was set to 0.01. Setting the learning rate to 0.005 (figure 4) resulted in slightly higher losses when the models were trained for the same number of epochs. Prior tests with a learning rate of 0.1 (figure 5) showed that the validation loss curve did not converge well with the training loss curve for any of the designed architectures. For the repeated training and final prediction on the test set, 25 epochs of training were chosen to be sufficient for a good generalization of the model while still providing good accuracy.

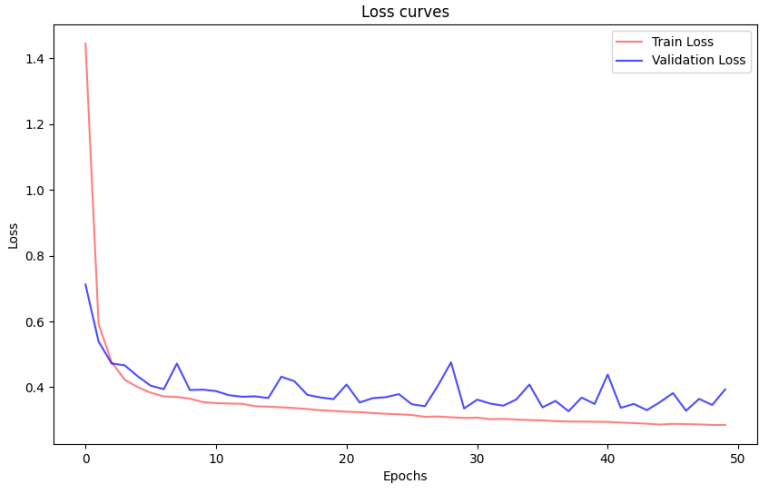

Figure 3: Training (blue) and validation (red) losses over number of epochs for model 1. Lr = 0.01, Total number of epochs = 50.

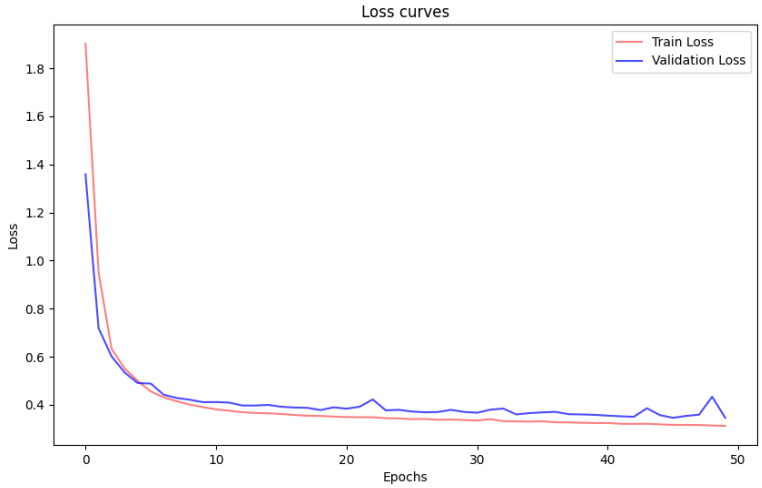

Figure 4: Training (blue) and validation (red) losses over number of epochs for model 1. Lr = 0.005, Total number of epochs = 50.

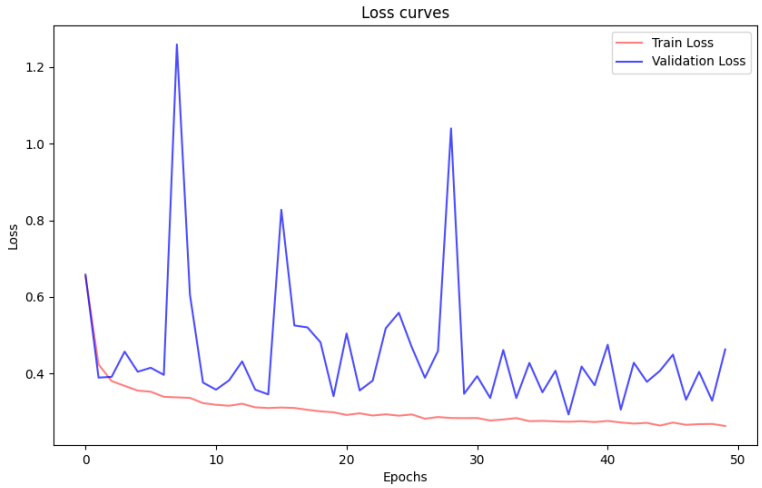

Figure 5: Training (blue) and validation (red) losses over number of epochs for model 1. Lr = 0.1, Total number of epochs = 50.

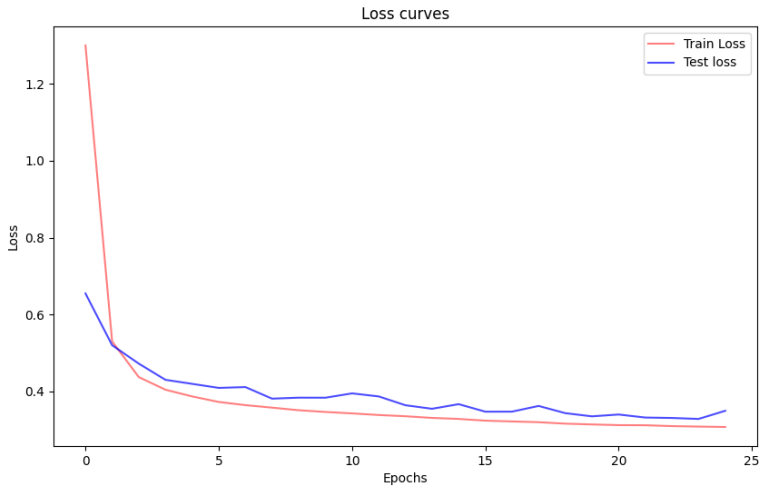

Figure 6: Training (blue
and test (red) losses over number of 25 epochs (Lr = 0.01). The training set consisted of the training and validation set used for model selection beforehand.

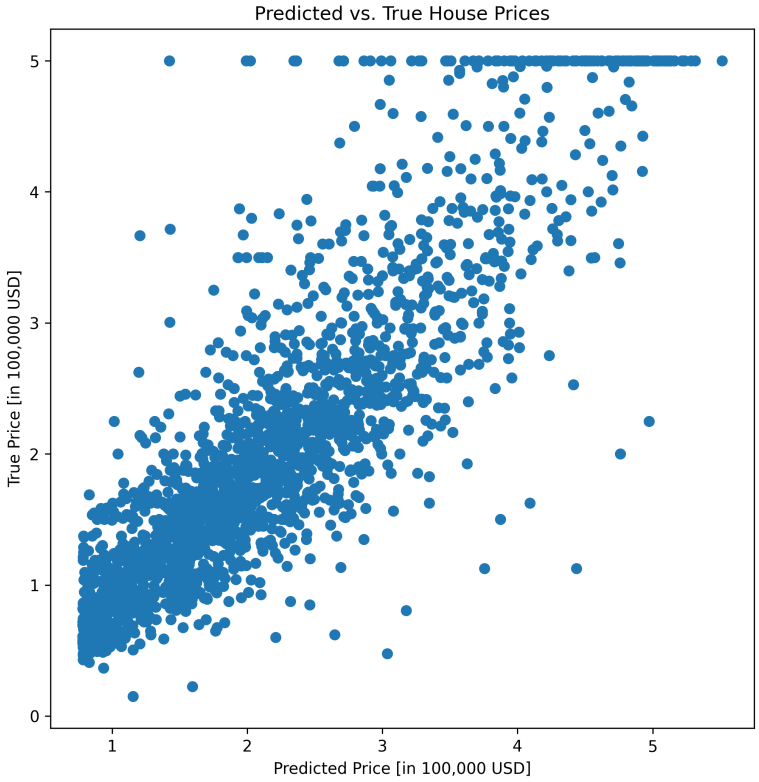

Figure 7: Scatterplot of true (x-axis) and predicted (y-axis) house prices predicted with model 1. House prices in 100,000 USD.

Figure 6 shows the loss curves during training on the training set which consists of the original training and validation set (n=18,576) as well as the loss curves for the test set (n=2064). The **final test loss was 0.349** while the **final training loss was 0.307**. The **lowest test loss was 0.328** in epoch 24 and the **lowest training loss was 0.307** in epoch 25. There is no clear evidence for over- or underfitting from our point of view, as the training and test losses decrease over the number of epochs and both converge in a similar manner. When true house prices and predicted house prices are plotted against each other, one can observe a linear correlation between the data points (figure 7). This indicates that the model is able to predict the house prices in most of the cases. However, the model's accuracy could be improved by testing other network structures or tuning hyperparameters.

2 Binary Classification Task

For this task, the model should be able to classify house prices into either being worth more or less than $200,000 USD. We used a similar model architecture having 8 input neurons, 4 hidden layers (16-32-16-8) and one output neuron. The activation functions for the input neurons and the neurons in the hidden layers were of *Rectified Linear Unit* type. The output activation function had to be changed to a *Sigmoid Activation function* to allow for the output of values between 0 and 1 (class 0 = lower than $200k USD, class 1 = higher than $200k USD). Further, we had to change the loss function from a *Mean Squared Error Loss Function (MSE)* to a *Binary Cross Entropy Loss Function (BCE)*. BCE Loss can to be used for binary classification tasks while MSE is used for regression tasks (predicted output is a logit). We increased the learning rate from 0.01 to 0.05 because we noticed to be stuck in a plateau where accuracy did not increase to over 60%. The model was trained for 15 epochs on a training sample size of 18,576 and tested on a test sample size of 2,064. The final training and test losses were **0.311 and 0.310**, respectively. The best training and test losses were **0.311 (epoch 15) and 0.310 (epoch 11)**, respec-

tively, with an accuracy achieved of **85.3%** in epoch 15 (figure 8).

In addition, we also tried a different variant of the training pipeline and the model architecture which could be, in a later stage, adjusted to predict multiple classes (e.g. house price below $100k USD, between $100k USD and $200k USD, more than $ 200k USD). In this case, the output layer activation function should be *log softmax*. As we have a two class problem, we changed the output layer to have two neurons. *Log softmax* allows to transform input values from the prior hidden layer onto a logarithmic scale which is not limited to values between 0 and 1 (as for the sigmoid output activator function), allowing multiclass prediction. Further, the loss function has to be changed to *NLLLoss* (Negative Log Likelihood Loss). Each column in the predicted output represents the logarithmic probability of the respective class for the respective input data. In order to calculate the accuracy of the model and validate the predicted output for the validation set, the maximum value per row of the predicted output has to be returned by using *logp.argmax(dim=1)* (PyTorch function) to identify the class with the highest predicted logarithmic probability.

Also, we created a confusion matrix (for the BCELoss variant) showing how many test samples were correctly or falsely classified by the model (figure 9). Regarding the multiclass design, a final average test loss of **0.289** on the test set (epoch 13) with an accuracy of **86.1%** could be achieved which is similar to the results achieved with *BCELoss*.

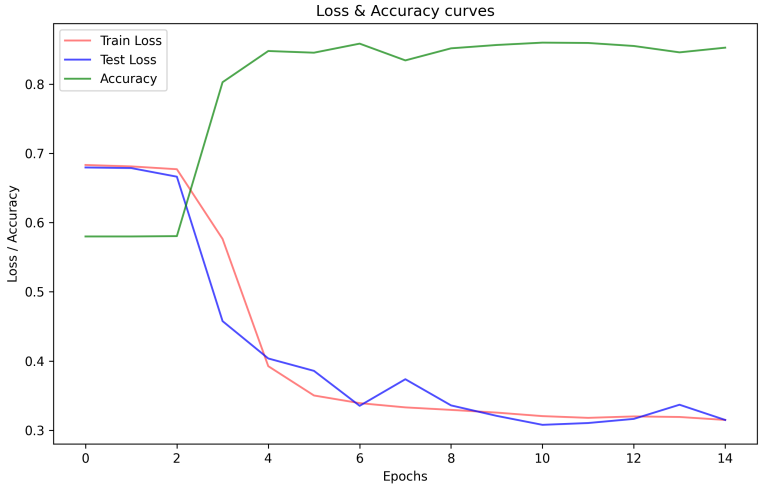

Figure 8: Training (red) and test (blue) losses of the binary classifier as well as prediction accuracy on the test dataset (green) over number of epochs.

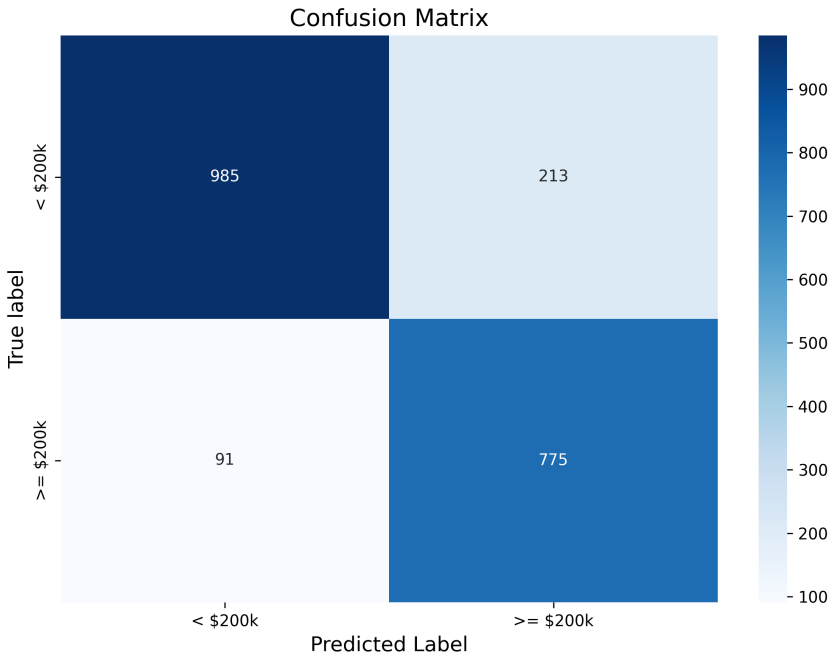

Figure 9: Confusion matrix of predicted and true labels with binary classification model.