University of Colorado Boulder **Amulya Ambati**
CSCI 5922: Neural Nets and Deep Learning **Lab1**
Prof. Danna Gurari **February 7, 2024**

Q1

Introduction

Predicting car insurance claims is a crucial task for insurance companies seeking to optimize risk management, pricing strategies, and resource allocation. By accurately forecasting the likelihood of future claims, insurers can make informed decisions regarding premiums, underwriting policies, and claims processing, ultimately improving operational efficiency and profitability while ensuring fair and competitive pricing for policyholders.
The Dataset is extracted from kaggle (Link to the dataset) and it contains information on policyholders and their attributes like policy tenure, age of the car, age of the car owner, the population density of the city, make and model of the car, power, engine type, etc, and the target variable indicating whether the policyholder files a claim in the next 6 months or not.

Data Preparation and Processing

By analyzing the target column which is 'is_claim', the percentage of 1's (claimed) is way less when compared to 0's (not claimed). Hence claim class looks like an imbalanced class.

target column value	percentage
0	93.6%
1	6.39 %

Train-Test Set

Because of imbalanced class data, a random train test split might not be correct because there is a chance that, all positive samples might go into training/testing and we might not have a chance to train the model/evaluate the prediction on insurance-claimed customers. Hence, proceed with a stratified train test split which allows for a similar distribution of targets in train and test sets.
Performed a train-test split of 75-25 %.The size of the data post-split is as follows:

Train	43,944
Test	14,648

Since the dataset is very imbalanced, after a trial run we observed that the model is predicting everything as 'not claimed' and positive class performance is very poor. Hence, performed downsampling on training data alone to balance the classes before passing it through the model. Data post downsampling,

Train	5,622
Test	14,648

Feature transformation

The dataset contained a few categorical columns such as area_cluster, engine_type, transmission_type, etc. Since the model cannot consume object or string values, some kind of encoding needs to be performed on these features. Post analyzing the number of unique categories present in the categorical columns, most of the columns have Yes/No or a small number of unique values. Hence, proceed with one hot encoding. Also, to avoid collinearity, while creating dummies for the above features, a category was dropped.
Post one-hot encoding, few features have values in a very wide range, hence performed standard scaling on those features, to facilitate faster and better convergence.
Finally, we are left with 100 features.

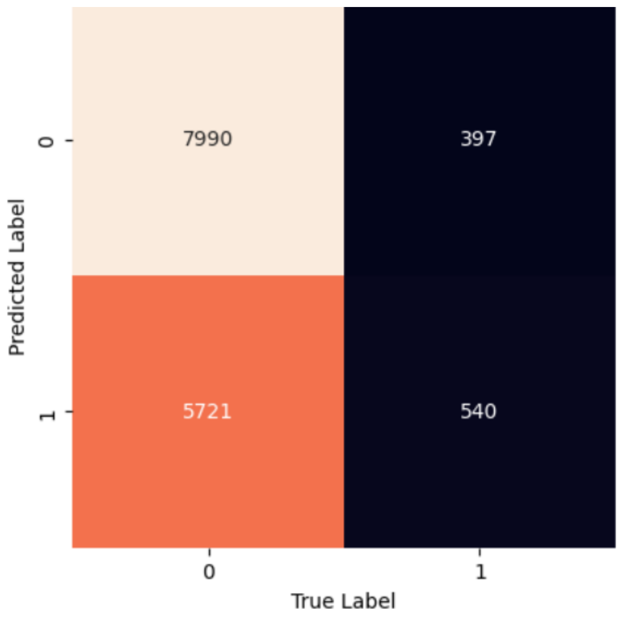
Train shape	(5,622,100)
Test shape	(14,648,100)

Modeling

Various Multi-layer perceptron architectures were tested, by varying numbers of layers, numbers of neurons per layer, and activation functions.
Few of the constant model parameters were
max_epochs = 500
Epochs = early_stopping(max_epochs)
learning rate = 0.001
batch size = 200
The reason for taking layer sizes 32 and 64 is, that since our input feature vector size was 100, we would like to transform that information into lower dimensional space.

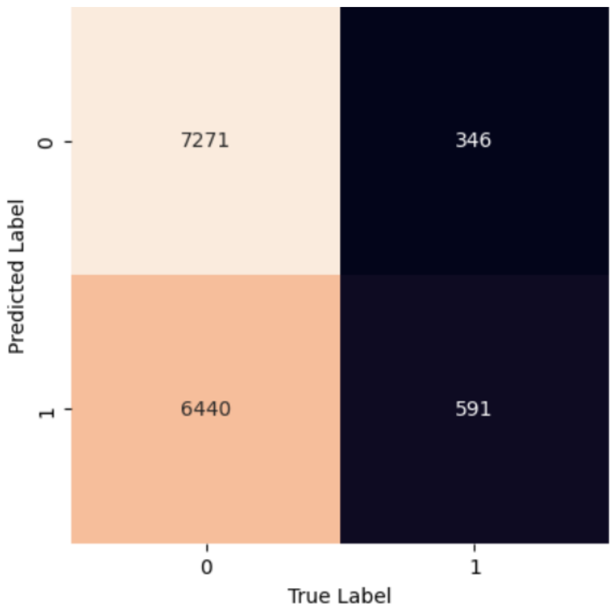
Layers and Neurons	Activation function	Train accuracy	Test accuracy	No. of epochs ran	Time taken (i
[32]	Tanh	59%	58%	89	3.3
[32]	Relu	61%	53%	234	7.8
[64]	Tanh	60%	45%	190	10.9
[64]	Relu	61%	56%	133	5.5
[64,32]	Tanh	59%	49%	56	4.19
[64,32]	Relu	63%	51%	196	10.4
[128,64,32]	Tanh	70%	55%	284	50.1
[128,64,32]	Relu	73%	55%	362	37.0

In-depth analysis of models by looking at confusion matrix, precision, and recall.
Model 1 :
Layers and Neurons : [32] Activation function: 'tanh'



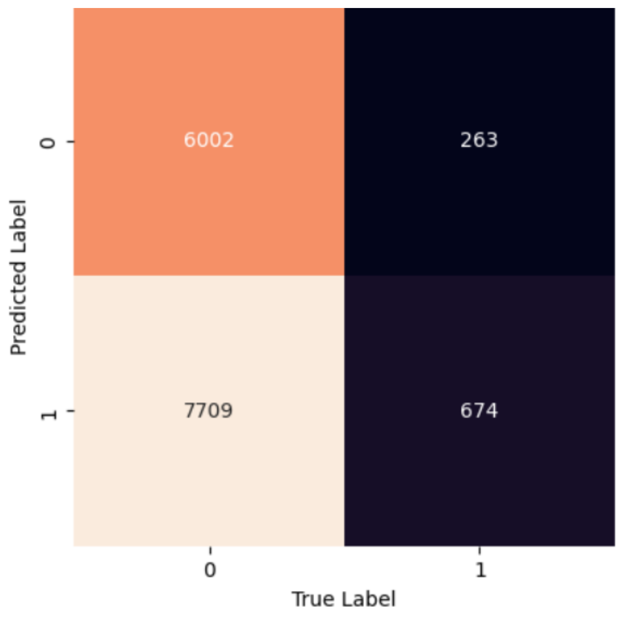
Target variable	Precision	Recall	F1-score
0	0.95	0.58	0.72
1	0.09	0.58	0.15

Model 2 :
Layers and Neurons : [32] Activation function: 'Relu'



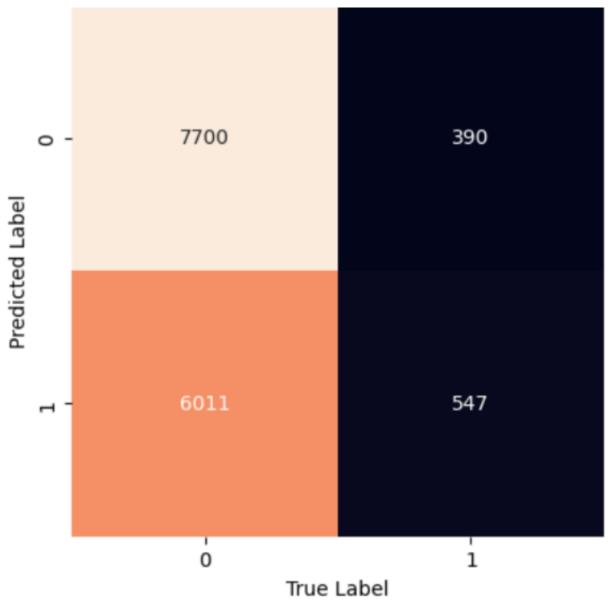
Target variable	Precision	Recall	F1-score
0	0.95	0.53	0.68
1	0.08	0.63	0.15

Model 3 :
Layers and Neurons : [64] Activation function: 'tanh'



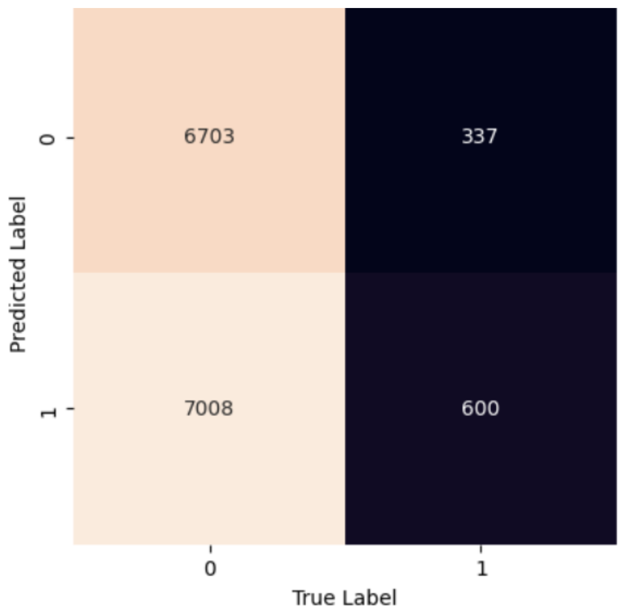
Target variable	Precision	Recall	F1-score
0	0.96	0.44	0.60
1	0.08	0.72	0.14

Model 4 :
Layers and Neurons : [64] Activation function: 'Relu'



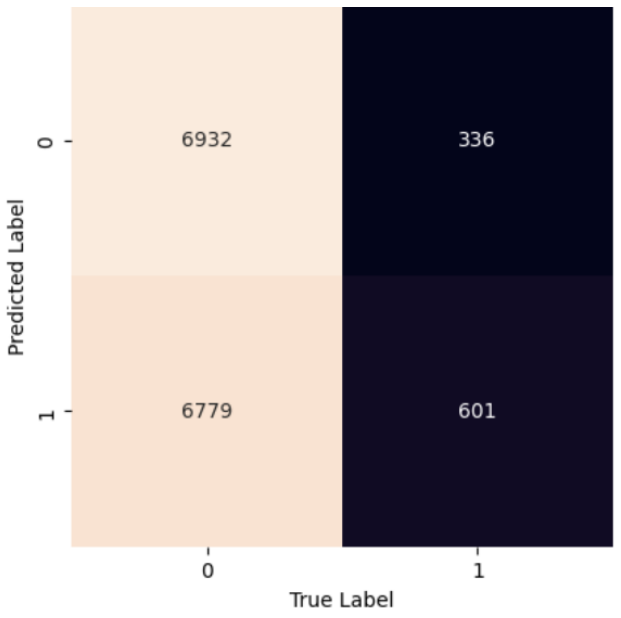
Target variable	Precision	Recall	F1-score
0	0.95	0.56	0.71
1	0.08	0.58	0.15

Model 5 :
Layers and Neurons : [64,32] Activation function: 'tanh'



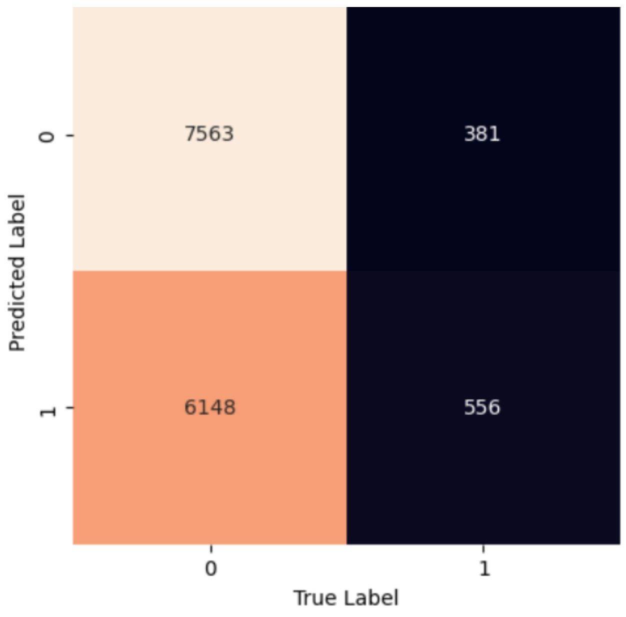
Target variable	Precision	Recall	F1-score
0	0.95	0.49	0.65
1	0.08	0.64	0.14

Model 6 :
Layers and Neurons : [64,32] Activation function: 'Relu'



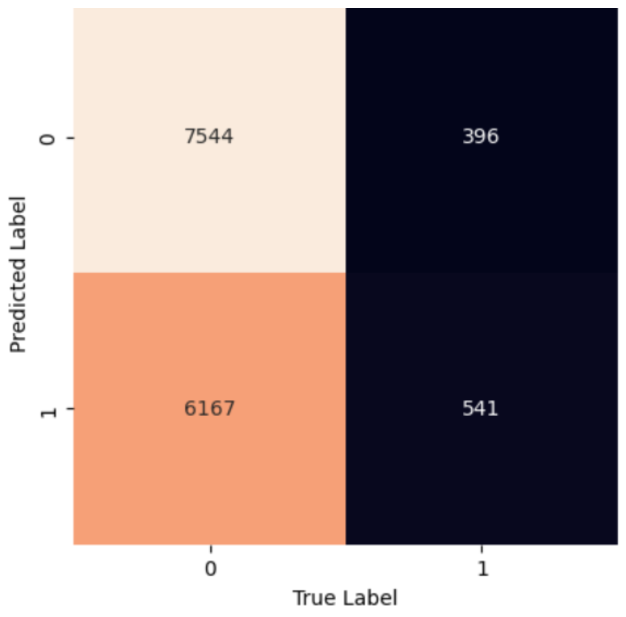
Target variable	Precision	Recall	F1-score
0	0.95	0.51	0.66
1	0.08	0.64	0.14

Model 7 :
Layers and Neurons : [128,64,32] Activation function: 'tanh'



Target variable	Precision	Recall	F1-score
0	0.95	0.55	0.70
1	0.08	0.59	0.15

Model 8 :
Layers and Neurons : [128,64,32] Activation function: 'Relu'



Target variable	Precision	Recall	F1-score
0	0.95	0.55	0.70
1	0.08	0.58	0.14

The performance of all 8 models is bad especially for class 0, as the average recall across all the models is approximately 50% and the precision for class 1 is also pretty low.

Observations/Patterns

Since early stopping is applied, all 8 different models ran for different numbers of epochs according to based on convergence and loss.

As a general trend, I observed and understand that with Relu activation function model is taking more iterations to converge.

Another trend that I observed is, that even though with Relu activation the model is taking more iterations, the time taken for the training is comparatively less, this might be because the time taken to compute the gradient of Relu is relatively less.

With increasing layers or number of neurons, train accuracy is increasing a bit. This is because the complexity of the model increases thereby able to fit the data better.

Q2

Introduction

In the modern financial landscape, credit scores play a pivotal role in determining an individual's financial health and access to various financial products and services. A credit score is a numerical representation of an individual's creditworthiness, calculated based on their credit history and other relevant financial factors. Lenders, such as banks and credit card companies, use credit scores to assess the risk of lending money or extending credit to individuals.

In this context, credit score classification refers to the categorization of credit scores into distinct groups based on predefined ranges. These classifications serve as a benchmark for assessing an individual's creditworthiness and determining their eligibility for credit products, interest rates, and loan terms.

The data for this analysis was sourced from Kaggle (Link to the dataset). This classifies the scores into standard, poor, and good with specific attributes and score ranges associated with each category and various other aspects. A total of 20 attributes were given using which we need to predict the credit class.

Data Preparation and Processing

By analyzing the target column which is 'Credit_Score', to understand the distribution of the target variable.

target column value	percentage
Good	17.83 %
Poor	29 %
Standard	53.16%

Train-Test Set

It's better to proceed with a stratified train test split which allows for a similar distribution of targets in train and test sets thereby better model training and better test performance

Performed a train-test split of 70-30 %.The size of the data post-split is as follows:

Train	69,972
Test	29,988

Feature transformation

The dataset contained a few categorical columns such as Payment_of_Min_Amount, Credit_Mix, Payment_Behaviour, etc. Since the model cannot consume object or string values, some kind of encoding needs to be performed on these features. Post analyzing the number of unique categories present in the categorical columns, most of the columns have a small number of unique values.

Hence, proceed with one hot encoding. Also, to avoid collinearity, while creating dummies for the above features, a category was dropped.

Post one-hot encoding, few features have values in a higher range like 50,000, etc, hence performed standard scaling on those features, to facilitate faster and better convergence.

Finally, we are left with 26 features to model upon.

Train shape	(69,972, 26)
Test shape	(29,988, 26)

Modeling

The aim here was to test the performance and learning curve by varying training data size. Hence 4 different experiments were performed with the same architecture and, the same hyperparameters but with different amounts of train data.

Model settings were as follows:

Layers = [64,32]

Activation = 'tanh'

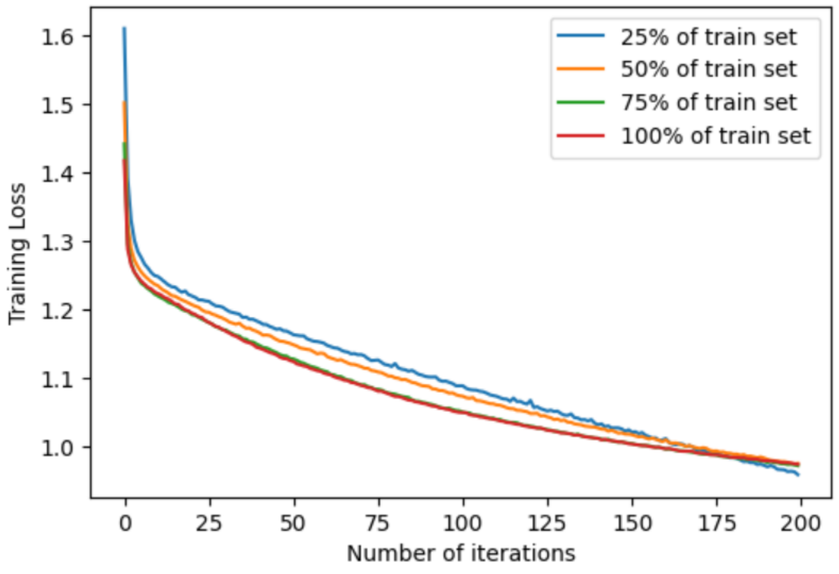
Epochs = 200

learning rate = 0.001

batch size = 200

Training data %	Training data size %	Train accuracy	Test accuracy	Time taken (in seconds)
25%	17,493	74.2%	63.84%	58
50%	34,986	74.5%	66%	110
75%	52,479	74.5%	66%	166
100%	69,972	74.7%	66.8%	271

The learning curve of the 4 experiments



Observations/Patterns

As expected, as the training data size increased, training time has also increased. Because the model needs to run on more batches(more iterations) in every single epoch. Moreover, the time taken should be linear with the amount of data, which is also seen in the above experiment.

Along with that, with increasing epochs, training loss is also going down. Hence if run for a few more epochs training loss might further decrease.

We would usually expect training loss to be low and test accuracy to be high with a higher amount of training data because the model has more examples to learn from. But in this case, this was not observed, maybe because the extra data points in 50%, 75%, and 100% are not adding any new information to the model.

One specific point to observe here is training loss on the first epoch across all four different experiments, for 100% of training data, the initial loss was less when compared to other experiments, because the data was trained on more batches and more weight updates have been occurred by the end of the first epoch when compared to other experiments.