Machine Learning
Final Presentation:
Artificial Neural Network

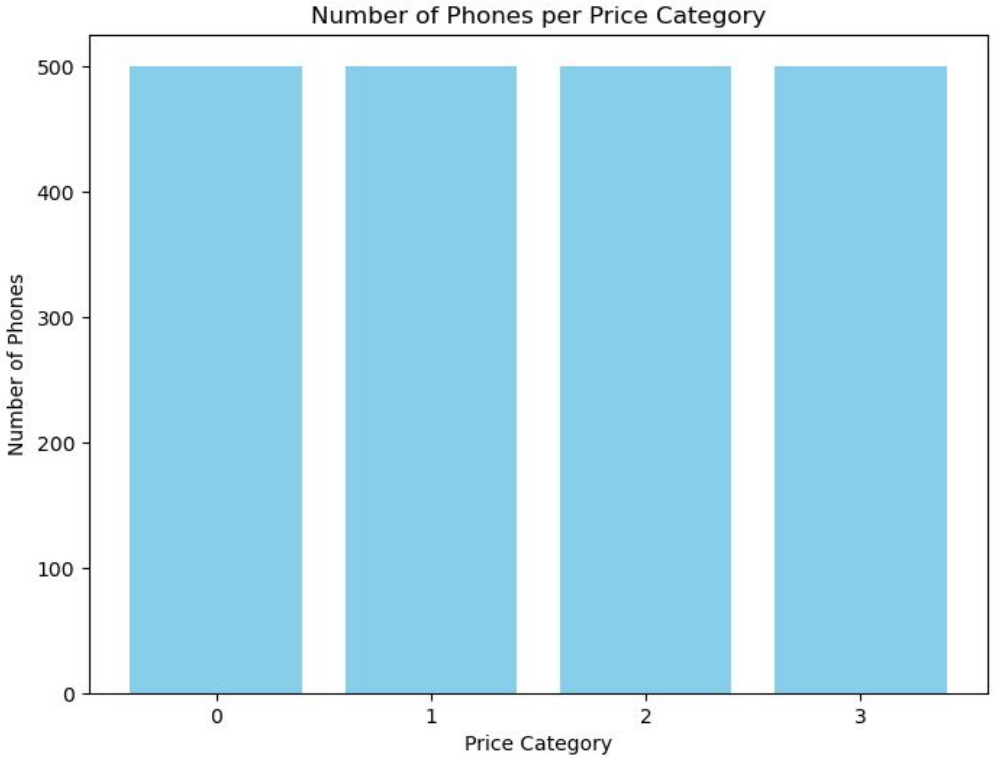
Student: Vinícius Santana da Silva Brandão
Professor: Daniel de Filgueiras Gomes

The problem - Phone Price Classification

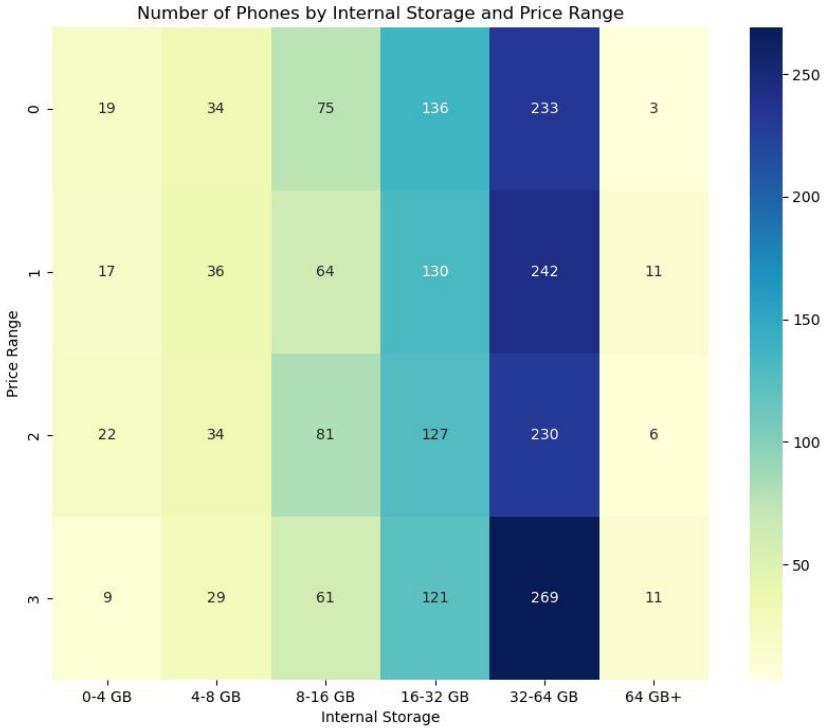
In this problem, we want to make an estimation of mobile phone prices based on their features (such as RAM, Internal Memory) in a competitive market. To accomplish this, we aim to analyze sales data from various mobile phone companies. The goal is to establish a relationship between specific features and the selling price, ultimately providing a price range that indicates the relative level of pricing.

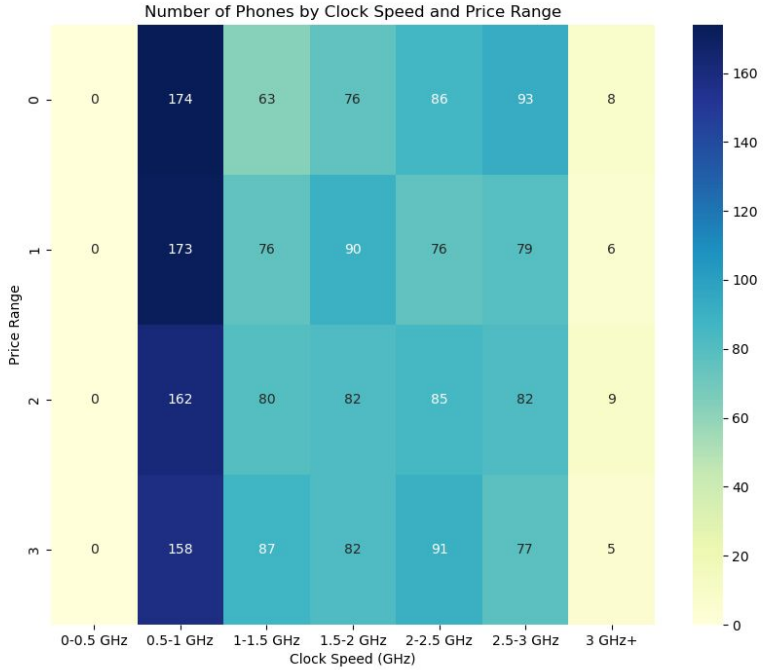
About the dataset

The dataset has 20 characteristics plus the classification, all of them were used to solve the problem. The characteristics were all phone specifications such as battery capacity, clock speeds, camera resolutions, dimensions, wifi and more. There are 4 different price categories, numbered from 0 to 3.

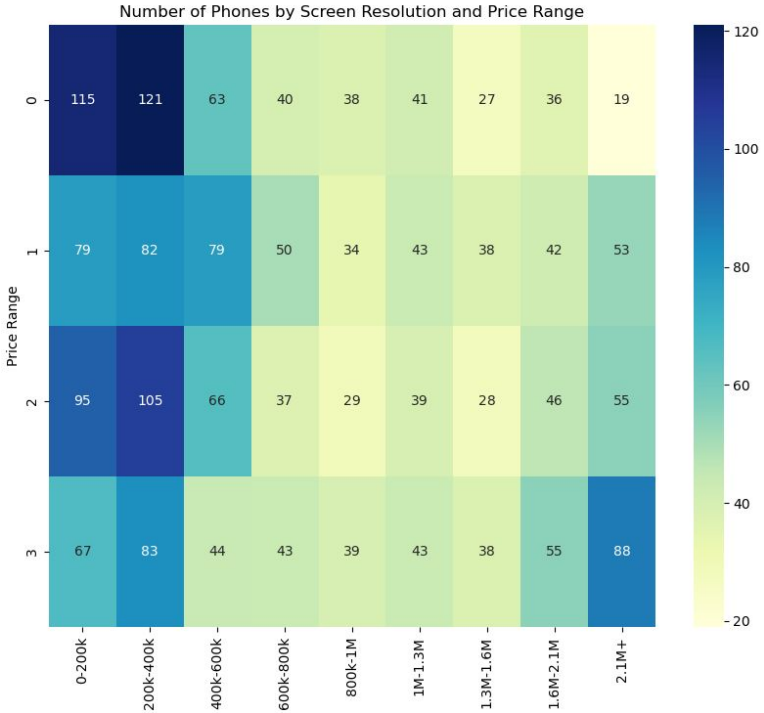


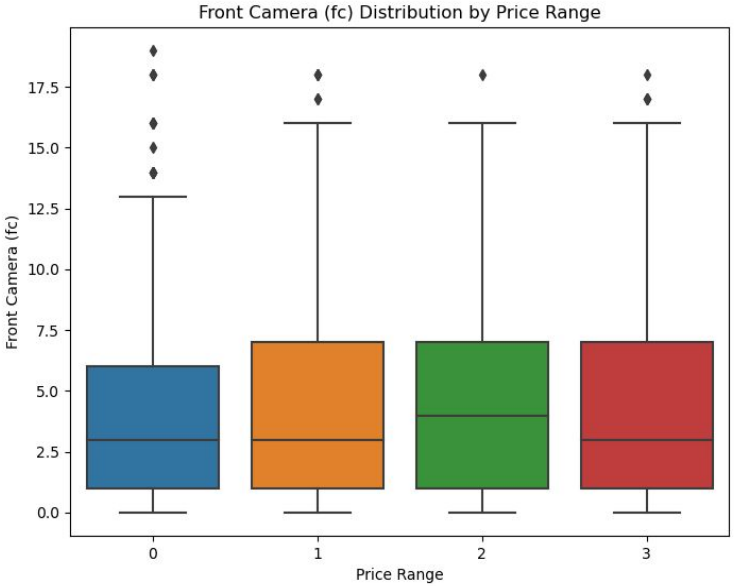
Phones by price category, evenly distributed. 4 categories with 500 phones each





Distribution of not so useful data





Distribution data more useful data

The fed data

The inputs given to the network were all the phones' characteristics normalized They were all normalized using

x normalized = (X - mean) / std.

In total, there were 2000 phones evaluated separated in a train set (1600 phones) and 2 test sets (200 phones each)

The outputs were the estimation for each phones' categories.

The solution process

To solve this simple problem, not more than a simple solution is required. So, the objective was to use an artificial neural network with at least 1 hidden layer, optimization and different activation functions, so that they could be utilized to their best cases.

Unfortunately, no regularization or cross validation was implemented.

The neural network

```
# Initializing the network and defining stricture:
n_inputs = len(X_train_normalized[0])
n_outputs = 4  # 4 -> output classes based on the provided dataset

dense1 = Layer_Dense(n_inputs, 32)  # Inicial Layer with 32 Neurons
# ReLU actovation function for all layers except output
activation1 = activation_ReLU()
dense2 = Layer_Dense(32, 16)  # Hidden layer with 16 neurons
activation2 = activation_ReLU()
dense3 = Layer_Dense(16, 8)  # Hidden layer with 8 neurons
activation3 = activation_ReLU()
dense4 = Layer_Dense(8, n_outputs)  # Output layer with N outputs to match the categories
                                    # In this case, N is 4
activation4 = activation_softmax()  # Softmax activation for output layer
```

The chosen network has 2 hidden layers, an input layer and an output layer. The activation functions chosen for the input and hidden layers were ReLU and the output layer used a softmax function.

The parameters chosen

```
optimizer = Optimizer_SGD(learning_rate=0.1)  # Adjust the learning rate as needed

# setting hyperparameters
num_epochs = 1500
```

0.1 Learning rate brought a good result, with quick convergence and a satisfactory precision for this project's intent.

1500 epochs is a little further than what would be optimal for this problem, but is better for graph visualization.

```
test1_size = 0.1
test_size = 0.1
random_state = 133
```

The train/test split was 80% train and 20% test, no extra validation method was introduced for training. This has probably induced a larger training bias than if it had some validation method

More about the neural network

```
# Forward pass
dense1_output = dense1.forward(X_train_normalized)
activation1_output = activation1.forward(dense1_output)
dense2_output = dense2.forward(activation1_output)
activation2_output = activation2.forward(dense2_output)
dense3_output = dense3.forward(activation2_output)
activation3_output = activation3.forward(dense3_output)
dense4_output = dense4.forward(activation3_output)
train_output = activation4.forward(dense4_output)

# Calculates Loss
loss = softmax_loss.forward(train_output, y_train)

# Backward pass
y_one_hot = np.eye(n_outputs)[y_train]

grad_loss = (train_output - y_one_hot) / len(y_train)
softmax_loss.backward(grad_loss, y_train)
dense4.backward(grad_loss)
activation3.backward(dense4.dinputs)
dense3.backward(activation3.dinputs)
activation2.backward(dense3.dinputs)
dense2.backward(activation2.dinputs)
activation1.backward(dense2.dinputs)
dense1.backward(activation1.dinputs)
```

- The network is being trained using batches, which is going to update the weights and biases only once for every dataset pass.
- The back propagation is being done manually, calculating loss, using a -log(correct class confidence) function, which is in itself a simplification of the categorical cross entropy function for machine learning.
- This loss is applied in the backward pass to update the network's information and enforce the learning process.

Optimization

```python
class Optimizer_SGD:
# Initialize optimizer - set settings,
# learning rate of 1. is default for this optimizer
    def __init__(self, learning_rate=1.0):
        self.learning_rate = learning_rate
        self.iterations = 0

    # Update parameters
    def update_params(self, layer):
        layer.weights += -self.learning_rate * layer.dweights
        layer.biases += -self.learning_rate * layer.dbiases
        self.iterations += 1
```

```python
optimizer = Optimizer_SGD(learning_rate=0.1)
```

The optimization is made via Stochastic Gradient Descendent (SGD), which seeks to find the global minimum for the loss function, bringing the closest possible network to it's optimal results. The learning rate will adjust the intensity with which the network will be updated in each epoch.

Good learning rate examples:
Good Learning Rate
Good Enough Learning Rate

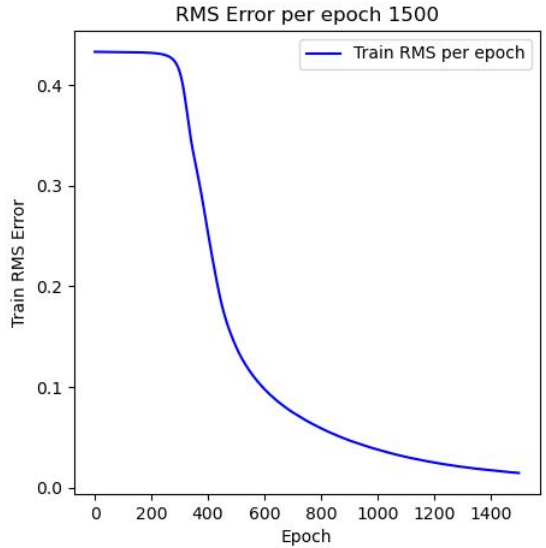
Results: accuracy and confusion matrix

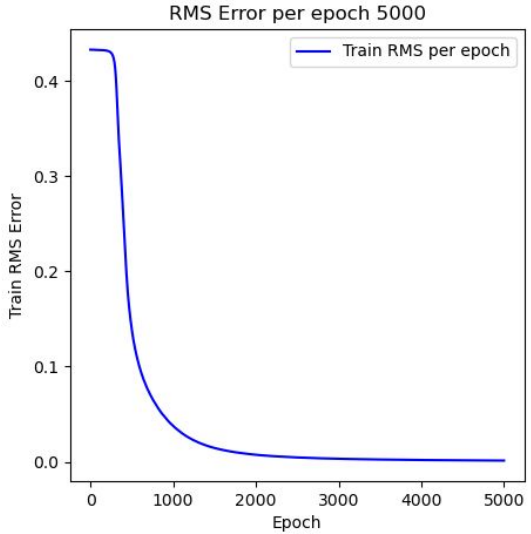
```
Epoch 1500/1500, Loss: 0.7489, RMS Error: 0.0145
Train Accuracy: 100.00%
test1 Accuracy: 90.00%
Test Accuracy: 93.00%
Confusion Matrix for test1:
[[43  0  0  0]
 [ 6 40  0  0]
 [ 0  4 48  1]
 [ 0  0  9 49]]

Confusion Matrix for Test:
[[46  0  0  0]
 [ 4 48  0  0]
 [ 0  2 46  3]
 [ 0  0  5 46]]
```

These are the final results for the training with the configurations shown in this presentation. This is a good enough accuracy for test and test1.

The mislabeling happens mainly with adjacent groups, and the network is biased towards classifying down the phones, with 30 downwards classifications and 4 upwards in this particular setting.

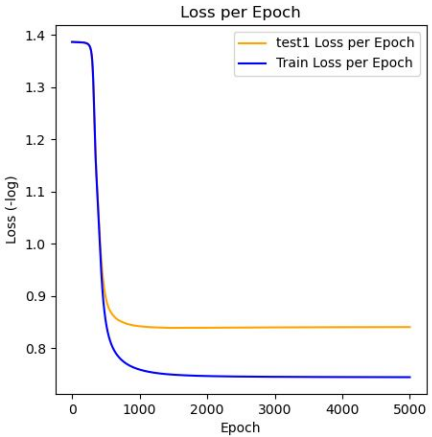
Results: RMS error per epoch

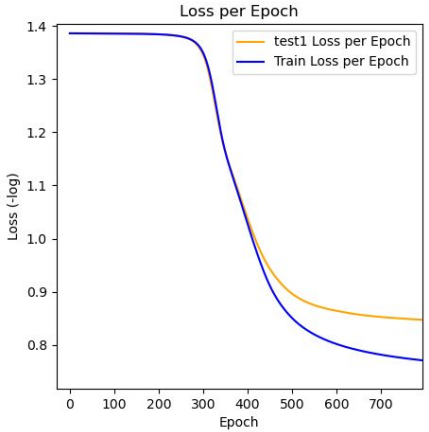


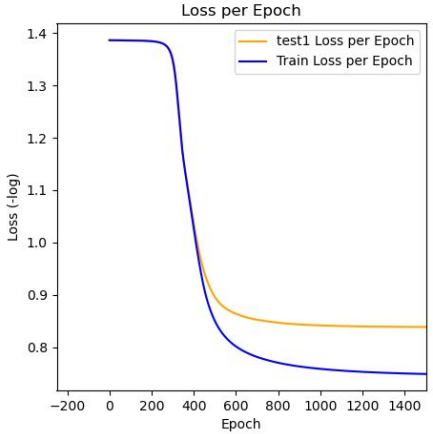


The RMS error converges around 1500 ~ 2000 epochs on the training data, showing no more significant progress over 2000 passes.

Results: Loss Per Epoch







The loss converges between 700 ~ 1500 epochs, for the training dataset and between 400 and 800 for the test1 set.

This means that over a 1500 pass run, the network is overfitted, but it is still very effective given the results shown previously

Questions

Sources & code

Problem:

Mobile Price Classification

consulted material:

Neural Networks from Scratch

The code is available at:

https://github.com/TeuPremium/Phone_Price_Prediction

student email:vinicius.brandao@ufpe.br