Understand the Dataset

To understand the dataset, we first read the problem description on Kaggle and explored the features along with their distributions. We recognize that it might have been helpful to plot our own distributions to better understand the relationships between features and the target variable (price range). However, given how common this dataset is and the ideas behind it, we were able to correctly estimate the correlations between each feature and the price range. We did not find any features that stood out as requiring special handling. Furthermore, Kaggle provided distributions of the features, which allowed us to visually confirm that there were no overly strong correlations between any two features.

Preprocess the Data

We checked all features for missing values and found none, which simplified the preprocessing. We then examined the categorical features to determine how best to use their values. Since they were already encoded as binary (e.g., 3G, 4G, WiFi represented as 0/1), no further encoding was necessary. Additionally, because we did not find any features that were highly correlated, we did not drop or merge any of them. Finally, we applied a standard scaler to standardize the data. Overall, we were satisfied with the quality of the dataset.

Build an MLP Model

Next, we built a five-layer multilayer perceptron (MLP). The network architecture started with 722 neurons, followed by 128, 64, 32 and 16, leading to a final output layer of 3 neurons. We applied dropout with varying rates per layer to reduce the chance of overfitting while ensuring we did not drop too many neurons. This added robustness and allowed us to create an ensemble-like effect to improve performance. We also used the leaky ReLU activation function at each layer to address linearity and further enhance model performance.

Train and Validate

For training and validation, we first set aside 15% of the data as a test set for final evaluation. The remaining data was split into 77% for training and 33% for validation. We initially tried an 85%/15% split, but the small validation set size led to poor validation results, so we adjusted accordingly. Since the dataset was relatively small, ensuring a reasonable validation set size was important.

During training, we implemented early stopping to prevent overfitting and ensure training stopped at the optimal point. We also applied weight decay to the Adam optimizer for regularization and used learning rate scheduling to dynamically adjust the learning rate, which

helped stabilize convergence. With these techniques, we obtained strong results after multiple rounds of training and validation. The best-performing model was then used on the test set.

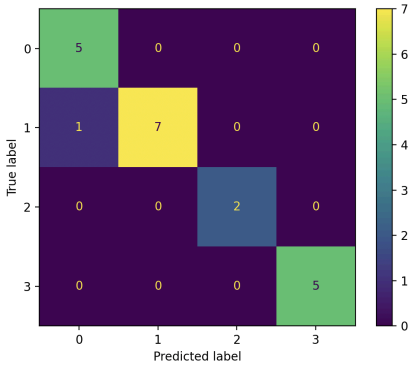
Evaluation

We evaluated the model by minimizing the loss function and maximizing accuracy. Our model successfully predicted the classes with high accuracy. We included a confusion matrix and also evaluated our model based on recall, which is the true positive rate, and the F1-score, which emphasizes the true positive rate.

```
Classification Report
              precision    recall  f1-score   support

           0       0.83      1.00      0.91         5
           1       1.00      0.88      0.93         8
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         5

    accuracy                           0.95        20
   macro avg       0.96      0.97      0.96        20
weighted avg       0.96      0.95      0.95        20
```



Final Prediction

The final model achieved an accuracy of **91.76%**, meaning it correctly predicts phone price ranges 91% of the time.

External Resources

- https://www.w3schools.com/python/python_file_write.asp

- https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity
- https://docs.pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html#torch.nn.LeakyReLU
- https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html