ELL409 - Machine Intelligence and Learning Semester Project Report

Cuisine Prediction

Kriti Garg, Yagya Goyal, Kartikey Agarwal

November 24, 2024

Abstract

This project explores cuisine prediction using a machine learning approach applied to the dataset provided. By leveraging a feedforward neural network with TF-IDF feature extraction and class balancing via SMOTE, the model predicts the cuisine type based on a list of ingredients. The results demonstrate the potential of simple neural network architectures for multi-class classification tasks in text-based datasets.

Our code and data can be found at this link: Cuisine Prediction ELL409

1 Problem Statement

The goal of this project is to develop a machine learning model capable of predicting the type of cuisine for a given recipe based on its list of ingredients. Using a dataset containing recipes with ingredients and their corresponding cuisines, the task involves training an Artificial Neural Network (ANN) to classify cuisines like Indian, Mexican, Moroccan, etc. The dataset is provided in JSON format, with ingredients as input features and cuisine type as the target variable. The final evaluation metric is overall accuracy, measured by comparing predictions on a test dataset to true labels. Results are submitted in a specified CSV format.

2 Introduction

Cuisine prediction is a multi-class classification problem where the goal is to predict the type of cuisine based on the ingredients in a recipe. This project focuses on developing an efficient machine learning pipeline to handle the task, starting from data preprocessing and feature extraction to training a feedforward neural network. The provided dataset is used as a benchmark, which includes recipes labeled with their respective cuisines and ingredient lists. This report highlights the methodology, experiments, and outcomes of this approach.

3 Methodology

3.1 Data Preprocessing

The dataset comprises recipes with their ingredients and corresponding cuisines. We observed that the dataset was highly imbalanced. To address class imbalance, SMOTE (Synthetic Minority Oversampling Technique) was applied to the training data, ensuring all classes were equally represented.

1. Ingredient tokens were preprocessed by replacing spaces with underscores (e.g., ‘"soy sauce"‘ becomes ‘"soy_sauce"‘) to ensure uniformity for text analysis, and ensure multi-word ingredients were treated as single ingredient only and not 3 ingredients if it read "grated pamesan cheese".

2. Each recipe's list of ingredients was joined into a single string (e.g., "sugar salt butter"). This was done separately for the training and test datasets.

3. TF-IDF (Term Frequency-Inverse Document Frequency) was used to transform the ingredient lists into a vectorized numerical format (limited to 4508 ingredients/features for this case).

3.2 Model Architecture

Feedforward - The front pass
The feedforward neural network was designed with:

1. Input layer matching the number of TF-IDF features.

2. Two hidden layers with 128 and 64 neurons, using ReLU activations and dropout for regularization.

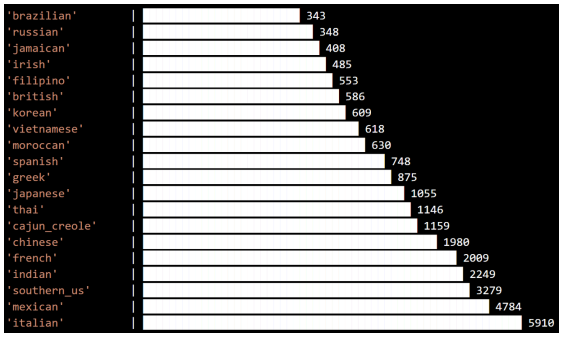

Figure 1: Imbalance in dataset

3. A softmax output layer for multi-class classification.

Every layer has an unsaid linear layer component, which is essential to transform data in a learnable, flexible way.

While this is the final architecture established, we went through a lot of steps to establish which one was the best suited for our problem statement. These steps are described in the following section.

The backward pass Except for the feed-forward neural network architechture, here is a small description of our back propagation. We set up an Adam optimizer, and used it to back propagate gradients for each batch in each epoch, i.e. we used mini batch gradient descent.

3.3 Training, Optimization and Evaluation

Data Splitting and Conversion

1. **Training and Validation Split:** train_test_split divides the resampled training data into training and validation sets (80 %- 20% split).

2. **Tensor Conversion:** The split data is converted into PyTorch tensors for use in the neural network.

3. **Dataloaders:** PyTorch DataLoader objects are created for both training and validation sets to manage batching during training.

Training the Model

1. **Loss Function:** CrossEntropyLoss is used for multi-class classification.

2. **Optimizer:** Adam optimizer with a learning rate of 0.001 is used for training.

3. **Training Loop:** For a set number of epochs. The model is trained instance by instance using the training DataLoader. The number of epochs are kept large enough to see the accuracy converging to a value.

4. **After each epoch:** The validation set is evaluated, and metrics such as loss and accuracy are calculated.

Testing and Prediction:

1. The trained model predicts the cuisines for the test data (X_test).

2. The predicted labels are converted back to cuisine names using the LabelEncoder.

Evaluation Metrics: Although the choice of evaluation metrics could be diverse, it was clearly mentioned in the aim of the project to choose accuracy as the evaluation metric and hence we did not do a lot of trials in this domain.

Here are a few models that we trained and compared:

1. Balanced vs Imbalanced data.

2. Preprocessed vs Untouched data.

3. Training With Dropout vs Without Dropout layer.

4. Comparison Between different number of layer and number of neurons per layer.

5. Comparison Between different batch sizes

6. Changing the maximum number of features to take into account.

The plots and accuracy results can be found in the next section.

4 Results

Here are the plots for each of our comparisons.

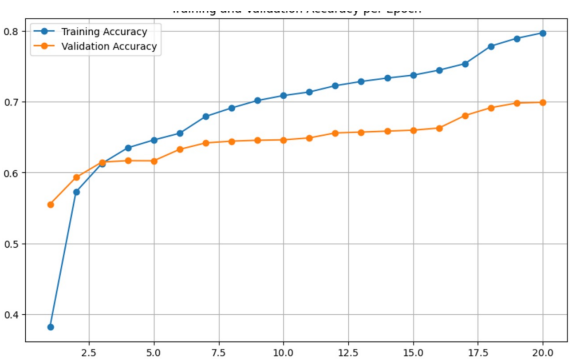

Figure 2: Without Data Balancing

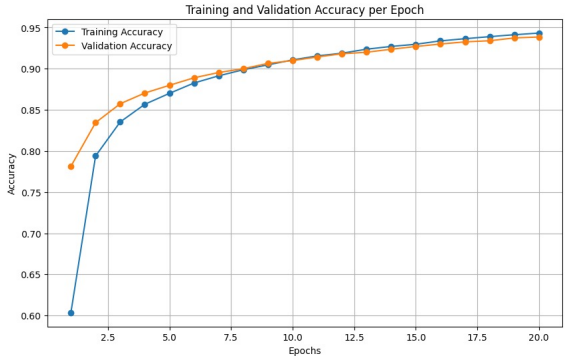

Figure 3: Without Preprocessing

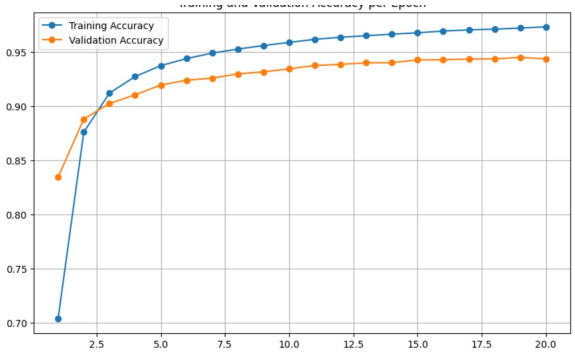

Figure 4: Without Dropout

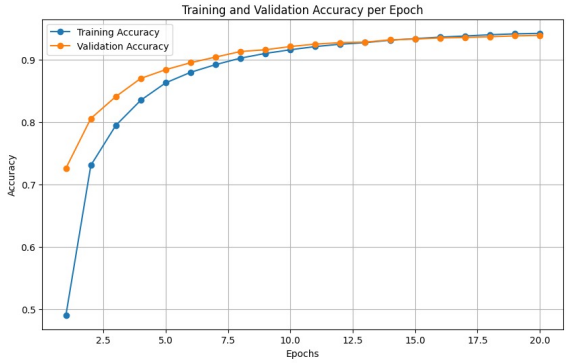

Figure 5: Using 3 hidden layers

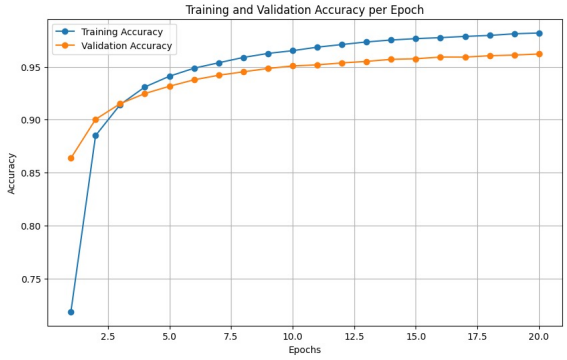

Figure 6: Using total of 1 layer

Observations

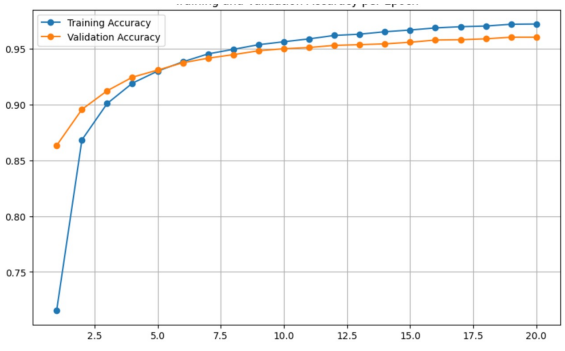

Figure 7: Using Batch Size = 10

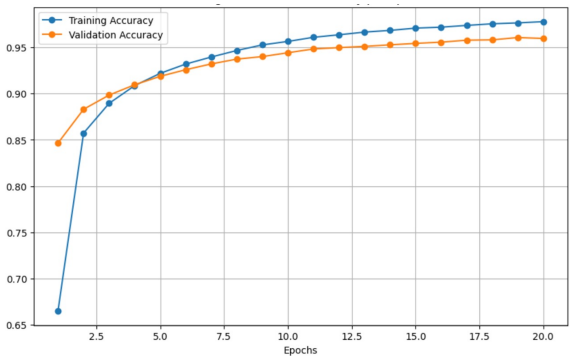

Figure 8: Changed Number of Neurons to (256,64)

1. As we can clearly see data balancing saved us from the hazard of overfitting on the imbalanced data.

2. Changing Batch size to 10, did not have any effect on the accuracy, rather it just took a lot longer to train, hence, the bigger batch sizes work better.

3. While all other changes did not bring in very considerable changes in accuracy, we observed that when judged on the metrics of overfitting, underfitting, efficiency and accuracy together, our model performed best, and we anticipate it to perform the best among this lot, on unseen data.

4. With our final model, trained on the 80-20 split data, we got a **training accuracy of 96.94%** and a **validation accuracy of 95.42%**

5 Discussion

1. TF-IDF Effectiveness: TF-IDF proved effective in capturing text features for multi-class classification tasks. Limiting the feature size to 4508 balanced performance and efficiency. This feature of the code gives the

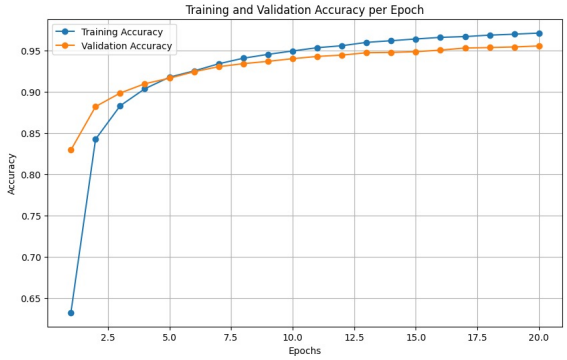

Figure 9: Final Model accuracy results

ingredients (features) weights according to their frequency of occurence. Qualitatively speaking, an ingredient which is more common, will have a lower weight.

2. Handling Imbalanced Data: SMOTE significantly improved class representation, ensuring fair learning across cuisines.

3. Model Simplicity and Performance: Despite being a simple feedforward neural network, the model achieved competitive accuracy, highlighting its suitability for this task.

Conclusions

Despite being a relatively straightforward architecture, the model proved to be suitable for the multi-class classification task, highlighting the importance of careful preprocessing and data balancing in text-based machine learning tasks.

Acknowledgements

We would like to thank Anant sir, for guiding us through the project, with valuable insights that helped us get the best out of it.

References

[1] Learn with Jay, Adam Optimizer

[2] Krish Naik, Deep Learing Videos

Appendix

This part contains the actual numbers of the tests and experiments conducted. It can be viewed on this link:
Appendix - Numbers