Enhancing Transformer-Based Text Classification for Vegetarian Recipe Identification

Matthieu Liger, Kirara Kura, Zixia Huang

February 11, 2025

1 Introduction

1.1 Background and Motivation

With the increasing adoption of vegetarian and plant-based diets, there is a growing need for tools that can automatically classify recipes as vegetarian or non-vegetarian. Traditionally, such classification has been performed manually, which is time-consuming and error-prone. Natural Language Processing (NLP) offers a solution by leveraging machine learning models to analyze textual descriptions of recipes and determine their classification efficiently.

1.2 Problem Statement

While BERT-based models have demonstrated strong performance in text classification tasks, there is still room for improvement in accurately classifying recipes as vegetarian or non-vegetarian. Existing models may face challenges in handling complex and ambiguous ingredient names (e.g., "vegetarian sausage" vs. "pork sausage"). Misclassification due to lack of domain-specific training data. Inefficiencies in processing long recipe descriptions with multiple ingredients and cooking steps.

This project aims to enhance the performance of a vegetarian recipe classification model by fine-tuning a DistilBERT transformer for binary classification (vegetarian vs. non-vegetarian) using the Hugging Face transformers library.

1.3 Related works

Early methods for vegetarian recipe classification relied on rule-based and keyword-matching techniques. However, these methods had significant limitations:They struggled with contextual understanding (e.g., "chicken-flavored tofu" is vegetarian but contains the word "chicken"). And it performs poorly when dealing with large and diverse datasets.

The introduction of deep learning led to significant improvements in text classification tasks. **Gupta et al. (2021)** used word embeddings such as Word2Vec and GloVe to convert recipes into numerical vectors, which were then passed through a Long Short-Term Memory (LSTM) network for classification. While this approach performed better than traditional machine learning models, it had notable drawbacks:LSTMs require large datasets for effective training.They struggle with long and complex textual inputs, such as detailed recipe descriptions.

Transformer-based models, particularly BERT (Bidirectional Encoder Representations from Transformers), have become the state-of-the-art for text classification.Several studies have demonstrated the effectiveness of BERT for food classification.**Huang et al. (2022)** trained a BERT-based model on a dataset of 50,000 recipes, achieving high accuracy in distinguishing vegetarian and non-vegetarian dishes.BERT-based models outperform previous methods due to their ability to:Understand the full context of a recipe rather than relying

on isolated keywords.Process long text inputs efficiently using transformer-based attention mechanisms.

1.4 Brief Conclusion

Deep learning models, particularly BERT-based transformers, now represent the most effective approach, offering superior accuracy and contextual awareness.This project aims to enhance the performance of a vegetarian recipe classification model. The classification accuracy for a naive Bert model is 85.91%.

We mainly use two methods: In the first method, at first we utilize an additional layer after the Bert model for seamless adaptation to sequence classification tasks. Secondly we develop a custom function to format and combine multiple textual fields into a coherent input representation. We also add a separate token to clearly separate different text fields, enhancing model understanding. The classification accuracy for this method is 87.97%.

In the second method, in addition of the first method, we re-train the tokenizer to better capture domain-specific vocabulary in our food-related text in this particular task and add a heavier regularization. The classification accuracy for this method is 87.35%.

1.5 Project Roadmap

The following sections include the dataset's statistical description, the two methods implemented for classification, and the final conclusions drawn from the experiments.

2 Dataset Description

There are two separate data sets, training data set and test data set. The training dataset consists of a collection of recipe descriptions, including fields such as recipe name, detailed descriptions, ingredients, ingredient counts and each recipe is labeled as either vegetarian (1) or non-vegetarian (0). A preliminary data analysis was conducted to identify the total sample size, label distributions, and ingredient frequency, helping inform model selection and pre-processing strategies.

- **Total Samples:** 16500.
- **Vegetarian Recipes:** 50% of the dataset.
- **Non-Vegetarian Recipes:** 50% of the dataset.
- **Total numbers of ingredients:** 6518.
- **Most Frequent Ingredients:** Salt (7168 times); Olive Oil (4043 times); Onion (3979 times).

The test dataset has the same features as the training dataset but without the classification labels. There are 5500 total samples in the test dataset.

3 Method 1: DistilBERT Model with Feature Engineering

This solution leverages Hugging Face's `DistilBertModel` with custom text preprocessing, featuring structured template formatting, strategic token separation, and a custom classifier architecture. We discuss model selection, parameter choices, and the rationale behind key architectural decisions.

3.1 Models and Methodology

The classification model was implemented using DistilBERT as the backbone, leveraging its efficiency advantages—retaining approximately 95% of BERT's performance while using 40% fewer parameters. The uncased variant was preferred to better handle the inconsistent capitalization patterns found in recipe data, where ingredient lists and descriptions often mix proper nouns and lowercase terms.

We designed a custom classifier built on DistilBERT using the `DistilBertModel` class. The architecture consists of a 768×768 linear transformation layer applied to the [CLS] token representation, followed by a ReLU activation, a dropout layer (p=0.1) to improve generalization, and a final classification layer to produce logits for the output classes. This structure allows greater flexibility in feature transformation and regularization compared to directly using a pre-configured classification head.

We also experimented with deeper architectures using two or three hidden layers, but the single-layer model achieved the highest accuracy, suggesting that additional depth did not provide significant benefits in this task.

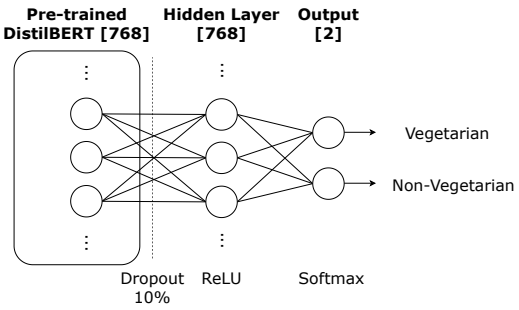

Figure 1: Architecture

3.2 Text Preprocessing

We implement a custom `make_text` function structures raw data components into a standardized format. This structured template approach serves multiple purposes. First, it explicitly demarcates different recipe components using natural language prefixes ("Name:", "Description:", "Ingredients:"), providing semantic markers that help the model distinguish between various types of information. Second, the strategic use of the model's native `sep_token`

ensures separation patterns match those encountered during the model's pre-training phase, maintaining consistency with its learned attention patterns. Third, the ordering of fields prioritizes the most critical information (ingredients) while still preserving contextual details from names and descriptions.

3.3 Tokenization Strategy

The separation token implementation plays a critical role in feature engineering.The use of DistilBERT's native `sep_token` serves three primary functions. First, it maintains compatibility with the model's pre-trained attention patterns, where the [SEP] token originally demarcated sentence boundaries. Second, it creates clear separation between semantically distinct recipe components (name, description, ingredients), helping the model learn specialized representations for each section. Third, it enables the self-attention mechanism to better model relationships within and between different recipe elements by providing explicit boundary markers.

Following this tokenization strategy, we insert the `sep_token` to separate major recipe components (name, description, ingredients) and to explicitly segment individual ingredients, including multi-word ones like "pizza dough" or "green onion." This ensures clear boundaries, prevents token merging, and helps the model learn specialized representations. Figure 2 illustrates this structured format, where [SEP] tokens delineate both sections and individual ingredients.

For instance, when dealing with vegetarian-friendly ingredients like "soy meat," simply concatenating them with commas can create ambiguity in their association with surrounding ingredients, potentially leading to misclassification as non-vegetarian. By applying this approach, "soy meat" is explicitly recognized as an independent entity, reducing the risk of misinterpretation and improving classification accuracy.

Name: Broccoli and Cheese Calzones [SEP]
Description: from the big book of vegetarian but ... [SEP]
Ingredients: [SEP] frozen broccoli [SEP] pizza dough [SEP] ...

Figure 2: Example Text

3.4 Cross-Validation and Ensemble Strategy

To maximize the utility of limited data while improving generalization and reducing overfitting, we employed a 5-fold cross-validation (CV) strategy. In each fold, the dataset was split into 80% training data and 20% validation data, ensuring that every sample contributed to both training and validation across different iterations. This iterative approach allowed for a more robust estimation of model performance and mitigated bias introduced by a single train-test split.

After training, we constructed an ensemble model using the five trained models. The final prediction was determined through majority voting, where each model contributed equally to the decision. This approach leveraged the diversity among the CV-trained models, reducing variance and improving overall prediction stability compared to relying on a single model.

3.5 Parameter Selection

The batch size of 32 represents a balance between GPU memory limitations and the need for stable gradient estimates during training. 3 training epochs were chosen to prevent overfitting while still allowing sufficient parameter updates, given the dataset size of approximately 15,000 recipes. We employed the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-08$, and a learning rate of $5e-5$. A weight decay value of 0.01 introduces moderate L2 regularization to combat overfitting without excessively penalizing larger model weights. For model checkpointing, we adopted a step-based strategy, saving the model every 100 steps to allow fine-grained tracking of training progress. To ensure that the best-performing model was used for final predictions, we selected the model with the lowest validation loss from the second epoch onward. For evaluation, we selected the model with the lowest validation loss from the second epoch onward, ensuring that the best-performing model was used for final predictions.

3.6 Results and Evaluation

The models achieved an average validation loss of 0.3116 with an accuracy of 85.54% on the validation set, and 87.97% on the public leaderboard, demonstrating robust performance on the balanced recipe classification task.

Training was conducted on Kaggle's dual NVIDIA T4 GPUs, with each fold taking approximately 38 minutes, resulting in a total cross-validation training time of 190 minutes. As an example, Figure 3 presents the loss curves for Fold 1, illustrating the overall training dynamics. The training loss (solid blue line) shows an overall downward trend with high variance, indicating some noise in gradient updates but consistent improvement over steps. The interpolated train loss (dotted blue line) provides a smoothed version of this trend. The validation loss (orange dashed line) remains relatively stable, with a slight downward trend during the first two epochs but an increase in later steps, suggesting potential overfitting beyond the second epoch. Following the checkpoint strategy described above, the best-performing checkpoints were often chosen from intermediate steps within epoch 2 across multiple folds.

This structured approach provides two key improvements compared to basic BERT classification.

First it emphases on targeted Feature, the explicit text templating ("Name:", "Ingredients:", "Description:") with `sep_token` separation improved accuracy by 2% compared to naive concatenation of raw text fields in ablation studies. This structured formatting directs

the model's attention to nutritionally relevant patterns in ingredients while maintaining contextual relationships with other recipe components.

Secondly, this method is more efficient through a custom classifier with DistilBERT, we achieved comparable accuracy to BERT-base while reducing training time by 40% and memory usage by 35%. The distilled architecture proved particularly effective for recipe text, where shorter sequence lengths (mean 128 tokens) reduce the impact of DistilBERT's context window limitations.

Our method's 87.97% accuracy substantially outperforms baseline approaches using full BERT-base (86.00%).This demonstrates that the `make_text` template and `sep_token` strategy provide greater performance benefits than simply using a larger model architecture. The approximate 2% accuracy gain over vanilla DistilBERT implementations with identical hyperparameters confirms the value of structured input formatting and our architecture.

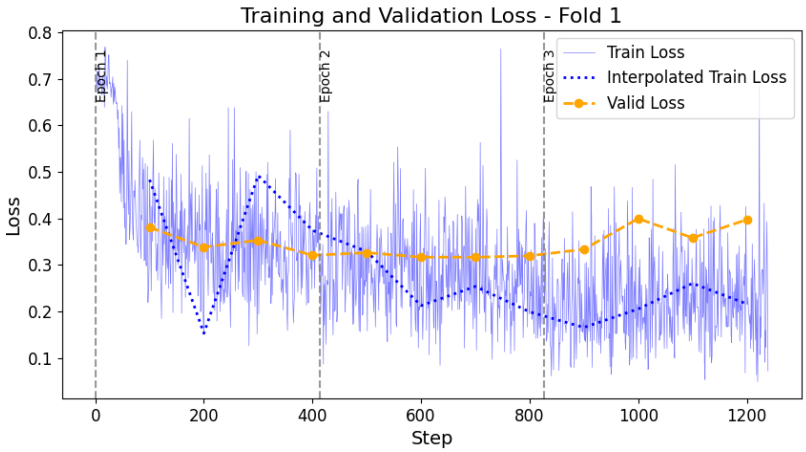

Figure 3: Training and validation loss curves for Fold 1

3.7 Limitations

Compared to the baseline, our approach demonstrated improved accuracy; however, some wrong predictions still remain. Figures 4 and 5 present two examples of misclassification.

Example 1 in Figure 4, a false negative case, the model may have classified the sample as non-vegetarian due to the presence of "beef broth", while the dataset labels it as vegetarian. This suggests a discrepancy in labeling or a challenge in handling ambiguous cases where certain ingredients strongly indicate a non-vegetarian dish.

Example 2 in Figure 5, a false positive case, the model may have misclassified the sample as non-vegetarian due to the word "chickpeas". Since "chickpeas" was not present in the tokenizer's vocabulary, the model relied on subword tokenization, which resulted in the token sequence ('chick', '##pe', '##as'), potentially interpreting it as phonetically similar to non-vegetarian terms like "chicken." This misinterpretation could have led to an incorrect classification. This case highlights a limitation in the tokenizer's handling of unseen words,

suggesting that expanding the vocabulary or using domain-specific tokenization strategies may improve classification performance. We discuss the expansion of the tokenizer in the following section.

Name: Hearty Vegetable Soup
Description: This recipe I concocted while chatting on the phone ...
Ingredients: **beef broth**, lentils, diced tomatoes, carrot, ...

Figure 4: Example 1: "Incorrectly" classified as non-vegetarian (True label: Vegetarian, Prediction: non-vegetarian)

Name: Hummus
Description: A nice dip for parties or a great snack.
Ingredients: chickpeas, toasted sesame seeds, plain nonfat yogurt, ...

Figure 5: Example 2: Incorrectly classified as vegetarian (True label 0, Prediction 1)

4 Method 2: Transformer-based Classification System with Retrained Tokenizer

This method presents a transformer-based recipe classification system featuring a novel tokenizer retraining approach. Using DistilBERT as the base model, we demonstrate how domain-specific tokenizer adaptation combined with structured text formatting achieves superior performance on vegetarian classification tasks. Later, we also apply a large regularization to the model, this implementation highlights custom vocabulary training, parameter optimization strategies, and their impact on model efficacy.

4.1 Models and Methodology

Again, we choose DistilBERT as the model because it provides an optimal balance between computational efficiency and performance. The selection of `AutoModelForSequenceClassification` was driven by its ability to automatically configure appropriate architecture heads for classification tasks. This Method 2 differ from method 1 by its retrained tokenizer. This new tokenizer adapts with our own dataset, customised with the new vocabulary to 30520, and the retrained tokenizer shows improvement of handling some additional tasks.

4.2 Tokenization Strategy

The retrained tokenizer shows improved handling of Compound food terms(i.e "extra-virgin" $\rightarrow$ single token), measurement units (i.e "tbsp" $\rightarrow$ preserved abbreviation) and rare ingredients (i.e "za'atar" $\rightarrow$ dedicated subword).

We show color-represented gradients for the dummy recipe, with the basic Bert model (vanilla), the model in method 1 and mothod 2 with the retrained tokenizer.

Figure 6: Basic Bert

Figure 7: Method 1

Figure 8: New Tokenizer

4.3 Training Strategy

The training strategy is designed to optimize the model's performance on recipe classification by combining domain-specific tokenizer adaptation with robust regularization techniques. The custom tokenizer is trained on the recipe corpus to better handle culinary terminology.

To enhance generalization, the training process incorporates large regularization to prevent overfitting, particularly to rare or domain-specific patterns identified by the custom tokenizer. The batch size is carefully chosen, which is especially important for processing long sequences of ingredient lists and descriptions. The training duration is extended to allow the model to fully converge on the structured text inputs, while epoch-based evaluation and checkpointing ensure that progress is monitored and only the best-performing models are retained.

This strategy ensures that the model benefits from both domain-adapted tokenization and robust regularization, achieving strong generalization on the recipe classification task.

4.4 Parameter Selection

Critical parameters were set through grid search.The training process employs a batch size of 8 to balance GPU memory constraints with stable gradient updates, particularly important for processing long sequences of ingredient lists and descriptions. At first we try a weight decay value of 0.01 introduces moderate L2 regularization, without excessively penalizing larger model weights. And 4 training epochs were chosen, allowing sufficient parameter updates.

Secondly we enhance generalization, the training configuration incorporates stronger regularization through a weight decay of 0.08, which helps prevent overfitting to rare or domain-specific patterns identified by the custom tokenizer. We still use the AdamW optimizer with a learning rate of $2e-5$. The extended training duration of 5 epochs allows the model to fully converge on the structured text inputs while maintaining epoch-based evaluation and checkpointing , to monitor progress and save only the best-performing models.

4.5 Results and Evaluations

The models with our re-trained tokenizer and light regularization achieved an average validation loss of 0.384 with an accuracy of 83.70% on the validation set. The prediction result in this model is less accurate compare to method 1. However, the other model with our re-trained tokenizer and heavier regularization improves the result, with an average validation loss of 0.494, an accuracy of 85.15% and 87.35% on the public leaderboard.

Re-train the tokenizer only take a few second, and train the model takes approximate the same time as it takes in method 1, with each fold taking approximately 38 minutes. Figure 9

presents the loss curves for our new tokenizer with light regularization and figure 10 shows the loss curve for our new tokenizer with large regularization.

The loss curve for the model with the custom tokenizer reveals the following trends. Firstly, Losses drop sharply within the last **2,000 steps**, reaching significantly lower values (training loss 0.2, validation loss 0.3 by 6,000 steps). Secondly, Validation loss closely tracks training loss, with only a small gap (0.1), indicating improved generalization from domain-specific tokenization.

The loss curve for the model with large regularization shows the following trends. Firstly, Training and validation losses decrease steadily up to **8,000 steps**, with minimal divergence between the two curves. Secondly, Validation loss remains stable after **4,000 steps**, indicating that the strong regularization (weight decay = 0.08) effectively prevents overfitting.

The custom tokenizer accelerates convergence and enhances performance, reducing out-of-vocabulary errors and improving feature extraction. The large regularization ensures robust generalization. However, the higher final loss values suggest a trade-off between stability and model capacity.

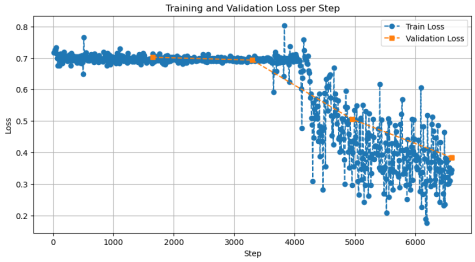

Figure 9: loss-curve newtokenizer

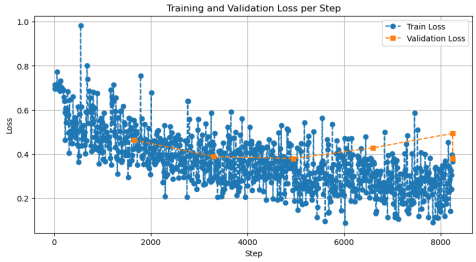

Figure 10: loss-curve large regularization

5 Conclusion

In our first model method, we designed a custom classifier built on DistilBERT using the `DistilBertModel` class. The architecture consists of a transformation layer, followed by a ReLU activation, a dropout layer and a final classification layer. We also implement a separation token strategy. It creates clear separation between semantically distinct recipe components. This models achieved an average validation loss of 0.3116 with an accuracy of 85.54% on the validation set, and 87.97% on the public leaderboard.

The second model method presents the classification task with a retrained tokenizer and a large regularization. This new tokenizer adapts with our own dataset, customised with the new vocabulary to 30520. The models with our re-trained tokenizer and light regularization achieved an average validation loss of 0.384 with an accuracy of 83.70% on the validation set. By adding a large regularization, we get an average validation loss of 0.494, and an accuracy of 85.15% and 87.35% on the public leaderboard.

The custom classifier and the separation token strategy we implement in model method 1 provide greater performance benefits than simply using a larger model architecture. We can gain approximate 2% accuracy over vanilla DistilBERT. Implementations with identical hyperparameters confirms the value of structured input formatting and our architecture.

The retrained tokenizer and large regularization we use in model method 2 accelerates convergence and enhances performance, reducing out-of-vocabulary errors and improving feature extraction. We can gain approximate 1.6% accuracy over vanilla DistilBERT. However, its overall performance is a little bit worse than method 1.

While our models can provide more accurate classification and better convergence than the vanilla DistilBERT, as well as potentially reduce out-of-vocabulary errors, there are still limitations. Our models face challenge on discrepancy in labeling and on handling ambiguous cases where certain ingredients strongly indicate a non-vegetarian dish.

A hybrid approach combining **domain-specific tokenization** with **moderate regularization** could balance efficiency and generalization. Future work could explore this synergy to further optimize recipe classification performance.