Homework 6: Reclassifying Recipes

Due April 1st at 10pm

Part 1: Expanding Our Network

Last week we built a regression classifier for classifying recipes into courses. We hand-selected features to feed into our model, and achieved fairly good performance. But it's annoying to have to build the features ourselves!

In this part of the assignment, you'll build a better classifier by using a neural network to extract features from the titles of the recipes.

As a human, you can probably guess the category of most of these recipes from their titles alone. But up until this point, we have not been making good use of our recipe titles. Maybe you hand-crafted features to capture some of the information in the titles last week, but you had to guess which words were important.

This week we will outsource the work of feature selection to another, more powerful network. We will take our recipe titles and turn them into feature vectors by using a powerful language model called DistilBERT, which, given a word, outputs a 768-dimension **embedding** (feature representation) of the word.

I have given you a new file to work with called **bert_model.py**. You can run this file in two modes: "train", to train your model, or "interactive", as described in Task 3. To run in "train" mode, run:

```
python3 bert_model.py train
```

A Note on Training Time and Model Checkpointing

We will work with large input representations this week. This will make the code much slower. **You should expect your models to take around 30 minutes to run; perhaps an hour, depending on your computer.** (If you run into difficulty, you can run your model on a lab machine. Let me know if you need help with this.)

To make things easier, I have set up the starter code to **checkpoint** your model: it saves progress checkpoints as it goes, so that if you stop and start training without losing your progress. **To retrain from scratch, you must delete these checkpoint files (found in the ckpt directory).**

Task 1: Incorporating Word Embeddings

The model that I have given you extends our regression model with DistilBERT embeddings. The **prep_bert_data** function takes the recipe titles and **tokenizes** them: translates them into lists of indices in the DistilBERT vocabulary. This involves truncating and padding them to the same length.

For instance, the tokenizer function maps the title "Tuscan-Style Potato Soup" to the following indices:

0:	<s>
565:	T
687:	us
7424:	can
12:	-
31774:	Style
38855:	Potato
31905:	Soup
2:	</s>
1:	<pad>
1:	<pad>
1:	<pad>

(Notice that DistilBERT uses subword tokens in its vocabulary, so some words get split into multiple indices.)

The model that I have given you is still a regression model, but it takes these much larger, automatically-extracted vectors of features.

Paste in your regression model code into the make_model function. The input to your layers is now named last_hidden_state rather than inputs.

Because the embedding layer adds a dimension, we will need to **flatten** our input back to its original 2D shape before passing it to the Dense layer for prediction.

Run the model and observe how well it does. Record its loss and accuracy, and make some observations on how well it does across recipe classes.

(You will see a number of warnings when you run your code. This is because we are taking layers from a pretrained model, but we are freezing the weights. You can ignore these warnings.)

Task 2: Layering Up

We're using embeddings now, but our model is still a regression model. Now it's time to add **more parameters** to our model and make it into a neural network.

Let's add **hidden layers** to our network. You should add these **before** your Flatten layer, and you should **not** give them an activation function.

The first layer should take in the output from DistilBERT (last_hidden_states). Each subsequent layer should take as input the output of the previous layer.

Add 2 more dense layers to your network. Retrain the model and observe how well it does. Record your model loss and accuracy, and make some observations on how well it does across recipe classes. Did the hidden layers improve the model?

Task 3: Adversarial Input and Model Interpretability

When we add hidden layers, there's a trade-off in interpretability. Before, we could measure how useful each feature was to our model by looking at its weight. Now, our best tool for understanding how the model is making its decisions is to manipulate our input. This is called a **probe task**.

You can run your program in "interactive" mode to interactively probe your model with new recipe titles. To run in "interactive" mode, run:

```
python3 bert_model.py interactive
```

Play around with this, and try to find some **minimal pairs**: pairs of recipe titles that are almost the same, but are classified differently by the model.

Construct 3 minimal pairs that illustrate aspects of how the model makes its decisions. For each example, make a hypothesis about which features in the input the model is paying attention to, and discuss how your minimal pair supports this hypothesis. At least one of your examples should illustrate a **model error**: a case where a human can guess the correct category, but the model cannot.

Part 4: Reading Response

Question 1 (10 points)

In Chapter 7, Shane discusses three problems that plague machine learning classification: class imbalance, overfitting, and biases in the data. For each of these issues, discuss a way that we could test whether our recipe classifier suffers from it.