Homework 6: Recipes for Recipe Classification

Due Oct. 30th at 10pm

Part 1: Recipe Regression

In this assignment, you will build classifiers for predicting the course category of a recipe. I have given you a dataset of recipes scraped from Yummly by Min et al. (2016).

We are going to try to predict which course a receipt belongs in. In the original dataset, there are 12 course categories. To make this task easier, we will focus on predicting only the first 6 categories:

0: Main Dishes
1: Desserts
2: Beverages
3: Soups
4: Salads
5: Condiments and Sauces

I have filtered the dataset to include only recipes in these 6 categories. I have also split the data into training and test files: **yummly_data_top6_train.tsvand yummly_data_top6_test.tsv**. Each recipe has the following features:

- file_id: the JSON file id, in case you want to get other features from the original recipe
- name: the recipe name
- rating: the average rating the recipe received
- servings: the number of servings the recipe makes
- sugar: how much sugar is in the recipe
- fat: how much fat is in the recipe
- protein: how much protein is in the recipe
- cuisine: a string consisting of all cuisine tags associated with the recipe
- cuisine_number_1: a numerical code for the recipe's first cuisine tag
- cuisine_number_2: a numerical code for the recipe's second cuisine tag if there is one, otherwise None
- cuisine_number_3: a numerical code for the recipe's third cuisine tag if there is one, otherwise None
- id: the recipe's original id
- time: how long the recipe takes to make
- category: the recipe's course category as a string
- category_number: a numerical code for the course category

Your goal is to write a multinomial regression classifier that uses a combination of a recipe's features to predict its category.

Task 1: Setting Up Your Regression Classifier (10 points)

To build your regression classifier, you should use the Keras Python library like we did in our class cat/dog classifier example. You may find it helpful to refer back to this file.

You may also use this Keras tutorial: `https://www.tensorflow.org/tutorials/load_data/csv`. The first part will show how to load a CSV dataset into a regression model.

I have given you a file called **yum_regression_starter.py** which contains starter code for your regression model. It already contains code to reads a file in the format of yummly_data_top6_train.tsvand processes it into Pandas dataframes.

Your task is to fill in the **make_model** function to create a multinomial regression model. Your model should be very similar to the one we built in class for cat/dog classification.

Test out your model by training it with just one or two features as input.

You may need to install some packages in order to get this script to work. I have tested this with Python 3.8, Tensorflow 2.7 and Keras 2.7; it may not work for other versions. **Let me know ASAP if you have issues at this step!**

You can specify which features to include in your training data by listing column names in selected_features:

```
selected_features = ["rating", "time","servings"]
```

Important: You should **NOT** change how train_labels or test_labels are defined.

Record which features you tried, and how well your model did. You should record its loss and accuracy on both the train and validation sets.

Task 2: Understanding the Regression Equation (10 points)

Here is a set of model weights and biases:

Feature	0(Mains)	1(Desserts)	2(Beverages)	3(Soups)	4(Salads)	5(Sauces)
rating	0.035	-0.23	-0.077	-0.06	-0.10	0.04
time	0.3	0.3	0.3	0.3	0.3	0.3
servings	0.177	0.25	-0.023	0.13	0.18	-0.019
bias	0.45	0.094	-0.15	-0.13	0.08	-0.42

A. Given these model parameters, calculate the pre-softmax score for Recipe 24841, Roasted Eggplant Cannelloni, in Category 0 (Mains).

B. Given these model parameters, calculate the post-softmax scores for Recipe 24841, Roasted Eggplant Cannelloni, in Category 0 (Mains).

Task 3: Adding More Features (5 points)

When your model is working, try it with all the numerical features that I have given you: rating, time, servings, sugar, fat, protein, cuisine_number_1, cuisine_number_2, cuisine_number_3.

Your model should achieve about 64% accuracy on the training set and about 67% accuracy on the validation set.

Record your model's loss and accuracy on both the train and validation sets.

Part 2: Feature Selection

Now that our basic model is working, it is time to think about our features. Which of these features are the most useful? In this part, you will improve your classifier by picking better features.

Save your work for this part of the homework by making a copy of your regression file. Call it **yum_regression_improved.py**.

Task 4: Interpreting Model Weights (10 points)

I have given you two functions to help you think about which features are most useful for the model. One is the **stats_by_label function**, which prints out descriptive statistics for each feature by category label. This can help you get an intuitive sense of which features you expect to be best at distinguishing among categories.

The second is the **print_coefficients** function. This function prints out a table of regression weights for a given model. The columns are categories; the rows are features.

A negative weight for a particular feature/category pair means that as the value of the feature increases, the likelihood of the category decreases. A positive weight indicates a correlation (positive linear relationship) between the feature and the category.

Identify 3 trends you see in the regression weights. For each observed trend, say whether you think the descriptive statistics support this weighting, go against it, or provide no evidence.

Task 5: Evaluation Metrics (10 points)

So far we have looked only at overall model performance. But our recipe collection suffers from **class imbalance**: there are way more recipes in the main course category than in the others.

Write a function called print_performance_by_class that prints a summary of accuracy by category. Your function should take two parameters: a set of labels (such as test_labels) and a set of predictions derived from calling model.predict() on the corresponding dataset.

Your function should print a summary like the one below. Note that I am rounding to 3 places to make the output more readable.

```
Accuracy by Category:
Category 0 : 0.913
```

Category 1 : 0.0
Category 2 : 0.0
Category 3 : 0.161
Category 4 : 0.0
Category 5 : 0.0

Task 6: Feature Selection (10 points)

You may find that your model does well only on certain classes, as in the example output that I have shown above. Your next task is to do feature engineering to improve your model's performance.

Design some new features for your model. For full credit, your model must achieve 75% accuracy on the test set.

There are several different approaches that you can take. Here are some suggestions:

- The recipe title is likely to be informative for this task. You could write functions to assign features based on strings that appear in recipe titles.
- The cuisine categories that I have given you are not very helpful, because the model can't assign different weights to different cuisines. You could add additional features to make better use of this information.

Your new features **should NOT make use of the category or category_number features**. This would let your model cheat by peeking at the answers!

Here is an example of how you might add a new feature to the data (in this case, whether the recipe title contains the word "strawberry":

```
def add_strawberry_column(df):
    names = df["name"]
    has_strawberry = [ ]
    for name in names:
        if "strawberry" in name.lower():
            has_strawberry.append(1)
        else:
            has_strawberry.append(0)
    df["strawberry"] = has_strawberry
    return df
train_data = add_strawberry_column(train_data)
test_data = add_strawberry_column(test_data)
```

As you add features, you can experiment with training the model for more epochs. If you train it too long, however, you will **overfit** to the training data. You should make sure that the validation loss continues to decrease— if it does not decrease for several epochs, you are training too long.

Record which features you added, how many epochs you trained for, and your model's best performance. As before, record your loss and accuracy on both the train and validation sets. **You should also write a discussion of how these features impacted accuracy by recipe class.**

Part 3: Reading Response

Question 1 (10 points)

Chapter 8 discusses *adversarial attacks* on AI: input designed to cause AI to malfunction. As Shane describes, these input may be undetectable to humans: humans see skiers in the picture on page 201, while the vision model is baffled.

Similarly, self-driving cars make errors that are very different from humans, such as driving straight into a white semi truck because it is too light to register.[1] However, human drivers are also notoriously error-prone. Do you think this mismatch between human and AI behavior is problematic even if AI outperforms humans on a task?

[1]`https://arstechnica.com/cars/2019/05/feds-autopilot-was-active-during-deadly-march-tesl`