Food.com Recipes: Final Report

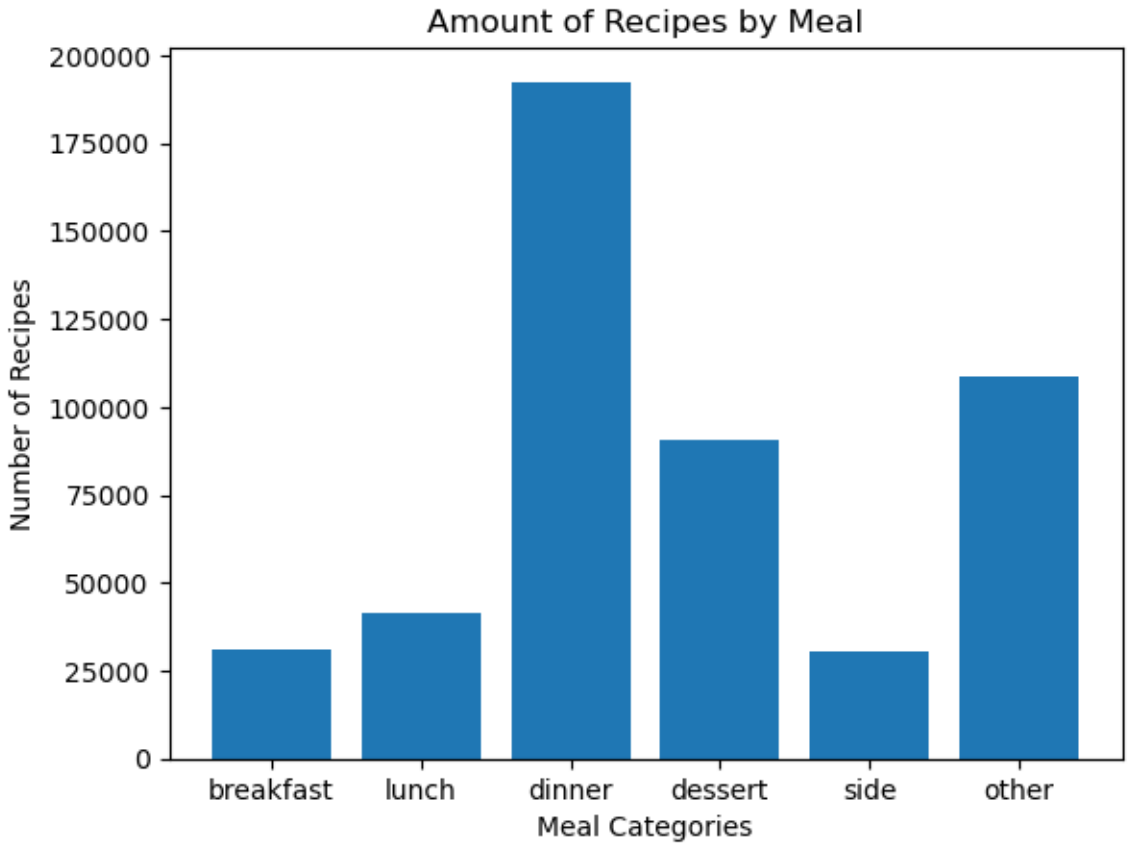
Spencer Wilson

Introduction

Food.com is one of the largest recipe aggregators on the web. They have over 500 thousand recipes submitted by users over 10 years. I wanted to see if I could use the ingredients list of a recipe to predict what meal the recipe was for (breakfast, lunch, dinner, dessert, or side). Since about a fifth of the recipes aren't labeled, it would be nice to be able to predict a course for the remaining recipes.

EDA

My data is completely text-based, so there aren't really any summary statistics to speak of. However, here are the counts of recipes for different courses:



As you can see, the most common category was dinner, but there were a lot of recipes that weren't in any of these categories. From an inspection of these recipes, they tended to be only tagged things like "low-sodium" and "30 minutes or less", which are not useful for the purpose of knowing which meal they are. These recipes were dropped from the dataset before proceeding with analysis.

Methods

Feature Engineering

Feature engineering was an important part of the process, since the ingredient lists were all text data. I used a function in python called “literal_eval()” that evaluates a string as though it were python code, which was useful because I needed to convert string literal lists of ingredients into an actual list in python. Once I had the ingredients, I turned them into “sentences” by replacing the spaces in them with underscores and joining them all together with spaces in between. This way, I could use a `CountVectorizer` from sklearn to vectorize the ingredient lists. Once the ingredient lists were all vectorized, it was ready to be used as input to any of a variety of machine learning algorithms.

Models

Model Name	**Description**	**Hyperparameters**	**Performance**
Ridge Classification	Linear method that is suited for multi-level classification	Tol: a stopping criteria, set at 1e-2, solver: sparse_cg is well suited for larger scale data.	Test accuracy: .733 Very fast to fit, good accuracy. Tends to confuse lunch and dinner.
Naive Bayes	Bayes classification; is a fair candidate for text analysis	A wide range of alpha values were tested with a grid search.	Test accuracy: .701 Very bad at classifying breakfast.
Random Forest	Random forest classification. Usually doesn’t perform well for text analysis.	Criterion: measures the quality of a split max_depth: depth of tree n_estimators: number of trees in the forest. Random Search tuned these parameters.	Test accuracy: .500 As expected, the random forest did not perform very well.
LinearSVC	A support vector classifier; another linear method and should perform well.	C: regularization parameter. Value of 0.1 was determined to work the best.	Test accuracy: .736 Similar issue to the Ridge Classifier.
ANN	A global average pooling layer is used to condense the sparse vectorized input	Different sizes and numbers of dense layers on top of the network were used, in addition to tuning the learning rate.	Test accuracy: .715 This one was very good at classifying dinner, but very bad at any of the other categories, calling most of them all dinner.

Model Selection

Tree based models don’t usually perform well for text classification like this, according to what I’ve read. My results with the random forest were unsurprising. Linear models were stronger choices, and they both seemed to get me about 70% accuracy.

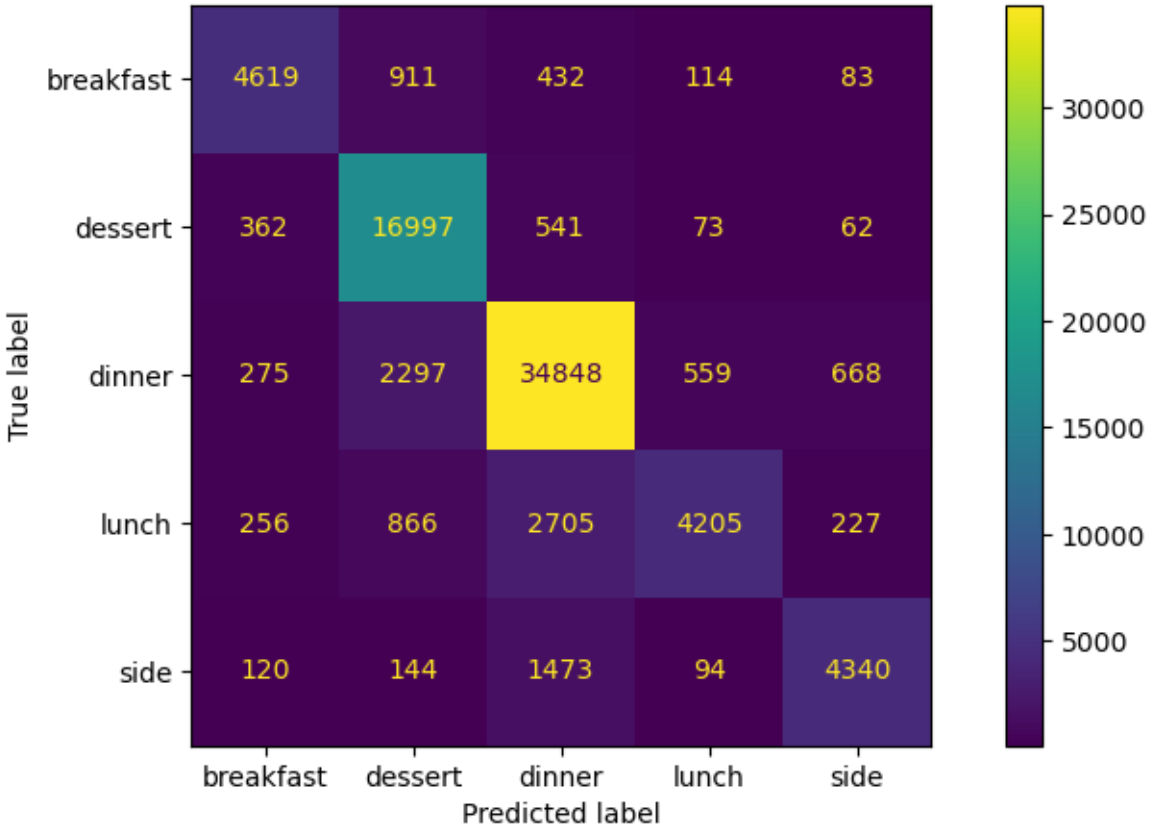
The ANN was difficult to fit because of the high class imbalance; it favored just marking nearly everything as dinner. All of the algorithms that I tried converged fairly nicely.

In the end, the reason why I preferred the LinearSVC is because it gave the best overall results, especially when examining the confusion matrices. It was the only classifier to achieve a majority of correct predictions across every level. The ANN was promising, but it took a long time to fit and I couldn't completely overcome the class imbalance issues.

The LinearSVC was the best

The SVM was the model that was the most accurate and the easiest to tune. With default parameters, it beat every other classifier I had made. It had a great confusion matrix. It's only drawback was the time it takes to fit; it was the slowest besides the ANN of the bunch.

The only hyperparameter to tune is C, which controls the strength of the regularization. In my Randomized search, I also tried a few parameters of the vectorization, just in case the n-grams or the number of vectors could optimize the model. As for performance metrics, it had a test accuracy of .736 and did a good job of minimizing type I and type II error. See the confusion matrix for the test data:

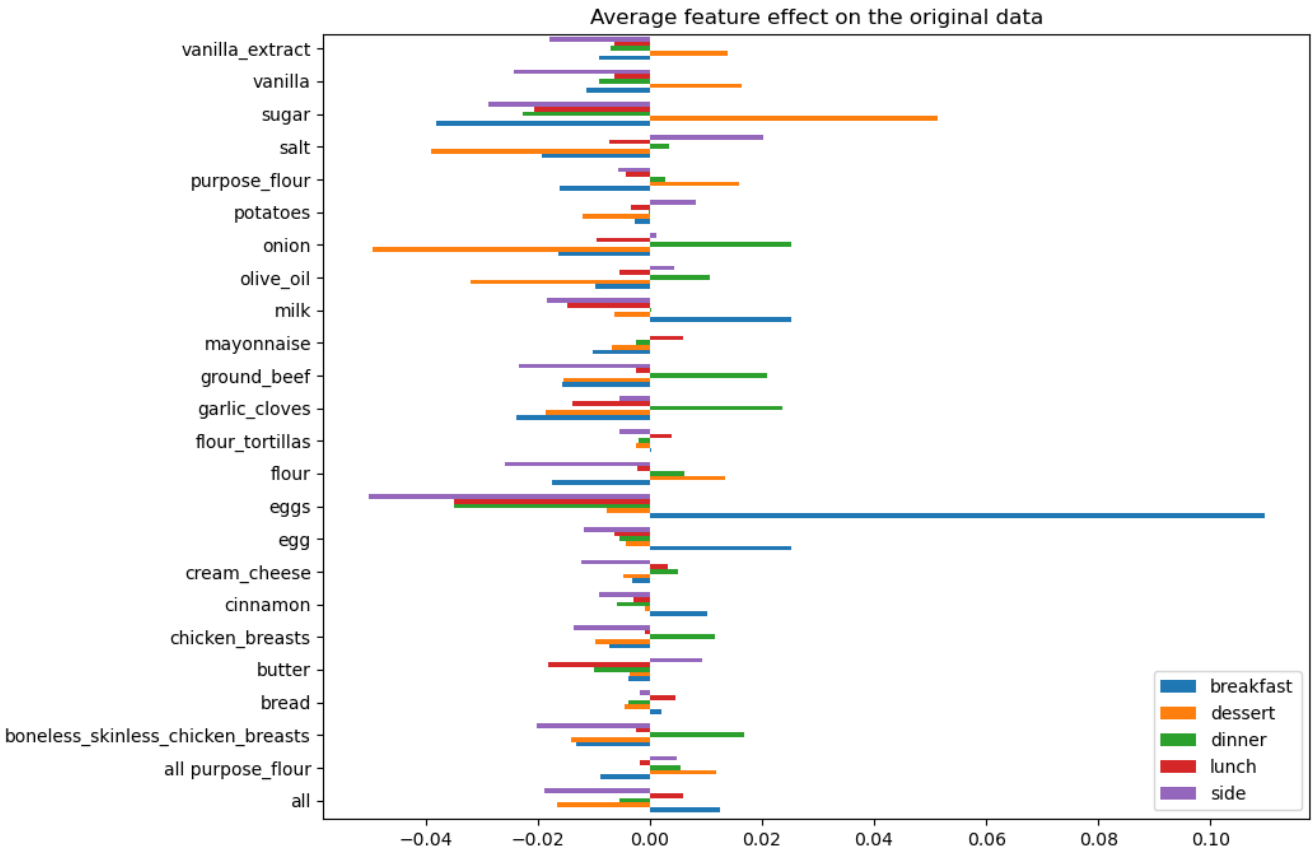


The diagonal is very clear on this matrix and in particular the reduced confusion between breakfast and dessert is impressive. A possible reason for the confusion between dinner and lunch is that these meals are so similar. Given the user-defined nature of this data, it is impressive to classify the meals so well.

I also looked at the words that had the greatest feature importance in the model:

Breakfast	Lunch	Dinner	Dessert	Side
eggs	mayonnaise	onion	sugar	salt
milk	bread	garlic_cloves	vanilla	butter
egg	flour_tortillas	ground_beef	all_purpose_flour	potatoes
cinnamon	cream_cheese	chicken_breasts	vanilla_extract	olive_oil

This seems accurate to my gut! We can also look at the feature effect for all of the top words:



Of course, the model couldn't get everything correct. For example, it classified a caramel corn as lunch when it should have been a dessert, and a kale smoothie recipe got classified as dinner instead of breakfast. There were also entries in the dataset that seem to have been labelled incorrectly. A pudding

pie was labelled as a dinner food, but the algorithm thought it was dessert. I think I trust the algorithm for that one.

Conclusion

In conclusion, the LinearSVC was the most effective model that I used to predict the mealtimes of recipes from Food.com. It was able to overcome the class imbalance and provide workable predictions. Unfortunately, I couldn't get any of my models to perform better than about 75% accuracy, but this is likely due in part to the high amount of noise in the data and the inconsistent feature names.

In the future, it would be cool to use other features in the dataset like the instructions or the descriptions to help the classification, or for other tasks like intelligent searching or as refinement for an LLM to generate new recipes. I bet that with some time, a transfer learning model like Bert might perform better on the ingredients classification task, although it might struggle somewhat because it isn't trained on lists of ingredients like I'm using. I feel like I've only been able to scratch the surface of the potential here, and it would be really cool to see what else could be done.