Dataset

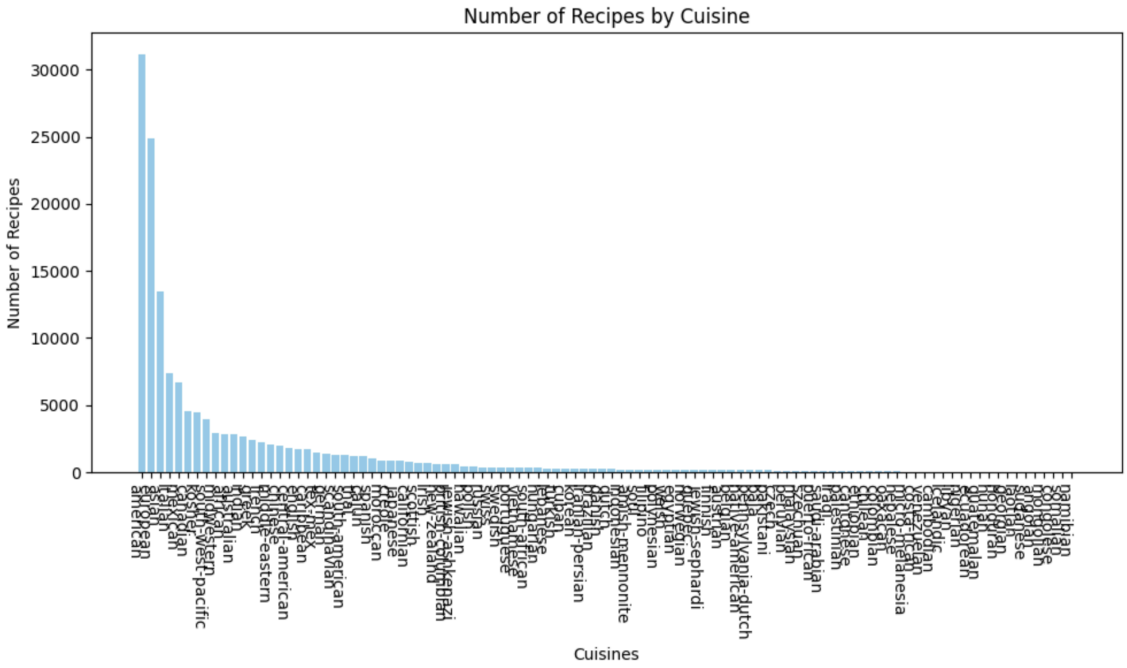
The dataset consists of 231,637 recipes with the following key attributes:

- **Tags**: 552 unique tags. Each recipe is tagged with a list of many descriptive tags, covering many different descriptive angles: flavor ("tasty", "savory"), difficulty ("easy"), cuisine ("asian"), cook time ("60-minutes"), etc. All cuisines are manually removed from the list of tags.
- **Cuisines**: 102 manually identified cuisines. Due to the absence of NLP tools for automatic cuisine tagging, our group manually selected these from the tag list.
    - Recipes can belong to multiple cuisines, with tags ranging from broad (e.g., *Asian*) to specific (e.g., *micro-Melanesian*).
    - 63.8% of recipes are tagged with at least one cuisine, amounting to 147,921 recipes.
- **Ingredients**: 11,674 unique ingredient strings. While there are many unique strings for ingredients, many ingredients are similar and can be grouped into further categories (e.g. "button mushroom", "button mushrooms", "white mushroom", and "mushroom" are all counted as separate ingredients). To avoid vector sparsity from handling thousands of unique ingredients with only tens of ingredients per recipe, we use Word2Vec to convert these vectors into encodings. This technique is used in both our RF classifier and FM + TF-IDF approach.
- **Description, Name, Recipe Steps**: Strings, or a list of strings, describing the recipe.
- **Time**: A floating point number ranging from roughly 0-200 (minutes) which represents how long the recipe takes to cook
- **Nutrition**: A list of 7 floats covering the nutritional value of the recipe (calories, protein, carbs, etc).

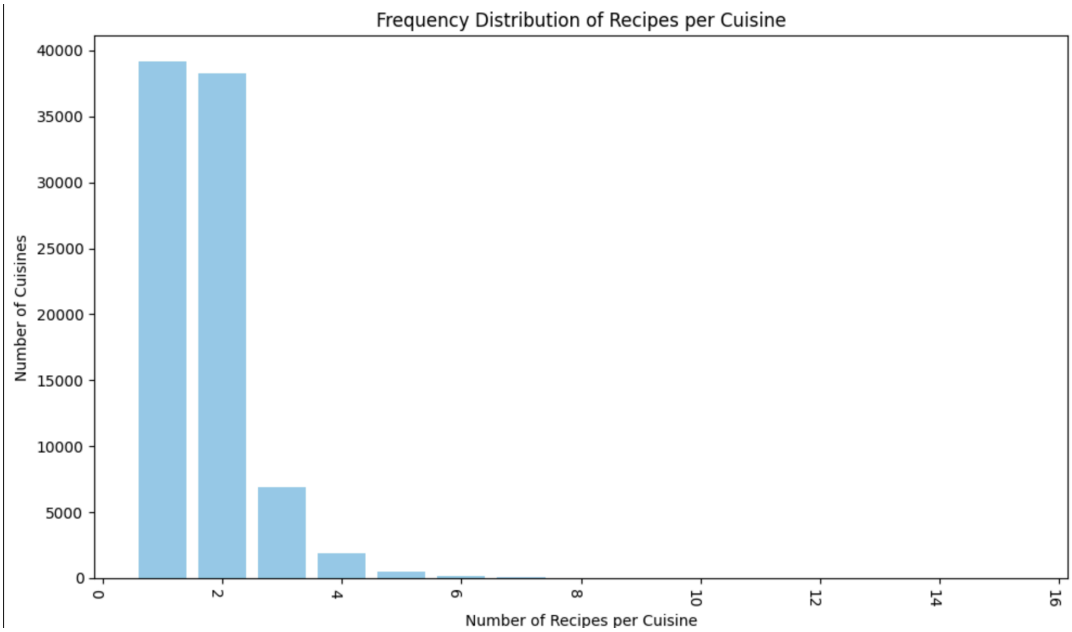
Statistical Insights

- **Cuisine Distribution**:
    - Mean recipes per cuisine: 1450.2.
    - Median: 224.
    - Maximum: 31,179 (*American* cuisine).
    - Minimum: 6 (*Namibian* cuisine).

- Standard deviation: 4195.3.



- **Recipe Distribution**:
  - Mean cuisines per recipe: 1.7.
  - Median: 2.
  - Maximum: 15 (*oven baked sweet plantains*).
  - Minimum: 1.
  - Standard deviation: 0.8.
  - 55.0% of recipes that have cuisine tags have more than one cuisine tag



Predictive Task

Cuisine Classification Task

Our primary goal is to classify the cuisine of a given recipe using features like ingredients, tags, and descriptions. Additionally, we aim to explore relationships between cuisines by leveraging features such as ingredient similarity.

Challenges and Considerations

The dataset presents significant class imbalance, with highly skewed representation across cuisines. For instance, "American" cuisine accounts for over 31,000 recipes, while "Namibian" has only six. This imbalance poses challenges for training a classification model, as underrepresented cuisines may be difficult to predict accurately. Additionally, the multi-label nature of the task (recipes tagged with multiple cuisines) complicates modeling and evaluation. In order to combat this, we wanted to try simple classifiers that would only predict the top most likely cuisine given the data, and accuracy was measured if that cuisine was in the list of cuisine tags a recipe had.

Literature

In the first paper that we examined, A Cooking Recipe Multi-Label Classification Approach for Food Restriction Identification[1] (Larissa F. S. Britto, Luciano D. S. Pacifico, Emilia G. Oliveira, Teresa B. Ludermir), the researchers utilize the same dataset as in Generating Personalized Recipes from Historical User Preferences[2] (Bodhisattwa Prasad Majumder , Shuyang Li, Jianmo Ni, Julian McAuley). In the original paper by Professor McAuley et al., a new “augmented” dataset is created based on the dataset that we are utilizing in our experiments for this assignment. This dataset is used in several different research papers related to recommendations based on food and/or recipes and is quite robust.

In the paper by Larissa et al., the predictive task is identifying food restrictions in various cooking recipes. So, only the list of the ingredients in the recipe and the recipe categories (i.e. the type of restriction) are taken into consideration during training the model. The model itself is actually quite basic. It’s simply a random forest classifier using the previously mentioned features. This leads to quite good performance since for this specific task, identifying the list of key ingredients can lead to a high probability of correct classification for the restriction.

By nature, the predictive task we are tackling in this assignment is quite similar to the classification of a recipe. We are, after all, classifying the recipe based on its features into a category (but instead of categorizing a food restriction, we are categorizing the cuisine of the recipe). This is essentially what the latent factor model (see the section about **CuisineScore** above) is doing. We are assigning weights to specific ingredients, which ideally influence the type of classification we are making for the cuisine. Ingredients that are more common for specific

More complex approaches utilize natural language processing for text features like the recipe description. State of the art in literature can achieve around 85-90% accuracy, which was a little beyond our best results or 79% accuracy on the test sets. Further accuracy seems unlikely due to the unavoidable noise in the dataset (due to error like mislabeling) and the highly unbalanced nature of our classes. Textual features encoded with Word2Vec (Tomas) and TF-IDF both provided rich and powerful contextual features to help classify the data. In our simplest approach, we looked for

Sources:

1. Britto, Larissa, Luciano Pacífico, Emilia Oliveira, and Teresa Ludermir. " A Cooking Recipe Multi-Label Classification Approach for Food Restriction Identification."

Proceedings of the 17th National Meeting on Artificial and Computational Intelligence, Evento Online, 2020. SBC, 2020, pp.246-257.
2. Majumder, Bodhisattwa Prasad, et al. “Generating Personalized Recipes from Historical User Preferences.” arXiv.Org, 31 Aug. 2019, arxiv.org/abs/1909.00105.
3. Mikolov, Tomas, et al. "Efficient Estimation of Word Representations in Vector Space." arXiv preprint arXiv:1301.3781 (2013).
4. Ramos, Juan. "Using TF-IDF to Determine Word Relevance in Document Queries." Department of Computer Science, Rutgers University.

Models

Baseline Approach

The first approach we tried was just predicting the most common cuisine for every recipe. This baseline approach was chosen because as stated before, “American” cuisine accounts for the majority of recipes, so we wanted to see if we could improve upon this baseline approach using different methods and models we learned in class.

Simple Feature-Based Approach

We wanted to start with a simple feature-based classifier to see how this would function. In order to make this classifier, we chose 3 features that we thought would be useful in predicting the type of cuisine of the recipe: tags, ingredients, and description,

There was other preprocessing and data cleaning that needed to be done in order to create a feature-based classifier. For each recipe, tags included both the cuisine tag and the other tags not related to cuisine. Therefore, for each recipe, we separated the overall “tags” feature into 2 different features: “other_tags” and “cuisine_tags”. After separating this, we created a dictionary data structure that linked the frequency of all other_tags to cuisine_tags.

Tag_to_cuisine[other_tag][cuisine_tag] = frequency$_{\text{other\_tag,cuisine}}$

For ingredients, no data cleaning was necessary because the input was already a clean list of ingredients, so we only created a dictionary data structure that linked the frequency that linked the frequency of all ingredients to cuisine_tags.

Ingredient_to_cuisine[ingredient][cuisine_tag] = frequency$_{\text{ingredient,cuisine}}$

Finally, for the description, we did a simple text-based search to see if any cuisines in our set of cuisines were included in the description.
descriptionMatch = {1 if cuisine in description else 0}

With this plan the textual based features could be converted to a numerical score of each cuisine for each recipe. The equation to calculate this score is as follows

$$CusineScore_{cuisine} = \alpha * (\sum_{otherTag} (Frequency_{otherTag, cuisine})) + \beta * (\sum_{ingredient} (Frequency_{ingredient, cuisine})) + \gamma * (DescriptionMatch_{cuisine})$$

Singular Value Decomposition (SVD) Approach

The SVD model decomposes a matrix into three smaller matrices. In this case, the matrix represents the interactions between users and recipes, where rows correspond to users, columns correspond to recipes, and the values in the matrix represent encoded cuisine classes. The model predicts the most likely cuisine for a given user-recipe pair.

1. **User matrix:** This matrix represents user preferences across several “hidden” factors.
2. **Singular values matrix:** A diagonal matrix containing the importance of each hidden factor.
3. **Recipe matrix:** This matrix represents recipes in terms of the same hidden factors.

These hidden factors capture relationships between users and recipes, such as a user’s tendency to prefer certain cuisines and recipes’ associations with those cuisines.

The goal during training is to minimize the difference between the actual and predicted values for the observed data. The actual value is the encoded cuisine class for a user-recipe pair. The predicted value is computed as the dot product of the corresponding user vector and recipe vector, scaled by the singular values.

For a single user-recipe pair:

- Compute the predicted value by taking the dot product of the user vector, the singular values, and the recipe vector.
- Calculate the error as the difference between the actual and predicted values.

- Update the user and recipe matrices based on the error, using a learning rate to control the step size.

After training, the model predicts the cuisine class for a new user-recipe pair by computing the dot product of the user and recipe vectors. The result is a continuous value that represents the most likely encoded cuisine class. To obtain the predicted cuisine, the continuous value is rounded to the nearest integer and mapped back to the original cuisine labels.

XGBoost with One-vs-Rest (OvR)

Math Behind the Process

1. **Binary Classification:**

$$L = \sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] + \text{Regularization}$$

2. **OvR Decomposition:**

   For K labels (e.g., cuisines), the model builds K independent classifiers, each using the above loss function to predict one label at a time.

The XGBoost model with a One-vs-Rest strategy was chosen to address the challenge of predicting cuisines in a multi-label classification task. This approach enables the training of separate classifiers for each cuisine, accommodating the possibility of recipes belonging to multiple cuisines. XGBoost was selected because of its robustness against class imbalance and its efficiency in handling sparse data. Additionally, the algorithm offers interpretability through feature importance analysis, which is helpful for understanding the key contributors to predictions.

The primary challenge encountered was the significant class imbalance in the dataset, with larger classes like “other” dominating the distribution, while smaller classes such as “french” and “japanese” had relatively few samples. Despite these difficulties, the model achieved an accuracy of 67%, which reflects its overall correctness in prediction. The macro-averaged F1-Score, however, was 0.29, indicating poorer performance for smaller classes. The weighted average F1-Score of 0.61 highlights the disproportionate influence of the larger classes on the

model's performance. A low Hamming Loss of 0.07 suggests that the model made relatively few incorrect label predictions overall.

The results showed that the model effectively predicted dominant classes like "other," achieving a recall of 97% for this class. However, smaller classes such as "thai" and "japanese" had recall values below 10%, reflecting the difficulty in predicting less represented cuisines. This imbalance in performance underscores the need for further refinements.

Improvements could be achieved by addressing the class imbalance through techniques like oversampling or undersampling, or by applying class weighting during training. Including additional features derived from textual data such as tags or descriptions might enhance the model's ability to differentiate between cuisines. Moreover, advanced techniques like threshold tuning or deep learning architectures tailored to multi-label classification could further improve performance, especially for minority classes.

In conclusion, while the model provides reasonable accuracy for dominant classes, it requires optimization to achieve balanced performance across all cuisines.

Random Forest Classifier

From the findings from A Cooking Recipe Multi-Label Classification Approach for Food Restriction Identification, it seemed like using a random forest classifier approach with only the ingredients as features was appropriate.

First, we had to modify the input data to include the cuisine of the food. This wasn't very easy since the data comes unlabeled with respect to the cuisine of the food (we are only given descriptions and tags, not a definitive cuisine). So, we naively labeled foods by seeing if their respective cuisine frequently appeared in the recipe description or tags. We considered the following 28 cuisines: "japanese", "italian", "indian", "korean", "thai", "mexican", "chinese", "indonesian", "vietnamese", "filipino", "greek", "french", "malaysian", "turkish", "peruvian","nigerian", "spanish", "lebanese", "jamaican", "american", "brazilian", "persian", "moroccan", "german", "cuban", "polish", "dominican", and "colombian". If none of these cuisines were found, the entry was ignored and not used in our data for training the model. After labelling the data using the method above, we fit a random forest classifier with 500 estimators using the ingredients as features.

However, since our ingredients are given as a list of strings, we needed to convert them to a float representation for the random forest classification algorithm to work. We did so by tokenizing the ingredients list after converting the list of strings into one big string (separated by white space). Then, we “vectorized” these string representations of the ingredients by using the TF-IDF vectorizer in sklearn with the training set. After these labels were created, an 85% train test split was done and then we trained the RFC.

Neural Network Classifier + TF-IDF Model

This model leverages two powerful text-to-vector models (TF-IDF and Word2Vec) that were covered in class. The TF-IDF algorithm and the pretrained Word2Vec model embeds all of our textual features; such as our ingredient list, name, description, and recipe steps; into vector space. These features are then concatenated with our numerical features including the recipe cook time, nutrition, and multi-hot representations of the tags.

The dimensionality of the Word2Vec and TF-IDF embeddings are important. We chose 300 and 5000 respectively as the size of these embeddings. After a full concatenation, the resulting input vectors were of dimensionality 5717. The feature breakdown is below:

- Text features: 5000
- Numerical features: 3
- Nutrition features: 7
- Tag features: 405
- Ingredient vector features: 300

We passed this through a deep neural net with a bottleneck architecture. Our layers were: an input layer accepting 5,715 features, compressed down to a 16-dimensional representation, followed by dropout (0.1) and batch normalization for regularization, another hidden layer projecting up to 128 dimensions, and finally an output layer that classifies into one of 102 cuisines. Before training, we scaled each column of our input data matrix, making sure that we only use the training data to scale.

This model was quite robust with regards to its hyperparameters. We chose a typical learning rate of 0.001 and the first hidden dimension (a sort of "encoder" to a much lower dimensional latent space) of 16 to prevent overfitting.

This model is a natural choice for this task: dense neural nets are very good at extracting complex patterns from high dimensional and varied data (we have many different data features). We combined both Word2Vec and TF-IDF to get the best of both types of vector embeddings. Word2Vec allows us to capture the semantic relationships between similar ingredients, while TF-IDF leverages a simpler and larger embedding to extract more information from the description/recipe steps.

Finally, this model is also pretty computationally efficient, especially since Word2Vec was pretrained and allowed us to extract semantic information without training on a large corpus ourselves. The first hidden dimension of size 16 forces our model to generalize, rather than overfit on just the training data.

Results

Accuracy was measured through the following equation:

$$Score \ = \ \sum_{i=1}^{n} if \ predicted \ cuisine \in actual \ cuisine, \ score \ += \ 1$$

$$Accuracy \ = \ \frac{score}{length \ dataset}$$

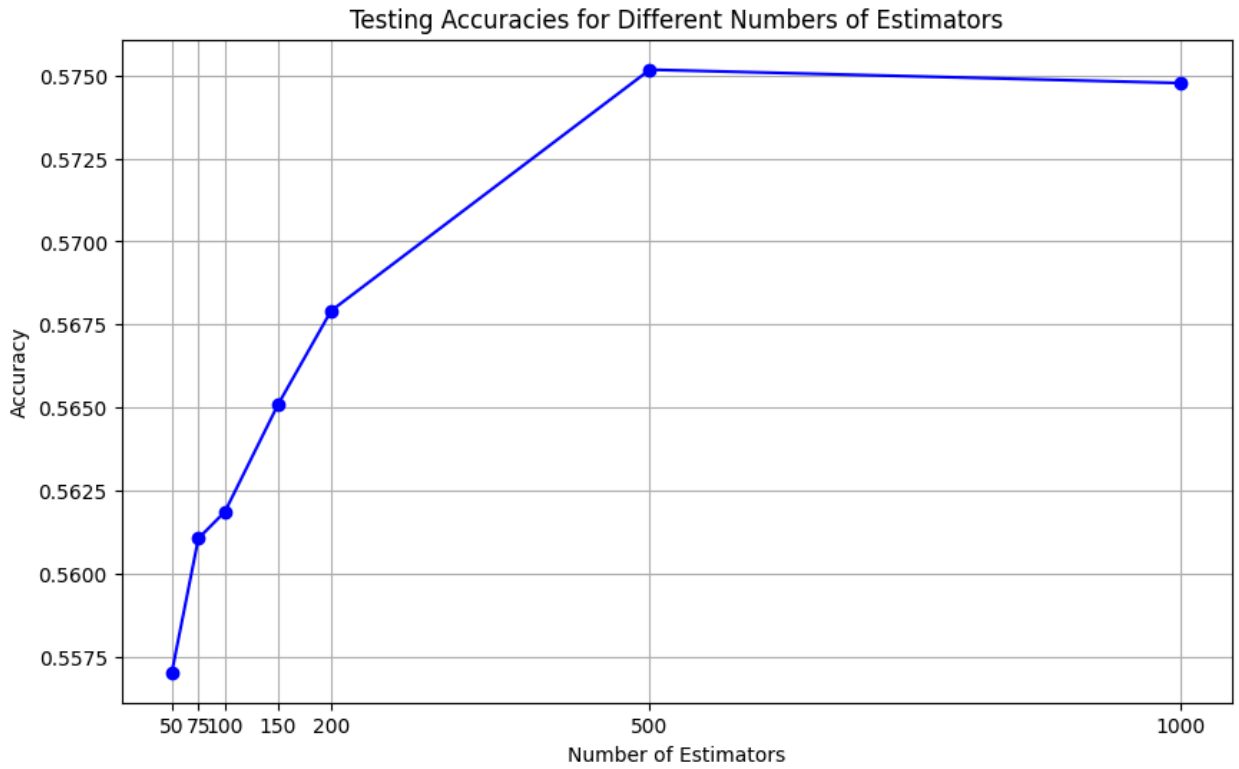
Our baseline approach, which simply predicted the most common cuisine (American), achieved an accuracy of 39%. This served as a starting point to assess the effectiveness of a more sophisticated model.

By employing a simple feature-based approach, we were able to improve our accuracy to 48%. In this model, the description weight played a crucial role. We assigned an extremely high weight to the description feature, which ensured that the cuisine mentioned in the description was almost always selected as the predicted cuisine. This gave the model a significant boost in performance, as the description often provides key insights into the cuisine type of a recipe.

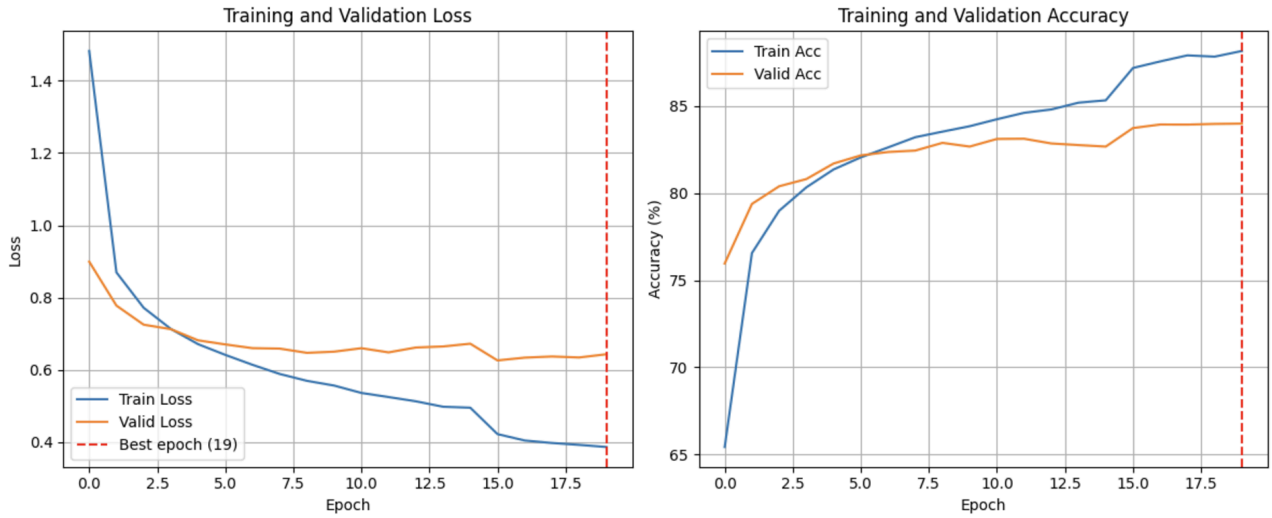
The weights for other_tags and ingredients were optimized using gradient descent. However, during the optimization process, we observed that most combinations of weights for these two features led to similar accuracy results, typically ranging from 47% to 48%. This suggested that the impact of these features was relatively minimal compared to the description weight.

The substantial impact of the description weight highlights the importance of textual features in predicting cuisine type. Since the description often contains explicit references to the cuisine, giving it a higher weight ensures that the model captures this critical information. In contrast, the other_tags and ingredients features, although important, did not have as strong of an effect on accuracy. Despite optimizing these features through gradient descent, the results were relatively stable, indicating that there may be diminishing returns when adjusting their weights.

For the RFC, accuracy was simply the proportion of correctly classified samples. We found that even though our classification task was far more complex (now have 28 classes instead of 1), the performance was still on par with the individual classification task using only the ingredients as features. We were getting around 55-60% accuracy when predicting cuisines. This is quite a powerful model, since it can accurately classify a recipe given the ingredients more than half the time. We experimented with the number of estimators and found that the more estimators we had, the better our performance on the testing set was. Up to a certain point, this is reasonable, but after the 200 mark, there is a good chance that the model was simply overfitting to the testing data. Here is the plot of the accuracy vs. the number of estimators:



Finally, our neural net classifier (using word embeddings) was able to achieve a final accuracy of **80.22**% on our test set. The accuracy was measured using the same metric as above. The learning curves were stable, showing a reasonable generalization gap of 2.2% at our final epoch and no overfitting:




```
Best validation accuracy: 83.99% (epoch 19)
```

We see a crossing-over point between the train and validation losses around the 5th epoch. This is likely due to dropout/batchnorm regularization effects penalizing the model during training more heavily than during validation. However, the model eventually begins to perform better on training versus validation. We terminate training when the model begins to overfit on training, climbing to an 87% accuracy while validation accuracy remains stable. This accuracy is a reasonable expectation of performance from our model.

Concretely, here are the label predictions, sorted by the most to least common classes. Note that the supports do not reflect the true underlying class distributions because we used weighted sampling to help teach our model to predict rare classes.

Cuisine	Precision	Recall	F1-Score	Support
american	0.88	0.96	0.92	6,159
european	0.73	0.83	0.78	2,734
asian	0.79	0.88	0.83	2,390
mexican	0.83	0.84	0.84	1,085
south-west-pacific	0.66	0.54	0.59	786
canadian	0.75	0.54	0.63	690
african	0.71	0.61	0.65	469

greek	0.77	0.69	0.73	434
kosher	0.99	0.9	0.94	411
french	0.66	0.47	0.55	384

Note that we have steadily decreasing recall scores as the classes get rarer. Our model still biases toward predicting the most common labels. This is to be expected due to the overrepresentation of common labels like "american" in our training set.

Again, as expected, much rarer labels are never predicted:

Cuisine	Precision	Recall	F1-Score	Support
namibian	0	0	0	1
congolese	0	0	0	2
mongolian	0	0	0	2
somalian	0	0	0	2
angolan	0	0	0	3
georgian	0	0	0	3
laotian	0	0	0	3
soul	0	0	0	3
sudanese	0	0	0	3
hunan	0	0	0	5

Overall, this model performs quite well on the test set and shows no signs of overfitting. Notably, we rely on the expressiveness of our input vectors, which are encoded and scaled with Word2Vec and TF-IDF for the list of ingredients and recipe text, respectively. These NLP tools allow us to leverage the text for prediction, which is allows to use the richest