What's Cooking?

Machine Learning Engineer Nanodegree
Capstone Project

Albert Pan
November 23, 2018

1 Definition

1.1 Project Overview

Food plays a major part in any culture around the world. In fact, the geographical characteristics and cultural associations of a region directly influence their cuisines. By investigating the ingredients used in various cuisines, we can gain a better understanding of the geographical and cultural landscape of different regions.

Han Su et. al.[1] has worked on investigating if recipe cuisines could be identified by their ingredients, using data from food.com. They treated each ingredient as a feature and examined the common ingredients for each cuisine. Their study provides good insight on how to approach this project and what results we could expect to achieve.

For this project, we can use machine learning techniques to attain useful predictions with our data. This project will give me an opportunity to work with real-world datasets and to learn about the relation between ingredients and cuisines. This kind of research can be expanded to other fields of study involving text classification.

[1] Su, Han & Lin, Ting-Wei & Li, C.-T & Shan, Man-Kwan & Chang, Janet. (2014). Automatic recipe cuisine classification by ingredients. UbiComp 2014 - Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. 565-570. 10.1145/2638728.2641335.

1.2 Problem Statement

Using the data provided by Yummly[2], the challenge is to predict the cuisine of the dish from its list of ingredients. More specifically, this would be multi-class classfication problem, as we have 20 different cuisines we can predict. We can use a machine learning model that utilizes this data to predict the appropriate cuisine.

My strategy for solving this problem is as follows:

1. Download data from Kaggle
2. Explore data with visualizations
3. Preprocess data and extract features
4. Train and test model
5. Tune hyperparameters

1.3 Metrics

As previously mentioned, our training data seems to be unbalanced, and thus I will be using the F_1-score to evaluate the model. The F_1-score is the weighted average of the precision and recall, and can be expressed mathematically with the following form:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1}$$

We can also take a look at the precision and recall metrics on their own. Precision can be expressed with the following equation, where $dish_x$ is a dish of a particular cuisine x:

$$precision = \frac{dish_x \, correct}{total \, dish_x} \tag{2}$$

where $dish_x \, correct$ is the number of dishes of cuisine x that are correctly classified as x, and $total \, dish_x$ is the the total number of dishes that were classified as x.

[2] https://www.kaggle.com/c/whats-cooking-kernels-only/data

Recall can expressed with:

$$recall = \frac{dish_x\ correct}{dish_x\ correct + dish_x\ incorrect} \tag{3}$$

where $dish_x\ correct$ is the number of dishes of cuisine x that are correctly classified as x, and $dish_x\ incorrect$ is the number of dishes of cuisine x that are incorrectly classified.

2 Analysis

2.1 Data Exploration

The dataset that I will be using is provided by Yummly, and it consists of two JSON files. The most important file is the *train.json* file, which has 39774 rows of data, and this is the data that we will be using to train our model. The *test.json* file has 9944 rows of data, and this is the data that we will be using to evaluate our model.

id	**cuisine**	**ingredients**
10259	greek	[romaine lettuce, black olives, grape tomatoes, ...]
22213	indian	[water, vegetable oil, wheat, salt]
12734	italian	[chopped tomatoes, fresh basil, garlic, ...]
41995	mexican	[ground cinnamon, fresh cilantro, chili powder, ...]
2941	thai	[sugar, hot chili, asian fish sauce, lime juice]

Table 1: A few examples from our training dataset.

The training data contains three fields, *id*, *cuisine*, and *ingredients*. The *id* field is a unique integer identifier for a particular dish, and is not needed for modeling or analysis. The other two fields, *cuisine* and *ingredients* are very important. *ingredients* consists of the list of ingredients for each dish, while *cuisine* denotes the cuisine. We will be extracting our features from *ingredients* and we will be trying to predict the target class *cuisine*.

2.2 Exploratory Visualization

To better understand the data, we can take a look at the distribution of cuisines in our training data:

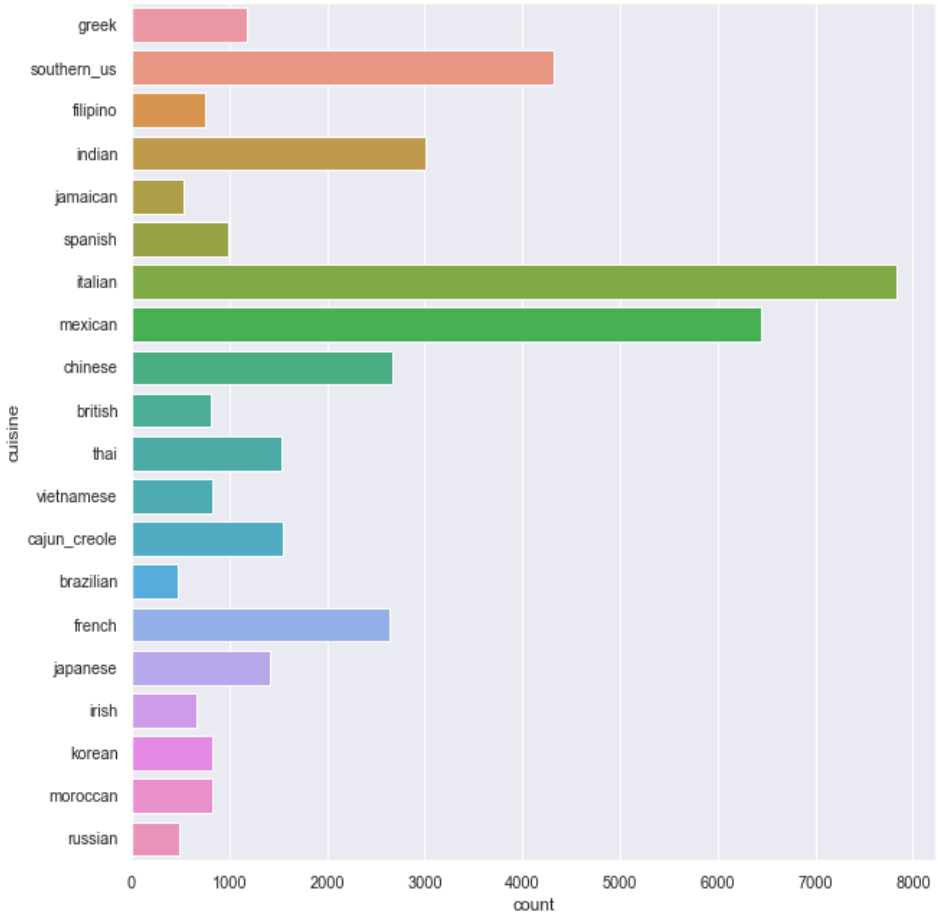

Figure 1: Cuisine distribution in training dataset

We see that our training data is unbalanced, as there are much more "italian"

and "mexican" cuisines that any of the other cuisines. This is why we decided to use the F_1-score as our evaluation metric, as the F_1-score will take into account both precision and recall.

We can also take a closer look at our ingredients. We have 6714 kinds of distinct ingredients, and there are a few ingredients in particular that are more common than the rest:

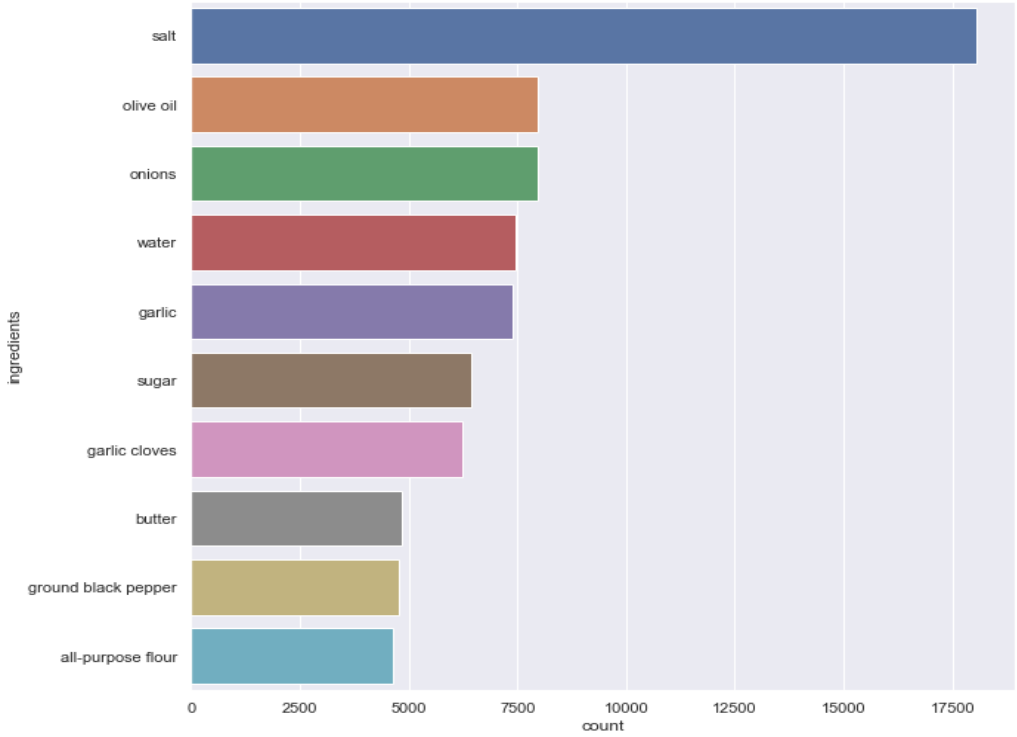

Figure 2: 10 most common ingredients

We can see that a good number of ingredients are common to many different recipes and cuisines. In particular, ingredients such as *salt*, *olive oil*, *onions*, and *water* are very common, and this makes sense. *Salt* is one of the ingredients that improves the flavor of a recipe, *olive oil* allows for a better transfer of heat for cooking, and so on. All of the ingredients shown in Figure 2 are essential to cooking, and it really captures the ingredients that most cuisines have in their dishes.

One important point to mention is that some ingredients are very similar to each other in both name and in usage. For example, let's take a look at

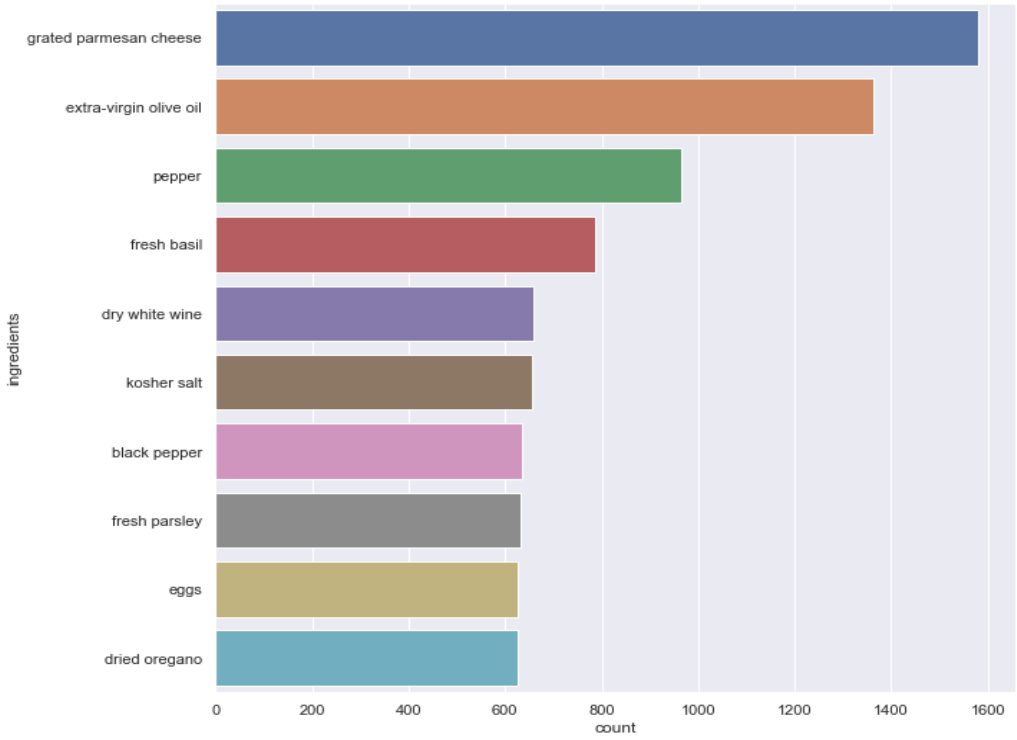

Figure 3: Italian: 10 most common ingredients

garlic and *garlic cloves*. They are both *garlic* ingredients, and mainly serve the same purpose in a dish. Other examples include *eggs* and *large eggs*, as well as *ground black pepper* and *black pepper*. Thus, we will combine these ingredients into one homogenous ingredient.

Since the ingredients mentioned above are common to many cuisines, they may not be very helpful for distinguishing one cuisine from another. We can filter out the most common ingredients from the ingredients used in *Italian* and *Mexican* dishes and see what they tells us:

Once we filtered out the most common ingredients, we can see the ingredients that are representative of their respective cuisines, as evidenced in figures 3 and 4. These are the ingredients that will distinguish a certain cuisine from another cuisine, and we will be focusing on extracting these ingredients in our data preprocessing.

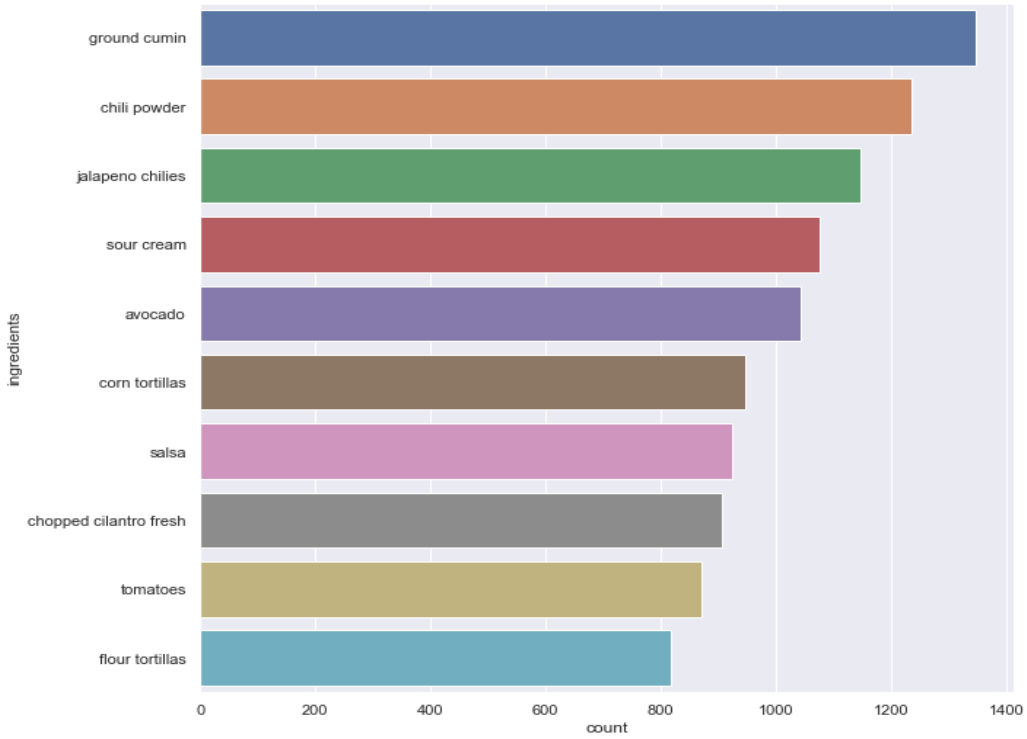

Figure 4: Mexican: 10 most common ingredients

2.3 Algorithms and Techniques

This is a classification problem, and so I will be using algorithms and techniques that are suited for classification problems. In particular, I will be using the following models/algorithms:

- Support Vector Machines (LinearSVC[3]): Support vector machines, or SVMs, are a set of supervised learning models that uses decision boundaries for both classification and regressions problems. SVMs construct a hyperplane that allows us to linearly separate data by transforming the feature space. For this problem, I decided to use a linear kernel because it is less computationally expensive and allows us to use multi-class classification.

- Naive Bayes (MultinomialNB[4]): Naive Bayes is a supervised learning algorithm that is based on Bayes' Theorem[5], and it assumes that all of the features are independent. This particular algorithm has been historically used for SPAM detection for its speed and accuracy. For our problem, we will be using MultinomialNB (Multinomial Naive Bayes model), which is a multi-class classifier implementation of Naive Bayes.

- Decision Trees (DecisionTreeClassifier[6]): Decision Trees learn decision rules and generates trees that can be used for both classification and regression problems. Using the input features, it can decide which subtree to follow until it reaches the bottom of the tree, in which it will return its prediction. We will be using the DecisionTreeClassifier from the *scikit-learn* library to perform multi-class classification.

- Random Forests (RandomForestClassifier[7]): Random forests are ensemble methods that can used to build models for classification and regression models[8]. They operate by constructing a series of decision trees and outputting a combined prediction from all of the trees. Random Forests aim to correct the decision tree's tendency to overfit to

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[5] https://en.wikipedia.org/wiki/Bayes%27_theorem
[6] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[8] https://en.wikipedia.org/wiki/Random_forest

the training data. We will be using the RandomForestClassifier, which is capable of multi-class classification.

- Gradient Boosted Trees (XGBClassifier [9]): Gradient boosting is a technique used in both classification and regression problems. and it builds a prediction model by creating an ensemble of weak learners, usually with decision trees[10]. It builds the model by utilizing the optimization of an arbitrary differentiable loss function. We will be using XGBClassifier from the *xgboost* library.

2.4 Benchmark

For our baseline benchmark, we can use the metric obtained by predicting the most common cuisine in the training and testing datasets. The most common cuisine is *Italian*, and our benchmark model will predict Italian to all recipes. This would give us a benchmark F_1-score of 0.3292.

3 Methodology

3.1 Data Preprocessing

Our training dataset is divided into *ingredients* and *cuisine*. *cuisine* is already ready for implementation. What we do have to do is preprocess each recipe's ingredients and uniformly format them. We will perform the following steps to preprocess the data:

1. Remove non-letter characters (punctuation) using regular expressions.
2. Convert all letters to lowercase.
3. Remove descriptor words (e.g. chopped, minced, toasted, etc.) and combined similar ingredients into homogenous groups.
4. Apply a lemmatization process to the words.
5. Convert words to vectors using tf-idf.

[9] https://xgboost.readthedocs.io/en/latest/python/python_api.html
[10] https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Lemmatization refers to the process of grouping together different inflected forms of words so that they can be analyzed as a single term[11]. This means that words such as *tomatoes* will be transformed into *tomato*, thus allowing us to treat them as the same term. I used *nltk*'s WordNetLemmatizer[12] to perform lemmatization on our ingredients.

tf-idf is a method in which you can extract information from a string. It evaluates the weight of the value of words according to the number of times that it appears in a text and offsets it by the number of times it appears in the entire corpus. In our problem, tf-idf seems to work well because the number of times an ingredient appears in the recipe of a particular cuisine could help us determine if that specific ingredient is indicative to that cuisine. We will use the output of *scikit-learn*'s TfidfVectorizer[13] to obtain the count matrix that we will use as input for our model.

We will also convert our *cuisine* class to integer representation using *scikit-learn*'s label encoder[14] so that the model can recognize our target class.

3.2 Implementation

We did not change our evaluation metric (F_1-score), as it works well in our case since we have an uneven distribution of data.

The supervised learning algorithms that we used were initialized with the default parameters. All of the models are from the *scikit-learn* library, with the exception of the XGBClassifier, which is from the *xgboost* library.

The test data can only be evaluated on the Kaggle competition, and so we split our training data into 70% training and 30% testing.

3.3 Refinement

We trained all of the models and got their training and testing scores. To refine the models, we tuned their hyper-parameters using *scikit-learn*'s GridSearchCV[15]. GridSearchCV allows us to perform an exhaustive search over all the hyper-parameters that we specify in a 5-fold cross-validated dataset. We then used the optimized classifiers and re-evaluated their performances.

[11] https://en.wikipedia.org/wiki/Lemmatisation
[12] https://www.nltk.org/_modules/nltk/stem/wordnet.html
[13] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[14] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[15] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

4 Results

4.1 Model Evaluation and Validation

4.2 Justification

5 Conclusion

5.1 Free-Form Visualization

5.2 Reflection

5.3 Improvement