Recipe Type Classification Using Ingredient Lists

Project Proposal

Huaxin Pang
Zayd Hammoudeh

CS256 – Fall 2015

Table of Contents

Recipe Type Classification Using Ingredient Lists

1. Introduction

We can write this section later. For now, I wanted to focus on describing my algorithm plan.

2. Project Goal

Develop an algorithm that can classify a recipe's type of cuisine (e.g. "Italian", "Indian", "Cajun/Creole", etc.) based solely off a list of ingredients.

3. Abstracting an Ingredient List into a Data Point

Each record in the training set represents a single recipe. The components of the record are a list of ingredients, a class value (i.e. cuisine type), and an identification number. Records are formatted using JavaScript Object Notation (JSON) as shown in figure 1.


```
{
    "id": 24717,
    "cuisine": "indian",
    "ingredients": [
        "tumeric",
        "vegetable stock",
        "tomatoes",
        "garam masala",
        "naan",
        "red lentils",
        "red chili peppers",
        "onions",
        "spinach",
        "sweet potatoes"
    ]
},
```


Figure 1 – Example Record for a Recipe of Cuisine Type "Indian"

A record of the type shown in Figure 1 does not naturally lend itself well to a machine learning algorithm since the core data (i.e. ingredient list) is relatively unstructured. As such, a structure must be applied to the record in order to adapt it to traditional algorithms.

The simplest encoding scheme for records of this type is transform the ingredients list into an N dimensional vector, where N is the number of ingredients in the entire data set. Hence, each attribute in the vector would be binary value representing whether a particular ingredient is present in the current record. Using this approach, it then becomes possible to use machine learning algorithms that support nominal attributes.

4. Algorithm Overview

The high dimensionality of encoding scheme described in section 3 does not lend itself well to many of the machine learning algorithms discussed in class. For example, a decision tree would be prohibitively large in size; similarly, a rule based classifier would be potentially difficult to construct and would have hundreds or thousands of rules.

In our view, there are two algorithms that at first thought appear the most promising fit for this dataset. The first is a neural network which is well suited for these types of classification problems with large numbers of dimensions. However, given the remaining duration of the semester, we foresee that this approach may be prohibitively complex.

A more direct solution than a neural network would be to use a K-Nearest Neighbor (K-NN) classifier as we foresee that recipes of the same cuisine type will have high levels of similarity of ingredients. What is more, with fusion cuisine becoming increasingly prevalent, K-NN may allow us to predict a set of closely aligned cuisine types when an ingredient list fits well into more than one category.

To use K-NN for this problem, the three key challenges that need to be solved are:

1. Dataset Preprocessing and Cleanup
2. Distance Metric Definition
3. Selection of K

The subsequent sections describe our planned approach for these three challenges.

5. Stemming Ingredient Names

In order to achieve the best results out of any machine learning algorithm, it is important that the records be as stylistically consistent as possible. For example, it is often considered disadvantageous for a training dataset to represent the same piece of information using multiple different notations. In contrast, the Yummly dataset has significant variation because the ingredient lists were written by countless different people, who each have their own notational style. For example, the ingredient cilantro appears as “chopped cilantro fresh”, “fresh cilantro”, “cilantro leaves”, “chopped cilantro” as well as just “cilantro” depending on the recipe. Similarly, garlic appears as “garlic cloves”, “minced garlic”, “chopped garlic”, “crushed garlic” and simply “garlic”. As such, some degree of “stemming” of the ingredient

names/descriptions will be required in order to eliminate cases of distinction without a difference. We expect this process to be somewhat iterative as we work to refine the global ingredient list without compromising the integrity of the data.

6. Inter-Recipe Distance Measure

A key component in the K-Nearest Neighbors algorithm is metric used to quantify the difference between two records.

The Modified Value Difference Metric (MVDM), shown in equation (1), is commonly used to quantify the difference between two nominal attribute values; we will modify this approach slightly by using MVDM to quantify the difference between two ingredients I_1 and I_2. Hence, if two ingredients (e.g. basil and tomatoes) are commonly associated with the same cuisine type (e.g. Italian), then the distance between the two ingredients will be low (minimum 0). In contrast, if two ingredients (e.g. Kimchi and garam masala) are never in the same type of cuisine, then the distance will be high (with a maximum value of 2).

$$d(I_1, I_2) = \sum_{i=1}^{k} \left| \frac{n_{1,i}}{n_1} - \frac{n_{2,i}}{n_2} \right| \quad (1)$$

For our algorithm, k is the number of cuisine types; n_1 and n_2 are the total number of recipes that have ingredients I_1 and I_2 respectively. $n_{1,i}$ is the number of recipes of cuisine type i that have ingredient I_1; similarly, $n_{2,i}$ is the number of recipes of cuisine type i that have ingredient I_2.

Using the training data provided on Kaggle, we will use MVDM to calculate the similarly between every possible pair of ingredients in the entire dataset. This inter-ingredient distance information will then be used to calculate the distance between two recipes $R_1$and R_2 using equation (2).

$$d(R_1, R_2) = \frac{1}{M \cdot N} \cdot \sum_{i=1}^{M} \sum_{j=1}^{N} d(I_1, I_2) \quad (2)$$

M represents the number of ingredients in recipe R_1 while N represents the number of ingredients in recipe R_2. (2) calculates the sum of the distances between each pair of ingredients. To ensure that recipes are not penalized for having more ingredients, we normalize the sum of sums by the total number of ingredient pairs (i.e. $M \cdot N$).

7. Determining "K"

Depending on what time allows, there are different possible approaches that we can use to determine the optimal value of K. One option would be to use cross validation to divide the data set into a set of d disjoint partitions. For each training run, we could look at how accurately different values of K classify the data. Using that information, the simplest option would be to select a single value of K for our algorithm. An alternative option we may choose to explore is to allow the algorithm to use multiple values of K, each of which has its own weight; after the class value has been selected for each value of K, the algorithm could then select the class value with the highest total weight as the final result.

8. Conclusion

I can write this section later. For now, I wanted to focus on describing my algorithm plan.