Cuisine Classification by Recipe

By Brian Cheang (bmc78), David Jin (dj256), Alvin Qu (aq38)

Keywords: *machine learning, multi-class text classification, probabilistic classification, logistic regression*

Introduction

A large part of a culture's makeup is its cuisine. Throughout history, different recipes have been spread via trade, word of mouth, or through generational migration. The world's most popular dishes differ in their flavor profiles largely due to their ingredients. Regional disparities in crops, livestock, and spices have shaped today's diverse portfolio of flavor profiles. At the same time, however, globalization and trade have intertwined many of our cultures, resulting in an overlap in ingredients used in different cuisines. For example, East Asian trade led to similarities between Chinese, Japanese, Korean, and Tibetan cuisine. A Chinese classic, *mantou* (steamed, sweetened bun), appears in Japan as *manzu*, in Korea as *mandu*, and in Tibet as *momo*.

As a result, the motivation behind our project was to be able to attempt to understand cultural differences and similarities by quantifying their respective cuisine disparities through ingredients. Although this project initially started as a non-probabilistic classification, where we wanted to predict the cuisine given a set of ingredients, we eventually transitioned and wanted to answer the bigger question of the relationships between ingredients and cuisines by being able to predict a percentage match between a set of ingredients to cuisines. Instead of asking the question: "What is typically in a Cajun dish? What about a Japanese recipe?", we instead posed the more interesting contrary: "If I have eggs and soy sauce, what cuisines would I be more likely to make?"

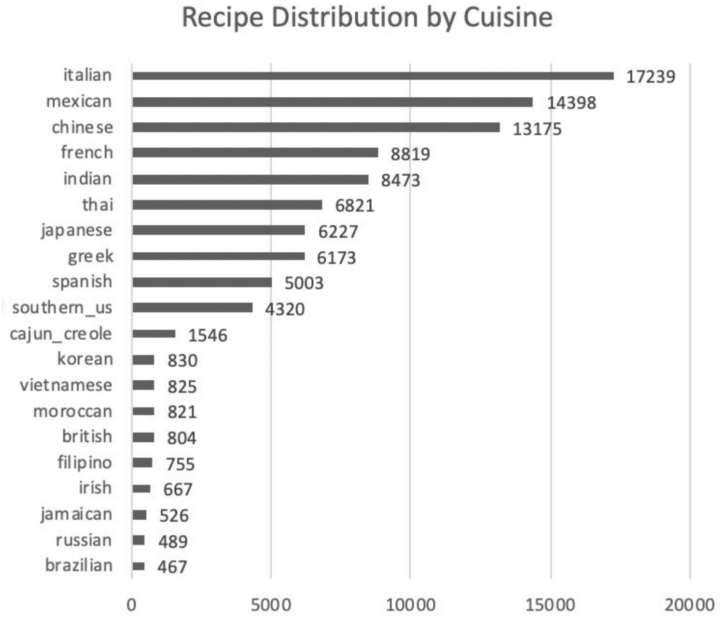
Dataset

Our dataset of 108,322 recipes and 20 cuisine types was gathered from two sources:

- The first source is *Recipe Ingredients Dataset*[1] from a previous Kaggle competition. The data was in the structure of an array of JSON objects.
- The second source is *66k-Yummly*[2], a dataset of around 66,000 recipes from Recipe Recommendation website, Yummly.

Below is a sample entry in our combined dataset:

```
{
  "cuisine": "american",
  "ingredients": [
      "1 (18.25 ounce) package chocolate cake mix",
      "1 1/3 cups water",
      "1/2 cup vegetable oil",
      "3 eggs",
      "15 large fresh strawberries",
      "2 cups white frosting",
      "36 fresh blueberries, rinsed and dried" ]
}
```



One thing to note is that the distribution of data is biased, with a large proportion of Italian, Mexican, and Chinese recipes. Training and testing data is created using a stratified .75-.25 split. This means that both 75% of the dataset is used for training, 25% for testing. Stratified train test split ensures that both datasets have the same proportion of label values so training and evaluation are representative of the total dataset.

Text Cleaning

Before training our models for classification, we applied several text cleaning procedures to remove noise from our dataset. Though information about cuisine type can be revealed through the proportion and amount of ingredients in a recipe, the formatting and units of ingredients was inconsistent throughout the dataset. Thus, we decided to filter out numeric characters from the dataset and focus on training the model to classify based only on whether an ingredient is used. All non-alphabetic characters were removed. This step is included to remove the many occurrences of numbers when measuring units in the ingredients list (eg. ‘4 oz salmon’).

Each word in the dataset is converted to their base form through lemmatization. This step is crucial for improving the accuracy of the model as lemmatizing the text will return all ingredients to singular form (ex: ‘eggs’ -> ‘egg’). Thus preventing the text vectorizer from recognizing singular and plural forms of each ingredient as two separate features.

Feature Engineering

An important aspect of our classification performance is the vectorization of text. Text vectorization is extremely important to connect our application’s input, user-inputted text, to our model’s input. Machine-learning models largely rely on linear-algebra operations for their prediction computations.

Specifically, a processor's GPU is more fit to handle vectors, a structure that allows for parallel computation, making training and testing on large data-sets possible.

In our case, we need to use a "word vectorizer" to convert our text data into vectors that our model can understand. We do this by first building a vocabulary of most frequent words that our vectorizer uses to tokenize text into vectors. Each vector has a list of integer values, which acts as an integer count that indicates the number of occurrences of each feature in an input. For example, a vocabulary of ["salmon", "salt", "rye", "water", "paprika"] and user input of "a salmon filet and a pinch of salt" would vectorize to [1, 1, 0, 0, 0]. After tuning parameters, we found that our classification model worked best on training features from a word count vectorizer with the following specifications and parameters:

- Binary Vectorization
  - Text is vectorized using only 0s and 1s to indicate the presence or absence of each feature.
- Ngram Range of 1 - 2
  - Extracted features are either single words or two adjacent words.
- 7000 Maximum Features
  - Based on previous research, the original dataset has roughly under 7000 unique ingredients. As the size of our training dataset is relatively small, features of length 7000 is not an issue.
- Custom list of stopwords removed
  - Adjectives (i.e. "Finely", "Fresh", "Large", "Sliced", "Cooked"…)
  - Traditional English Stopwords (i.e. "and", "or", "I"…)
  - Common Typos in our Dataset (i.e. "sauc", "appl", …)
    - We initially implemented this procedure by using online spell checking libraries to filter out misspelled words. However, upon exploring filtered out features, we discovered that many niche foreign ingredients were not recognized by common spell checking libraries and were incorrectly filtered out. As a result, we resorted to manually hardcoding a set of typos.

Non-probabilistic Classification

Baseline Models

- Baseline 1: *Proportional Predictor (accuracy = 10%)*
  Our initial baseline model randomly produces a cuisine prediction with probabilities based on each cuisine's proportion in the training dataset. For example, as the dataset is made up of 14% Mexican cuisines, the model will output Italian with a probability of 14%.
- Baseline 2: *Italian (accuracy = 18%)*
  The second baseline model predicts the majority cuisine label ('Italian') for every input.
- *LogReg2015 - B. Li, M. Wang (accuracy = 78.4%)*
  Previous research[3] on the same classification task yielded high classification accuracy using a Logistic Regression model on the Yummly-66K dataset.

Logistic Regression

There were a few reasons why we chose to use logistic regression for our model. First, logistic regression works better the larger the data set is. Next, a logistic regression model is a better fit for our dataset, since we cannot assume that the vectors obtained from vectorizing our recipe dataset is normally distributed, especially when we have a lot more data from certain cuisine than others. Lastly, logistic regression is better suited to make probabilistic classifications than models based on, for example, Naive-Bayes, or linear regression, because it makes non-binary predictions.

Our logistic regression model employs stochastic gradient descent for minimizing an objective function (negative log-likelihood) on each update to make for an increasingly accurate classifier. To prevent overfitting, we modify the objective function to add regularization. Although, our logistic regression achieved highest accuracy with L1 regularization, we opted for L2 regression instead, sacrificing a 1% improvement for the robustness and stability of a ridge regression.

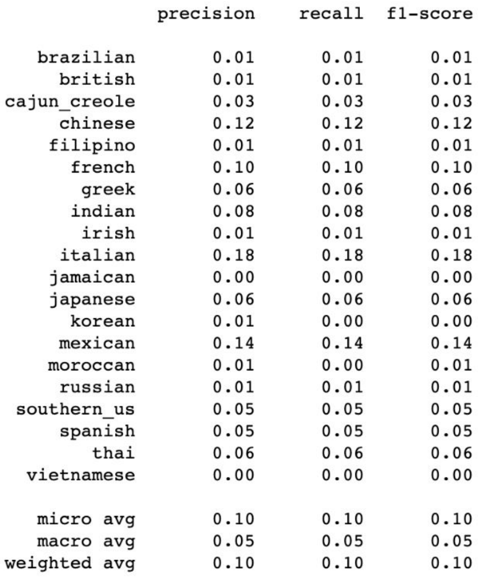
The objective/hypothesis function is a function that we want or believe to be similar to the true function in terms of classification based on input. In our case, our hypothesis function makes use of the logistic sigmoid function. The logistic sigmoid function, unlike the linear function, transforms any input to a probability value between 0 and 1.

For the purposes of our project, we use gradient-descent to update the objective function. The gradient-descent function allows update the z-value, changing the logistic sigmoid function, which determines how and what we classify input. The logistic regression model trained on our training vectors output a one-dimensional vector of 20 entries, each representing a unique label. Each value represents the probability that, given the input data, the data maps to a certain class. An example entry in this vector could be: P(Y = “Korean” | x = “one filet of salmon) = .032. Typically, linear regression models use cut-off values to make a binary classification, but because we are dealing with a multi-class problem, we have two strategies for probabilistic and non-probabilistic. For the probabilistic model, we simply return the probability vector, as each row value represents the percentage match of input and cuisine. For our non-probabilistic model, we take the maximum probability from the vector and take the respective class as our classification, as it represents the corresponding class with the highest probability.

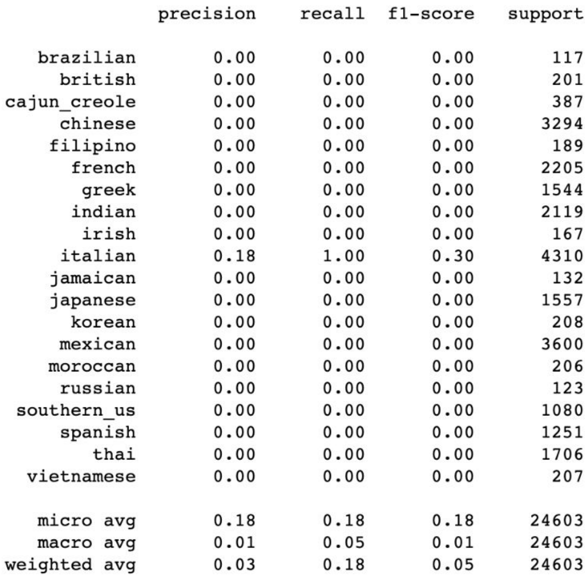
One thing to note about our logistic regression model that made a non-probabilistic prediction is that it is not strictly a classifier by itself, meaning that it does not automatically make a classification based on our input, but makes a classification decision based off the vector output that contains the conditional probabilities, as discussed above.

Model Evaluation

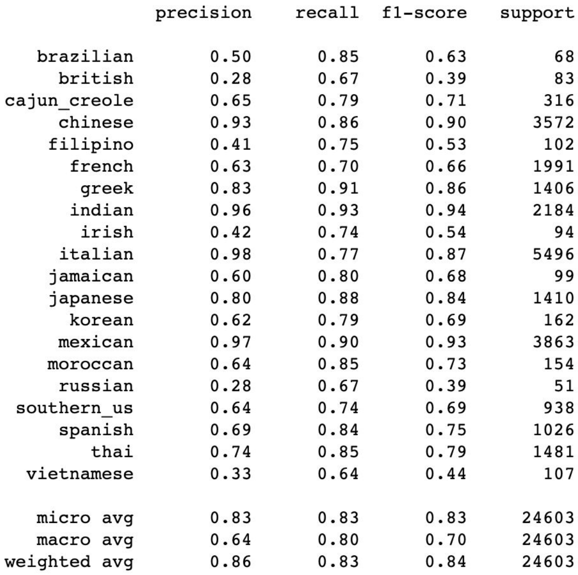
The per-class precision, recall, f-1 score breakdown for Baseline 1 (top left), Baseline 2 (top right), and our final Logistic Regression model (bottom) are provided below:



```
              precision    recall  f1-score

   brazilian       0.01      0.01      0.01
     british       0.01      0.01      0.01
cajun_creole       0.03      0.03      0.03
     chinese       0.12      0.12      0.12
    filipino       0.01      0.01      0.01
      french       0.10      0.10      0.10
       greek       0.06      0.06      0.06
      indian       0.08      0.08      0.08
       irish       0.01      0.01      0.01
     italian       0.18      0.18      0.18
    jamaican       0.00      0.00      0.00
    japanese       0.06      0.06      0.06
      korean       0.01      0.00      0.00
     mexican       0.14      0.14      0.14
    moroccan       0.01      0.00      0.01
     russian       0.01      0.01      0.01
 southern_us       0.05      0.05      0.05
     spanish       0.05      0.05      0.05
        thai       0.06      0.06      0.06
  vietnamese       0.00      0.00      0.00

   micro avg       0.10      0.10      0.10
   macro avg       0.05      0.05      0.05
weighted avg       0.10      0.10      0.10
```



```
              precision    recall  f1-score   support

   brazilian       0.00      0.00      0.00       117
     british       0.00      0.00      0.00       201
cajun_creole       0.00      0.00      0.00       387
     chinese       0.00      0.00      0.00      3294
    filipino       0.00      0.00      0.00       189
      french       0.00      0.00      0.00      2205
       greek       0.00      0.00      0.00      1544
      indian       0.00      0.00      0.00      2119
       irish       0.00      0.00      0.00       167
     italian       0.18      1.00      0.30      4310
    jamaican       0.00      0.00      0.00       132
    japanese       0.00      0.00      0.00      1557
      korean       0.00      0.00      0.00       208
     mexican       0.00      0.00      0.00      3600
    moroccan       0.00      0.00      0.00       206
     russian       0.00      0.00      0.00       123
 southern_us       0.00      0.00      0.00      1080
     spanish       0.00      0.00      0.00      1251
        thai       0.00      0.00      0.00      1706
  vietnamese       0.00      0.00      0.00       207

   micro avg       0.18      0.18      0.18     24603
   macro avg       0.01      0.05      0.01     24603
weighted avg       0.03      0.18      0.05     24603
```



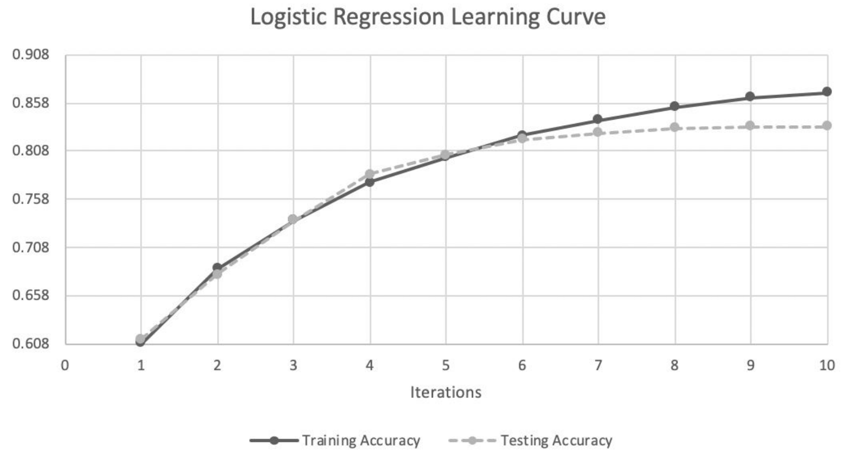
```
              precision    recall  f1-score   support

   brazilian       0.50      0.85      0.63        68
     british       0.28      0.67      0.39        83
cajun_creole       0.65      0.79      0.71       316
     chinese       0.93      0.86      0.90      3572
    filipino       0.41      0.75      0.53       102
      french       0.63      0.70      0.66      1991
       greek       0.83      0.91      0.86      1406
      indian       0.96      0.93      0.94      2184
       irish       0.42      0.74      0.54        94
     italian       0.98      0.77      0.87      5496
    jamaican       0.60      0.80      0.68        99
    japanese       0.80      0.88      0.84      1410
      korean       0.62      0.79      0.69       162
     mexican       0.97      0.90      0.93      3863
    moroccan       0.64      0.85      0.73       154
     russian       0.28      0.67      0.39        51
 southern_us       0.64      0.74      0.69       938
     spanish       0.69      0.84      0.75      1026
        thai       0.74      0.85      0.79      1481
  vietnamese       0.33      0.64      0.44       107

   micro avg       0.83      0.83      0.83     24603
   macro avg       0.64      0.80      0.70     24603
weighted avg       0.86      0.83      0.84     24603
```

The main metrics used to evaluate our model's performance were the per-class precision and the micro average precision score. Discussions of accuracy for non-probabilistic classification in our paper refer to the micro average precision score. This is an averaged precision score over all classes that takes into account the amount of data contributed by each class. With a heavily imbalanced dataset, we would like to prioritize the classification ability of the more frequent classes. As our model had significantly better micro average precision than macro average precision (all classes treated equal), it suggests that our model is more learned in classifying the majority cuisines. This difference is logical as minority cuisines provide a smaller training set. The model is then more likely to be confronted with an unfamiliar recipe in the corresponding testing set.

The model significantly outperformed the two trivial baseline models in each of the 20 classes as well as in total accuracy. We believe that the logistic regression model was able to achieve high accuracy as a cuisine classifier as logistic regression models typically performs well with large training set with non-Gaussian distributions. Our model was even successful in outperforming the LogReg2015 model

(78.4% Accuracy) from previous research for the same classification problem. We believe the largest factors for improvement between our Logistic Regression model and LogReg2015 were as follows:

1. Dataset Size
    a. We combined the dataset used in LogReg2015 with an additional source of over 40,000 labeled recipes. By increasing the dataset significantly, a model is provided with many more training examples and is able to learn more information.
2. Vectorizer Parameter Optimization
    a. Initially, we started with a baseline sum count vectorizer of 7011 total unique unigram features, our logistic regression model achieved a testing accuracy of 79.0%.
    b. By enforcing a binary count (x_i = 1 if feature *i* appears in input, 0 otherwise), the vectorizer accuracy improved to 80.9%. By simplifying the training features to vectors with binary inputs, the logistic regression model was trained to classify based on the existence of ingredients and not their occurrence frequency in recipes, thus leading to better generalization for unseen recipes.
    c. Next, by filtering out our set of custom stopwords, we were able to reduce the noise of our training data and further improve classification accuracy to 82.7%. This was largely due to the removal of typos in noisy recipe data such as “ca”. Although not an actual ingredient, different recipes with “cake mix”, “carrot”, “catfish”, etc. would all share the “ca” feature and cause a misrepresentation of shared similarity.
    d. Lastly, by using both unigrams and bigram features[4], we were able to obtain a slight improvement to 83.3% classification accuracy. This is due to the fact that there are many important and frequently used two-word ingredients. For example, “soy sauce” is a staple ingredient in many Asian recipes. By using bigram features as well, our model learns to recognize ingredients such as “soy sauce” as an ingredient feature instead of two separate occurrences of “soy” and “sauce”.
3. Early Stopping in Model Training



    a. By plotting the learning curve of the logistic regression model, we discover that the training accuracy plateaus after 10 iterations while testing accuracy plateaus after around 8-9 iterations. The slight gap between training and testing accuracy at max iterations of 10 suggests that there may be an issue of overfitting.[5] As a result, we trained our logistic regression model with a max of 9 iterations. Forcing the model to stop early prevents the model parameters from being overtrained to the training data. As a result, our final logistic regression model improved to the final high of 83.4% classification accuracy.

Probabilistic Classification

Motivation

We began to look for ways to expand the practicality of our application in order to enhance user experience. Throughout our extensive user testing with dishes from popular recipe websites, we noticed a large number of fusion recipes, which are not easily classified into a single cuisine type. With the increasing ubiquity of fusion cuisine, we deemed it important to reflect this trend and implement changes to our model to output cuisine probabilities instead of a single binary classification. For each input of ingredients or recipe, our model would predict and output the percentage split across all the 20 cuisine classes (eg. 78% Chinese, 13% Korean, *etc.*). Such a modification would allow us to categorize recipes and express cuisine types with more versatility.

Baseline Model

In constructing our baseline model for the multilabel classification, we simply output the same, constant prediction across all inputs. The output vector has 20 values (the cuisine types) that sum to 1, where each value represents the percentage of recipes in the total training set for each cuisine type.

Logistic Regression Model

To create our probabilistic logistic regression classifier, we modified our non-probabilistic model. As previously discussed, the Logistic Regression model assigns a specific fractional weight to each cuisine class for given an input, and classifies an input based on the cuisine with the largest weight. By eliminating this final step, the modified model now outputs a one-dimensional vector of 20 percentage values. For each input, the model produces an array of 20 percentages, each value indicating the model's confidence of the recipe belonging to the corresponding cuisine.

Evaluation

A. *Related Work*

As a result of our new output type, we needed to implement a new evaluation metrics. In search for an existing evaluation method for probabilistic-classification tasks, we stumbled upon several relevant research papers[6][7]. After reading, we realized that the main challenge that we faced was the dependency on dataset labels being multi-labeled.

There are existing methods to evaluate multi-label classification models, such as hamming loss, 0/1 loss, log loss, and ranking loss, but they all involve comparative calculations across the different labels, with the assumption that the training/testing data has a correct and existing multi-label classification. Unfortunately, our dataset only has correct labels in the form of single-label classification. Even in an attempt to adjust for the difference in label type, our dataset only has labels in the form of vectors of 20 labels each where only one label value is 1, while the rest are 0. With such a vector, any of the aforementioned loss functions would not be well-suited for evaluation as the calculations for the labels with true value of 0 in our dataset would simply be overlooked or incorrectly penalized.

Since we were unable to find a grounded evaluation method that was well-suited for our specific combination of single-label dataset and multi-label model, we had to develop our own metric that utilizes comparisons of similarity across the cuisine types, inspired by methods utilized in *NLP*.

B. *Evaluation using Similarity*

In addition to the previously discussed issue of fusion foods, there also exists varying discrepancies between the different ethnicities in terms of similarity in culture, which inherently affects the similarity in ingredients utilized. Thus, rather than evaluating correct predictions with a score of 1 and misclassifications with a score of 0, we began to develop evaluation metrics that would allow consideration of those discrepancies by implementing a range in terms of accuracy for each output. For example, a ‘*Chinese*’ recipe mislabeled as ‘*Korean*’, being the most heavily weighed, can potentially be penalized less than a *‘Chinese’* recipe mislabeled as *‘French’*.

One way to achieve this is by constructing similarity distributions across vectorized ethnicity labels, where each pair of classes have a percentage value assigned (*e.g.* similarity(*‘Chinese’*, *‘Korean’*) = 0.675) that measure how alike two cuisines are. Thus, we can use these similarities values to assign a “score” to the prediction while allowing for misclassifications to be penalized differently with consideration of similarity/dissimilarity between certain cuisines. This “score” will lie within a continuous range from 0 to 1 rather than the traditionally binary value of 0 or 1 for correct/incorrect. With this, we are able to “reward” the predictions accordingly if the model predicts the correct label as the second highest value or if the model outputs a similar cuisine as the highest valued label. Additionally, rather than solely rewarding 1 for a correct classification, where the highest valued label is the true label, we are now able to proportionally “reward” the prediction based on how “confident” the model is in its prediction (60% *Chinese* vs 95% *Chinese*).

C. *Similarity Metrics*

Before the implementation of the following similarity measures, we first vectorized each cuisine class *c* into an array consisting of normalized (by total number of recipes in dataset with label *c*) counts of each ingredient.

- *Jaccard Similarity*

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad J(A,B) = \frac{\sum_{x \in \text{ingredients}} x \in A \text{ and } x \in B}{\sum_{x \in \text{ingredients}} x \in A \text{ or } x \in B}$$

  - Jaccard similarity considers cuisines as sets rather than vectors and returns set similarity with the use of simple AND/OR’s. We chose to implement this metric due to its logical approach, as we would consider cuisines to be more similar if they share a larger number of distinct ingredients with each other.

- *Generalized Jaccard Similarity*

$$\boldsymbol{GJ(A,B)} = \frac{\sum_{x\in\text{ingredients}} \min(\text{tf}(x,\boldsymbol{A}),\text{tf}(x,\boldsymbol{B}))}{\sum_{x\in\text{ingredients}} \max(\text{tf}(x,\boldsymbol{A}),\text{tf}(x,\boldsymbol{B}))}$$

  - The similarity measure above fails to take into account how often each ingredient appears in recipes with a certain cuisine label. To account for this, we can use Generalized Jaccard, an extension of Jaccard to weighted sets. For each ingredient x and each cuisine C, we compute the frequency weight tf(x,C) as the ratio between the number of times recipes with label CC contains xx, and the total number of ingredients used in CC.

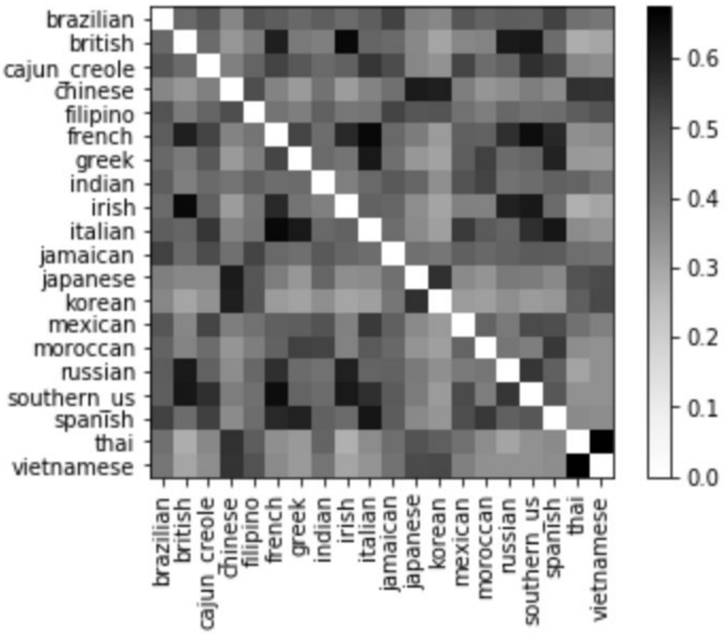
- *Cosine Similarity*

$$\boldsymbol{cossim(\vec{a},\vec{b})} = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\cdot\|\vec{b}\|}$$

  - Cosine similarity metric returns the normalized dot product of multi-dimensional two vectors. More specifically, it determines the cosine of the angle between the two class vectors. We chose to include this metric due to its efficiency as well as lack of consideration for the magnitude of the vectors, thus avoiding any potential noise from the variability in number of recipes, even if normalized. The use of the angle indicates "similarity" in terms of the closeness of direction of the vectors.

D. *Similarity Matrix*

Due to the uniqueness all metrics described above, where each exhibit their own strengths in evaluating similarity between cuisine classes, we decided to average the pairwise similarity values from the three metrics to obtain a combined similarity matrix. The following heatmap displays the resulting similarities between each pair of class (diagonals are set to 0.0 rather than 1.0 only for the heatmap to better display the range of similarity across the rest):

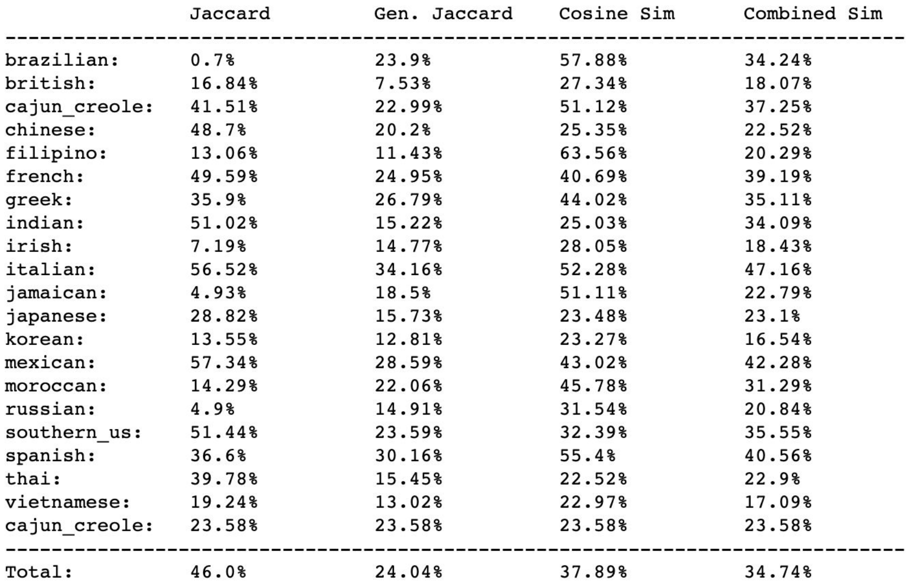


```
Average Similarity
==================
0.44172032731
```

In the evaluation process, we loop through each data point in the test set and perform the following calculations. First, we assign a “reward score” of $s_r = 0$ as accumulator to the current recipe *r*. Then, we take the actual label $c_a \in C$ of the recipe and loop over the 20 y_i’s, where y_i is the percentage/fractional value predicted by our model for cuisine $c_i \in C$. For each cuisine *i*, we update s_r += $[y_i \times similarity(c_a, c_i)]$. However, we only want to reward misclassifications that have above-average similarity to the actual label. As shown, the average similarity across all pairs of cuisines is ≈ 44.17%, so we set that as the cutoff and only update the accumulator and “reward” if $similarity(c_a, c_i) \geq 0.44172$. On the other hand, $similarity(c, c) = 1.0$. After the loop, we have an evaluation score within the range of 0 and 1, since y_i sum to 1 and all $similarity() \leq 1$. that reflects the accuracy of our model’s prediction. With the use of inverted indices, we are able to access the similarities from the matrix above in the described process.

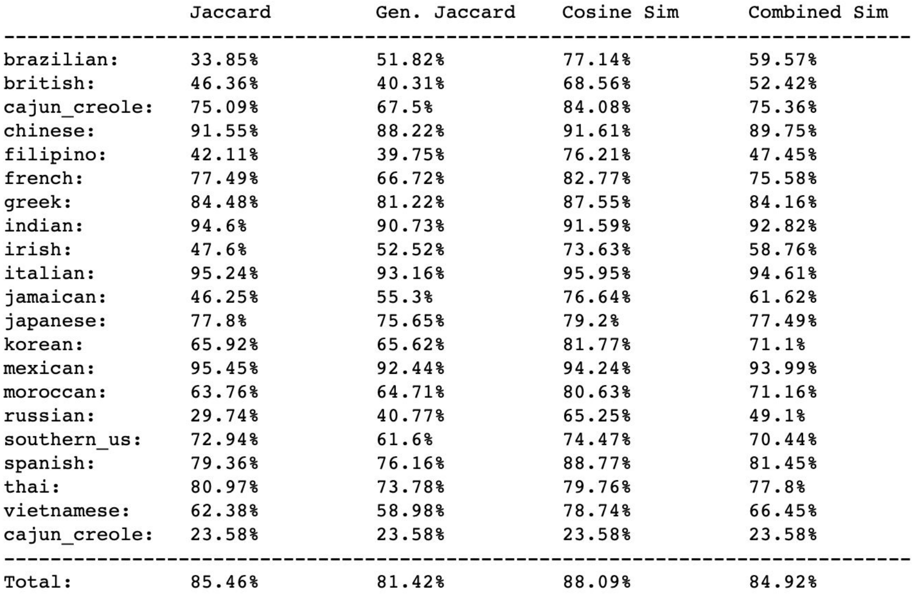
E. *Performances*

- *Baseline Model*



```
                Jaccard         Gen. Jaccard    Cosine Sim      Combined Sim
----------------------------------------------------------------------------
brazilian:      0.7%            23.9%           57.88%          34.24%
british:        16.84%          7.53%           27.34%          18.07%
cajun_creole:   41.51%          22.99%          51.12%          37.25%
chinese:        48.7%           20.2%           25.35%          22.52%
filipino:       13.06%          11.43%          63.56%          20.29%
french:         49.59%          24.95%          40.69%          39.19%
greek:          35.9%           26.79%          44.02%          35.11%
indian:         51.02%          15.22%          25.03%          34.09%
irish:          7.19%           14.77%          28.05%          18.43%
italian:        56.52%          34.16%          52.28%          47.16%
jamaican:       4.93%           18.5%           51.11%          22.79%
japanese:       28.82%          15.73%          23.48%          23.1%
korean:         13.55%          12.81%          23.27%          16.54%
mexican:        57.34%          28.59%          43.02%          42.28%
moroccan:       14.29%          22.06%          45.78%          31.29%
russian:        4.9%            14.91%          31.54%          20.84%
southern_us:    51.44%          23.59%          32.39%          35.55%
spanish:        36.6%           30.16%          55.4%           40.56%
thai:           39.78%          15.45%          22.52%          22.9%
vietnamese:     19.24%          13.02%          22.97%          17.09%
cajun_creole:   23.58%          23.58%          23.58%          23.58%
----------------------------------------------------------------------------
Total:          46.0%           24.04%          37.89%          34.74%
```

- *Logistic Regression Model*



```
                Jaccard         Gen. Jaccard    Cosine Sim      Combined Sim
----------------------------------------------------------------------------
brazilian:      33.85%          51.82%          77.14%          59.57%
british:        46.36%          40.31%          68.56%          52.42%
cajun_creole:   75.09%          67.5%           84.08%          75.36%
chinese:        91.55%          88.22%          91.61%          89.75%
filipino:       42.11%          39.75%          76.21%          47.45%
french:         77.49%          66.72%          82.77%          75.58%
greek:          84.48%          81.22%          87.55%          84.16%
indian:         94.6%           90.73%          91.59%          92.82%
irish:          47.6%           52.52%          73.63%          58.76%
italian:        95.24%          93.16%          95.95%          94.61%
jamaican:       46.25%          55.3%           76.64%          61.62%
japanese:       77.8%           75.65%          79.2%           77.49%
korean:         65.92%          65.62%          81.77%          71.1%
mexican:        95.45%          92.44%          94.24%          93.99%
moroccan:       63.76%          64.71%          80.63%          71.16%
russian:        29.74%          40.77%          65.25%          49.1%
southern_us:    72.94%          61.6%           74.47%          70.44%
spanish:        79.36%          76.16%          88.77%          81.45%
thai:           80.97%          73.78%          79.76%          77.8%
vietnamese:     62.38%          58.98%          78.74%          66.45%
cajun_creole:   23.58%          23.58%          23.58%          23.58%
----------------------------------------------------------------------------
Total:          85.46%          81.42%          88.09%          84.92%
```

In the actual evaluation, we created the above accuracy table to breakdown and better analyze the model's performance, such as where it succeeds or fails rather than if it succeeds or fails. The rows indicate percentage accuracies for recipes with the respective correct cuisine label in our dataset. The columns separate accuracies of the model across the three distinct similarity metrics used as well as the combined measure. For example, in the table for the logistic regression model, the 95.95% value at (*italian*, *Cosine Sim*) can be interpreted as the model's accuracy in predicting Italian recipes, using the Cosine Similarity metric in evaluation.

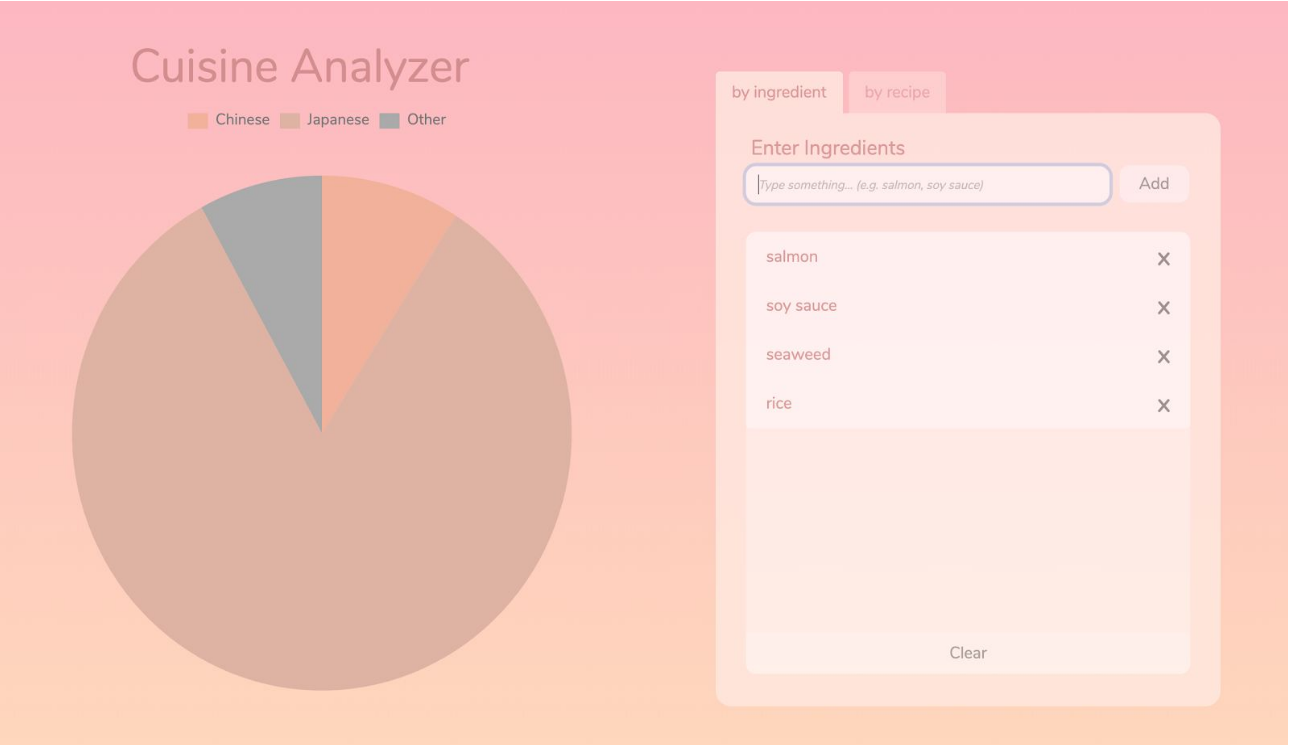
As shown, the logistic regression model far outperforms the baseline probabilistic model across the use of all similarity measures and all classes, as expected. Additionally, it is interesting to note that the accuracies increased not only greatly, but also proportionately from the baseline probabilistic model to the logistic regression model.

Further, we can observe that, for many of the classes, there exists variations across similarity measures for the same class. This difference can be attributed to the inherent difference in the similarity measures, as discussed earlier. The accuracy difference between evaluations using Jaccard and Generalized Jaccard stem from Gen. Jaccard taking into account term frequency of each ingredient and Jaccard not. Across the majority of classes, Cosine Similarity evaluates a consistently higher percentage. In creating the similarity matrix with Cosine Similarity only, we had noticed that the average closeness between classes were higher than the Jaccard measures. This pattern results from the characteristic of Cosine Similarity, which, rather than set similarity, considers the angular/directional closeness between cuisine vectors. It can be inferred that this property possibly allows ingredients that are both commonly shared across cuisines and necessarily used in majority of dishes, such as oil and salt, skewing the direction of the ingredient vectors in the respective dimensions of those ingredients, even after normalization. Thus, this recognizes many cuisines to be more closely related and thus "rewarding" misclassifications more often, which explains the overall higher percentage. However, for the entirety of the test set ("Total"), this difference is less noticeable across similarity measures, and the combined similarity column provides us a balanced view.

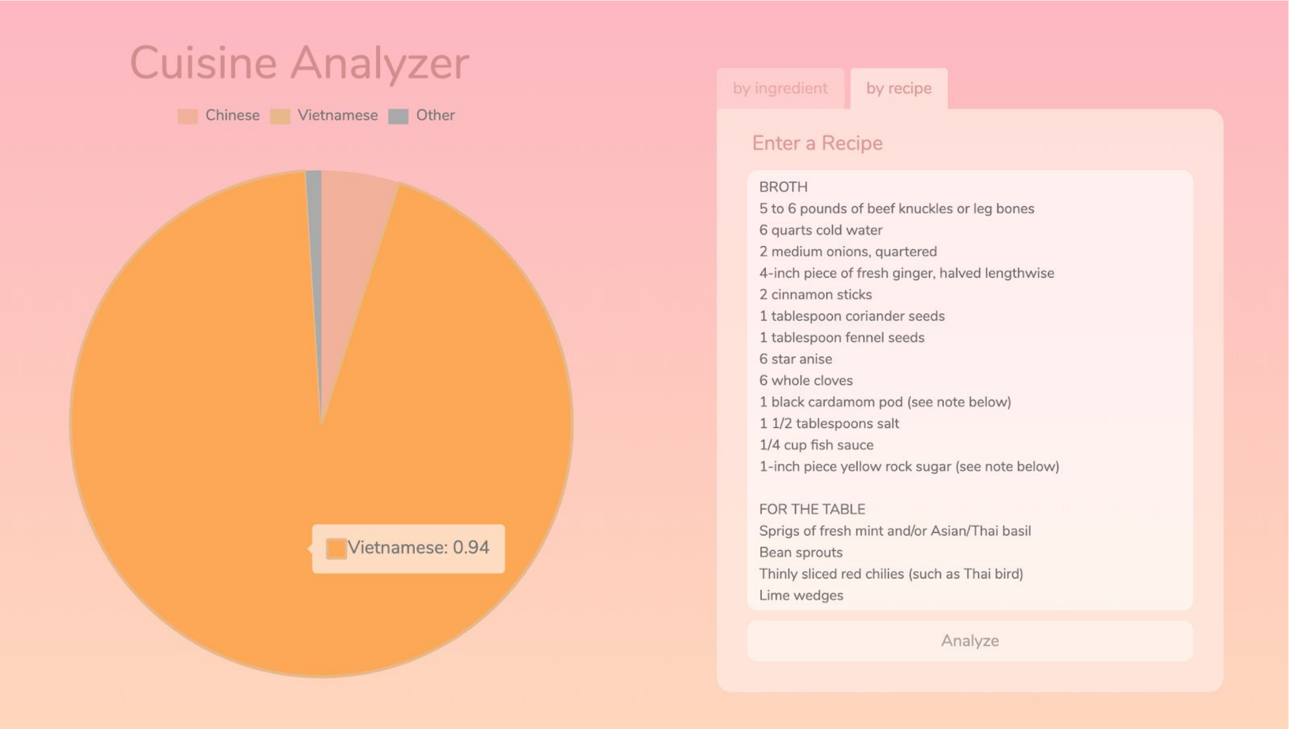
Lastly, in accordance with observations from non-probabilistic model evaluation, the results suggest that our model is more learned in classifying the majority cuisines. This is desired as, with an unfortunately heavily-imbalanced dataset, we would like to prioritize the classification ability of the more frequent classes. Thus, the "reward" mechanism in our custom evaluation using similarity measures does not deviate heavily from traditional eval previously done as it reflects similar patterns for the same model's performance. Similarly, the "Total" evaluation is only slightly higher than traditional evaluation on the model's non-probabilistic variant. This makes sense as the "reward" for misclassifications that are similar is balanced by the "penalization" for correct classifications that aren't very confident.

Web Application

To make our model user-interactive, we created a web-application using Flask, Bootstrap, and JQuery. Given text input, the application displays an interactive pie-chart representing the breakdown of percent match between input and cuisines determined by our model. The first page tab allows user-input to be an ingredient at a time, with the pie-chart dynamically updating each input. The second page tab allows user input to be an entire text block, allowing the user to classify any provided recipe.



salmon ×
soy sauce ×
seaweed ×
rice ×



BROTH
5 to 6 pounds of beef knuckles or leg bones
6 quarts cold water
2 medium onions, quartered
4-inch piece of fresh ginger, halved lengthwise
2 cinnamon sticks
1 tablespoon coriander seeds
1 tablespoon fennel seeds
6 star anise
6 whole cloves
1 black cardamom pod (see note below)
1 1/2 tablespoons salt
1/4 cup fish sauce
1-inch piece yellow rock sugar (see note below)

FOR THE TABLE
Sprigs of fresh mint and/or Asian/Thai basil
Bean sprouts
Thinly sliced red chilies (such as Thai bird)
Lime wedges

**recipe for Vietnamese Pho Noodles*

References

[1] Kaggle. Recipe Ingredients Dataset. https://www.kaggle.com/kaggle/recipe-ingredients-dataset.

[2] Min, Weiqing, et al. “You Are What You Eat: Exploring Rich Recipe Information for Cross-Region Food Analysis.” http://isia.ict.ac.cn/dataset/Yummly-66K.html.

[3] Li, Boqi, and Mingyu Wang. *Cuisine Classification from Ingredients*. 2015, cs229.stanford.edu/proj2015/313_report.pdf.

[4] Ifrim, Georgiana, et al. “Fast Logistic Regression for Text Categorization with Variable-Length n-Grams.” *ACM Digital Library*, 24 Aug. 2007, dl.acm.org/citation.cfm?id=1401936.

[5] “Logistic Classifier Overfitting and Regularization.” *Code Project*, 3 Oct. 2014, www.codeproject.com/Articles/824680/Logistic-Classifier-Overfitting-and-Regularization.

[6] Wang, Hongning, et al. “A Generative Probabilistic Model for Multi-Label Classification.” *A Generative Probabilistic Model for Multi-Label Classification*, 2008, www.cs.virginia.edu/~hw5x/paper/wang-MultiLabelClassification.pdf.

[7] Guo, Yuhong, and Wei Xue. “Probabilistic Multi-Label Classification with Sparse Feature Learning.” www.ijcai.org/Proceedings/13/Papers/206.pdf.