Where is the Chef From?

CS229 (Fall 2018) Final Report
Manish Pandit (manish7), Annie Pitkin (apitkin) & Hengkai Qiu (hq2128)
https://github.com/manishpandit/chef

Introduction

Between the unprecedented rise in human migration and the impact of globalization over the last two centuries, local cuisines around the world have fuzed. A chef's use of internationally infused ingredients has enhanced cuisines while retaining local identities. Our goal is to predict a recipe's country of origin given only a list of ingredients. In this multiclass classification problem we hope to gain insights into the factors and ingredients that distinguish a country's cuisine.

Related Work

This was a 2017 Kaggle competition. The public leaderboards top entry has 82.78% accuracy.

Dataset and Features

The public dataset is from the Kaggle competition, *What's for Dinner?* The data is provided in JSON format. Each example in the dataset contains the recipe identification, type of cuisine and a list of ingredients. The data consists of 39,774 unique recipes from 20 countries with 428,275 ingredients (6,714 unique). There are an average of 11 ingredients per recipe. Consequently, if we treat each ingredient as a feature we will end up with a sparse design matrix.

Feature Engineering

Several feature engineering challenges emerge from the fact that some of the ingredients are commonly used across multiple cuisines (for example, salt, oil and water). Figure 1 shows the top 10 ingredients found in the dataset across all cuisines. Figure 2 shows the distribution of recipes across various cuisines. The distribution is uneven; some countries are represented in higher volume compared to others.

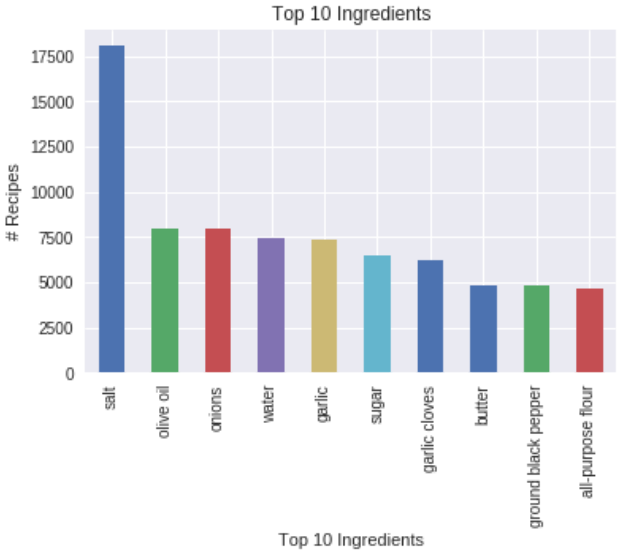

Figure 1

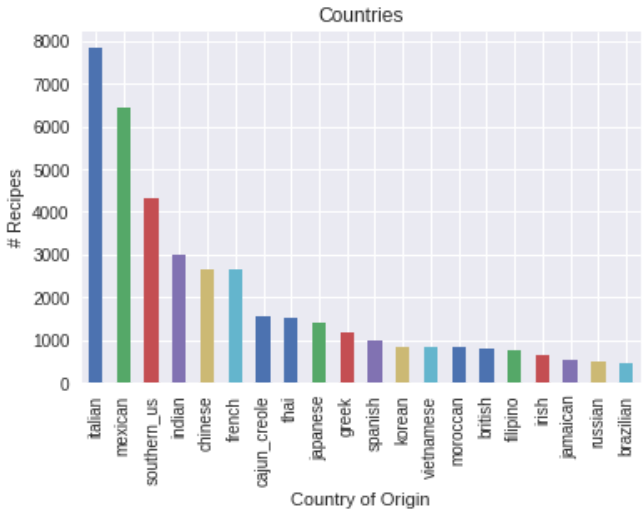

Figure 2

Figure 3 demonstrates the use of a particular ingredient (soy sauce) across various cuisines. This example shows a strong connection between a small set of countries and an ingredient.

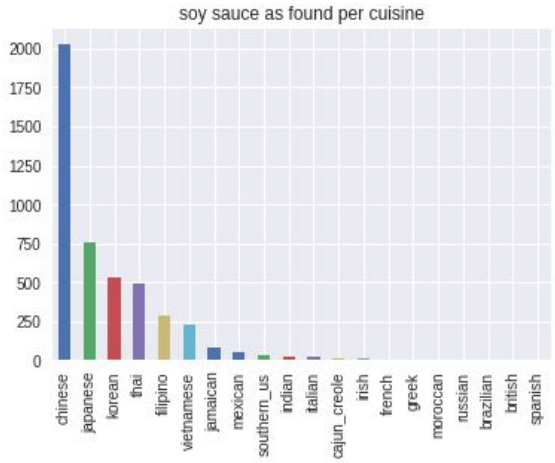

Figure 3

A high level analysis of the data reveals the necessity of data cleansing. The following list includes some examples of data clean up that we will address in subsequent sections:

1. Misspelled ingredient names.
2. Singular vs plural (i.e. onion vs onions).
3. Preparatory step included in the ingredient name (i.e. diced tomatoes vs chopped tomatoes).

Initial Design Matrix

We used Binary Encoding to extract the ingredients as individual elements and mapped the ingredients to dictionaries. The *(m, n)* design matrix employs a sparse representation of each unique ingredient. The matrix consists of zeros and ones to indicate if an ingredient exists in the recipe. In this design matrix *n* = 6700 (or the number of unique ingredients).

Second, we created a corresponding *(m, 1)* matrix with values ranging from 0-19 to represent the country of origin for each recipe in the dataset. Lastly, we split the data into 37,785 examples for training and 1,989 examples for test - this is a roughly 95% / 5% split.

Methods

We trained the training set on seven multi-class classification algorithms, optimized the model hyperparameters for some of the potential winners by Grid Search, performed cross validation check on the tuned models and then made predictions on the test data set. Subsequently we visualized the model performance through multiple evaluation matrices (Cross Validation Score, Testing Score and Confusion Matrix). We analyzed the errors and made the necessary adjustments to previous processes and structures. As shown in subsequent sections, the change from a Binary Encoding to TF-IDF Vectorizer design matrix was only applied after witnessing the TF-IDF design matrix resulted in much better testing accuracy.

Algorithms

1. Logistic Regression: binary classification algorithm using sigmoid function, i.e. $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. By using one-vs-rest (OvR) scheme and cross-entropy loss, we are able to solve multi-class problems.
2. Multiple Naive Bayes: successful classifier based upon the principle of Maximum A Posteriori (MAP). Given a problem with K classes with prior probabilities we can assign the class label to an unknown example with features $x = (x_1, \ldots, x_N)$ such that $c = argmax_c P(C = c \,||\, x_1, \ldots, x_N)$
3. Multi-layer Perceptron Classifier: Multi-layer Feedforward Neural Networks provide a natural extension to the multiclass problem.
4. Support Vector Machine: maximize the minimum distance from the separating hyperplane to the nearest example. We used ‘linearSVC + one-vs-rest(OVR) scheme’ and ‘SVC + one-vs-one scheme’ to solve multi-class problems.
5. Passive Aggressive Classifier: family of online learning algorithms. Similar to Perceptron except PA Classifiers do not require a learning rate. However, contrary to the Perceptron they include a regularization parameter *C*. We used ‘squared_hinge’ as loss function.

6. Decision Tree Classifier: non-parametric supervised learning method used for classification and regression, the goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
7. Random Forest: meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default)
8. *k-Nearest Neighbors: measures the distance from an example to every other training example, identifies k smallest distances to each class and outputs class label based on the most represented class in these k classes. Since running this algorithm with sparse features becomes very time-consuming after a couple of trials we did not implement this algorithm.

Preliminary Experiments

To train this model we used Binary Encoding to create the design matrix and implemented a neural network using the Tensorflow and Kera packages in Colab using Google's TPU. Initially we implemented a simple one layer neural network outlined below to output a baseline model and understand the data.

Baseline Neural Network 5 iterations with Sparse Cross Entropy loss function			
Input Layer (*m, n*) list of ingredients →	**Hidden Layer** 1024 neurons ReLU activation function 20% dropout rate (regularize and prevent overfitting) →	**Output Layer (*n, 1*)** 20 neurons Softmax activation function Optimizer = adam Mini batch gradient descent →	**Prediction** country with the highest probability

The results were very encouraging but not optimal. The training accuracy of the NN was approximately 97% and the test accuracy was roughly 78%. The model overfitted the training data. Our initial error analysis revealed:

1. Data cleanup opportunity. In particular, remove extra words that do not add value to learning from ingredients. These extra words erroneously make non-unique ingredients appear distinct.
2. Tune the NN and address the high variance.
3. Apply additional multiclass classification algorithms for comparison.
4. Rethink about Binary Encoding. We felt that this was not the optimal way to extract text data from documents, and also ideally we would like to reduce the number of features so that it will be more efficient for algorithms like SVM.

Further Experiments

With a foundational understanding of the data we shifted our focus to cleaning the input data. We concentrated on stemming the words, such that 'olives' would become 'olive' since the plural should not lead to two different ingredients. Similarly, words like 'low-fat' and 'low fat' should have the same meaning. After these preprocessing steps we tested with few algorithms, but this instead reduced our testing score by 1-2%. Ultimately we decided to put the data cleaning aside for now and moved on toward redesigning the design matrix.

Design Matrix Revisited

We implemented TF-IDF vectorizer to extract a bag of words from the recipe data by setting stop_words to 'english' and binary to True. Second, derived the *(m, n)* TF-IDF design matrix which returns a normalized count of ingredients based on how many times an ingredient appears across all recipes. With

the TF-IDF vectorizer *n* = 3000. This is much less than the *n* value found in binary encoding. We used TF-IDF because we figured having an indication of the frequency of a certain ingredients would provide additional information as opposed to simply Binary Encoding.

Results

When we began this problem we did not pre-process any of the data. Our initial results had accuracies ranging from 64% to 79%. After analyzing the errors we tried to homogenize the ingredients but found it did not help performance. Then we implemented the TF-IDF Vectorizer, tested again and gained 0.5% to 1.0% improvement on the testing score. This increase to a testing score of 81% moved us within top 50 on Kaggle public leaderboard. Finally, we decided to tune our model and ran a grid search over the hyperparameters, especially the regularization parameters. We received our best testing score of 82% from the Support Vector Machine. This result is within the top 10 on the Kaggle public leaderboard.

We used a Support Vector Machine with C =10, gamma =1 and kernel = rbf to obtain our highest accuracy of 82%. The logistic regression classifier and the multilayer Perceptron neural network both produced accuracies of 80%. The passive aggressive classifier using a squared hinge loss function resulted in 79% accuracy. With multiple naive bayes and a smoothing parameter of 0.1 we achieved 74% accuracy. The decision tree and random forest with information gain entropy had the lowest accuracy of 64% and 66%, respectively. These scores are reflected in Table 1 below.

Algorithm	*Initial Accuracy on Test Set*	*HP Opt*	*CV #*	*CV Score on Training Set*	*Final Accuracy on Test Set*
MLP Neural Net	*77%*	*Yes*	*-*	*-*	*80%*
Logistic Reg	*79%*	*Yes*	*5*	*79%*	*80%*
SVM	*79%*	*Yes*	*5*	*81%*	*82%*
Decision Tree	*64%*	*No*	*-*	*-*	*-*
Passive Agg	*74%*	*Yes*	*5*	*75%*	*75%*
Random Forest	*66%*	*No*	*-*	*-*	*-*
Multinomial NB	*69%*	*Yes*	*5*	*73%*	*74%*

Table 1

We quantified and analyzed various performance metrics including accuracy, precision, recall, f1-score , support and confusion matrices. Figure 4 is a confusion matrix of classification results from a logistic regression model.

Confusion Matrix

We evaluated the classification accuracy by computing the confusion matrix. Each row corresponds to the true cuisine label. We normalized the results by dividing by the number of recipes for each cuisine in the test data. The diagonal elements represent the proportion of samples for each cuisine whose predicted label was equal to the true label, while off-diagonal elements were mislabeled by the classifier. In other words, the higher the diagonal values of the confusion matrix the better since this indicates a greater number of correct predictions.

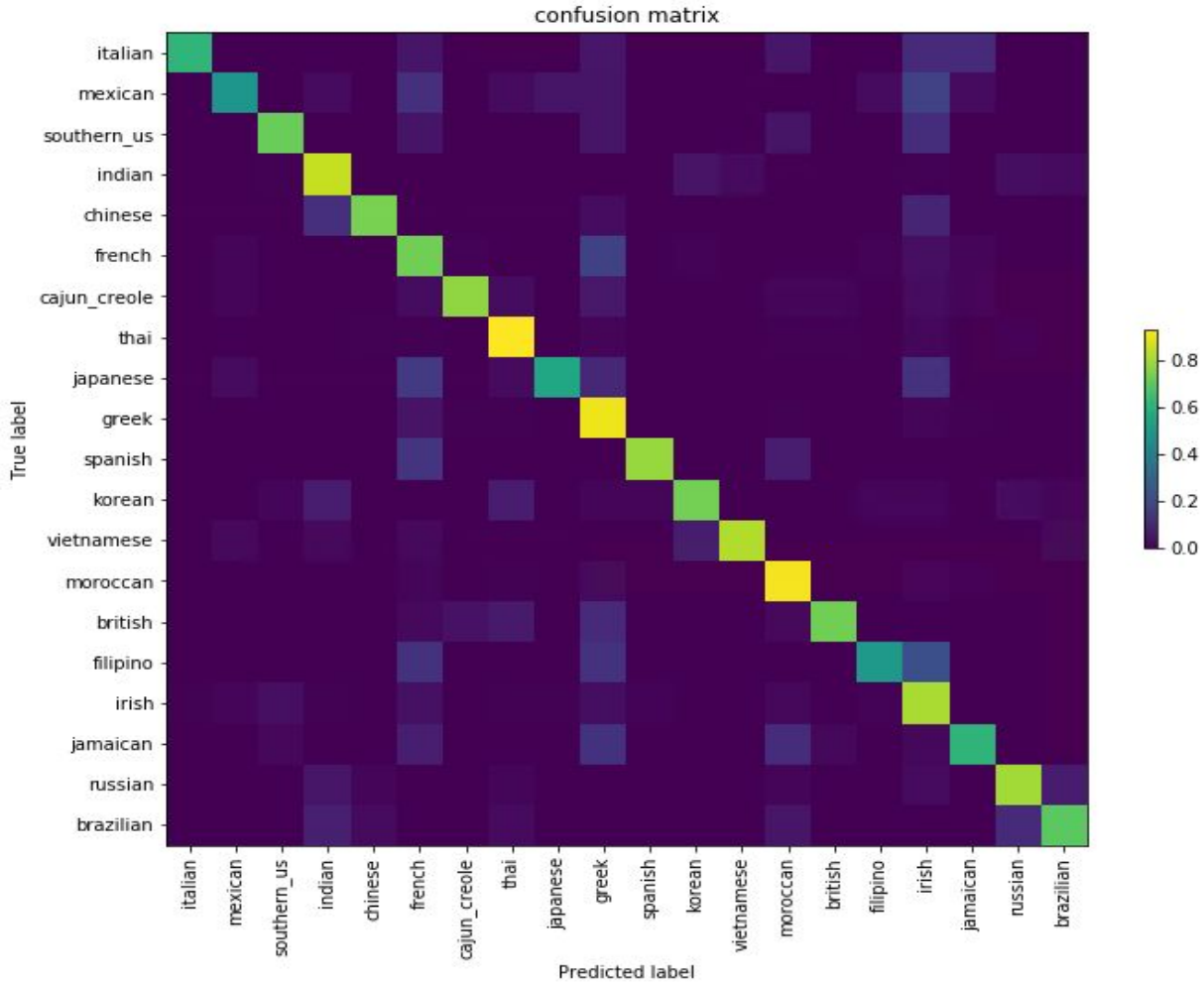

Figure 4

One of the key observations from analysis was the similarity in accuracy scores between training performance and test performance. This indicates a relatively low variance. To put it another way, the model was not overfitting. This observation lead to our questioning ways to further reduce the bias. We considered two options:

1. Extended feature vectors.
2. Hyperparameter optimization.

We extended our feature vector considerably through collecting additional player statistics as mentioned in the feature engineering section. We also applied grid search over various hyper parameters on several of our models.

Future Work

1. We used a TF-IDF to extract a bag of words by setting stop_words to a sentinel, but ideally we could analyze the recipe data and create our own list of stop words in order to extract a bag of ingredients. This would be a good next step to improve our results. Another avenue of further work would be to use stemming and lemmatization to reduce inflectional forms.

Contributions

Manish Pandit: Research, analysis, coding, and documentation.
Annie Pitkin: Research, analysis, coding, and documentation.
Hengkai Qiu: Research, analysis, coding, and documentation.

References

1. *Multiclass classification. https://en.wikipedia.org/wiki/Multiclass_classification*
2. *Softmax function: https://en.wikipedia.org/wiki/Softmax_function*
3. *Naive Bayes Classifier: https://en.wikipedia.org/wiki/Naive_Bayes_classifier*
4. *Mohamed, Aly (2005). "Survey on multiclass classification methods" (PDF). Technical Report, Caltech.*
5. *Kaggle Challenge - What's for dinner? Public leaderboard: https://www.kaggle.com/c/whats-cooking-kernels-only/leaderboard*
6. *Kaggle dataset:* https://www.kaggle.com/kaggle/recipe-ingredients-dataset/home