What to Cook and What Else to Cook?

Kenneth Lee [1] **Roger Zhou** [1] **Libin Feng** [1] **Ying Cheng** [1]

Abstract

We explore the potential of creating new recipes via text data. Our goal has two folds. First, we aim to classify the cuisine based on ingredients. Second, we want to predict an ingredient that is missing from a given list of ingredients and a cuisine name. The first task can be formulated as a multi-class classification problem. To convert the text into numerical signals, we can use TFIDF vectorizer, CountVectorizer, word embedding based on Word2Vec model. We are able to achieve 0.85 micro-averaged F1 scores for the multi-classification task with multilayer perceptrons and bag-of-words model. The second task can be viewed as a recommender system. We adopt two approaches, vectorizers and standard text processing, in working with the recipe text, followed by examining the recommendation performance under each approach. For the vectorizers, we compare the recommended ingredients; under the "text_preprocess" method, the collaborative filtering model results in a much better "top n accuracy" metric than the baseline popularity model.

1. Introduction

With the increasing number of recipe sharing channels, more cooking recipes associated with ingredients and cooking procedures are made available online. Many recipe sharing platforms have been developing recipe recommendation mechanism. One important aspect is to recommend cuisine based on selected ingredients.Recently, it has been shown that cuisine types can be automatically classified based on ingredients (Ghewari & Raiyani, 2015; Su et al., 2014). Such knowledge can be applied to building recommender system (Kuo et al., 2012; Teng et al., 2012). Moreover, the recent advance in deep learning gives rise to a more sophisticated approach to categorizing cuisines and recognizing ingredients concurrently (Chen & Ngo, 2016). Thus, we will compare how various machine learning approaches perform on classifying multiple cuisines and explore methods on building a recommendation system. Section 2 outlines the methods we undertake to approach the multi-class classification problem as well as the recommender system. The result is then provided in Section 3 along with insights we gained.

[1]Department of Statistics, University of California, Davis, CA, United States. Correspondence to: Kenneth Lee <honlee@ucdavis.edu>, Roger Zhou <luzhou@ucdavis.edu>, Libin Feng <lbfeng@ucdavis.edu>, Ying Cheng <ygcheng@ucdavis.edu >, Github repository <https://github.com/kenneth-lee-ch/whattocook>.

2. Methods

2.1. Data Collection

The dataset is from a competition named "What's cooking" on Kaggle, an online data science community. The data has 39774 rows and 3 columns (id, cuisine, ingredients)

2.2. Exploratory Data Analysis

We explore the dataset as follows:

- Examine whether it is an imbalanced in terms of the distribution of the cuisine.
- Count the number of occurrence for each ingredient in the data and rank them.
- Check the distribution of the number of ingredients.
- Find out which cuisine type usually have a large number of ingredients.

2.3. Data Preprocessing

We then preprocess the ingredient names in the following ways:

1. Remove all the "(.oz)" description from the ingredient name.
2. Change all the ingredient names to lowercase.
3. Lemmatize each word.
4. Deaccent each word.
5. Remove any non-letter from the text.

2.4. Cuisine Classification

2.4.1. Vectorization

Next, we transform the ingredient texts by using some bag-of-word models such as TFIDF vectorizer, CountVectorizer, and a pretrained word embedding model called Word2Vec.

CountVectorizer: It generates a matrix where each column represents a unique ingredient name in the data and each row represents one training sample. Each entry is filled by counting the number of occurrence of the ingredient corresponding to its column.

TFIDF Vectorizer: It is similar to CountVectorizer, but it calculates a TFIDF statistic, tfidf(S, i, D), based on the entire dataset and the document in which the word occurs. Each list of ingredients is represented by a vector using each statistic as an element in the vector. The TFIDF statistic tends to be low for words that have higher frequency. Let D be the dataset, S be the list of ingredients in one sample, i be the name of the ingredient. The statistics is given by (3).

$$\text{tf}(S, i) = \frac{\text{word count in the sample}}{\text{total number of terms in the sample}} \tag{1}$$

$$\text{idf}(i, D) = \frac{log(\text{number of samples})}{\text{Number of samples with this word}} \tag{2}$$

$$\text{tfidf}(S,i, D) = \text{tf}(S, i)\text{idf}(i, D) \tag{3}$$

Word2Vec: It is a learned real-valued vectorial representation for a set of predefined vocabularies. It turns each ingredient in each sample to a 300 dimensional vector. We then sum all the vectors within the sample and divide each entry by the number of elements in that sample to create an embedding matrix.

2.4.2. Feature Extraction

- Comparison between PCA and t-SNE
  - Afterwards, we use both Principal Component Analysis (PCA) and T-distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimensionality of the vectorized data down to 2 components. We then compare the clusters visualizations between PCA and t-SNE.
- K-means Clustering for mapping labels
  - We then apply K-Means clustering to the vectorized data and output 20 components and map them to the true labels of cuisines to see if K-Means recovers a similar number of data points for each cuisine.

2.4.3. Evaluation Metrics

Since classifying cuisines is a multi-class classification problem and the dataset has an imbalanced distribution for two output classes, precision and recall will be a more informative metric than the Receiver Operating Characteristic (ROC) curve to evaluate the classifiers we use in this work (Saito & Rehmsmeier, 2015). More specifically, we will use micro F1 score to measure the model performance as we are only interested in maximizing the number of correct predictions the classifier makes.

Precision and Recall: As shown by (4), precision is computed as the ratio of the number of true positives divided by the sum of the true positives and false positives. It measures how precise our model is at predicting the positive class.

Recall is a ratio of the number of true positive divided by the sum of true positive and false negative as shown by (5). It measures how good our model is at predicting the actual positive class.

One way to measure the robustness of the classifiers is to calculate their micro F_1, scores and compare them. In the multi-class classification setting, we consider all classes when computing precision and recall via micro-averaging. We decide to use micro-averaged F_1, which is the harmonic mean of micro averaged precision and micro averaged recall. We use this metric because we want to take the small number of classes into account and rank our classifier based on how well they can classify those small number of some cuisines types in an imbalanced data setting.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \tag{4}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \tag{5}$$

2.4.4. Model building and Implementation details

We choose to compare logistic regression (LR), decision trees (CART), random forest (RF), linear discriminant analysis (LDA), Adaboost (Ada), K-nearest neighbours (KNN), Gaussian Naive Bayes (NB), and multilayer perception (MLP). We briefly describe these models below. We select these methods because we want to have good combination of linear, non-linear classifiers, bagging and boosting methods for comparison. Support Vector Machine (SVC) was considered once but it requires heavy computations which is difficult on home PCs, so we put aside this model this time.

- CART (Non-Linear): Classification based on the values of the available features.
- RF (Bagging): Averaging out of multiple decision trees

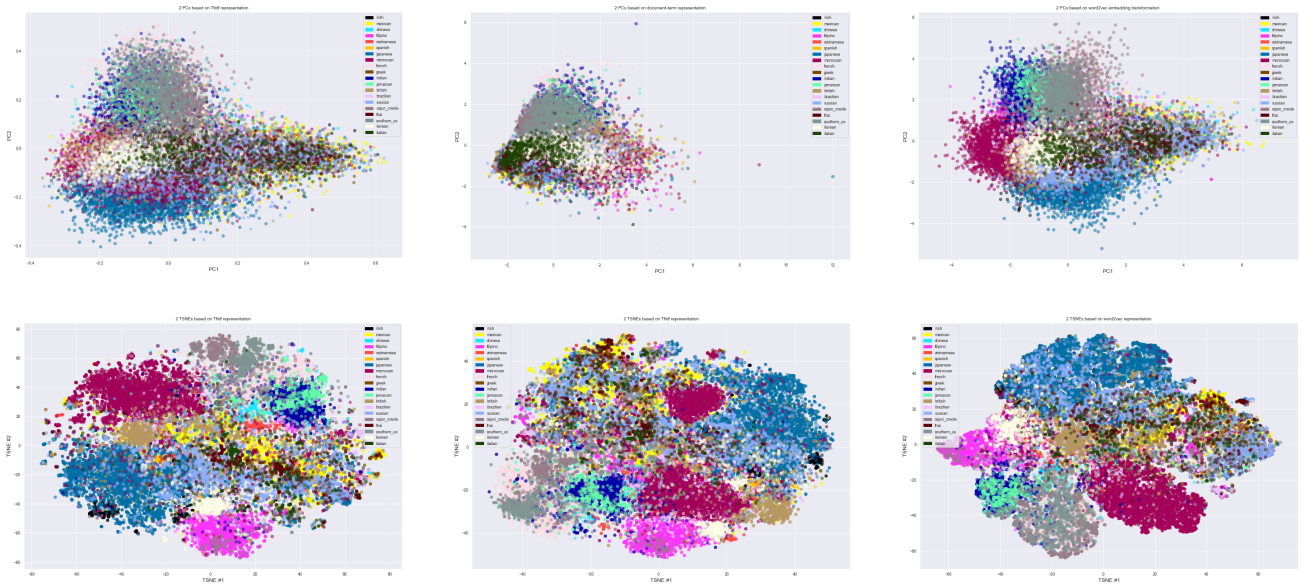

Figure 1. PCA vs. t-SNE in visualizing various cuisines: The top row shows 2 principal components, the bottom row shows 2 t-SNE components. From the left to the right, it shows three different types of vectorized text data: TFIDF representation, document-term matrix, embedding based Word2Vec model. We see that the clusters are shown more clearly by t-SNE, especially for the embedding based on Word2Vec model. A normalized representation e.g. TFIDF representation tend to have sparse data points on the graphs (top left) as compared to the PCA result from document-term matrix (top middle).

to solve overfitting problems of decision trees.

- KNN (Non-linear): Classification based on distance and majority votes.
- LDA (Linear) : Find a linear combination of features such as density functions that separates multiple classes.
- LR (Linear) : Modeling the probability of certain classes (one vs. rest in multi-class scenario).
- MLP: Using input layers, at least one hidden layers and output layers with multiple nodes and different activation functions to construct ANN and find weights that minimizes the loss.
- NB (Non-linear): Classification based on conditional probability and normal distribution assumptions.
- Ada (Boosting): Converting a set of weak classifiers into a strong classifier.

The vectorized data was randomly shuffled and split into 70% training and 30% test set. We conduct 5-fold stratified cross-validation on the training set to spot check each of these models with default parameters set by scikit-learn packages in Python besides LR and MLP. We implement logistic regression with the parameter "multiclass" being set to "ovr", which denotes one vs. rest.

Then, we continue to fine tune some of the promising algorithms by conducting a grid search with 5-fold stratified cross-validation, again on the training set.

In addition, to come up with a MLP model, we likewise utilize the same cross-validation strategy as described above. We use Adam optimizer, sparse categorical cross-entropy loss, 10 epochs (due to excessive amount of training time), and set the output layer with softmax activation and 20 units. Then, we fine-tuned the following components:

- **Weight initialization**: We set "kernel_initializer" to be either he-uniform or normal.
- **Number of additional hidden layers**: 2 and 3.
- **Activation functions**: for the first hidden layer, we have tried using ReLU, sigmoid, and softmax.
- **Number of nodes**: We have tried using 20 and 30 units for each hidden layer.
- **Batch normalization**: We have also tried adding batch normalization layer in between hidden layers.

2.4.5. Micro-averaged precision score on Test Set

After we finished tuning some promising classification models, we fit eight models again using the respective training

set that produces the highest scores in the tuning part. For each model, one vs. rest strategy is used to produce the micro-average precision score. Note that precision is the same as F1 score in micro-averaging in a multi-class classification setting.

Also, we use the whole training data set to fit the model and predict on the 30% test set rather than using 5-fold cross-validation solely based on the training set, thus the result of micro average precision will be different from the micro F1 score in the spot-check part. The micro-averaged precision-recall curves are plotted based on the calculated predicted probabilities or decision functions (scores) using the test data set.

2.5. Recommender System

In the second task, we aim to build a recommender system (RecSys), with the goal of recommend new ingredients based on a given recipe, i.e. a list of ingredients. Our methods will return a ranked list of recommended ingredients for the user to choose from.

2.5.1. RecSys based on normalized CountVectorizer and normalized TFIDF vectorizer

By using a normalized vectorizer, we represent all recipes and the given ingredients in the form of unit vector. Both CountVectorizer and TFIDF vectorizer are used. Then we calculate the inner product between the vector of given ingredients and all recipes to obtain the similarities. We find the recipe with the largest similarity with the given ingredients and recommend the ingredients of that recipe. We use recommendation index to indicate how much we recommend an ingredient. For a recommended ingredient, we count how many times it occurs together in a recipe with each given ingredient and sum them up. We define this total number of simultaneous occurrences in recipes of a recommended ingredient and the given ingredients as the recommendation index. The higher the recommendation index is, the more we recommend an ingredient.

2.5.2. RecSys based on text_preprocess

Instead of transforming the corpus into sparse matrices, the "text_preprocess" method that we developed allows us to work with the recipes directly so that we can address the problem of splitting everything up into single words and recommending ingredients that do not make sense. Therefore, in the second approach to the RecSys, after splitting the corpus into training and testing corpuses, we will construct models on the training corpus and evaluation metric based on the test corpus, all under the "text_preprocess" method. Note that in this approach, we always tell our model to ignore the last ingredient in the recipe in order to facilitate the recommendation as well as the "top n " calculation (as defined below).

- **Top n Accuracy:** To evaluate the performance of the model, we define the top n accuracy calculation, where we randomly select 50 ingredients that are not in the recipe given by the user, and ask the model to rank them combined with an actual ingredient from the recipe (i.e. the last ingredient). We set $n = 5$ throughout this task. In the end, we compute the percentage the model successfully rank the actual ingredient within the top 5 of the 51 ingredients (Recommender Systems in Python 101).
- **Popularity Model:** As a baseline measure of RecSys performance under the "text_preprocess" method, we build a model that summarizes the number of occurrences of each ingredient in the list (of ingredients to be ranked) in the training recipes, and ranks the ingredients from the most popular to the least popular.
- **Collaborative Filtering Model:** This model takes advantage of the similarity between the recipes and makes recommendations only based on the recipes similar to the recipe given by the user. Specifically, the similarity is defined by *cosine similarity*, which is also known as the Pearson correlation, between 2 recipes. Of course, to establish the calculation of cosine similarity, we need to expand the recipes into recipe vectors/matrices (i.e. a matrix with mostly 0's and some 1's corresponding to the ingredients present in the recipes). Once the preparation is completed, we will fetch the $1,000$ recipes that are the most "cosine similar" to the user-specified recipe, and then rank the list of 51 ingredients based on their occurrences in these $1,000$ recipes.

3. Results and Discussion

3.1. Cuisines Classification

3.1.1. Exploratory Data Analysis

Based on exploratory data analysis, we observe the following:

- This is an imbalanced data set. Italian cuisine has the highest number of occurrence in the data set, whereas Brazilian has the least number of occurrence.
- Salt, olive oil, onion, water, and garlic are the top 5 ingredients by occurrence in the data.
- Most cuisines have around 10 ingredients in the data.
- Most cuisines that have over 30 ingredients are Mexican and Indian food.

- When we reduce TFIDF representation of the ingredients down to 2 principal components, we found that sugar and sauce are the most loading based on the largest absolute values.

3.1.2. Visualizing cuisines based on PCA and t-SNE

As shown by figure 1, we see that the clusters are shown more clearly by t-SNE, especially for the embedding based on Word2Vec model. This is probably due to the learned representation from the pretrained word embedding Word2Vec and the underlying ingredients structure among all the cuisines. We have also noticed that the data points on the graph for plotting two principal components are more sparse when the data has been normalized.

3.1.3. K-Means Clustering for Mapping Labels

We see from figure 2. that most data points get mapped to Italian, Chinese, Indian, and Mexican. It is expected because the data is imbalanced and Italian, Chinese, Southern US, Indian, and Mexican are among the top five cuisines in the data in terms of the number of samples and K-Means performs well on this dataset. We did not present all the confusion matrices based on other types of vectorization because they present a similar result.

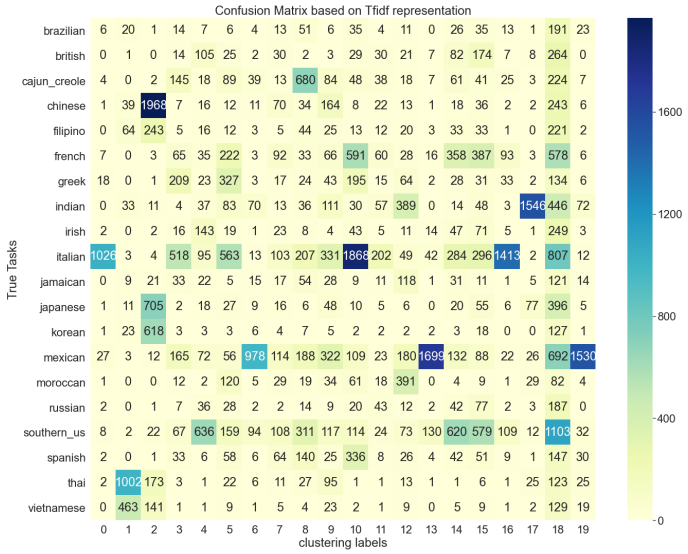

Figure 2. Confusion matrix for K-Means Clustering mapping labels based on TFIDF representation. The higher the number it is, the darker blue the color will be.

3.1.4. Model Performance

We summarize the spot-check result in Table 1. We observe the following:

- We see that logistic regression consistently obtain the highest micro F1 score in the representation based on bag-of-words model.
- KNN usually performs better on normalized data setting.
- Interestingly, naive Bayes has the lowest score for both TFIDF and document-term matrices. One potential reason can be due to the feature independence assumption behind naive Bayes, which is not the case for the data we provide.
- We also see that Random Forest consistently perform better than the decision tree classifier. This is typical because RF consists of multiple single trees with each based on a random sample of the training data to provide more stable and accurate result.
- We can see that the bag-of-words models (TFIDF representation and document-term matrix) provide better model performance as they preserves more information than the embedding matrix we built on Word2Vec model.
- For neural network models with bag-of-words vectorization, using ReLU activation as first activation function while adding batch normalization layers yields higher precision scores.
- For neural network models with Word2Vec vectorization, using sigmoid activation as first activation function while not using batch normalization yields higher precision scores.
- Increasing width of layers tend to increase the performance of the MLP models. In this case, 30 nodes for each layer in each MLP model yields higher precision scores than using 20 nodes for each layer. However, it also requires longer training time.
- The number of epoch is only set to 10 due to the excessive amount of training time. However, MLP still performs well on this task. We strongly believe that increasing the number of epoch can be beneficial, further investigation is needed. All MLP models obtained higher precision scores by using "he_uniform" over normal as the weight initialization distribution. Further try is need to see if other weight initialization methods have better performances.

Model\Data	TFIDF	Document-Term Matrix (DTM)	Normalized DTM	Word2Vec	Normalized Word2Vec
LR	**0.769**	**0.778**	0.748	0.623	0.596
LDA	0.748	0.736	0.693	**0.683**	**0.665**
KNN	0.724	0.626	0.693	0.608	0.616
CART	0.608	0.632	0.610	0.515	0.523
NB	0.274	0.273	0.279	0.486	0.497
RF	0.685	0.695	0.678	0.612	0.620
Ada	0.526	0.546	0.519	0.474	0.483
MLP	0.758	0.760	**0.758**	0.636	0.640

Table 1. Model Comparison based on spot-check (without tuning parameters except for MLP) : Logistic Regression (LR), linear discriminant analysis (LDA), K-nearest neigbhors (KNN), Decision Tree Classifier (CART), Naive Bayes (NB), Adaboost(Ada), multilayer perceptron (MLP). These scores are based on micro-F1 score.

Model	Data	AP
LR	DTM	0.84
LDA	TFIDF	0.75
KNN	TFIDF	0.79
CART	DTM	0.45
NB	Normalized Word2Vec	0.22
RF	DTM	0.83
Ada	DTM	0.76
MLP	DTM	**0.85**

Table 2. Validation and micro-average precision score for Logistic Regression (LR), linear discriminant analysis (LDA), K-nearest neigbhors (KNN), Decision Tree Classifier (CART), Naive Bayes (NB), Adaboost(Ada), multilayer perceptron (MLP).

Then, we fine-tuned only those algorithms that show promises based on our initial spot-check results. These models include LR, LDA, and KNN. However, the performance of all of these algorithms do not improve significantly. LR still remains to have 0.778 score based on document-term matrix, while LDA also remains 0.748 based on TFIDF. After tuning KNN, we obtain 0.737 score based on TFIDF representation with the number of neighbors be set to 15.

For the micro-average precision-recall curves (see jupyter notebook) and the micro-average precision score(AP), LR(0.84) and MLP(0.85) has the highest AP among eight models, which is consistent with the spot check precision scores in the ranking but those two APs are ≈ 0.1 higher than the spot check results respectively. On the other side, the AP of KNN(0.79) happen to be higher than LDA(0.75) and the AP for RF is 0.14 higher than spot check results. Naive Bayes model did not yield desired result(≈ 0.48) based on Word2Vec matrix but got low AP similar to spot check precision scores with DTM or TFIDF matrices. In all, these APs are different from the spot check scores for most of the models.

3.2. Recommender System

3.2.1. Recommender based on normalized CountVectorizer and normalized TFIDF vectorizer

We randomly generate a list of ingredients and apply the proposed recommender system. Our results indicate that the recommender based on normalized CountVectorizer and normalized TFIDF vectorizer sometimes give similar recommendations and sometimes give very different recommendations. Table 3 shows one example.

3.2.2. RecSys based on text_preprocess

We split the recipe corpus into 80% training corpus and 20% test corpus. Following the "top n accuracy" metric defined in section 2.5.2, we have obtained the below results:

- The popularity model returned a "top n" of 0.8068, meaning that over 80% of the time, the popularity model ranks the actual ingredients among the top 5 of the list of 51 ingredients.

- The collaborative filtering model returned a "top n" metric of 0.9863, which is significantly higher than the popularity model. However, the evaluation process took dramatically longer because of the higher computation complexity.

We have noticed a trade-off in the construction of the collaborative model. We decided to use $1,000$ recipes that are the most similar to the user-specified recipe. While this choice is ad hoc, when we increase the size of the neighborhood around the given recipe, we are introducing bias to the cuisines that are small in numbers. In other words, we are bound to use recipes from other cuisines to recommend ingredients for the recipe in which the user is interested, when the neighborhood size becomes larger.

Given ingedients	Recommended ingredients based on normalized Countervectorizer	Recommendation index	Recommended ingredients based on TFIDF vectorizer	Recommendation index
korma	cream	1723	salt	5185
hatcho	peanut	843	pepper	3716
catsup			butter	2717
cant			milk	1952
raw			parsley	512
knockwurst			potato	350
dungeness				
rice				
scotch				
young				
links				
chocolate				
treacle				
turnip				
vanilla				
deli				
kipper				

Table 3. An example of the recommender based on normalized countervectorizer and normalized TFIDF vectorizer

References

Chen, J. and Ngo, C.-W. Deep-based ingredient recognition for cooking recipe retrieval. In *Proceedings of the 24th ACM international conference on Multimedia*, pp. 32–41, 2016.

Ghewari, R. and Raiyani, S. Predicting cuisine from ingredients. *University of California San Diego*, 2015.

Kuo, F.-F., Li, C.-T., Shan, M.-K., and Lee, S.-Y. Intelligent menu planning: Recommending set of recipes by ingredients. In *Proceedings of the ACM multimedia 2012 workshop on Multimedia for cooking and eating activities*, pp. 1–6, 2012.

Saito, T. and Rehmsmeier, M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10 (3):e0118432, 2015.

Su, H., Lin, T.-W., Li, C.-T., Shan, M.-K., and Chang, J. Automatic recipe cuisine classification by ingredients. In *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: adjunct publication*, pp. 565–570, 2014.

Teng, C.-Y., Lin, Y.-R., and Adamic, L. A. Recipe recommendation using ingredient networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, pp. 298–307, 2012.