Machine Learning Engineer Nanodegree

Capstone Project

Qixiang Zhang
Jul 3rd, 2018

What's Cooking? | Kaggle

A featured Kaggle playground competition

I. Definition

Background

This project is built upon one of Kaggle's playground competition - *What's Cooking?*.

Grew up in China, studied in the United States for 10 years, traveled to some parts of the world, and inspired by Anthony Bourdain, I love eating and making food. Being a home-cook myself, I always enjoy mixing a few ingredients to produce meals that my family love (most of the time).

Thank Yummly for the dataset and Kaggle for hosting the competition. I will leverage what I have learned from Udacity's Machine Learning Nanodegree program to explore machine learning models and produce the optimal results for this capstone project.

This is a classic multi-class classification problem. From Udacity, I have learned to use binary classification algorithms to classify 2 classes. I will explore algorithms for multi-class classification.

I found these cool papers and sources below regarding multiclass classification:

- Survey on Multiclass Classification Methods [1]
- A Comparison of Methods for Multi-class Support Vector Machines [2]
- Towards Maximizing the Area Under the ROC Curve for Multi-Class Classification Problems [3]
- A simplified extension of the Area under the ROC to the multiclass domain [4]
- L1 AND L2 REGULARIZATION FOR MULTICLASS HINGE LOSS MODELS [5]
- On Logistic Regression: Gradients of the Log Loss, Multi-Class Classification, and Other Optimization Techniques [6]
- Machine Learning Basics Lecture 7: Multiclass Classification [7]

Problem Statement

The main goal is to be able to use an ingredients list to predict the most probable cuisine (e.g. Greek, Indian, Chinese, etc.) out of 20 total possible cuisines. The training dataset is clean and labeled. I can directly use defined models to train on the dataset and to classify recipes into 20 different classes. Specifically, to answer these questions below:

- What are the most used ingredients in each of the cuisines?
- How similar or different are these cuisines?
- Given a recipe (ingredient list), how accurately can the model classify its cuisine?

From the labeled dataset, we can learn: what are the most used ingredients, main ingredients for each cuisine, which cuisines are similar to each other (consists of similar ingredients), and finally use recipe ingredients to predict the most probable cuisines.

Metrics

Originally, I was going to use AUC_ROC to measure the performance of my models because the class samples are imbalanced. For example, Italian cuisine has 7838 recipes (19.71%) while Brazillian cuisine has only 467 recipes (1.17%). However, I decided to stick with the Kaggle's metric "accuracy" to measure the performances of my model for two reasons: the results will be measured under Kaggle's evaluation methodology and AUC_ROC may give my model a different result; AUC_ROC is very complicated to implement across all the models I will be using. To compensate the imbalanced data, I will split the train-validation data manually to maintain the weights for each class. I will also use Scikit-learn's StratifiedKFold to cross validate the models while doing the grid search for the best parameters for each model.

Project Steps

The workflow for solving the problem would follow the similar approach that provided by other projects in the Machine Learning Nanodegree:

1. Import the data
2. Study the data
3. Preprocess the data
4. Explore different models
5. Evaluate the models
6. Summary

II. Analysis

Data Exploration

What does the test (target) file `test.json` looks like?

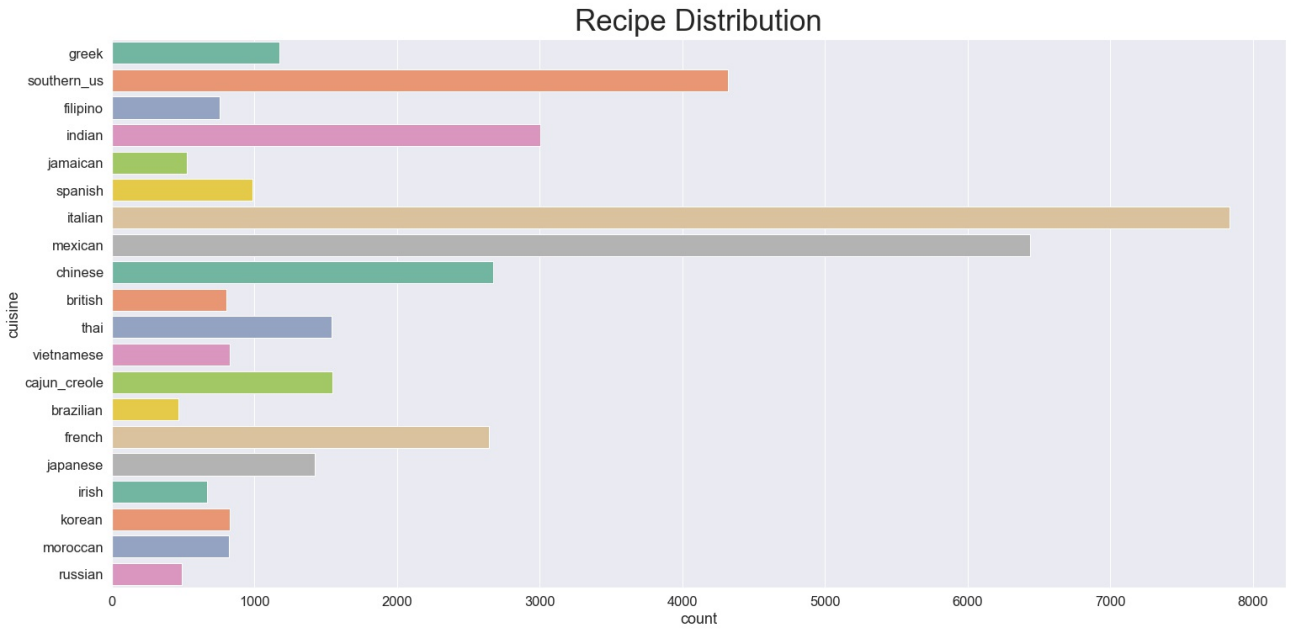
	id	ingredients
0	18009	[baking powder, eggs, all-purpose flour, raisi...
1	28583	[sugar, egg yolks, corn starch, cream of tarta...
2	41580	[sausage links, fennel bulb, fronds, olive oil...

What does the training dataset `train.json` look like?

	cuisine	id	ingredients
0	greek	10259	[romaine lettuce, black olives, grape tomatoes...
1	southern_us	25693	[plain flour, ground pepper, salt, tomatoes, g...
2	filipino	20130	[eggs, pepper, salt, mayonaise, cooking oil, g...

Let's look at the `train.json` in depth:

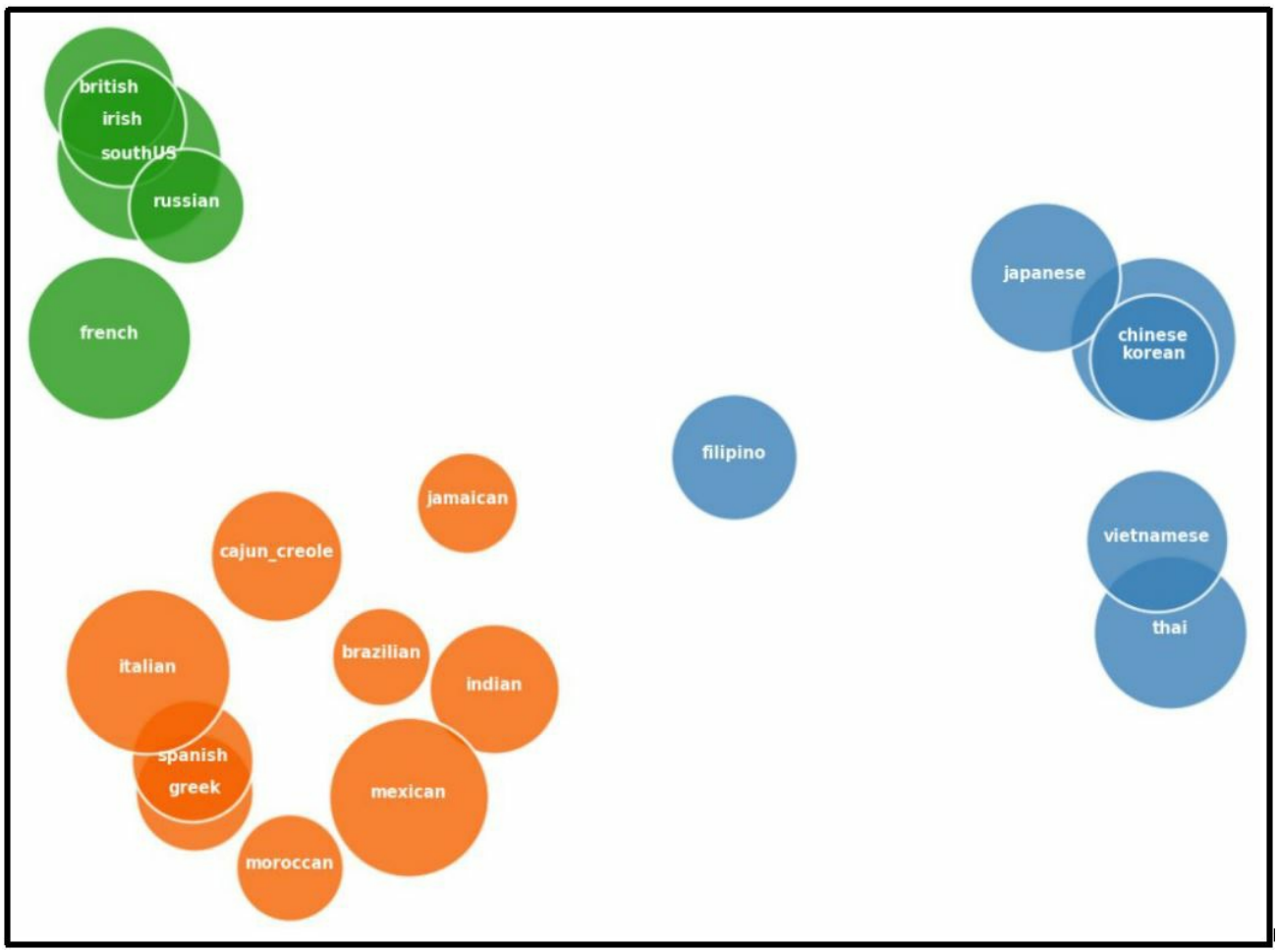
```
There are a total of 39774 ingredient lists (recipes),

20 types of cuisines:

    Weight     Recipe Cuisine

    19.71 %    7838     italian
    16.19 %    6438     mexican
    10.86 %    4320     southern_us
    7.55 %     3003    indian
    6.72 %     2673    chinese
    6.65 %     2646    french
    3.89 %     1546    cajun_creole
    3.87 %     1539    thai
    3.58 %     1423    japanese
    2.95 %     1175    greek
    2.49 %     989     spanish
    2.09 %     830     korean
    2.07 %     825     vietnamese
    2.06 %     821      moroccan
    2.02 %     804      british
    1.9 %      755     filipino
    1.68 %     667      irish
    1.32 %     526      jamaican
    1.23 %     489      russian
    1.17 %     467      brazilian
```



From the graph and data above, we can see that each cuisine has a quite different counts of recipes. Out of all recipes, Italian, Mexican, and Southern US dominants nearly half of the population. The dataset also has a long "tail" - 14 of the cuisines has less than 4% of the total counts of recipes.

How similar are these cuisines?

I found this graph in the competition kernel page [8] posted by alona_levy completed this question beautifully. All kudos to him showing the relationships between cuisines. The image is included as below:



Let's look at the ingredients!

```
There are a total of 6714 "unique" ingredients

Most common ingredients used:

('salt', 18049)
('onions', 7972)
('olive oil', 7972)
('water', 7457)
('garlic', 7380)
('sugar', 6434)
('garlic cloves', 6237)
('butter', 4848)
('ground black pepper', 4785)
('all-purpose flour', 4632)
('pepper', 4438)
```

From the list above, the 'galic' and 'galic cloves' are essentially the same ingredients. Therefore, there must be many same ingredients but with a slightly different names. I looked deeper into the dataset and found more instances. Here are some examples:

- **lettuce**: shredded lettuce, lettuce leaves, romaine lettuce, romaine lettuce leaves, etc.
- **eggs**: eggs, large eggs, extra large eggs
- **tomatoes**: tomatoes, diced tomatoes, fresh tomatoes

Algorithms and Techniques

Preprocessing

Within the ingredient list, although I have consolidated each ingredient from every recipe, some of the differentiated ingredients might still be the same (i.e. garlic vs. garlic cloves). Natural language processing is recommended for training better models.

Metric

The naïve `train_test_split` would not divide the dataset proportionally feeding into each model and will reduce the model's performance. Accuracy may not be the appropriate metric [9] for measuring the performances of each model since we have imbalanced classes. However, Kaggle uses accuracy to score every competitor. To best divide the dataset for training and validation, I will manually split the dataset to best maintain the weights for each class. Then, I will use StratifiedKFold to cross validate each model and GridSearchCV to tune each model. Lastly, I will use accuracy as a scoring function within the grid search method to measure the performance of each model under different set of parameters.

Modeling (algorithms)

The multiclass classification problem can be decomposed into several binary classification tasks that can be solved efficiently using binary classifiers, I will be using some of the scikit-learn modules as follows:

- Multinomial Naive Bayes
  - Naive Bayes assumes the Bayes' theorem, that independence among predictors (a particular feature in one class is unrelated to other features). The algorithm will calculate the likelihoods of each class using Bayes theorm and the class with the highest probability will be the output. Multinomial Naive Bayes algorithm assumes the input variables follows the bell curve (normal distribution) and works very well in classifying multiclass data and text classification in practice. It is easy and fast to predict class of test data set and designed for multi-class prediction. However, the challenge is that the assumption of independence does not hold well in most of the cases in real life. In this project, I use tf-idf to vectorize and highlight the feature importance (based on the count of input variables) before feeding into the models.
- Ensemble Methods (Random Forest, XGBoost, and LightGBM)
  - Random Forest is an ensemble of a group of decision trees (features) and trains each individual decision tree with different subsets of the training data. Each node of the decision tree is split using a randomly selected feature from the data, so the models created by the algorithm would not be correlated with each other. Eventually, the outliers would be eliminated through majority voting. Random forest is a good model to reduce overfitting and no need for feature normalization. Each individual decision trees can be trained in parallel. It can be used for multiclass classification out of the box. In this project, I will use Random Forest as a benchmark model.
  - XGBoost is a start-of-the-art model that became popular not long ago. It is also an ensemble method that creates a strong classification model by stacking "weaker" ones. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor until the training data is accurately predicted or reproduced by the model. It also adds an additional custom regularization term in the objective function.
  - LightGBM, different from other tree-based algorithms, splits the tree leaf wise, which reduce more loss than the level-wise algorithms to reach higher accuracy. However, it is a sensitive algorithm prone to overfitting. Adjusting the parameters (over 100 of them) can be very challenging with limited documentation available. I found help from this Medium Post [21]
- Stochastic Gradient Descent Classifier
  - Stochastic gradient descent is a iterative method for optimizing a differentiable objective function (a stochastic approximation of gradient descent optimization). Traditional gradient descent uses least square approach for fitting by calculating the entire data set, which is very expensive and slow to train. Stochastic gradient descent only uses a small sample of training set (selected stochastically) to calculate the parameters, which is faster. In Scikit-learn, stochastic gradient descent is a binary classification. By using the "one versus all" scheme provided by Scikit-learn, I can use it for multi-class classification for this project.
- Support Vector Machines
  - Support Vector Machine is a classification algorithm that uses the support vectors (data points nearest to the hyperplane,

which separates and classifies a set of data) to classify instances into two classes (binary). The goal is to choose a hyperplane with the greatest possible margin between hyperplane and any point within the training set, given a greater change of new data being classified correctly. Same Scikit-learn's "one versus all" scheme can be applied on this algorithm to turn this binary classification method into multi-class classification.

- Logistic Regression
  - Logistic regression is one of the most used algorithm for binary classification for its simplicity and efficiency to implement. It measures the relationship between the labels and features by estimating the probabilities using its underlying logistic function. Fundamentally, it is a linear method but it uses the logistic function to transform the predictions. Scikit-learn has the logistic regression with multinomial built-in but I will implement both the built-in multinomial and the "one vesus all" scheme module.
- Neural Network by Scikit-learn
  - Multi-layer perceptron classifier (MLPClassifier) is a feedforward neural network model (connections between nodes do not form a cycle unlike recurrent neural networks) that has been adopted by Scikit-learn. It optimizes the log-loss function using LBFGS or stochastic gradient descent.

For the methods above, before feeding preprocessed training data directly into the models, splitting the training data into training and validation is recommended. As stated in the `Metric` section, I will use Stratified KFold method to divide the data while maintain the weights for each class.

Beside the above algorithms, I also plan to design a deep convolutional neural network (CNN) from Keras (inspired by Jason Brownlee's article [10]). CNN is commonly used for image classification. A convolutional filter can be seen as a sliding matrix function applied to a matrix input. CNN consists of layers of these filters that perform different labeling (generalized features specific to the input data). By feeding the input data through CNN, the model will automatically learn its filters values and categorize the input data. Instead of image pixels, the input would be string of ingredients in forms of matrices (tf-idf regularization).

Benchmark - Random Forest (Scikit-learn.ensemble.RandomForestClassifier)

Random forest is an ensemble learning method for classification that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The nature of random forest is perfect for multi-class classification.

I fit the `train.json` on to the out-of-box random forest model and used the `test.json` to predict the cuisine column. After submit to Kaggle, I received an accuracy score of `0.70002`

III. Methodology

Preprocess the datasets

I found the ingredient column stores each ingredients in a list. I also found there are some characters that are not useful for classification (i.e. '()', numbers, '¢', etc.). One way to get rid of them is to use regular expression functions to find the pattern and remove the non-letter characters. Another powerful library that I found through the source1 [11] and source2 [12] is to use natural language process package ntlk.

Let's look at in detail!

What does the ingredients column look like?

I abstract some examples from the raw dataset (train.json file)

```
['(    oz.) tomato sauce', 'serrano chilies', 'golden raisins', 'firmly    packed brown sugar',
'chopped garlic', 'vinegar']
['manchego cheese', 'empanada', 'jalapeno chilies', 'garlic salt', '2 1/2 to 3 lb. chicken,
```

```
    cut into serving pieces', "hellmann' or best food real mayonnais", 'vegetable oil',
    'sliced green onions']
    ['butter', 'sugar', 'corn starch', 'Betty Crockerâ„¢ oatmeal cookie mix', 'YoplaitÂ® Greek 100
    blackberry pie yogurt', 'blackberries']
    ['Tipo 00 flour', 'semolina', 'extra-virgin olive oil', 'large free range egg']
```

These ingredients above are small samples in the training dataset. We have to preprocess the data to clean the 'ingredients' going into the models before training.

I will use regular expression [13] to get rid of non-letter characters, and then nltk to restore words to their original forms (i.e. 'tomatoes -> tomato')

1. I found this tutorial [13] very helpful for learning the basics about the regular expression library.
2. I learned most about **lemmatizing** and **stemming** from here [14].

Let's look at the preprocessed results!

```
['toma sauce', 'serrano chilies', 'golden raisin', 'firmly packed brown sugar',
'chopped garlic', 'vinegar']
['manchego cheese', 'empanada', 'jalapeno chilies', 'garlic salt', 'chicken cut
 in serving piece', 'hellmann or best food real mayonnais', 'vegetable oil',
'sliced green onion']
['butter', 'sugar', 'corn starch', 'Betty Crockerâ oatmeal cookie mix',
 'YoplaitÂ Greek blackberry pie yogurt', 'blackberry']
['Tipo flour', 'semolina', 'extra virgin olive oil', 'large free range egg']
```

Originally, I was going to remove all the words except for nouns. However, I found the nltk library does not usually pos_tag each word correctly (i.e. the word 'chicken' was tagged as a verb instead of a noun). Also, removing these words might throw away some important data for training.

I found another very useful technique: Tf-IDF or term frequency–inverse document frequency. According to Wikipedia, it is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

With this dataset, tf-idf will score ingredients based on the frequency of their occurrences, which will mitigate the "garlic vs chopped garlic" problem more efficiently.

Implementation

TfidfVectorizer

Originally, I planned to feed the lemmatized ingredients directly into these models. After reading this article [22], I found that by tf-idf vectorizing the texts into matrices as inputs are much more efficient. After cleaning up the input data (train.json and test.json) by removing units and some of the non-letter characters such as numbers and whitespaces. It still needs to be configured into a tf-idf matrix before feeding into different models. Instead of using strings as input, I will be using the tfidfvectorizer from Scikit-learn to configure the strings into tf-idf features as the input.

Implementation - metric

As I have previously mentioned, I originally planned to use AUC-ROC to measure the performance for the models but I decided to stick with Kaggle's original guidline to measure each model based on the categorization accuracy (the percent of dishes that you correctly classify).

Additionally, I will be using Scikit-learn's grid search with cross validation to search for the best parameters for the models (except for Keras deep learning model) that I will be using. The cost of running time and tweaking is very high (about a week).

Implementation - algorithms (models) I will be using:

This page gives a clear guide on Scikit-learn's available algorithms that I may be able to use. I choose to implement the following models for this project:

- Random Forest (Scikit-learn.ensemble.RandomForestClassifier)
- Naive Bayes (Scikit-learn.naive_bayes.MultinomialNB)
- Logistic Regression (Scikit-learn.linear_model.LogisticRegression)
- Neural Network by Scikit-learn (Scikit-learn.neural_network.MLPClassifier)
- Logistic Regression (Scikit-learn.linear_model.LogisticRegression) with the one-vs-all wrapper by Scikit-learn
- Support Vector Classification (Scikit-learn.svm.SVC¶) with the one-vs-all wrapper by Scikit-learn
- SGD (Scikit-learn.linear_model.SGDClassifier) with the one-vs-all wrapper by Scikit-learn
- XGBoost
- LightGBM
- Keras Deep Neural Network

The general procedure

1. I defined a function to split the train.json dataset into 85% of training data and 15% of testing data (for validation). I maintained the weight for each class (cuisine).
2. For both training and testing data,
    i. I TfidfVectorized the lemmatized string into matrices of TF-IDF features.
    ii. Labels (y's) are encoded using Scikit-learn's `LabelEncoder` module for some of the Scikit-learn's models.
3. I defined a function for finding the best estimators and performing cross-validation for each of the models. The inputs are classifiers, parameter dictionary, features input (x), and the expected labels output (y).
4. To train the deep neural network using Tensorflow, since Keras already has the cross validation function within the model, all I have to do is split the training, validation, and the testing datasets.

Challenges

- In the beginning, I did not expect the cross validation process is so expensive. It is the best practices to record the results and define functions as needed for better organization. Another trick that I used for perforamnce is to only use 10% of the entire data for fitting the models to debug any potential errors. Using only a small portion of the data, the model runs much faster (can usually finish within seconds to minutes with cross validation). I did not have to wait hours to see the results (either an error, or the expected accuracy score).
- For all of the Scikit-learn modules, I was able to directly use the `fit` and `predict` functions to train and predict on the tfidfvectorized features and raw labels (cuisines). For XGBoost, LightGBM, and deep CNN models, the labels have to be encoded for training the data. After training, remeber to `inverse_transform` (decode) back to the original labels.

Refinement

I have implemented the base model using Scikit-learn's random forest without adjusting any parameters. I received a result of 0.70002 (around top 80% on the public leaderboard on Kaggle when the competition was active) - I think it is not bad for my first Kaggle score without doing any tuning on the parameters and being able to use the Scikit-learn's base random forest model.

After GridSearchCV with a validation score of 0.7595 accuracy, Kaggle gives a new score of 0.75905 (around top 63%) for the tuned random forest model. My score went up almost 20% on the ladder! To achieve a higher accuracy I must explore other models! So I have used the same technique (GridSearchCV) to tune all the models.

Use the best parameters from grid search results for each models to fit the training data

Random forest

I tuned the max_features, criterion, and number of estimators among all the parameters. I set the number of estimators very high because there are a high number of ingredients. I let the model to test between 'gini' and 'entropy' as criterion, and also different number of features when looking for the best split.

I have also kept the random state as 0,and n_jobs as 2 to keep 2 cores for computing. The validation accuracy score is 0.7595.

```
# Best estimator after running the grid search cross validation for the random forest model
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features=3, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=2,
            oob_score=True, random_state=0, verbose=0, warm_start=False)
```

Multinomial Naive Bayes

The Scikit-learn's naive bayes class has a module for multinomial models. It is designed for classification with discrete features (e.g. word counts for text classification) - perfect for this problem. With text classification, it requires integer feature counts or tf-idf input.

Alpha is the only parameter that I have tuned. I tried [0.01, 0.02, 0.035, 0.04, 0.1, 0.5, 1] and found that 0.035 gives the highest validation score - 0.7424.

```
# Best estimator after running the grid search cross validation for the Multinomial Naive Bayes model
MultinomialNB(alpha=0.035, class_prior=None, fit_prior=True)
```

Logistic Regression

Scikit-learn has many variations of logistic regression models. I implemented and tuned the basic model. I tried many different solver and the 'C', which is the inverse of regularization strength. I found the combination of lbfgs and 5 gives me the best validation score - 0.7896.

```
# Best estimator after running the grid search cross validation for the Logistic Regression (Multinomial) model
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=2, penalty='l2', random_state=0, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

Neural Network by Scikit-learn

It was surprising and unsurprising to find the neural network model on Scikit-learn. I expected to build a neural network myself using Keras (which I did later) but I found this handy tool straight from Scikit-learn. I was also unsurprised to find it on Scikit-learn because they are very well known for providing the most comprehensive and organized machine learning API's.

There are many parameters that I have fiddled with such as the activation function, solver, and the learning rate. It gave a decent validation score of 0.7811.

```
# Best estimator after running the grid search cross validation for the Scikit-learn's neural network (MLPclassifier) model
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
```

```
        learning_rate_init=0.001, max_iter=200, momentum=0.9,
        nesterovs_momentum=True, power_t=0.5, random_state=0, shuffle=True,
        solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
        warm_start=False)
```

Logistic Regression with OVA

This is the second logsitic regression I implemented with the multi-class classification strategy of One-Versus-All. One-Versus-All is a classic multi-class classification strategies based on this material and Wikipeida. Scikit-learn provides a wrapper classification method, Scikit-learn.multiclass.OneVsRestClassifier, to turn binary classifiers (SVM, SGD, etc.) into multi-class classifiers.

Like the previous logistic regression base model, I tuned the solver and the 'C' parameters and received a validation score of 0.7958.

```
# Best estimator after running the grid search cross validation for the One-Versus-All Logistic Regression model
OneVsRestClassifier(estimator=LogisticRegression(C=10,
          class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=2,
          penalty='l2', random_state=0, solver='lbfgs', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=2)
```

SVM with OVA

Another model that I have implemented with the OVA wrapper. I tuned a number of parameters including the 'C', 'gamma', 'coef0', cache size, and kernel. This grid search took a very long time to run (about 11.5 hours on 6 cores of the CPU). Finally, I received an high validation score of 0.8082 (the highest so far)

```
# Best estimator after running the grid search cross validation for the One-Versus-All SVM model
SVC(C=3.25, cache_size=500, class_weight=None, coef0=0.0,\
  decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',\
  max_iter=-1, probability=False, random_state=0, shrinking=True,\
  tol=0.001, verbose=False)
```

SGD with OVA

Stochastic Gradient Descent has been my favorite classification method for its efficiency and performance. I tried to tune 2 of the parameters: loss function and the learning rate. With the modified huber loss and optimal learning rate, I received a validation score of 0.7797 (not bad but not the best).

```
# Best estimator after running the grid search cross validation for the One-Versus-All SGD model
OneVsRestClassifier(estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='modified_huber', max_iter=None,
       n_iter=None, n_jobs=2, penalty='l2', power_t=0.5, random_state=0,
       shuffle=True, tol=None, verbose=0, warm_start=False), n_jobs=2)
```

XGBoost

It is always exciting to explore something not in the textbook. XGBoost was recommended by a Udacity mentor. The API is straight-forward to read. It had a constraint on the input data - labels should be integers instead of strings. Therefore, I used Scikit-learn's LabelEncoder to fit the labels and later inverse transformed the labels back to strings as cuisine prediction.

I tuned a few parameters: learning rate, max depth of the tree, gamma, and number of estimators. This took a quite long time to grid

search (13.5 hours on 2 cores of the CPU).

```
# Best estimator after running the grid search cross validation for the XGBoost model
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=1, learning_rate=0.01, max_delta_step=0,
       max_depth=12, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=2, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
# remember to inverse transform the encoded cuisine column
```

LightGBM

Lke XGBoost, lightgbm was also introduced by the Udacity mentor. It is a similar algorithm as XGBoost. I tuned the learning rate and the number of estimators. However, it never stopped running… I gave up after the 20th hour.

```
# LightGBM model
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
        gamma=1, learning_rate=0.01, max_depth=6, min_child_samples=20,
        min_child_weight=0.001, min_split_gain=0.0, n_estimators=500,
        n_jobs=2, num_leaves=31, objective='multiclass', random_state=0,
        reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=0.8,
        subsample_for_bin=200000, subsample_freq=0)
```

Keras Deep CNN

With Keras documentation, guide1 [15], guide2 [16], and guide3 [17], I have successfully designed my neural network for text classification.

I started with the relu activation function with about half of the nodes with the input function. A 0.5 dropout layer is followed after it. Then I dense again with a tanh function followed by a higher discard rate dropout layer (0.67) to reduce dimension. Lastly, I condense the data with softmax and 20 nodes (20 cuisine) as outputs. I set the epoch (max iteration) as 10 while set the early stopping as patience as 1 by monitoring the total validation loss in each iteration.

Here is the detailed code for the architecture:

```
# define the layers
model = Sequential()
model.add(Dense(1024, input_shape=(2182,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(144, activation='tanh'))
model.add(Dropout(0.67))
model.add(Dense(20, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# parameters for neural network
epochs = 10
early_stopping = EarlyStopping(monitor='val_loss', patience=1)
```

Here is the results after fitting the training data:

```
Train on 33799 samples, validate on 5975 samples
Epoch 1/10
 - 13s - loss: 1.0982 - acc: 0.6840 - val_loss: 0.7255 - val_acc: 0.7868
```

```
Epoch 2/10
 - 13s - loss: 0.7309 - acc: 0.7840 - val_loss: 0.6944 - val_acc: 0.7883
Epoch 3/10
 - 13s - loss: 0.6135 - acc: 0.8191 - val_loss: 0.6721 - val_acc: 0.8012
Epoch 4/10
 - 13s - loss: 0.5297 - acc: 0.8427 - val_loss: 0.6842 - val_acc: 0.7975
```

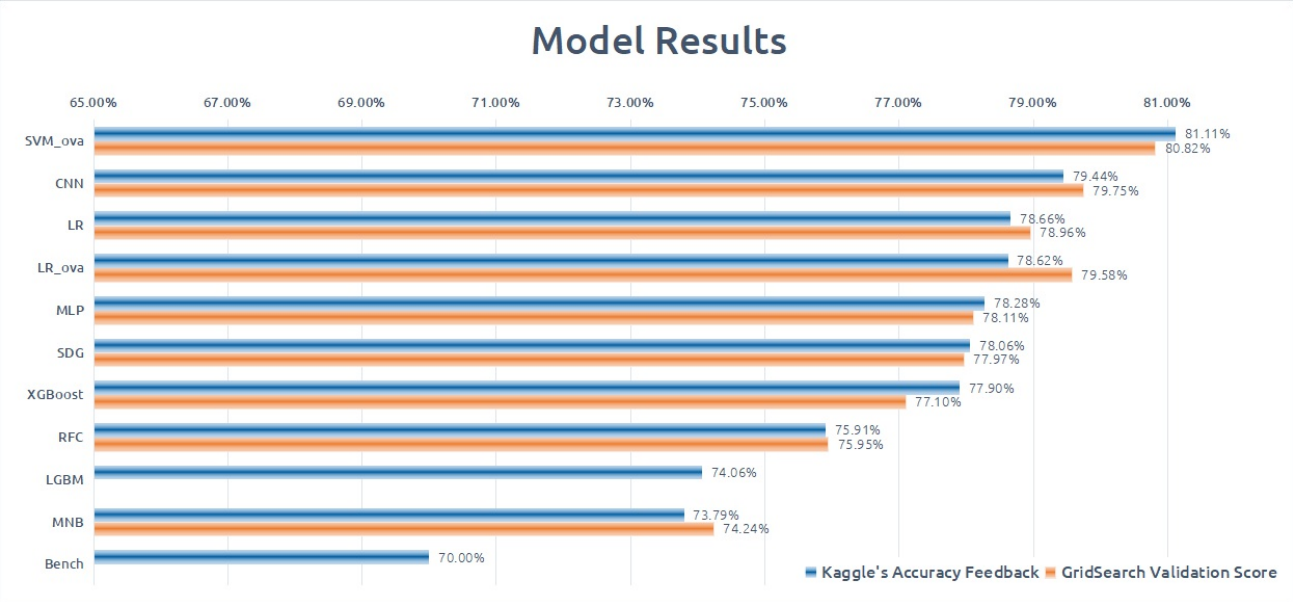
The validation score is `0.7975`, which is higher than all other models!

I learned the following while trying different architectures for neural network:

1. It does not require deep and complex layers for best output
2. The `dropout` layer with a high discard rate prevent the model from overfitting very efficiently.
3. The `ReLU` is the best activation function for the first layer
4. Middle layers can be implemented with `tanh`, `softsign`, or `sigmoid`, and the differences are minimal.

IV. Results

Model Evaluation and justification (sorted by Kaggle's score):



All the models end up producing much better results than the benchmark model (random forest out of the box). SVM (OVA) produced the highest accuracy **0.81114**. The deep learning with Keras scored the second highest as **0.79444**, followed by the logistic regression base model.

As expected, the model with the highest validation score scored the highest on the testing data (SVM). The StratifiedKFold maintained the weights for each classes the same as the total population. The grid search cross validation helped finding the best parameters for each model. Given that the validation tiers matches the final accuracy scores, the final models are reasonable and align with the solution expectations. The final parameters are appropriate and robust enough for the problem and the datasets.

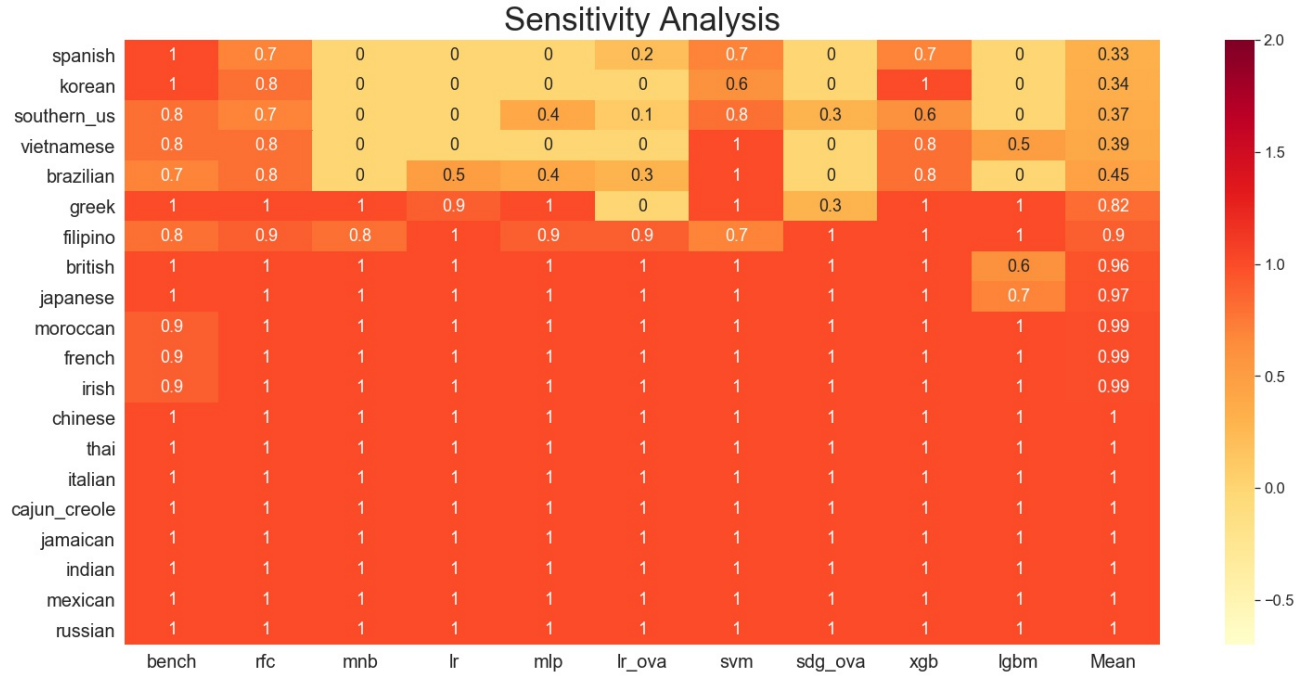
Sensitivity Analysis

An model with high accuracy is not always a robust model. The model may sometimes be either too complex or too naïve to sufficiently generalize to new data. The model could be underfitted. I ran a sensitivity analysis to fit each model ten times with different

training and testing sets to see how the predictions change for a specific group of recipes and cuisines pairs.

The sensitivity analysis test data is the same population as the train.json. I randomly selected 1 recipe from each cuisine aside for prediction. For the rest of the data point at each iteration (total 10), I split the data with 80-20 train-valid ratio and fit each model with the 80% of training data while leave out 20% of the data aside.

Below is the result showing the average accuracy for each model to classify each cuisine after 10 iterations. The last column is the mean score for all cuisines.



From the heatmap, we can see that all of the models perform very well in classifying cuisines including British, Japanese, Moroccan, French, Irish, Chinese, Thai, Italian, Cajun_Creole, Jamaican, Indian, Mexican, and Russian.

On average, models struggles to classify cuisines including Spanish, Korean, Southern US, Vietnamese, and Brazilian. Surprisingly, the Benchmark model seems to outperform the rest of the models on all cuisine.

However, SVM still outperforms all the models by a notch. This sensitivity analysis validate the robustness of the model's solution. In the future, to mitigate each model's weakness, I can attempt to stack [20] two models that are good at classifying different classes to reach more ideal results.

V. Conclusion

Reflection

When I encountered this problem during research, I found it very intriguing because I am a foodie and a home chef who is passionate about food. I thought it would be appropriate to apply what I have learned through the Udacity's Machine Learning nanodegree on this classic classification problem by using all the models and techniques that I have learned (supervised learning, sampling, validation, parameter tuning, deep learning, and more!).

As I approach this problem, I found myself still lack knowledge and experiences in preprocessing. I am spoiled with preprocessed clean data and all I had to do is to 'fit' it with a out-of-box Scikit-learn module. I can imagine that in the industry, many problems has to be solved with some serous configurations of discovered models, or even made from scratch.

I also learned many different strategies of classifying dataset with multiple classes: one-versus-all, pairs, and error-correction-output-coding. Scikit-learn also have the error-correction-output-code wrapper to turn binary classification algorithms into multi-class ones. I tried it on Logistic Regression, SVM, and SGD while ran into problems of configuring the input/output. Since the result was not competitive (less than 70% accuracy) I discarded them from this project.

This is also the first time that I encountered natural language processing problem. Lemmatizing/stemming and the tf-idf vectorizer technique are very useful in dealing with text classification and reducing dimensions before feeding the classification models. I found it challenging to generalize the patterns in the ingredient list. To best disregard the 'noises' in the training and testing dataset, we might have to define our own grammar rules. In the earlier section, I mentioned that I was ready to remove all non-nouns in the ingredients list with the pos_tag function labeled words. However, the labeled words were wrongly labeled sometimes. For example, one of the ingredient was '2 1/2 to 3 lb. chicken, cut into serving pieces'. The word 'chicken' was labeled as a verb instead of a noun, while the main ingredient in this string is the chicken. Chicken is the only ingredient mattered in this long string. Selecting only the nouns among the string will run the risk of giving up many key ingredients for correctly predicting the cuisine.

I did not enjoy the time waiting for the grid search results. In the beginning, I set the n_jobs parameters as high as possible, so '-1', which would be all 6 cores of my i5 - 8400 CPU. During the validation time, the computer freezes up often when I tried to open a new webpage for research. For future training, I will explore the option of reducing the number of core running or even seeking resource from AWS. I also planned to use KNN as one of my algorithms but the cross validation time was too long even for 10% of the data. I think it requires a lot of memory (which my computer does not have) for the full dataset.

The last thing I have learned is the sensitivity analysis. The models can usually be either too complex or too naïve. There should always be some well-formed tests to see if the models are overfitted or sensitive to outliers.

I have definitely learned a lot from this experience since it was my first Kaggle project. It was rewarding when I found my scores are competitive enough on the leaderboard. I also enjoyed reading some of the posting (kernels) on Kaggle website.

Improvement

There are many areas for improvements or considerations for the future projects:

1. AUC_ROC is still a very good metric to implement for imbalanced dataset. I tried to implement it in the grid search with the make_scorer function and the scoring parameter, but it did not work properly as expected. I shall consider implementing it in the future along with accuracy to measure the performance for each model.
2. I shall consider dig deeper in configuring my own corpus [18] using the NLTK library for future NLP problems. If I can capture all the essential ingredients for each cuisine, the accuracy score will be able to improve substantially.
3. I believe there must be a way of designing a better convolutional neural network architecture using Keras that scores higher accuracy thatn SVM.
4. I was going to try combining some algorithms together into a custom ensemble model with help1 [19], help2 [20] but I had trouble implementing it. The key of stacking the algorithms is to use the predictions of two or more algorithms as features for another algorithm as input before predicting the final labels. I had trouble putting features of tf-idf vectorizing, data fold diving, and cross validation together into a function. I also had trouble identifying how well each model predicting on each cuisine. To achieve this, I should learn by doing projects with smaller scale on datasets.

Reference

[1] Aly, Mohamed. "Survey on Multiclass Classification Methods" 2005
[2] Hsu, Chih-Wei and Lin, Chih-Jen. "A Comparison of Methods for Multi-class Support Vector Machines" 2011
[3] Kang, Ke, Wang, Rui, Chen, Tianshi. "Towards Maximizing the Area Under the ROC Curve for Multi-Class Classification Problems" 2011
[4] Landgrebe, Thomas, and Duin, Robert P.W.. "A simplified extension of the Area under the ROC to the multiclass domain" 2005
[5] Moore, Robert C. and DeNero, John. "L1 AND L2 REGULARIZATION FOR MULTICLASS HINGE LOSS MODELS" 2009
[6] Stratos, Karl. "On Logistic Regression: Gradients of the Log Loss, Multi-Class Classification, and Other Optimization Techniques"

2018
[7] Liang, Yingyu "Multiclass Classification" 2016
[8] alona_levy. "Cultural Diffusion by Recipes" Kaggle 2016
[9] Afonja, Tejumade. "Accuracy Paradox" 2017
[10] Brownlee, Jason. "Multi-Class Classification Tutorial with the Keras Deep Learning Library" 2016
[11] Brownlee, Jason. "How to Clean Text for Machine Learning with Python" 2017
[12] Arun. "Text Preprocessing and Machine Learning Modeling" 2017
[13] Schafer, Corey. "Python Tutorial: re Module - How to Write and Match Regular Expressions (Regex)" 2017
[14] Stanford University. "Stemming and lemmatization" 2008
[15] Brownlee, Jason. "The 5 Step Life-Cycle for Long Short-Term Memory Models in Keras" 2017
[16] Brownlee, Jason. "Best Practices for Document Classification with Deep Learning" 2017
[17] Brownlee, Jason. "Use Keras Deep Learning Models with Scikit-Learn in Python" 2016
[18] ntlk. "Creating New Corpus Reader Instances"
[19] Anisotropic. "Introduction to Ensembling/Stacking in Python" 2018
[20] Gorman, Ben. "A Kaggler's Guide to Model Stacking in Practice" 2016
[21] Mandot, Pushkar. "What is LightGBM, How to implement it? How to fine tune the parameters?". 2017
[22] Shaikh, Javed. "Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK" 2017