Capstone Project

Machine Learning Engineer Nanodegree

Definition

Project Overview

The geographic and cultural differences of a region are reflected directly on their food. This makes so that food is one of the main ways to know and connect with different cultures around the world. Understanding the recipes created from different kinds of cuisines lead you to understand better people from different origins.

In this project, I apply machine learning concepts to a database of recipes provided by a kaggle competition[1] to categorize the cuisine only from recipe ingredients. This is a supervised learning classification problem and the approach I used utilizes text analysis using term-frequency to find a relation between the ingredients and it's cuisine. This kind of approach can be expanded to various fields of study that need text classification.

Problem Statement

The goal of the project is to categorize a recipe in a cuisine using only a list of ingredients. This fits into a classification problem that uses supervised learning. The strategy to tackle the problem is:

1. Download the train and test data from Kaggle
2. Preprocessing and data wrangling
3. Train various classifiers from the training data
4. Evaluate the different classifiers with the test data
5. Optimize the hyperparameters of the best classifier

Metrics

Categorization accuracy is the main metric to be used in the classifier. It takes into account the percent of dishes correctly classified. The categorization accuracy was chosen specially because of the Kaggle competition that uses it to evaluate the submission.

$$accuracy = \frac{recipes\ correctly\ classified}{total\ of\ recipes}$$

[1] https://www.kaggle.com/c/whats-cooking. This is a finished competition that was active from September 9th 2015 to December 20th 2015.

But there are plenty of other metrics to use in classification problems that get a deeper look at how the model is performing. There are:

1. Confusion matrix that shows how the classes are misclassified with each other.
2. Precision shows the influence of the false positives in the classification of each category
3. Recall shows the influence of the false negatives in the classification of each category.
4. F1 score is a combination of precision and recall. It returns the harmonic mean between the two.

The highest score is not always the best way to measure if a technique is better than the other, I will also use training time to evaluate the model.

Analysis

Data Exploration

This competition utilized a dataset provided by Yummly ,in json, that contains 49718 recipes. The training set contains 39774 recipes labeled by cuisine. The test set contains 9944 recipes and is not labeled. All analysis are going to be made in the test set.

cuisine	**id**	**ingredients**
greek	10259	[romaine lettuce, black olives, grape tomatoes...
southern_us	25693	[plain flour, ground pepper, salt, tomatoes, g...
filipino	20130	[eggs, pepper, salt, mayonnaise, cooking oil, g...
indian	22213	[water, vegetable oil, wheat, salt]
indian	13162	[black pepper, shallots, cornflour, cayenne pe...

Table 1: Example of the dataset.

The train data is divided between id, cuisine and ingredients. Id is not important for modelling or analysis so I will ignore it. The other two are the ones that really matter in this dataset. I will talk more about them below:

1. Cuisine: The target feature to be predicted. It represents the origin of each recipe. In this dataset there are 20 different types of cuisine. (string)
2. Ingredients: The ingredients are the main data to be manipulated. They will be the predictors of this model. Each row comes with a list of ingredients for the specified recipe. (list of strings)

Exploratory Visualization

Analysing the dataset we can see that the recipes are not equally distributed. There's a lot more "Italian" recipes than the others. We can see it detailed in the image below.

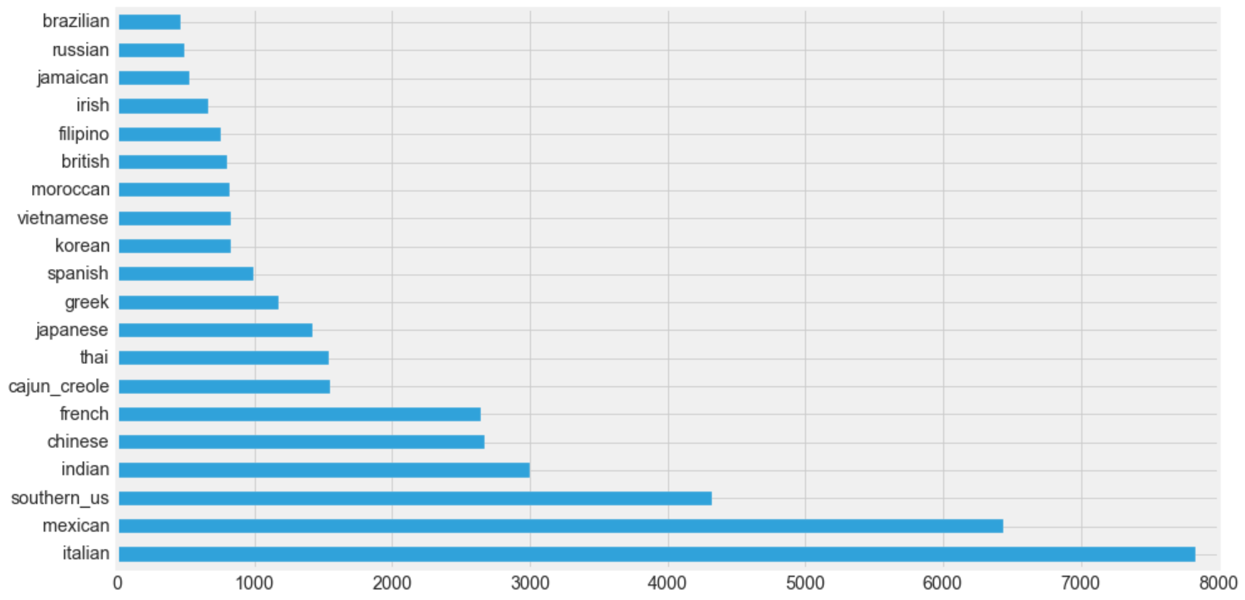


Image 1: Recipes per cuisine

On the ingredients column, there are 6558 types of ingredients. And most of them repeat throughout different recipes and cuisines. To really capture the types of ingredients correctly I needed to clean the data and remove all ambiguous listings of a ingredient.

The first step was to locate all symbols and numerals that were included in the ingredient lists and remove them. Then, the second step was to make all letter lowercase to construct a uniform list. And lastly, check for the same ingredients with different names. The way I did it was plot the 100 most common ingredients and look for duplicates. The most common were 'garlic'/'garlic cloves','eggs'/'large eggs' and 'black pepper'/'ground black pepper'. I changed them to a homogenous name and plotted the 10 most common ingredients. The 10 most common ingredients are listed on the image below.

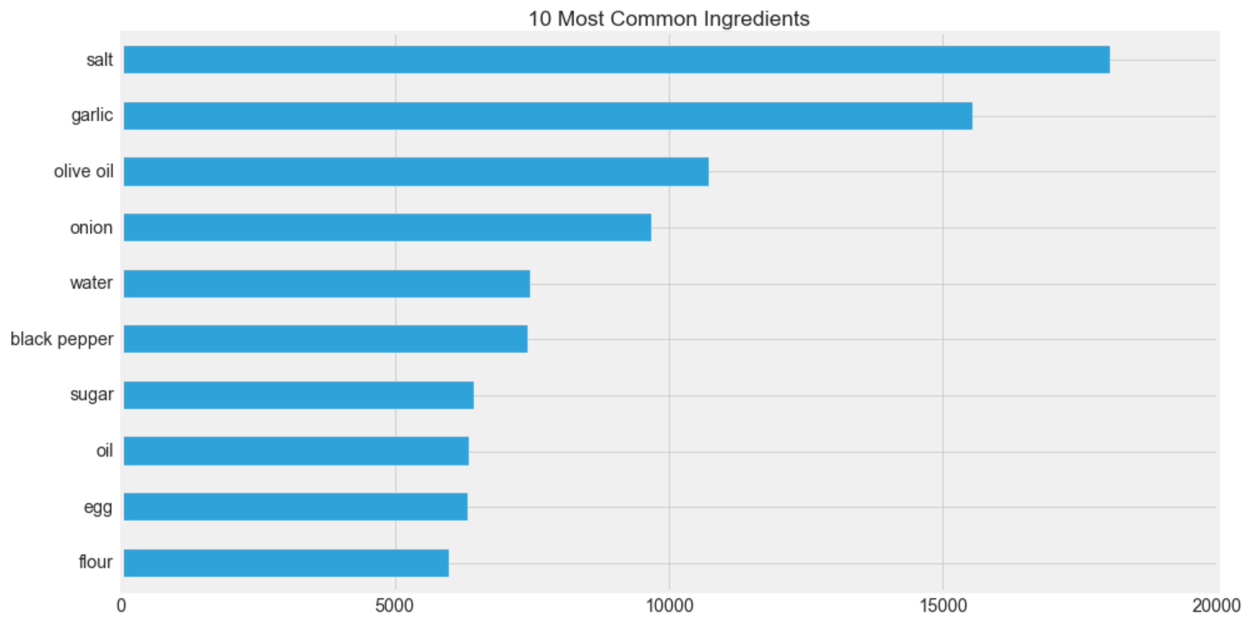


Image 2: The 10 most common ingredients

As we can see in the plot above, salt is the most dominant ingredient in this dataset. That is interesting because salt is one of the most important ingredients to have in your food. It boost the flavor of a recipe. On second place we have garlic, this ingredient is a type of bulb that is used in almost all kinds of cuisines when talking about savory recipes. It adds flavor to the recipe. On third place the olive oil, this ingredient is a fat extracted from the olive. It is very versatile in the kitchen, it can be used from a salad dressing to frying. This list of 10 most common ingredients really captured the ingredients you must have in a kitchen.

The ingredients above are global and probably represent a good part of the most common ingredients in each specific cuisine. So, to take care of that I removed salt, garlic, olive oil, onion, water, black pepper, sugar, oil, eggs and flour from the most common for each specific cuisine. I expect the result will tell a little more about the cuisines now that the most used ingredients were removed.

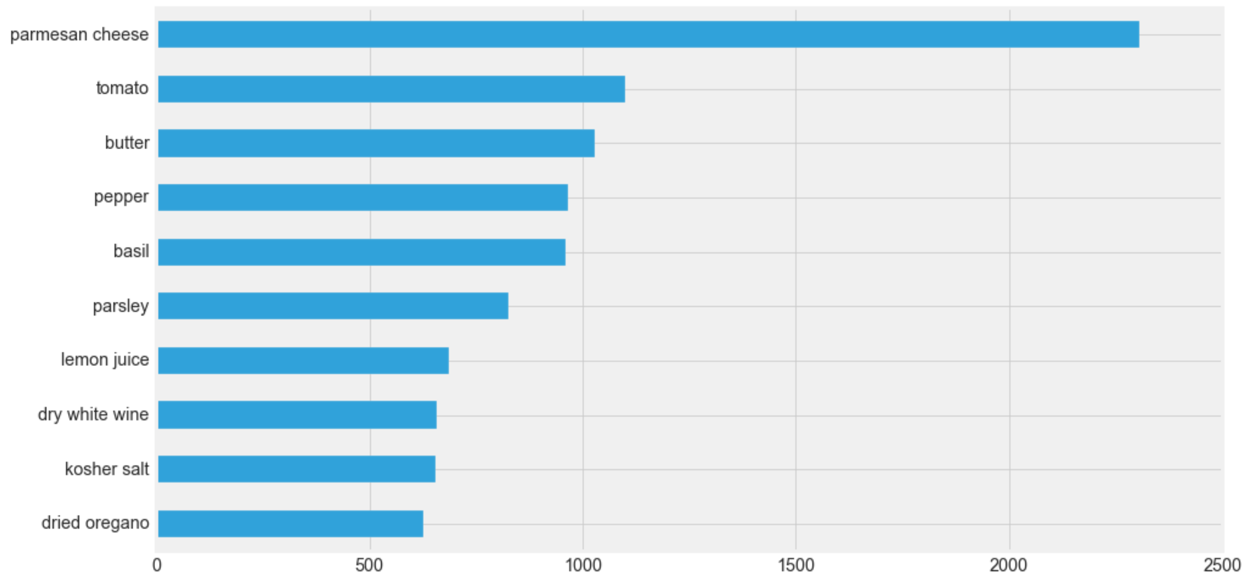


Image 3: Italian cuisine most common ingredients

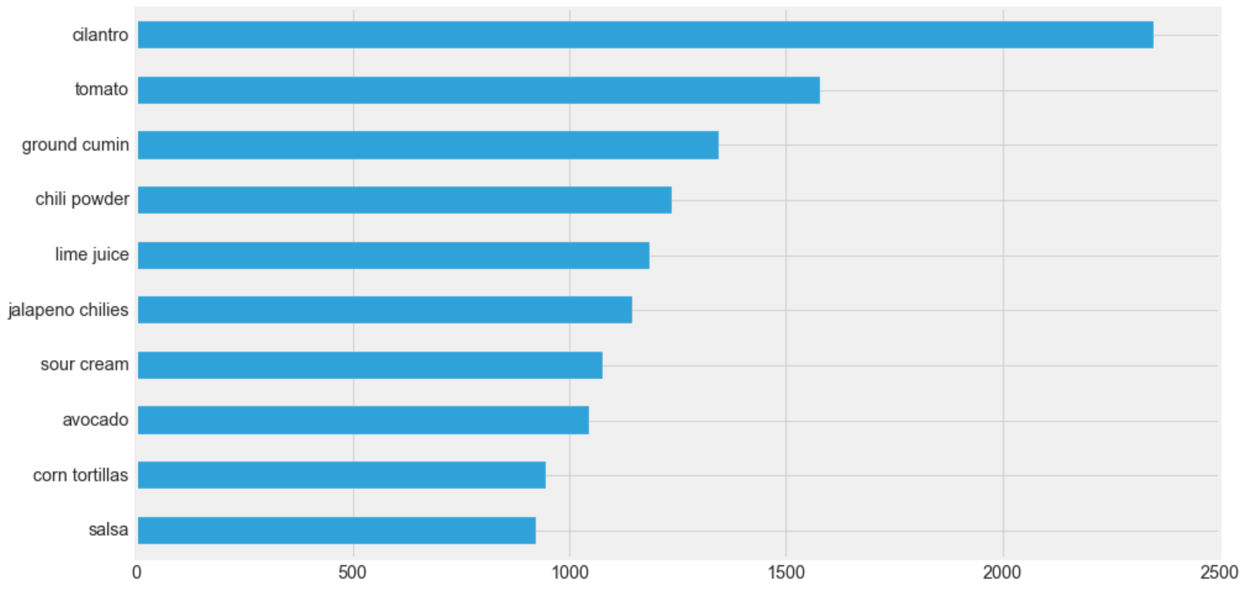


Image 4: Mexican cuisine most common ingredients

As we can see in the examples above, these ingredients represent the regional foods much better than using only the global most common ingredients.

Algorithms and Techniques

This is a classification problem, so the algorithms and techniques that I used are based on this. For this problem I explored different algorithms and techniques:

Support Vector Machines (LinearSVC[2])

Support vector machines are a set of supervised learning methods that uses decision boundaries for classification and regression problems. The support vector machine constructs a hyperplane as a decision surface so that the margin of separation between the positive and negative examples is maximized. The margin of separation is the distance between the hyperplane and a data point that is the closest to it. When you maximize the margin of separation, the decision surface is classified as the optimal hyperplane.

On problems such as this with nonlinearly separable data, the optimal hyperplane is not present in the feature space. To improve find the solution to this problem the kernel trick is deployed. The kernel trick "translate" the feature space to one where the data is linearly-separable and make the optimal hyperplane creation possible. Even then, when the algorithm is learning and the dataset is too varied there's a need for a soft margin of separation, it is governed by the hinge loss function.

Also, there is the "C" parameter that penalizes the model for misclassifying the categories. A high value of "C" would mean the model would reduce the margin of separation. On the opposite side, a low value of "C" (lower than 1) increase the margin of separation.

In a multi-class classification, the strategy used is a one-vs-rest classification. This means that each class has a classifier.

Decision Trees (DecisionTreeClassifier[3])

By learning simple decision rules, the algorithm generates a tree that can be used for regression and classification[4]. This tree method partitions the feature space into a set of rectangles and fit a simple model in each one. In a simple binary tree a partition means that a node from the tree splits into two branches. These two branches them can split into four until a whole tree is formed from the training data. The quality of a split is usually measured using the Gini Impurity or Information gain. Gini Impurity measures the probability of a randomly chosen class being incorrectly labeled and Information gain measures the decrease of randomness after a certain step is taken.

When a whole tree is created means the training has stopped. The final leaf of a tree determines the result of the classification problem, it outputs the result of an item following that path of the tree.

[2] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[3] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[4] http://scikit-learn.org/stable/modules/tree.html

Naive Bayes (MultinominalNB[5])

The Naive Bayes is a supervised learning algorithm that utilizes Bayes' Rule[6] and assumes that all features are independent. Bayes's Rule relates the prior probability of an event with the posterior probability of an event given a conditioning. When used with the assumption that everything is independent that simplifies the system and makes the probability of an event given another be the probability of each event multiplied. This reduces the processing power exponentially because all you are doing is multiplication. Despite this "naive" assumption Naive Bayes learns very well and it is faster than ohter machine learning algorithms.

Specifically for this project which has multiple classification classes, the probabilities are estimated with a multinomial distribution. The distribution is parametrized by vectors for each class, the size of the vector is given by the number of features and where each item in the vector represents the probability of that specific feature being present in a given class.

Neural Networks (MLPClassifier[7])

Artificial Neural Networks are a algorithm that intend to imitate the brain neural networks. ANN's use a perceptron as an neuron unit. Each perceptron has an activation function that maps the weighted inputs to each output of the neuron. The activation function also needs to be differentiable and it is usually an rectified linear funtion. A differentiable function is necessary because the backpropagation algorithm that is used for training the network needs to differentiate it. Backpropagation is a technique that uses forward propagation to generate the initial outputs and back propagates updating the weights with a error signal generated by comparing the initial output with the desired response.

Grouping a number of neurons makes a layer of perceptrons and for a multilayer perceptron you just have to connect multiple layers. The middle layers that are not connected to either the input or the output are called hidden layers. Also, each neuron has a high degree of connectivity with the other layers.

Benchmark

The benchmark for this problem is the simplest classifier possible. This classifier will predict the cuisine that appear most in the dataset, which is 'Italian', to all recipes. This benchmark give us a 19.26% classification accuracy.

The another one that can be used as a benchmark is randomly choosing a cuisine as an answer. The result is 5.22% classification accuracy.

[5] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[6] H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.
[7] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Methodology

Data Preprocessing

The dataset was divided between "cuisine" and "ingredients". Cuisine was already ready for the implementation. The main problem is to format and preprocess a recipe to have all ingredients uniformly formatted throughout all the dataset.

The steps of pre-processing were:

1. Check and remove all non-letters characters from the recipes using regular expressions.
2. Adjust the words to make all the letters lowercase.
3. Remove modifiers from the ingredients to make it represent the same ingredient. For example, some recipes might come with "fresh cilantro" and others have "fresh chopped cilantro", but there's no difference between these two ingredients. What we need is "cilantro". To do that I search for the 100 most common ingredients and analyse if they have these "modifiers" to remove them.
4. The last step is to apply a lemmatization process. The *lemma* is the dictionary form of a word, so the lemmatization process tries to change words to its dictionary form. This way they can be analysed as a single word. For example "eggs" and "egg", the lemmatizer removes the "s" at the end makes them both "egg", the dictionary form.I used the NLTK WordNetLemmatizer[8] to do it.

After processing the ingredients to make them uniform. The next step is transforming this large quantity of strings into data that can be inputted in the learning algorithms to be learned. The way to extract information from this bunch of strings if by using TF-IDF[9](Term frequency-inverse document frequency). TF-IDF is a method to weight the value of words according to the number of times it appear in the text and offset it by the number of times it appears overall in the corpus. This method seems to fit since the number of times an ingredient appears in the recipe of a specific cuisine could determine if that ingredient is fundamental to that cuisine. The output of the TfidfVetorizer is a count matrix that can be used as an input for the learner. A major problem when using the TF-IDF vectorizer was transforming the training and test dataset separately. They never had the same size since the ingredients were different between them. The solution was concatenate both datasets into one and create the a count matrix for them both.

The last thing before implementation is to encode the cuisine labels to be numeric with a LabelEncoder[10].

[8] http://www.nltk.org/api/nltk.stem.html#module-nltk.stem
[9] https://en.wikipedia.org/wiki/Tf–idf
[10] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

Implementation

Since the test data can only be evaluated by kaggle website. The initial testing was done by a train-test-split with two-thirds of the training dataset used for training and the remaining one-third for testing.

The implementation of the supervised learning algorithms is done initially with the default parameters for every algorithm. Also, together with the classification accuracy score, precision score, recall and f-1 score, I also calculated the time it took to train each one. All four different models tried were from the sklearn module from python.

For the SVM classifier I used the LinearSVC class inside the svm module in sklearn. This implements a support vector classifier that uses the liblinear C library. Compared to the SVC that uses the libsvm C library, liblinear is faster and more flexible in the choice of the C parameter and the loss functions. Also, LinearSVC permits multi-class classification utilizing a one-vs-rest strategy that fits one classifier per class. The penalty hyperparameter "C" is 1.0 by default and the loss function is 'squared_hinge'.

For the Decision Tree classifier I used the DecisionTreeClassifier class inside the tree module in sklearn. The default settings utilize the "Gini Impurity" for the quality splitting of a branch. With the minimal sample split being 2 by default it increases the depth of the decision tree until there are only 2 samples to split. This tends to overfit the model.

For Naive Bayes classifier I used the MultinomialNB class inside the naive_bayes module in sklearn. This class allows the Naive Bayes to be used as a classifier of multiple categories using a multinomial distribution. The default smoothing parameter (alpha) is 1.0. It also defaults to learn a prior class probability and not receive it.

For Multilayer Perceptron I used the MLPClassifier class inside the neural_network module in sklearn. This was the one where I didn’t use the default layer option because I did not want a single layer perceptron. I changed for a two hidden layer configuration so that it could be a proper MLP. Other than that, I did not change the parameters so this means that the activation function is a rectified linear unit and with a constant learning rate.

During the implementation I tried a number of times to use a dimensionality reduction technique like PCA and TruncatedSVD. The results were not on par with their counterparts utilizing the full feature set. PCA only captured around 2% of the variance for each principal component so the solution was stop using it.

Refinement

The model that got the best classification accuracy in the last section is refined in this section. The methods I chose for the refinement of the model were tuning the hyperparameters using GridsearchCV[11]. This means an exhaustive search over multiple parameters in a 5-fold cross-validated dataset. After that, using this tuned classifier I use it as a meta-estimator in a

[11] http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html

BaggingClassifier[12]. The bagging classifier will use the tuned model as a base estimator and use multiple estimators to random sections of the data.

[12] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Results

Model Evaluation and Validation

After the tests, the results for the models are:

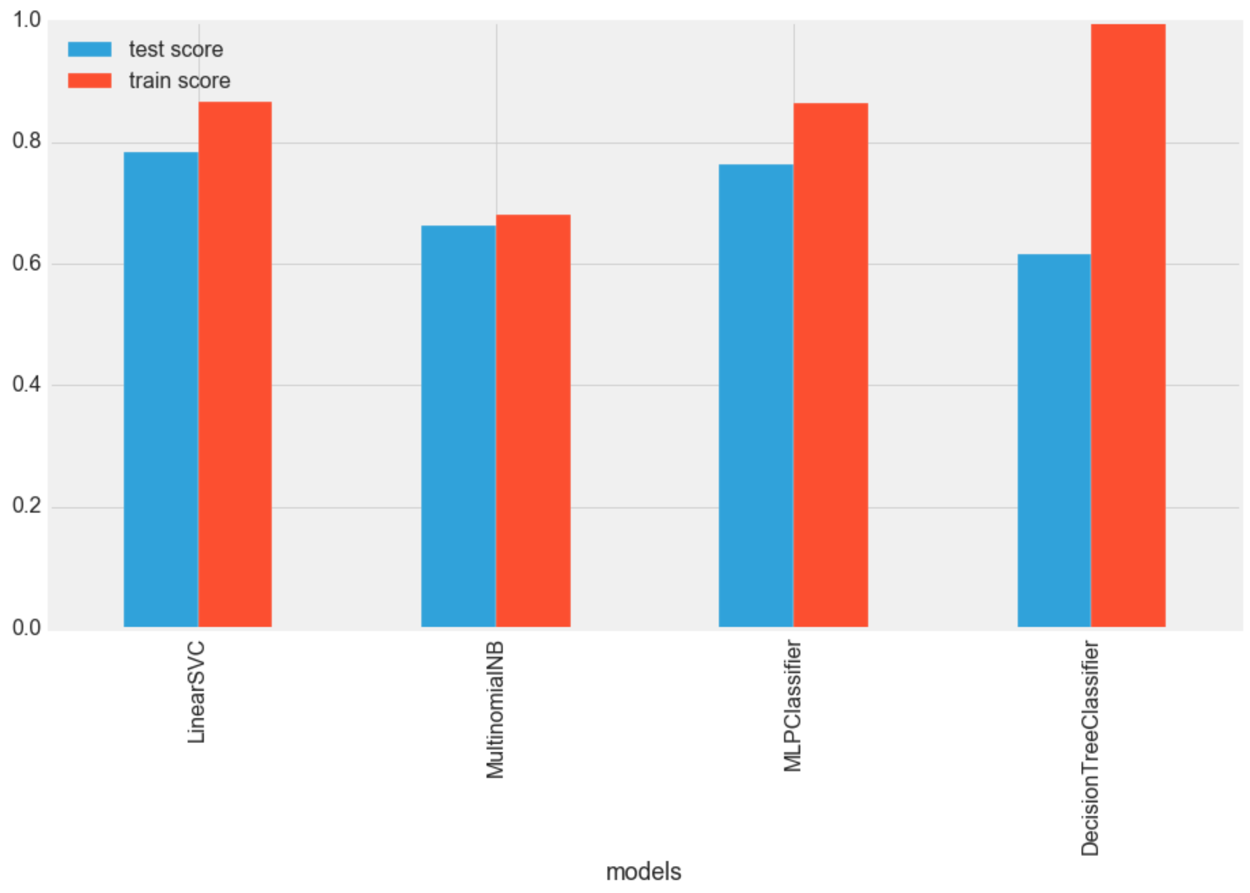


Image 5: Train and Test score plot comparison

After looking at the plot above we can see that LinearSVC and MLPClassifier are the ones the look the best in terms of test score.

The DecisionTreeClassifier has a training score of almost 1, that could mean that the model is overfitted in this configuration and why the test score is low compared to the other two models.

MultinomialNB does not have a performance comparable to the other models, it is not the result that was expected since Naive Bayes is usually used for text processing. It could be the features are too varied for it to learn and the assumption that the features are independent is not sufficient for training.

Before choosing a model to refine and tune, I needed to evaluate the other metric chosen from the beginning. The comparison between training time is showed below:

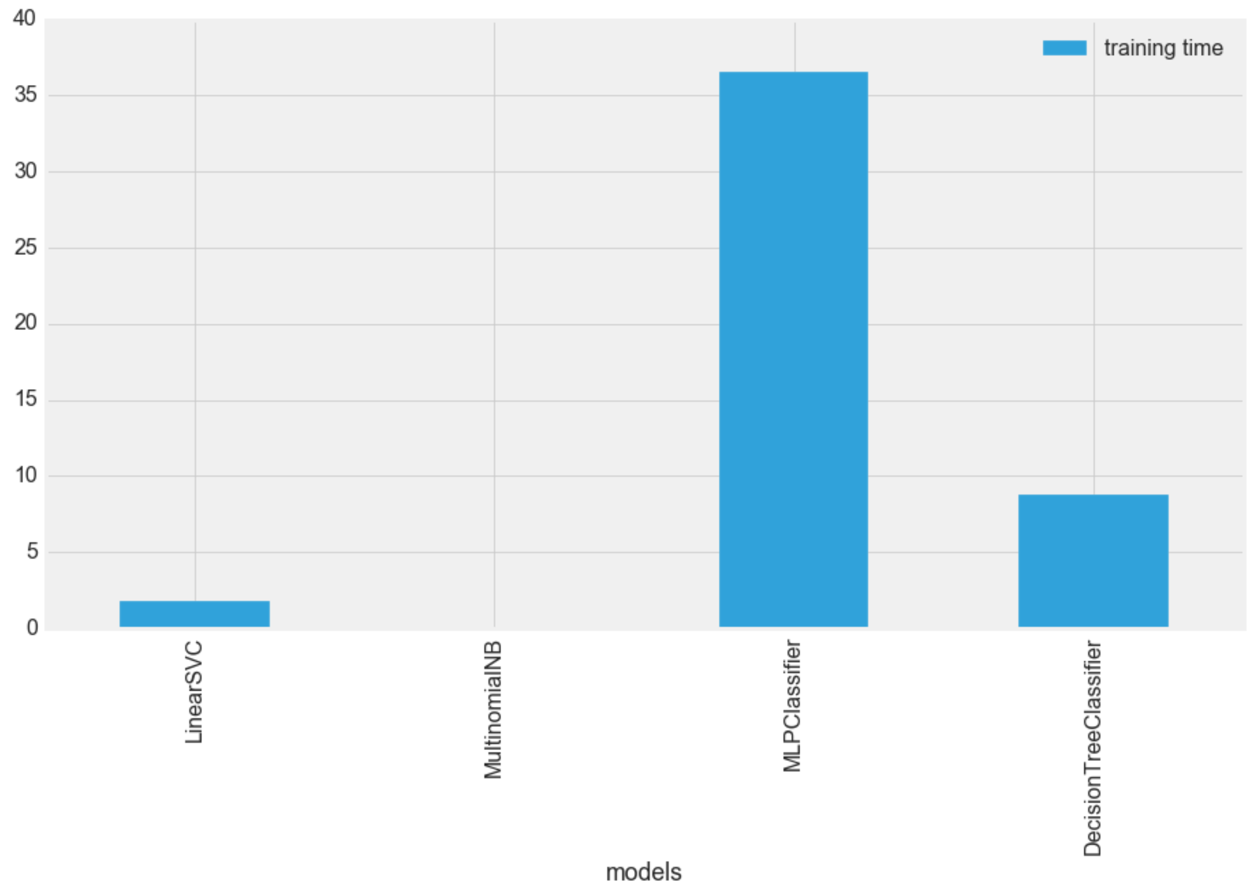


Image 6: Training time for the models

The training time leaves no doubt that the LinearSVC should be chosen for tuning. It is almost 20x faster than the MLPClassifier.

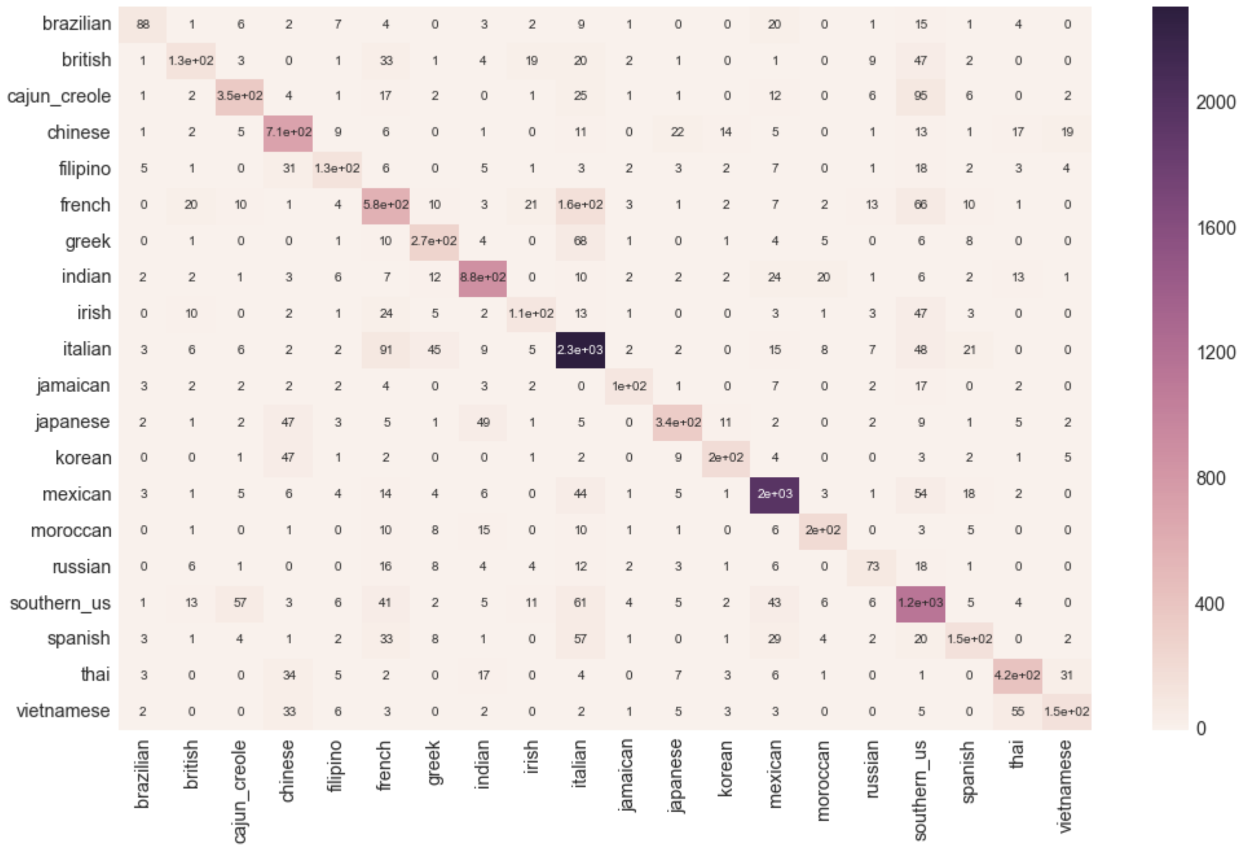
To tune the classifier I chose two hyperparameters that could influence the problem. First I chose to tune the "C" parameter, "C" represents the penalty for misclassifying a label. High values of C will make the model much more granular and penalize a lot for misclassification. Considering there are 20 classes in our model and the feature space is very vast I chose to tune it with low values below 1. These low values will produce a larger margin for error but consequently improve our score. Second I chose to increase the number of iterations the classifier makes. The current training score is around 0.9 so I thought by increasing the number of iterations it could improve model fitting and still don't overfit. The values utilized are below.

C	0.001	0.01	0.1	**0.5**	1
max_iterations	500	**1000**	2000	5000	10000

Table 2: Values utilized on the gridsearch

The best estimator was found to be one with "C" equals to 0.5 and "max_iterations" equals to 1000. It improved the training score from 0.7843 to 0.7857.

The last step is to make a bagging classifier utilizing the optimized classifier. While using BaggingClassifier[13] multiple times with different parameters I found that the default values are the highest that they can give and it gives the same performance of the original classifier without bagging.



	brazilian	british	cajun_creole	chinese	filipino	french	greek	indian	irish	italian	jamaican	japanese	korean	mexican	moroccan	russian	southern_us	spanish	thai	vietnamese
brazilian	88	1	6	2	7	4	0	3	2	9	1	0	0	20	0	1	15	1	4	0
british	1	1.3e+02	3	0	1	33	1	4	19	20	2	1	0	1	0	9	47	2	0	0
cajun_creole	1	2	3.5e+02	4	1	17	2	0	1	25	1	1	0	12	0	6	95	6	0	2
chinese	1	2	5	7.1e+02	9	6	0	1	0	11	0	22	14	5	0	1	13	1	17	19
filipino	5	1	0	31	1.3e+02	6	0	5	1	3	2	3	2	7	0	1	18	2	3	4
french	0	20	10	1	4	5.8e+02	10	3	21	1.6e+02	3	1	2	7	2	13	66	10	1	0
greek	0	1	0	0	1	10	2.7e+02	4	0	68	1	0	1	4	5	0	6	8	0	0
indian	2	2	1	3	6	7	12	8.8e+02	0	10	2	2	2	24	20	1	6	2	13	1
irish	0	10	0	2	1	24	5	2	1.1e+02	13	1	0	0	3	1	3	47	3	0	0
italian	3	6	6	2	2	91	45	9	5	2.3e+03	2	2	0	15	8	7	48	21	0	0
jamaican	3	2	2	2	2	4	0	3	2	0	1e+02	1	0	7	0	2	17	0	2	0
japanese	2	1	2	47	3	5	1	49	1	5	0	3.4e+02	11	2	0	2	9	1	5	2
korean	0	0	1	47	1	2	0	0	1	2	0	9	2e+02	4	0	0	3	2	1	5
mexican	3	1	5	6	4	14	4	6	0	44	1	5	1	2e+03	3	1	54	18	2	0
moroccan	0	1	0	1	0	10	8	15	0	10	1	1	0	6	2e+02	0	3	5	0	0
russian	0	6	1	0	0	16	8	4	4	12	2	3	1	6	0	73	18	1	0	0
southern_us	1	13	57	3	6	41	2	5	11	61	4	5	2	43	6	6	1.2e+03	5	4	0
spanish	3	1	4	1	2	33	8	1	0	57	1	0	1	29	4	2	20	1.5e+02	0	2
thai	3	0	0	34	5	2	0	17	0	4	0	7	3	6	1	0	1	0	4.2e+02	31
vietnamese	2	0	0	33	6	3	0	2	0	2	1	5	3	3	0	0	5	0	55	1.5e+02

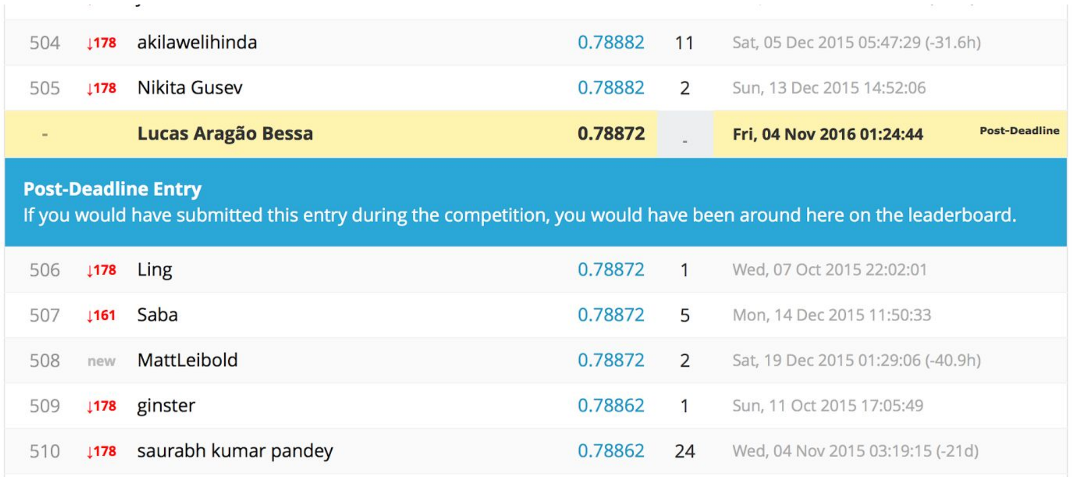
Image 7: Confusion matrix of the last classifier

We can see with some prior cooking knowledge that the higher number of misclassifications happen with similar cuisines. For example, different branches of asian cuisine (chinese, korean, japanese, indian) got misclassified between them in higher quantities than with other cuisines. Also, european cuisine (French and Italian) cuisine got the highest misclassification rate, around 160 french dishes had a false negative as an Italian dish. It also happened with cajun_creole and southern_us that are branches of the US cuisine. This shows that the model got "confused" when the geographic locations and cooking practices were similar between cuisines.

After all the results showed in this section, the final evaluation is the predicting the original test set that can only be tested doing a submission in the kaggle website. For the final

[13] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

submission we can train the classifier with the whole training dataset since we don't need to optimize it anymore. After creating a "submission.csv" output and uploading for submission the final score is 0.78872.



504	↓178	akilawelihinda	0.78882	11	Sat, 05 Dec 2015 05:47:29 (-31.6h)
505	↓178	Nikita Gusev	0.78882	2	Sun, 13 Dec 2015 14:52:06
-		**Lucas Aragão Bessa**	**0.78872**	-	**Fri, 04 Nov 2016 01:24:44** Post-Deadline

Post-Deadline Entry
If you would have submitted this entry during the competition, you would have been around here on the leaderboard.

506	↓178	Ling	0.78872	1	Wed, 07 Oct 2015 22:02:01
507	↓161	Saba	0.78872	5	Mon, 14 Dec 2015 11:50:33
508	new	MattLeibold	0.78872	2	Sat, 19 Dec 2015 01:29:06 (-40.9h)
509	↓178	ginster	0.78862	1	Sun, 11 Oct 2015 17:05:49
510	↓178	saurabh kumar pandey	0.78862	24	Wed, 04 Nov 2015 03:19:15 (-21d)

Image 8: Score and position in the competition

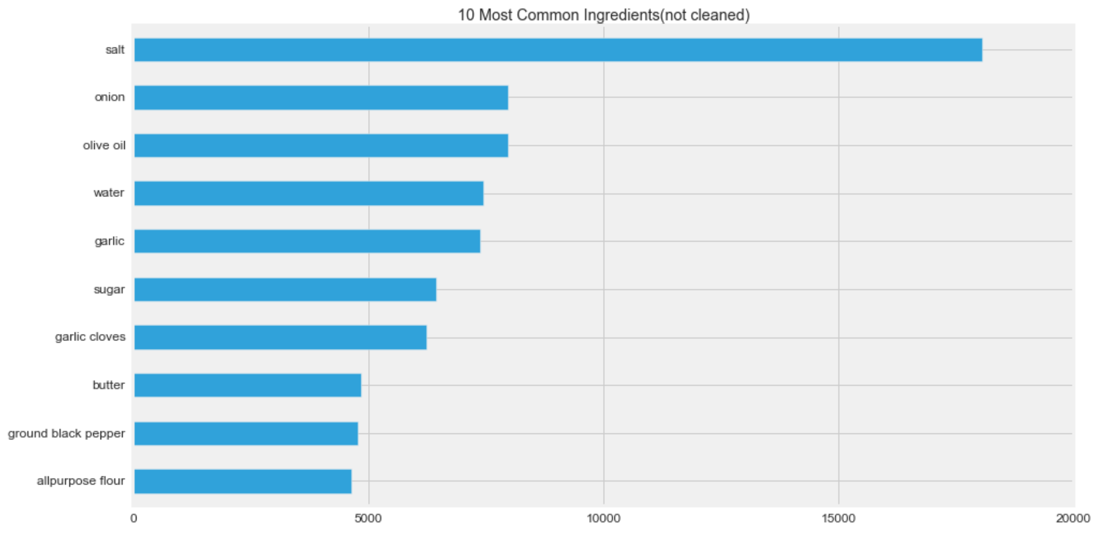
Justification

The final result is excellent relative to the benchmark of 0.1926(the same benchmark of Kaggle competition). The model classifies correctly around 79% of the recipes with a fast training time.This puts my result 4% accuracy under the top performer that had around 83%. We do not know the winning code but usually Kaggle winners use a lot of processing power. So being at 4% distance of the top performer is a good achievement.

Since misclassifying the recipes does not have high stakes, this model could perfectly be used to recommend tags of recipes in the Yummly website.

Conclusion

Free-Form Visualization



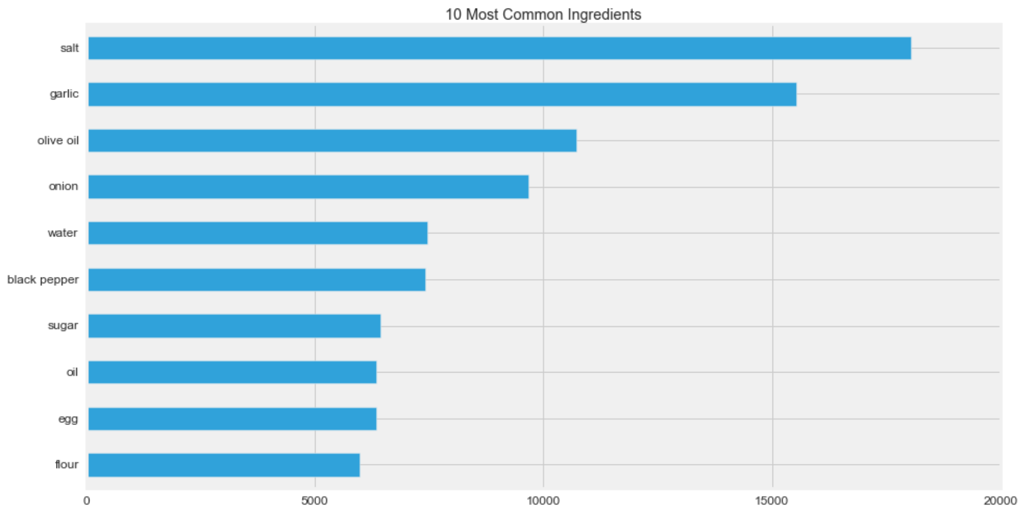


Image 9: Comparison between the most common ingredients with a cleaned dataset and a not cleaned dataset

The plot informs the importance of cleaning the ingredients data by removing unwanted terms from them. We can see that the first graph shows 2 instances of "garlic", them being "garlic" and "garlic cloves". It also diminishes the importance of certain products such as "olive oil" that could be "extravirgin olive oil".

From these two charts I can infer that data cleaning is the most important aspect when dealing with text analysis. Without proper preprocessing, all the data that you have is useless.

Studies show that 80% of the time in a project is spent cleaning the data[14]. If your model is not working properly great chances that the data is not processed correctly.

Reflection

In this project I had a dataset with around 40 thousand labeled recipes. These recipes had to be used to train a model to classify which cuisine a dish belongs to with only the list of ingredients. The steps were simple, first I cleaned the data to create a dataset that lists the ingredients correctly. Next, I trained different classifiers to find the best suited for this type of data. The support vector classifier was chosen for its performance and training time. Finally, I did a Grid search on two of the hyperparameters of the SVC to improve the performance. For the final submission I trained the optimized classifier on the whole training dataset and submitted the file to kaggle for analysis. The result was a model with around 79% of categorization accuracy.

This project was a good example of a very diverse dataset with an interesting purpose. Despite being a simple string classification and not a pages size one, this approach showed us that machine learning techniques can perform very well in text classification. This type of text classification can be applied to various fields of study.

One of the most interesting things in this project was checking the ingredients to see if they repeated and modifying them so that they were uniformly analysed. Other interesting things was that the dataset was so varied that while doing a PCA analysis it showed that each component only explained around 2% of the variance. This shows that text is a high-dimensionality problem that can't be reduced efficiently.

Improvement

The model could be improved with a larger dataset. A higher number of recipes would improve specially for a neural network classifier since it improves highly from larger quantities of data.

What could also be done is utilize prior cooking knowledge to pre-classify combinations of ingredients in with a higher probability for certain types of cuisine. For example, sweet ingredients together could mean a dessert, so this relation could be explored by favoring cuisines that have more dessert than others.

One final improvement would be having the name of the dish together with the ingredients. This could give us one more feature set to work with and improve the performance.

14
http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?_r=0