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. 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.

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

The highest score is not always the best way to measure if a technique is better than the other and since the competition is already over I also considered the training time to be an important variable in evaluating the model.

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

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.

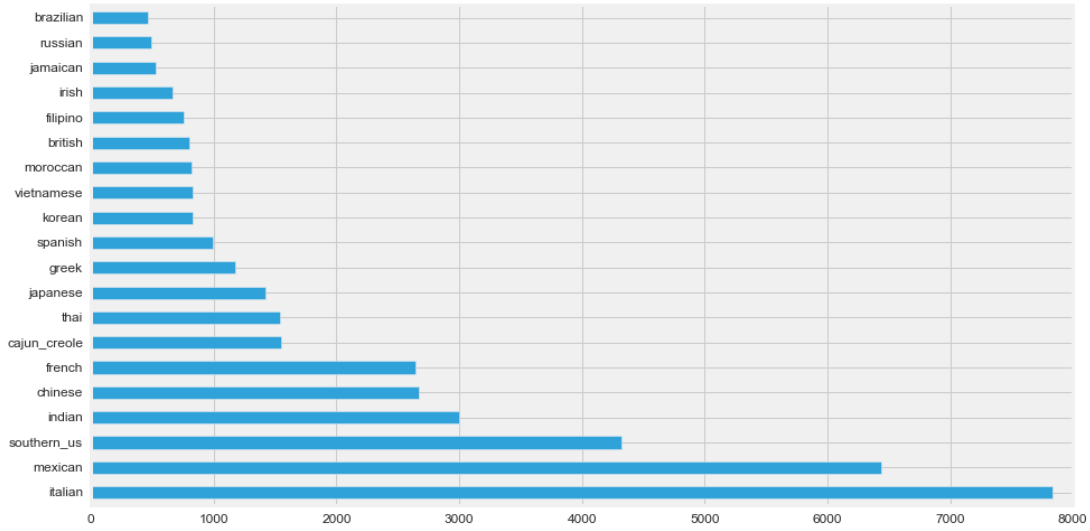


Image 1: Recipes per cuisine

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.

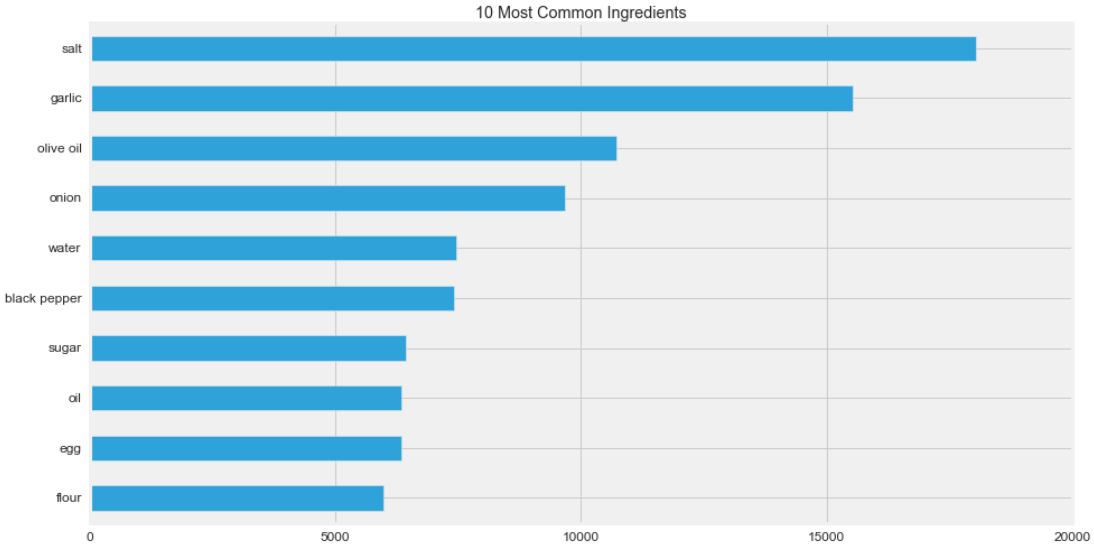


Image 2: The 10 most common ingredients

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.

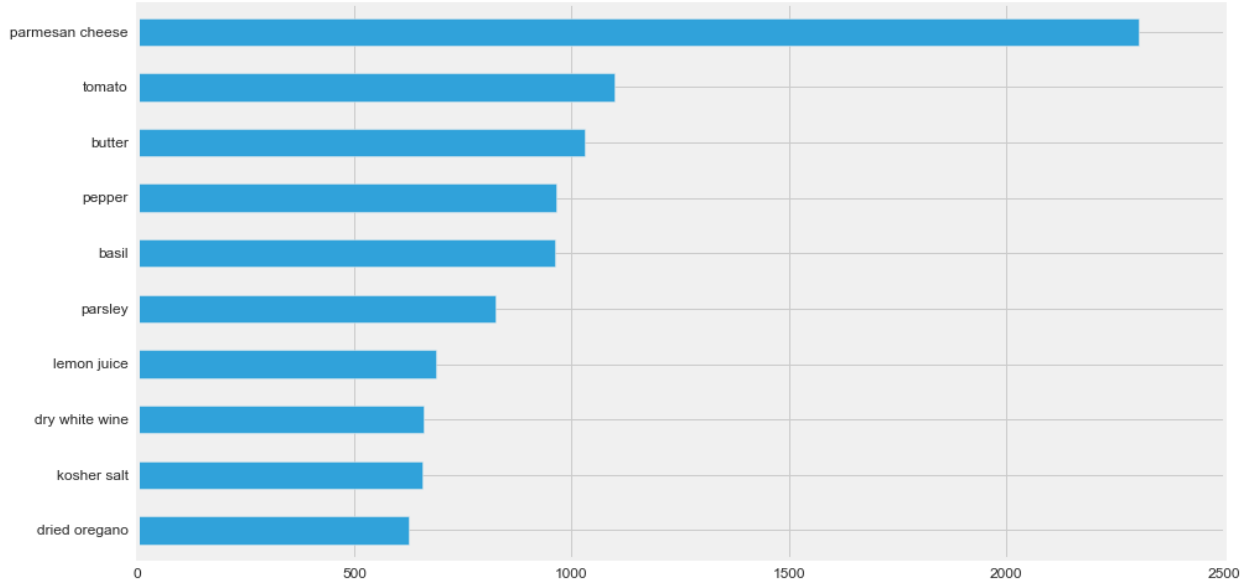


Image 3: Italian cuisine most common ingredients

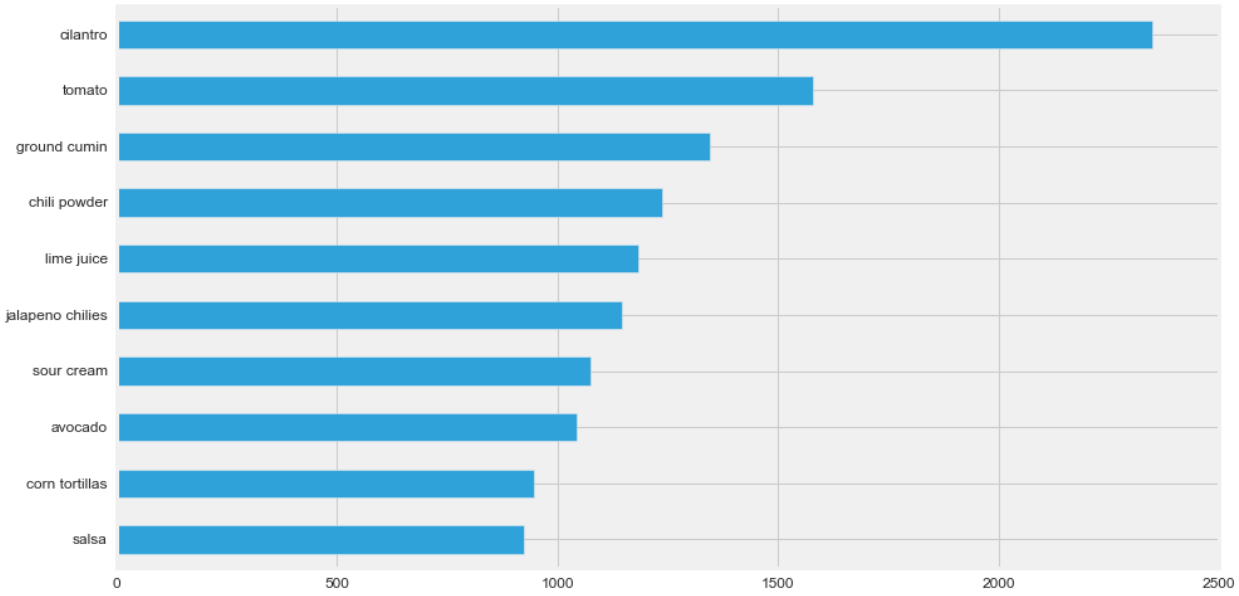


Image 4: Mexican cuisine most common ingredients

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. It constructs a hyperplane where it's possible to create linearly separable data by transforming the feature space[3]. In this analysis, I used a linear kernel because it is less computationally intensive. LinearSVC permits multi-class classification utilizing a one-vs-rest strategy that fits one classifier per class.
- Decision Trees (DecisionTreeClassifier[4]) - By learning simple decision rules, the algorithm generates a tree that can be used for regression and classification[5]. Based on the input features it classifies the target. The DecisionTreeClassifier is a classifier capable of multi-class classification.

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

- Naive Bayes (MultinominalNB[6]) - The Naive Bayes is a supervised learning algorithm that utilizes Bayes' Rule[7] and assumes that all features are independent. Is a fast algorithm that is historically used for SPAM protection. MultinomialNB is a multi-class classifier implementation of Naive Bayes.
- Neural Networks (MLPClassifier[8]) - Artificial Neural Networks are a algorithm that intend to imitate the brain neural networks. It learns a function by calculating the weights in multiple layers of "neurons"(perceptrons). It can be used for either classification or regression. MLPClassifier implements a multi-layered perceptron that is trained using the backpropagation algorithm and it supports multiclass classification by applying the softmax[9] as the output function.

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.

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 "large", "fresh", "minced" or "chopped".
4. The last step is to apply a lemmatization process. This process intends to group together the different inflected from of words so they can be analysed as a single item[10]. I used the nltk WordNetLemmatizer 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.

---

[6] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[7] H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

[8] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[9] https://en.wikipedia.org/wiki/Softmax_function

[10] *Collins English Dictionary*, entry for "lemmatise"

The way to extract information from this bunch of strings if by using TF-IDF[11](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.

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

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 I also calculated the time it took to train each one. All four different models tried were from the sklearn module from python.

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

[11] https://en.wikipedia.org/wiki/Tf–idf

Results

Model Evaluation and Validation

After the tests, the results for the models are:

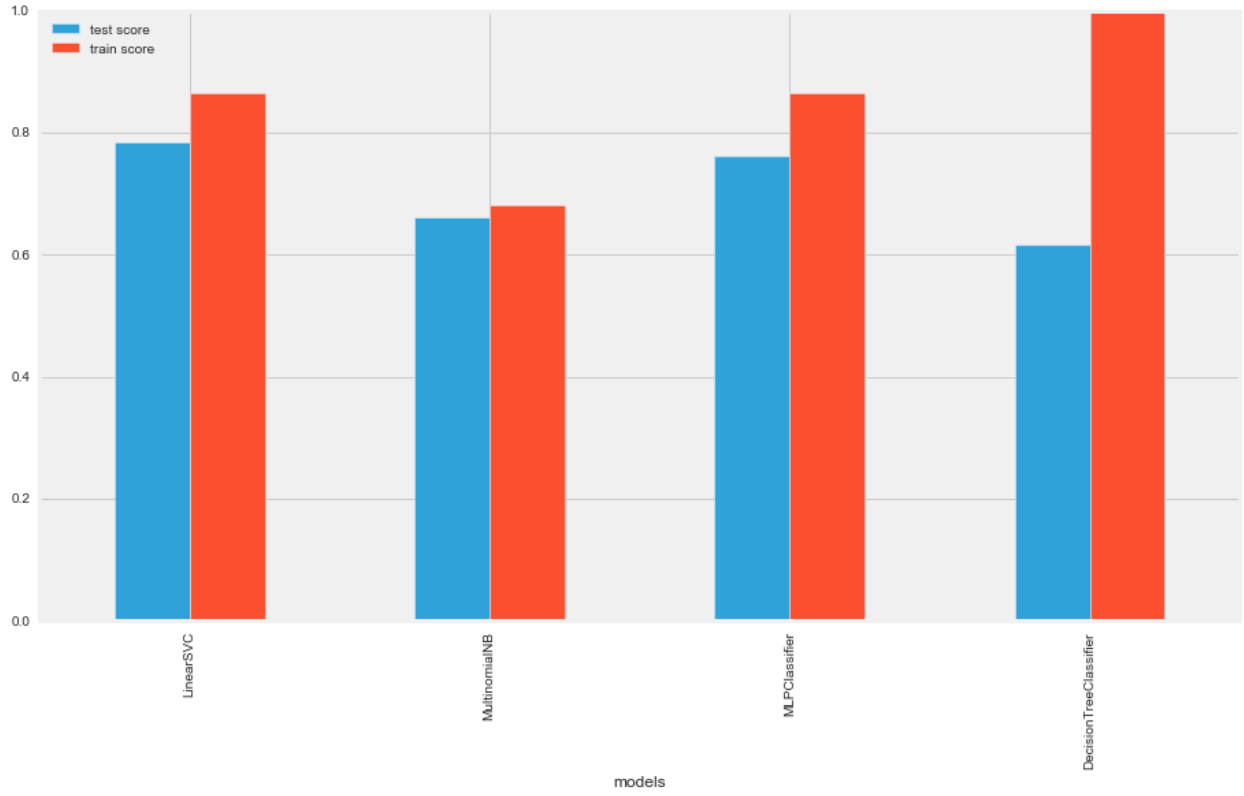


Image 5: Train and Test score plot comparison

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:

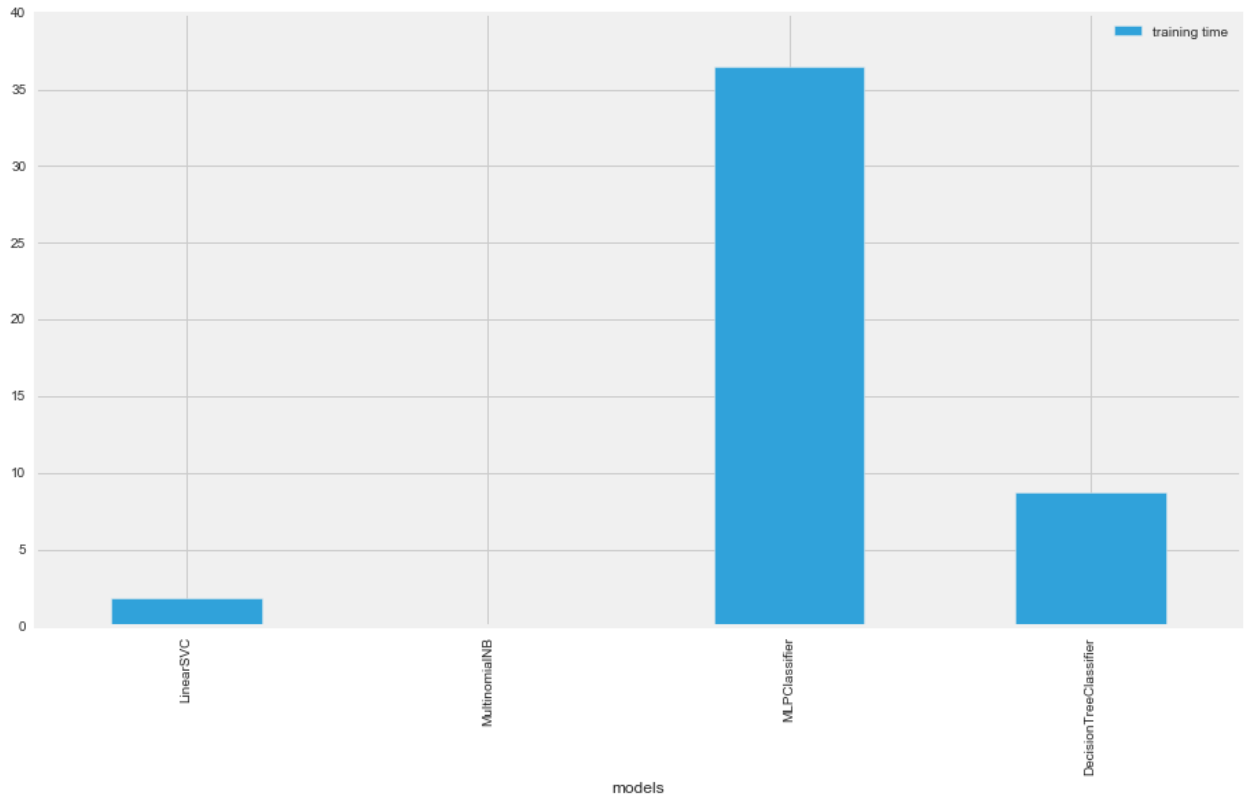


Image 6: Training time for the models

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.

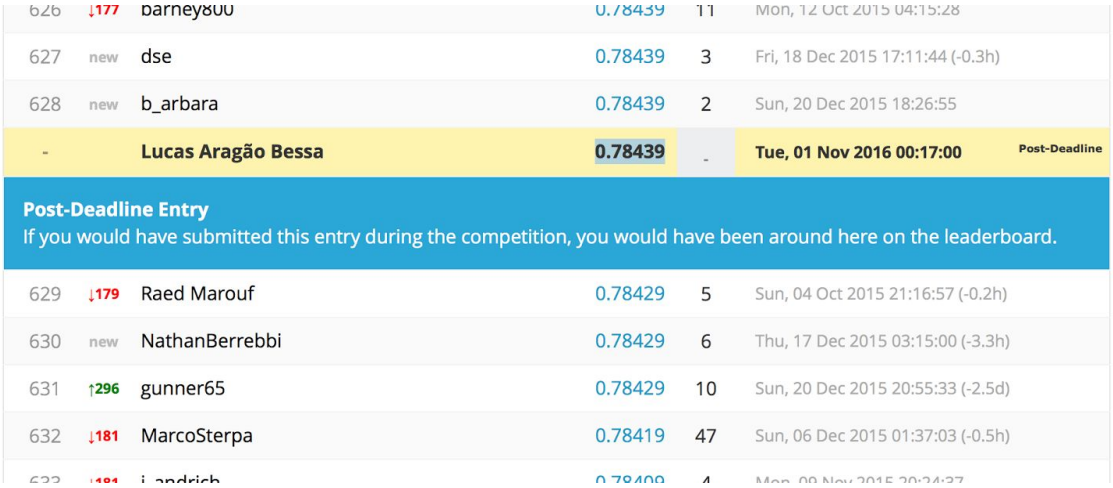
C	0.001	0.01	0.1	**0.5**	1
max_iterations	500	**1000**	2000	5000	10000

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[12] 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.

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. After creating a "submission.csv" output and uploading for submission the final score is 0.78439.



626	↓177	barney800	0.78439	11	Mon, 12 Oct 2015 04:15:28	
627	new	dse	0.78439	3	Fri, 18 Dec 2015 17:11:44 (-0.3h)	
628	new	b_arbara	0.78439	2	Sun, 20 Dec 2015 18:26:55	
-		**Lucas Aragão Bessa**	**0.78439**	-	**Tue, 01 Nov 2016 00:17:00**	**Post-Deadline**

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

629	↓179	Raed Marouf	0.78429	5	Sun, 04 Oct 2015 21:16:57 (-0.2h)
630	new	NathanBerrebbi	0.78429	6	Thu, 17 Dec 2015 03:15:00 (-3.3h)
631	↑296	gunner65	0.78429	10	Sun, 20 Dec 2015 20:55:33 (-2.5d)
632	↓181	MarcoSterpa	0.78419	47	Sun, 06 Dec 2015 01:37:03 (-0.5h)
633	↓181	j_andrich	0.78409	4	Mon, 09 Nov 2015 20:24:37

Image 7: Score and position in the competition

Justification

The final result is excellent relative to the benchmark of 0.1926. The model classifies correctly around 78% of the recipes with a fast training time. Since misclassifying the recipes does not have high stakes, this model could perfectly be used to recommend tags of recipes in the Yummly website.

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

Conclusion

Free-Form Visualization

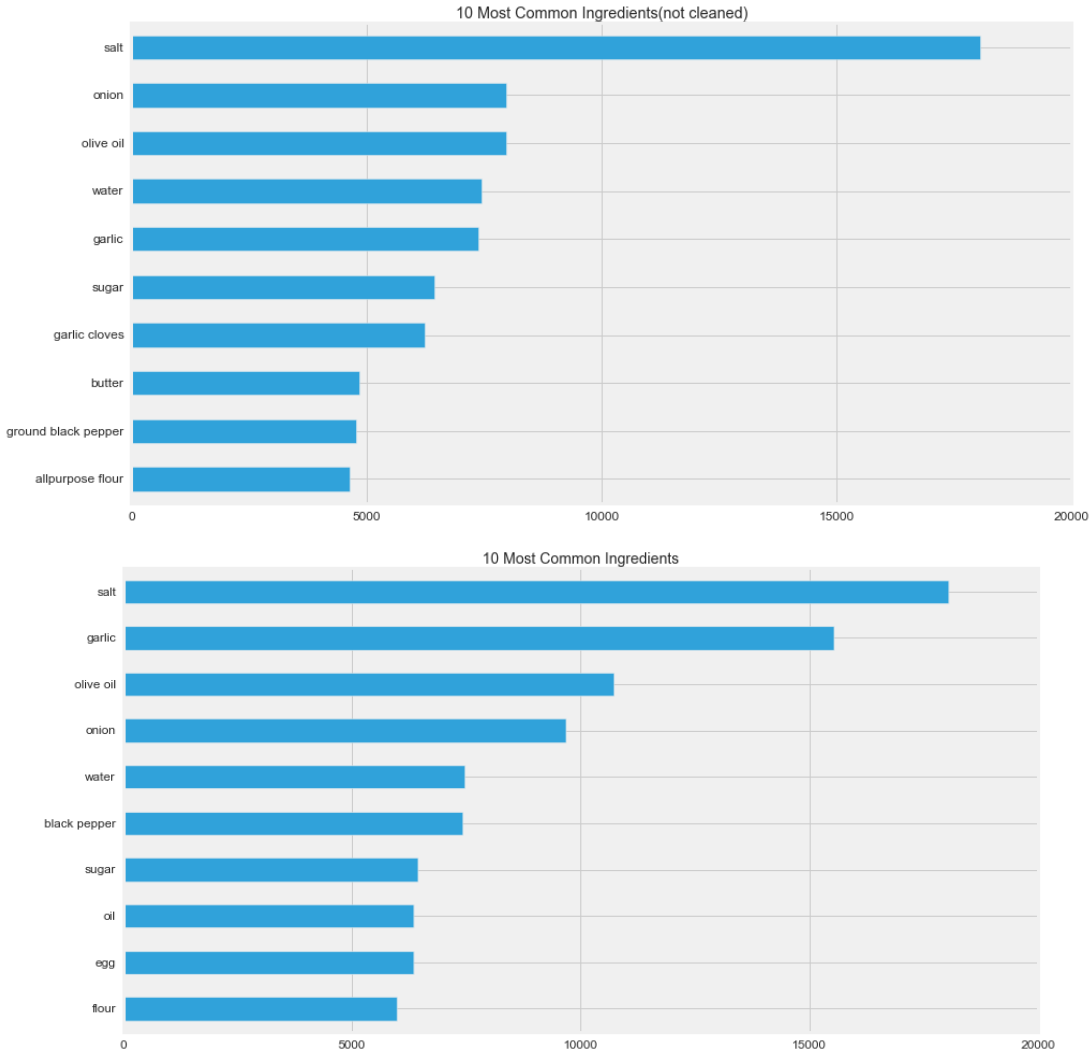


Image 8: Comparison between the most common ingredients with a cleaned dataset and a not cleaned dataset

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[13]. If your model is not working properly great chances that the data is not processed correctly.

Reflection

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.

---

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