Final Project - What's Cooking?

Kasyap Challapalli & Adarsh Kulkarni (On-campus students)

15 December 2015

Contents

---

1 Abstract

There is nothing better than home-cooked food. While it may not be exceptional, people are strongly associated with their home-town cuisines or with their country's cuisines. It gives a chance to identify themselves and experience that "home sweet home" feeling every time they taste something from their own cuisine. Some enthusiasts even try to replicate that feeling and try to cook on their own and maybe invent something new as well. But how would someone decide what cuisine he/she is cooking? We have a solution. Based on the ingredients used for cooking the recipe, our algorithm can predict what cuisine the dish is going to be.

2 Introduction

2.1 Motivation

This project is part of the ”Kaggle” Competition Playground which is a platform to test our Machine Learning Skills and compete against the entire world. Also, there is no language restriction for this competition, which makes this competition even tougher. For this particular topic, we have analyzed the training dataset given by Kaggle and we observed that the number of features for each training sample are not the same, a similar trait we have dealt with in one of our previous assignments. The only exception in this case is that we have to handle ”string values” instead of ”numeric values” for the features.

Also, we do not know the accuracy of our algorithm until we make a submission to Kaggle and there is a limit of how many submissions we make every day. These factors meant that we had to be very accurate in making our assumptions and testing them and we cannot put in any random tweaks that we desire. All these factors motivated us to choose this particular competition.

2.2 Problem Statement

As put by Kaggle, our problem statement is ”This playground competitions asks you to predict the category of a dish’s cuisine given a list of its ingredients”. Our interpretation of the problem statement is to train our classifiers on the given data and predict the labels for the test data which, in our case are the cuisines and then post process the data to make it fit for a submission.

3 Approach to the Solution

The approach we followed to solve the problem can be summarized in Figure 1. The main steps involved in the approach is understanding how to pre-process the data and identify the algorithms suitable for the data. Then, we need to post-process the data in the format specified by Kaggle and submit the file to find the accuracy.

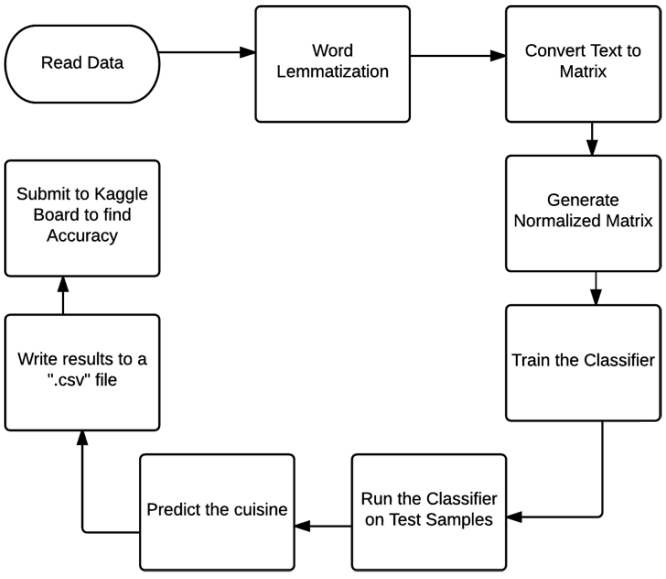


Figure 1: Approach to Solving the Problem

The following subsections summarize our pre-processing methodology.

3.1 Word Lemmatization

Word Lemmatization involves determination of the lemma for a given word. In our case, lemmatization will be helpful to reduce the chances of creating different features for the same ingredient. For example, one of the ingredients in a sample is 'Onion', there may be another sample with an ingredient 'Onions'. We do not want 'Onion' and 'Onions' to be different features. Word lemmatizing the dataset will change both to 'Onion'. The code for lemmatization is given below :

```
from nltk.stem.wordnet import WordNetLemmatizer
lmtzr = WordNetLemmatizer()
train_data['ingredients_string'] = [' '.join([lmtzr.lemmatize(re.sub('[^A-Za-z]', ' ', line))
 for line in lists]).encode('utf-8').strip() for lists in train_data['ingredients']]
test_data['ingredients_test'] = [' '.join([lmtzr.lemmatize(re.sub('[^A-Za-z]', ' ', line))
 for line in lists]).encode('utf-8').strip() for lists in test_ingredients]
```

3.2 Converting text to numerical data

Each training sample in the dataset consists of a set of ingredients and the cuisine. We need to convert these ingredients into feature vectors. The bag of words approach is used here, where an integer id is assigned to each word that occurs in the training set. The total number of features will be the number of distinct words in the corpus. The code snippet for the implementation is given below:

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer(max_features=3000)

X_train_counts = count_vect.fit_transform(ingredients_cuisine).toarray()
X_test_counts = count_vect.transform(test_words).toarray()
```

3.3 Normalized tf-idf representation

The matrix of numerical feature vectors generated in the previous step is transformed to a normalized term-frequency times inverse-document-frequency representation. This method will help in reducing the impact of tokens that occur very frequently in the matrix. These tokens also contain less information than the features that do not occur frequently in the dataset. The code snippet for TF-IDF representation is given below:

```
from sklearn.feature_extraction.text import TfidfTransformer

tf_transformer = TfidfTransformer()
X_train = tf_transformer.fit_transform(X_train_counts)
X_test = tf_transformer.transform(X_test_counts)
```

4 Methods

Before we decided the methods we need to use, we tried to analyze our problem. The goal was to solve a classification problem. Also, there were multiple classes (in our case, these were the cuisines that we needed to predict) which means we cannot stick to classifiers that can solve binary classification problems. Hence we decided to explore as many classification methods as possible that can deal with multiple classes. In other words, we had a multi-class classification problem to solve.

We decided to begin with the classification methods discussed in class, and also explore some variants which were not used or discussed so far. Owing to these factors, the following four methods were finally selected to deal with our classification problem.

4.1 Logistic Regression

Since we were dealing with multiple classes, we decided to use Multinomial Logistic Regression. This method is an extension of the regular Logistic Regression and is mostly used to explain the relation between one dependent variable which, in our case is the set of recipe ingredients and one or more independent variables which, in our case are the possible cuisines for the given set.

Logistic regression in our case would predict the probabilities for all the cuisines available based on the given ingredient set. Since all the cuisines cannot be used as possible labels for prediction, some labels are shortlisted. The labels in the short list are decided based on the training data. The predict method returns the cuisine with the highest probability as the predicted cuisine or predicted label. To improve the accuracy of the predictions even more, a grid search is performed over a set of tuning parameters and the best parameters are selected for the classifier. Implementation of Logistic Regression we used is shown in the appendix.

4.2 Support Vector Machines

SVM's are extremely helpful in classifying non-linearly separable and high dimensional data through the Kernel functions. The problem we need to solve also involves a high dimensional space because of the several number of classes involved. The Linear SVM uses the "One vs. Rest" method which is useful for multi-class classification. This will result in multiple classes as result each with a score. The class with the highest score is returned as the final result.

We used four kernels for solving our problem viz. Sigmoid Kernel, Polynomial Kernel, RBF Kernel and the Linear Kernel. The kernels help in transforming the data to a higher dimensional space so that it can be processed more easily. The SVM implementation we used is shown in the appendix.

4.3 Neural Networks

For a Multiclass Neural Network, we will have a separate class for each label we are trying to predict and in our case, we will have a separate class for each cuisine type. The network is then trained on the given training dataset. Based on this training it received, the network will now try to predict the classes/cuisines for the test data. For solving our problem, we have used the scikit-neuralnetwork library as we have not used this so far in our assignments. This particular library implements multi-layer perceptrons. And we are specifically using the "mlp" module. We have tried combinations of linear and non-linear activation functions in these two layers.

For the first Layer, we have varied the activation functions as "Tanh" and "Sigmoid" which are non-linear.For the second layer which is also the output layer, we had the option of using either a "Linear" or a "Softmax" activation function. Both these functions are linear and we have used the "Softmax" function because our problem is a classification problem. We have referenced the official documentation of the "scikit-neuralnetwork" library for all the choices we have used in our implementation. The implementation of our approach is shown in the appendix.

4.4 Tree Ensembles

Ensemble methods aim to combine the predictions of several base estimators to improve robustness of a single estimator. There are two methods of implementation. One of them is the averaging approach, an example of which is "Bagging". In this method, the raw classifier scores are converted to probabilities and then an average of the probabilities is calculated to determine the best classifier score. For our problem, instead of using the normal bagging approach, we have tried to implement "Random Forest" approach, which is a special case of Bagging.

Random Forest uses a modified tree algorithm which selects a random subsets of features at each split and this helps in reducing the correlation of the trees compared to the ordinary bagging method and increases the accuracy of classification. The forest will choose the classification having the most votes over all trees in the forest We chose this approach because we have not implemented this approach earlier. The Random Forest approach we implemented is shown in the appendix.

The other method we use is Gradient Boosting. Gradient boosting involves boosting the weak predictive models into strong models, by forming an ensemble of the weak models. The parameter "n_estimators" controls the

number of weak learners or regression trees that have to be boosted and converted to strong models. While Random Forest approach tries to reduce the variance in the results, Boosting on the other hand tries to reduce the bias effect on the results. Contrary to Random Forest which selects data randomly, Gradient Boosted Trees algorithm tries to find optimal linear combination of trees. A sample implementation of this Gradient Boosting Method is shown in the appendix.

5 Results

For our work, we do not have a code snippet which can evaluate the accuracy of our algorithm. Since this is a competition, we had to submit each result to the Kaggle website to know our accuracy score.

Table 1: Logistic Regression Results

Solver, Iteration	Accuracy
lbfgs, 50	78.66
lbfgs, 100	78.691
liblinear, 100	78.912
newton-cg, 100	78.932

Figure 2: Accuracy using Logistic Regression

From Figure 2, we can observe that the Logistic Regression reports the highest accuracy for ”newton-cg” solver over 100 iterations. The other solvers also come close but the newton-cg solver beats them all. The lbfgs solver is a limited memory version of the BFGS algorithm which stores only a few vectors that represent the approximation and hence gives lower accuracy than the other solvers.

Table 2: SVM Results

SVM Kernels	Accuracy
Sigmoid Kernel	19.268
Polynomial Kernel	19.268
RBF Kernel	71.5
Linear	78.942

Analysis of Figure 3 reveals that the Linear Kernel of the SVM reports the highest accuracy. In theory, the performance of the RBF kernel should be similar to the Linear SVM. In our case, we think that since the data is linearly

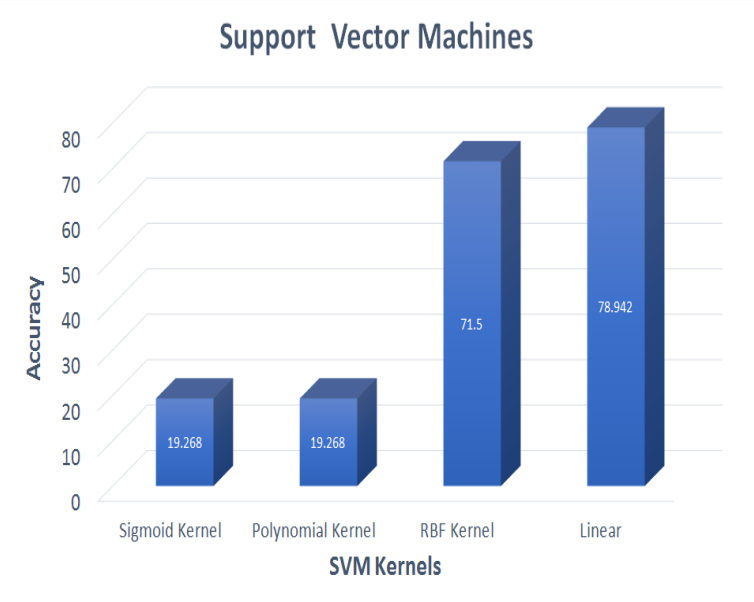
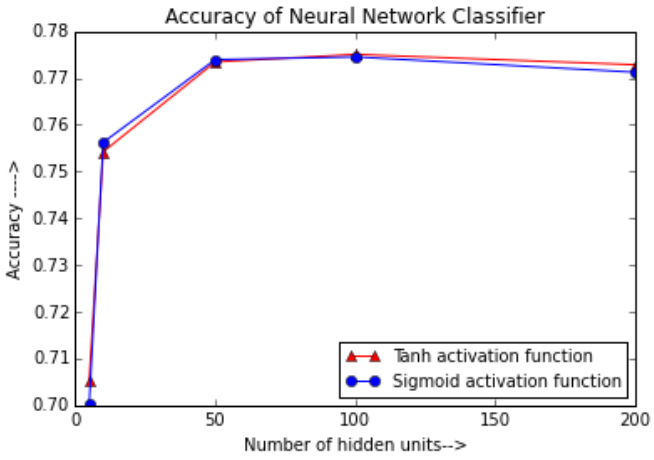


Figure 3: Accuracy using Support Vector Machines

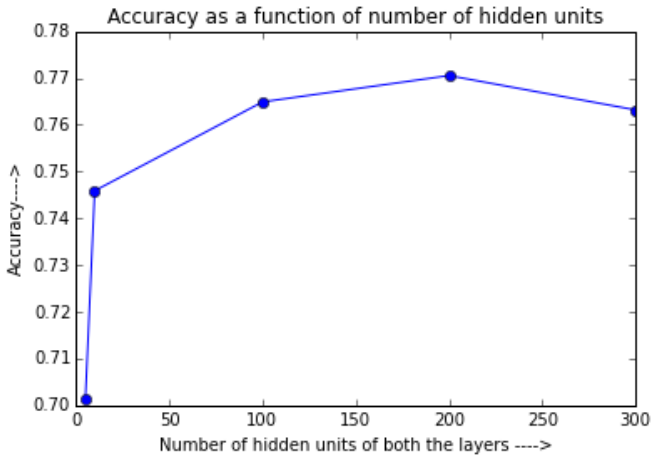
separable the RBF kernel exhibits overfitting and gives lower accuracy. RBF, Sigmoid and Polynomial kernels will exhibit higher accuracy if the data is not linearly separable and needs to be transformed into a higher dimension.

Table 3: Results for Neural Network

Results for One Hidden Layer			**Results for Two Hidden Layers**	
	Accuracy Results			**Accuracy Results**
No of Hidden units	**Tanh Activation**	**Sigmoid Activation**	**No of Hidden Units**	**Tanh Activation**
5	70.535	70.012	5	70.143
10	75.422	75.623	10	74.588
50	77.343	77.393	100	76.488
100	77.504	77.45	200	77.051
200	77.283	77.122	300	76.317



(a) Accuracy for Neural Networks with one hidden layer



(b) Accuracy for Neural Networks with two hidden layers

Figure 4: Accuracy plots for Neural Networks

For Neural Networks, we have used two networks - one network with just one hidden layer and the other one with two hidden layers. The results are shown in Figure 4.

Table 4: Results for Random Forest Algorithm

Random Forest Algorithm	
No. of Estimators	Accuracy
10	70.103
50	74.165
100	75.171
200	75.282
300	75.191
400	75.654
500	75.613
1000	75.654

Table 5: Results for Gradient Boost Algorithm

Gradient Boost Algorithm	
No. of Estimators	Accuracy
10	69.522
50	72.798
100	73.321
300	73.390
500	73.382

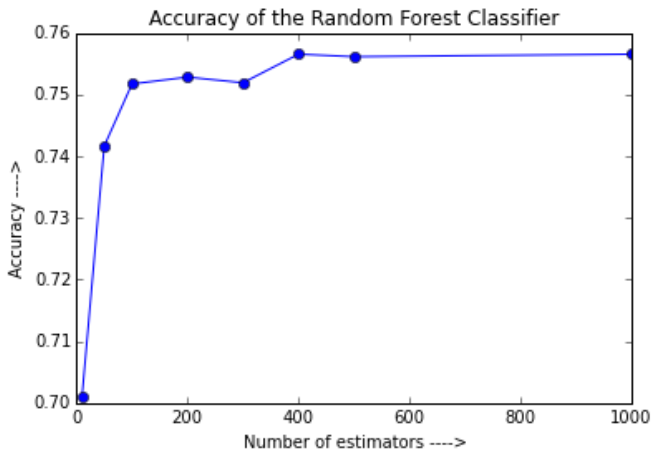
From Figure 4a, we can observe that both the activation functions ”Tanh” and ”Sigmoid” produce similar results and we have recorded the highest accuracy of 77.5% for neural networks using ”Tanh” function. Sigmoid resulted in its highest accuracy of 77.45% which is pretty close. Both these results have been obtained with 100 units in the hidden layer.

We can see that with the increase in the number of hidden units in a layer, the accuracy also of the classification also increases. This is because, as the number of hidden units increases the neural network is able to more closely approximate the data. Under-fitting is observed when the number of hidden units are very less, as the network cannot track the input data closely. When the number of hidden units is more than 100, over-fitting occurs resulting in decrease in accuracy.

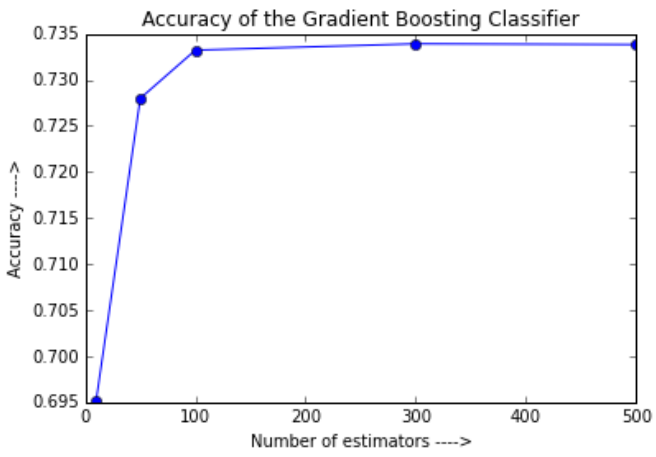
Figure 4b. represents the results of using two hidden layers in the network. For this network however, the highest accuracy of 77.4% was obtained with 200 hidden units in each layer. With increase in the number of hidden layers, there is a decrease in the effectiveness of the backpropagation algorithm. Using only one hidden layer provides good accuracy in most cases. The network can solve more complex problems if it has more hidden layers and units but it can also slow down the network and result in overfitting.

Figure 5 represents the accuracy results of using ensemble methods of Random Forest and Gradient Boost. Figure 5a represents the results of using Random Forest whereas Figure 5b represents the results of using Gradient Boosting.

With Random Forest, we have achieved the highest accuracy of 75.654% using 400 estimators. In case of Gradient Boost Classifier, we have achieved the highest accuracy of 73.4% using 300 estimators. From these results, we can safely conclude that for this particular dataset, Random Forest gives the best possible accuracy among the ensemble methods we used. The Gradient Boost classifier requires more careful tuning of parameters to get better accuracy than Random Forest. Without tuning, it is susceptible to overfitting which is observed in the results.



(a) Accuracy using Random Forest



(b) Accuracy using Gradient Boost

Figure 5: Accuracy plots for Random Forest and Gradient Boost Algorithm

6 Conclusions and Future Work

When we entered the competition, the top accuracy score on the Leader board was around 82.09%. We made this accuracy score our target and we have managed to reach close enough. As of the day we are submitting the report, the highest accuracy score is 82.85% and our own accuracy score is 78.94%. We came up pretty close and we have tried all possible methods we could think of in the given time frame. We have experimented on some methods like Gradient Boosting and explored some variants of the methods which were not tested in class assignments.

According to our results, for the given dataset, the best possible combination was that of using a Linear Kernel on a SVM. This particular combination resulted in our highest accuracy of 78.94%. As for the future work, we still have a few days left for the competition to end and we hope we can try to improve our accuracy score by at least 2%. We are still running some simulations, some of which have been running for 2 days straight. We hope such simulations might result in a better accuracy score.

Inspired from this competition, we would like to expand our work and try to predict a certain dish given the ingredients and their quantities. The main challenge in solving this problem would be the creation of a training dataset large enough to train the classifiers so that they can handle multiple features and multiple classes. Given time and resources, this would be our future course in this line of work.

7 Team member contributions

Kasyap Challapalli

- Processing data using Neural Networks and Logistic Regression
- Post-processing the data
- Graphic design of the Poster
- Sections 1, 2, 4.1, 4.3, 5 and 6 of the Report

Adarsh Kulkarni

- Pre-processing the data
- Processing the data using SVM classifier and Tree Ensembles
- Organization of content on the Poster
- Sections 3, 2, 4.2, 4.4, 5 and 6 of the Report

References

[1] Random Forest Algorithm
https://en.wikipedia.org/wiki/Random_forest

[2] Artificial Neural Network,
https://en.wikipedia.org/wiki/Artificial_neural_network

[3] Support Vector Machines,
https://www.quora.com/What-does-support-vector-machine-SVM-mean-in-laymans-terms

[4] Logistic Regression,
www.stat.cmu.edu/ cshalizi/uADA/12/lectures/ch12.pdf

[5] Sklearn - Working with text data,
http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

[6] Conduct and Interpret a Multinomial Logistic Regression - Statistics Solutions,
http://www.statisticssolutions.com/mlr

[7] sklearn.linear_model.LogisticRegression — scikit-learn 0.17 documentation,
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[8] Stata Data Analysis Examples: Multinomial Logistic Regression,
http://www.ats.ucla.edu/stat/stata/dae/mlogit.htm

[9] How to do multi class classification using Support Vector Machines (SVM) - Stack Overflow,
http://stackoverflow.com/questions/1958267/how-to-do-multi-class-classification-using-support-vector-machines-svm

[10] 1.4. Support Vector Machines — scikit-learn 0.17 documentation,
http://scikit-learn.org/stable/modules/svm.html

[11] Welcome to sknn's documentation! — scikit-neuralnetwork documentation,
http://scikit-neuralnetwork.readthedocs.org/en/latest/

[12] Building Decision Trees for the Multi-class Imbalance Problem, T. Ryan Hoens, Qi Qian, Nitesh V. Chawla, and Zhi-Hua Zhou,
https://www3.nd.edu/ dial/papers/PAKDD12.pdf

[13] 1.11. Ensemble methods — scikit-learn 0.17 documentation,
http://scikit-learn.org/stable/modules/ensemble.html

8 Appendix

8.1 Code for Logistic Regression

```
from sklearn.linear_model import LogisticRegression

parameters = {'C':[0.001, 0.01, 0.1, 1, 10, 100]}

clf = LogisticRegression()

classifier = grid_search.GridSearchCV(clf, parameters)
classifier=classifier.fit(X_train,cuisine)
predictions=classifier.predict(X_test)
```

8.2 Code for SVM

```
from sklearn.svm import LinearSVC
from sklearn.svm import SVC

parameters = {'degree':[1,2,3,4,5,6,7]}
C_range = np.logspace(-2, 2, 5)
gamma_range = np.logspace(-2, 2, 5)
param_grid = dict(gamma=gamma_range, C=C_range)

clf = SVC(kernel='rbf')
#clf = LinearSVC(max_iter=2000)

classifier = grid_search.GridSearchCV(clf, param_grid)
#classifier = grid_search.GridSearchCV(clf, param_grid=param_grid)
classifier=classifier.fit(X_train,cuisine)
predictions=classifier.predict(X_test)
```

8.3 Code for neural network

```
from sknn.mlp import Classifier, Layer
nn = Classifier(
   layers=[
      Layer("Tanh", units=300),Layer("Tanh", units=300),
      Layer("Softmax")],
   learning_rate=0.01,
   n_iter=30, regularize="L2")

nn.fit(X_train_counts, cuisine)
predictions = nn.predict(X_test_counts)
words_list = [' '.join(x) for x in predictions]
```

8.4 Code for Random Forest and Gradient Boost

```
from sklearn.ensemble import RandomForestClassifier

randomForest = RandomForestClassifier(n_estimators=40)
clf = randomForest.fit(X_train,cuisine)
cuisine_accuracy = randomForest.predict(X_test)
```

```
from sklearn.ensemble import GradientBoostingClassifier

boost = GradientBoostingClassifier(n_estimators=50,learning_rate=0.2)
clf = boost.fit(X_train,cuisine)
cuisine_accuracy = boost.predict(X_test)
```