ECS 171 Final Project: "What's Cooking"

Ge Cheng   Mazar Farran   Carl Glahn   Corbin Gomez   Weitang Liu
Kenan Nalbant   Alexander Parella   Michael Rea   Paul Salessi   Jiaming Xie

November 26th, 2015

1 Abstract

"What's Cooking" is a problem from *Kaggle* that concerns classifying different types of cuisine given a meal's ingredients. The training set that we will be using are samples of different meals that have an id number, a cuisine type, and an ingredients list; the test set will be similar but without the cuisine type, as this is the predicted field. To solve this multi-class classification problem, we used four different machine learning techniques: support vector machines, decision trees, naïve Bayes classifiers, and neural networks. Support vector machines granted the least amount of error followed by decision trees, then neural networks, and finally naïve Bayes classifiers. This study suggests that different machine learning algorithms will perform differently based on unique qualities of the given dataset.

2 Introduction

These machine learning algorithms fall within the realm of supervised learning, in that outputs with labels are present in the training set. Because these outputs are discreet, this problem is one of classification rather than one of regression. The data set of cuisine types and ingredients from *Kaggle* presents a situation in which feature values are also discreet—the ingredient is either present in the current meal or it is not. The amount of ingredients per meal is much lower than the amount of features; therefore, the matrix representing the dataset is extremely sparse—this is a problem that the algorithms need to handle in their computation. Solutions to these problems are significant because they will provide insight on the best ways to solve sparse, binary featured data sets.

This paper contributes findings on which classification methods tend to do better on extremely large, sparse data sets. We chose four classification methods that are diverse in their approaches to solving the problem. By running these different algorithms on the same data set, we can see which algorithm would perform the best in terms of training run time and testing error minimization. We have found that support vector machines work the best with a testing accuracy of 74.4%, decision trees follow with a testing accuracy of 61%, then neural networks with a testing accuracy of 20%, and lastly naïve Bayes classifiers with a testing accuracy of 6%.

3 Methods

3.1 Support Vector Machines

The motivation for using support vector machines comes from their ability to reduce dimensions and eliminate useless results. Our method works as follows:

1. We perform natural language processing to sanitize the data.

2. We know that there are around 6694 non-repeating ingredients (features). Because the sample-feature matrix is extremely large and sparse, 39774×6694 total elements with the majority being valued at 0, or not present, we then create a new class-feature (cuisine-ingredient) matrix based on if the specific feature (ingredient) appears in that class (cuisine). For example, if egg appears in one of the Chinese samples but never appears in any Greek samples, we put 1 at the Chinese-egg element, but 0 at the Greek-egg element. This 20×6694 matrix is the binary summary matrix. We also used a count summary matrix, also 20×6694 in size, that keeps track of the frequencies of ingredients in each cuisine.

3. We then perform 96% (or 98%) information PCA on the above class-feature matrices and the whole dataset. PCA reduced the dimensions of the class-feature matrices to 6694×19. These 19 features contain the most information about the training data. PCA reduced the dimension of the whole dataset to 6694×1408.

4. Build a 39774×6694 data matrix with 0 and 1 entries. The rows represent all the samples and the columns represent all the unique ingreidents in the whole training dataset. Multiply this data matrix with PCA_output. We get a 39774×19 matrix or 39774×1408 matrix, depending on if we're using the class-feature matrix or the sample-feature matrix.

5. By preserving those 19 features, we then use SVMs to train out data, starting with a linear kernel. We also used different kernels such as Radial basis, Laplacian, and ANOVA. This technique, utilizing SVMs, is a one vs. one method—it compares every two possible cuisines every time. During prediction, the class which receives most of the votes is selected.

3.2 Decision Trees

As a first approach to feature selection, we implemented the "Bag Of Words" model for every ingredient in relation to each of the 20 cuisines. The result was a document that provided an ordered list of ingredients by frequency in relation to each cuisine. From this list, we then determined that there were 107 unique ingredients that appeared in every cuisine in at least one recipe. Also, we determined that most ingredients only appeared in a small number of recipes. Using this knowledge, we created a function to calculate the density of 1's present in the dataset per ingredient in order to see which ingredients were most important. What we found was that 99% of the table was filled with 0's, which confirmed our earlier findings that the matrix dataset would be extremely sparse.

At this point we had three options. The first was to get rid of the ingredients that were sparse (our threshold for omission was that less than 1% of their columns were filled with 1's), which would remove the majority of our 6,714 ingredients. However, by doing so, we would lose many notions of associations between ingredients. The second option was the opposite, where we would take out the ingredients that were not sparse. If an ingredient appeared in every cuisine or nearly every cuisine, then we conjectured that the ingredient would not be informative. However, we would again lose the relationships of ingredients among the samples if we were to remove ingredients. This approach also had a problem of only eliminating 200 or so ingredients (using our earlier definition of sparse) which is not a significant portion of the 6671 total ingredients. The third approach was a mix of the first two in which we would both eliminate those ingredients that were relatively dense, and those which were very sparse. Here we would still be losing some associations in both frequently and infrequently occurring ingredients; however, we would greatly reduce the amount of features used in the training of our decision trees.

For our first test run we eliminated all ingredients with less than 1% density (approximately 400 of the samples had a 1 in that column). Out of the 6714 ingredients, only 194 met that criterion.

We also decided to use the random forest technique, in which multiple decision trees are used for prediction simultaneously. After eliminating all ingredients with less than 1% density, we generated 50 decision trees on our training set.

Finally, we split the complete dataset into training and testing sets, made a confusion matrix, and tested the model's accuracy against the testing set.

3.3 Neural Networks

Dimensionality reduction was performed with the Latent Semantic Indexing (LSI) algorithm, which is a dimensionality reduction algorithm for categorical datasets. The algorithm works by looking for trends and correlations between features in the feature set in order to omit redundant features that do not provide new information. This stems from the idea that features that are used in the same context often carry the same information and meaning. this was a key step for our training, as there were 6685 possible ingredients, and working with this number made the runtime and resources required prohibitively high. Using LSI allowed us to reduce the number of features to a manageable number, and testing was performed using multiple numbers of features.

After the reduction of the dimensionality of the data was performed, a number of network tuning possibilities were tested. One of these options was the dropout technique for preventing the neural network from overfitting the training data set, rendering it less effective on the testing set. This technique involves, with a specific probability, deactivating nodes in the network with each sample that is forwarded through the network. This prevents nodes in the network from co-adapting during training. During testing, the weights of each node are then multiplied by the dropout percentage, which effectively averages the thinned networks that were used during training. In our testing, we used a dropout rate of 50%, and tested the differences achieved by applying dropout.

We also applied annealing to our network in order to prevent overshooting a minimum, which may occur with a learning rate that is set too high. During annealing the learning rate is decreased by a certain amount with each epoch, and the steps that are taken get gradually smaller and smaller, which avoids the possibility of the network continually bouncing around the error function and taking steps that are too large. This was achieved by multiplying our learning rate by .8 with each epoch, iteratively decreasing the learning rate.

We also explored a variety of activation functions—in addition to the standard sigmoid function, we performed analysis using both the hyperbolic tangent (tanh) and rectified linear units (ReLU) as nodes in our network. The benefits of a ReLU node is the reduced likelihood of the gradient dropping to zero for high gradient values during back propagation, allowing for faster learning. Using hyperbolic tangent provides a similar benefit in that it too allows for faster convergence as it is zero-centered, preventing the all-negative or all-positive weights generated from a sigmoid neuron from causing the network to zig-zag haphazardly.

3.4 Naïve Bayes Classifier

For this project, a Java implementation of a multi-class version of a naïve Bayes classifier was made from scratch. It was used to determine the most likely cuisine to have a particular set of ingredients. The likelihoods and priors for all classes were computed during the training. Likelihoods were found by counting each occurrence of each ingredient for every class, and then dividing this count by the class frequency. Finally, the log of this value was taken so that each value could be added up, such that:

$$\boldsymbol{G(X_1, ..., X_n) = log\frac{\Pi P(FeatureX_i|Class1) \times P(Class1)}{\Pi P(FeatureX_i|Class2) \times P(Class2)}}$$

The priors were computed by dividing the class frequency by the total number of samples. No dimensionality reduction was performed on the data.

The naïve Bayes classifier was modified to handle multiple classes. The logs of the likelihood and prior were calculated for each class. The class with the highest log likelihood was chosen by the classifier.

This algorithm has a runtime of O(N), as each piece of data need only be touched a linear number of times.

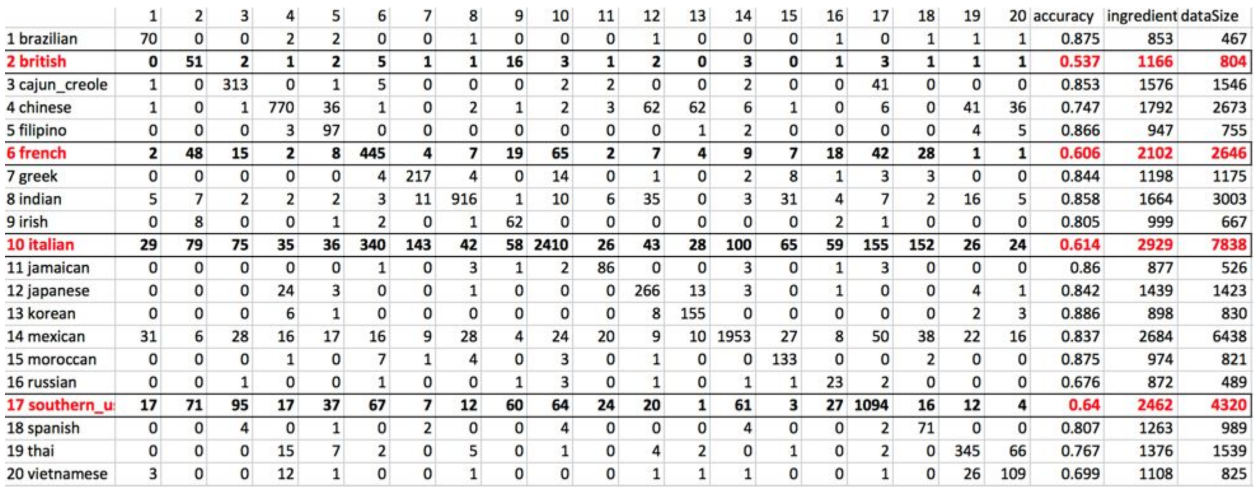
4 Results

4.1 Support Vector Machines

After training the data with Gaussian kernels and preprocessing the testing data, we tested the files on Kaggle, which gave back a 74.4% accuracy. It takes 6 hours in total to do PCA and to train the SVM on a server.

In order to understand our model better, we also split the original data into training and testing data with a 66% split rate. Then we used the testing data to calculate some basic statistics for the model. In general, it correctly classifies the 9586 testing instances out of 13258 with 72% accuracy, and 38.6% error rate. From the confusion matrix and the accuracy table below, we can see how the multi class classifier performs in different classes. In terms of precision, we can see that the classifier performed the best in the classes of Brazilian, Korean, and Moroccan cuisine with the precision rate higher than 87%. We can also see the cuisines that are relatively difficult to classify: British, French, Italian, and Southern US, which have less than 65% accuracy.

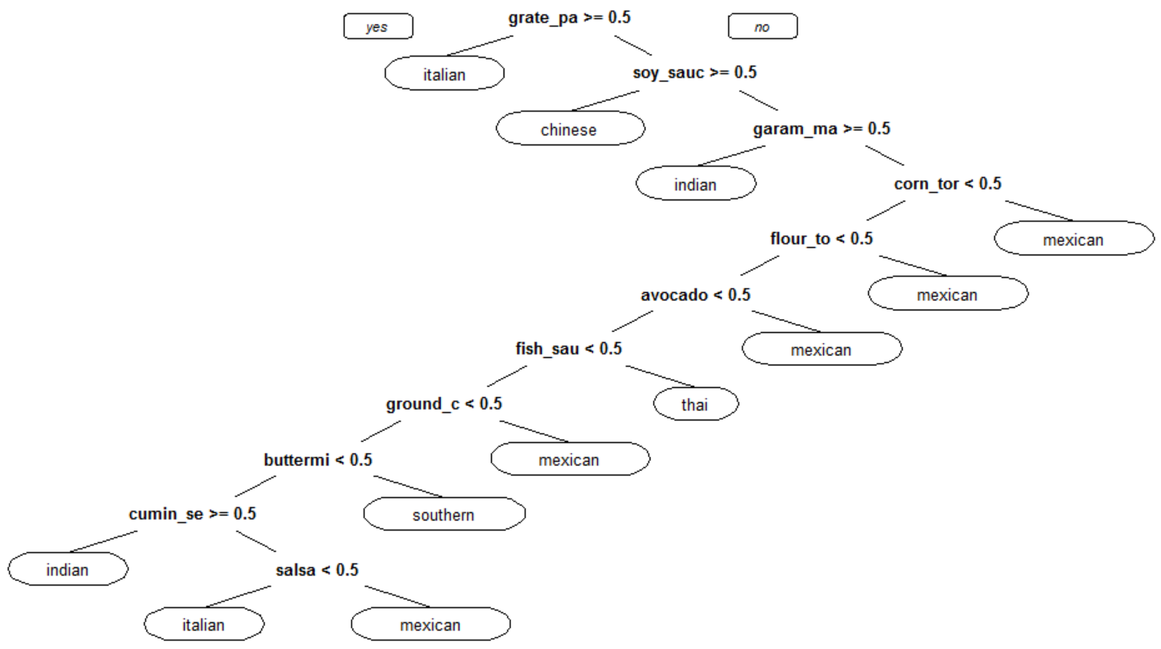
(Note: columns represent the actual cuisines, while the rows represent the predicted cuisines)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	accuracy	ingredient	dataSize
1 brazilian	70	0	0	2	2	0	0	1	0	0	0	1	0	0	0	1	0	1	1	1	0.875	853	467
2 british	**0**	**51**	**2**	**1**	**2**	**5**	**1**	**1**	**16**	**3**	**1**	**2**	**0**	**3**	**0**	**1**	**3**	**1**	**1**	**1**	**0.537**	**1166**	**804**
3 cajun_creole	1	0	313	0	1	5	0	0	0	2	2	0	0	2	0	0	41	0	0	0	0.853	1576	1546
4 chinese	1	0	1	770	36	1	0	2	1	2	3	62	62	6	1	0	6	0	41	36	0.747	1792	2673
5 filipino	0	0	0	3	97	0	0	0	0	0	0	0	1	2	0	0	0	0	4	5	0.866	947	755
6 french	**2**	**48**	**15**	**2**	**8**	**445**	**4**	**7**	**19**	**65**	**2**	**7**	**4**	**9**	**7**	**18**	**42**	**28**	**1**	**1**	**0.606**	**2102**	**2646**
7 greek	0	0	0	0	0	4	217	4	0	14	0	1	0	2	8	1	3	3	0	0	0.844	1198	1175
8 indian	5	7	2	2	2	3	11	916	1	10	6	35	0	3	31	4	7	2	16	5	0.858	1664	3003
9 irish	0	8	0	0	1	2	0	1	62	0	0	0	0	0	0	2	1	0	0	0	0.805	999	667
10 italian	**29**	**79**	**75**	**35**	**36**	**340**	**143**	**42**	**58**	**2410**	**26**	**43**	**28**	**100**	**65**	**59**	**155**	**152**	**26**	**24**	**0.614**	**2929**	**7838**
11 jamaican	0	0	0	0	0	1	0	3	1	2	86	0	0	3	0	1	3	0	0	0	0.86	877	526
12 japanese	0	0	0	24	3	0	0	1	0	0	0	266	13	3	0	1	0	0	4	1	0.842	1439	1423
13 korean	0	0	0	6	1	0	0	0	0	0	0	8	155	0	0	0	0	0	2	3	0.886	898	830
14 mexican	31	6	28	16	17	16	9	28	4	24	20	9	10	1953	27	8	50	38	22	16	0.837	2684	6438
15 moroccan	0	0	0	1	0	7	1	4	0	3	0	1	0	0	133	0	0	2	0	0	0.875	974	821
16 russian	0	0	1	0	0	1	0	0	1	3	0	1	0	1	1	23	2	0	0	0	0.676	872	489
17 southern_u	**17**	**71**	**95**	**17**	**37**	**67**	**7**	**12**	**60**	**64**	**24**	**20**	**1**	**61**	**3**	**27**	**1094**	**16**	**12**	**4**	**0.64**	**2462**	**4320**
18 spanish	0	0	4	0	1	0	2	0	0	4	0	0	0	4	0	0	2	71	0	0	0.807	1263	989
19 thai	0	0	0	15	7	2	0	5	0	1	0	4	2	0	1	0	2	0	345	66	0.767	1376	1539
20 vietnamese	3	0	0	12	1	0	0	1	0	0	0	1	1	1	0	0	1	0	26	109	0.699	1108	825

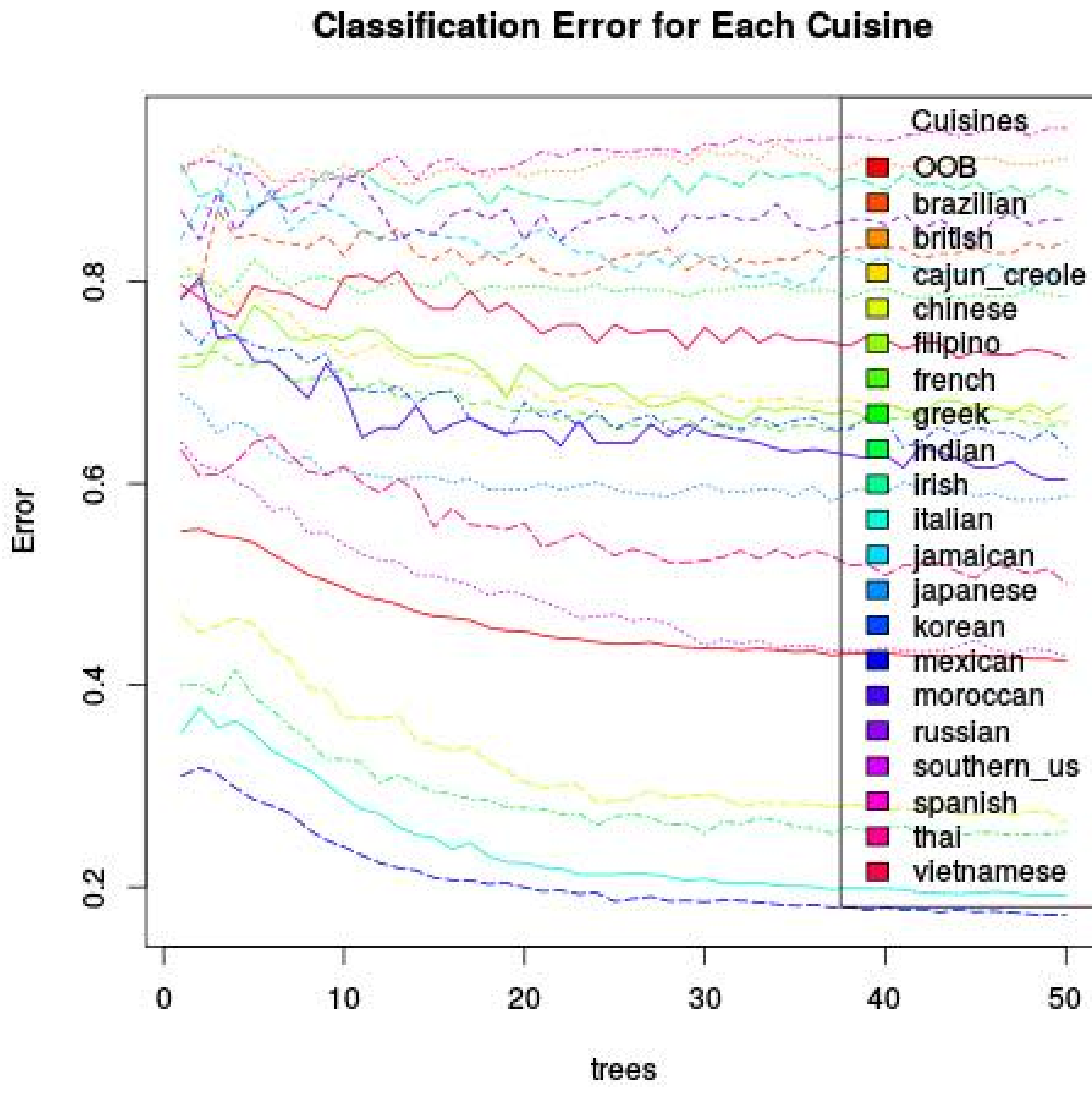
4.2 Decision Trees

After eliminating all ingredients with less than 1% density, the single decision tree trained with this smaller data set scored a 37% on *Kaggle*. The decision tree is displayed below.



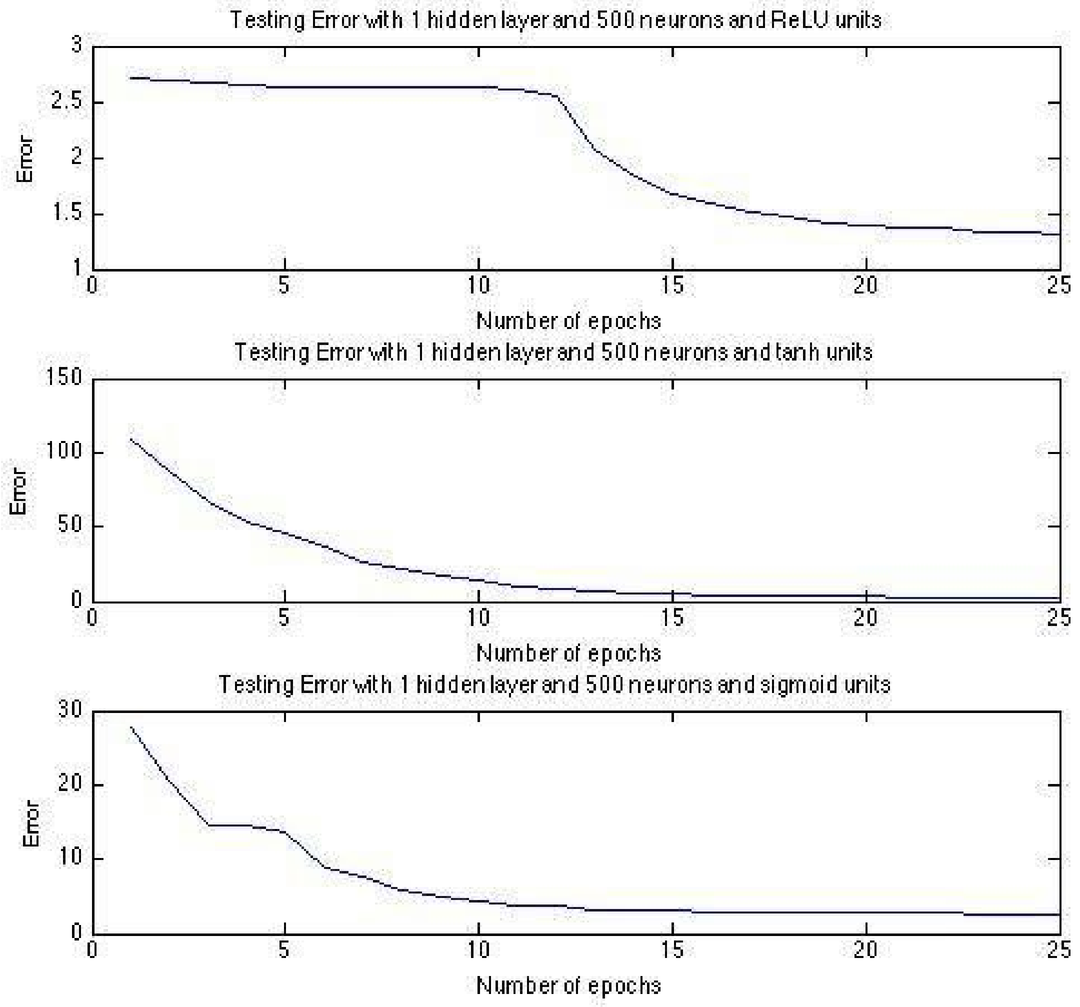
Using the random forest technique with 50 decision trees had an enormous impact on our accuracy level as it moved our score on *Kaggle* up to a 61% from the previous 37%.

After splitting our dataset into testing and training pitted against a confusion matrix, we generated 84.35% accuracy against the testing data.



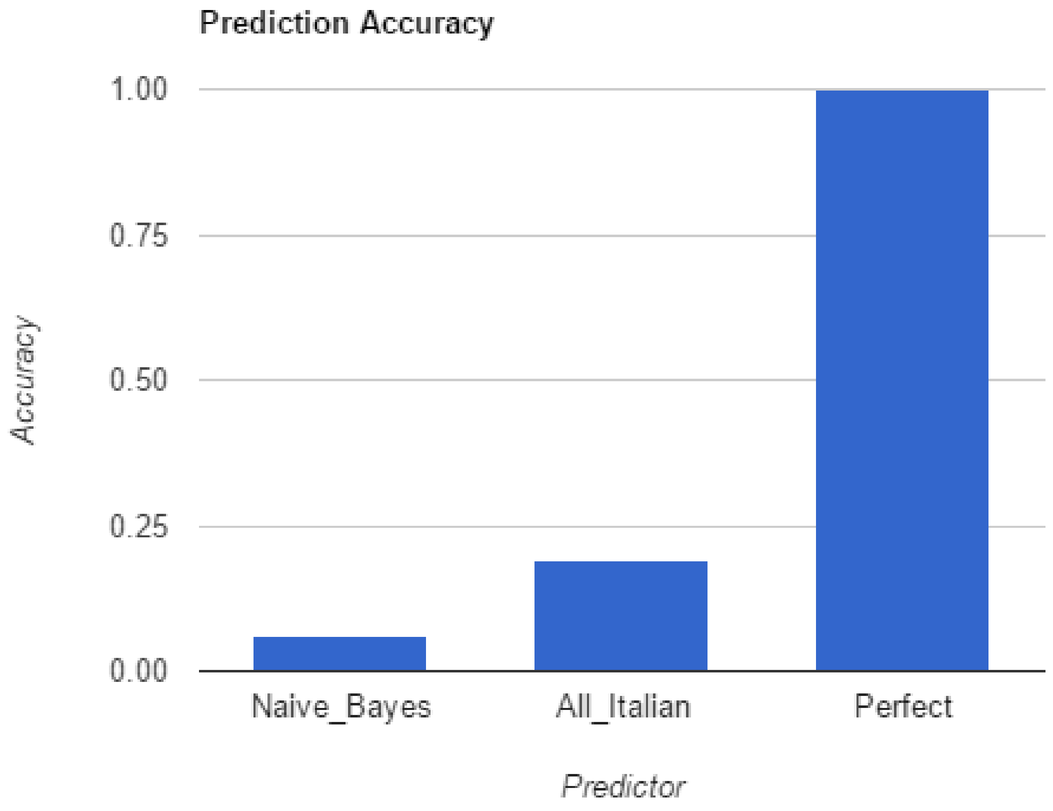
4.3 Neural Networks

The best results were achieved using a network consisting of one hidden layer of 500 neurons, with ReLU activations and dropout—this generated a 20% on *Kaggle*. This achieved both the best testing error and the best training error among multiple topologies. Annealing the learning rate had a very minimal impact on the performance of the network during our tests.



4.4 Naïve Bayes Classifier

The naïve Bayes classifier predicted the testing data with an accuracy of 6.2%. When run on the training data, the accuracy was also low. Here is a comparison between our predictor, assuming all cuisines are Italian, and a perfect predictor:



5 Discussion

5.1 Support Vector Machines

5.1.1 High Misclassification Rate in Some Cuisines

The classifier did not perform well in the the classes of British, French, Italian, and Southern US. As we explored the data further, we found that those four cuisines have quite diverse ingredients which are very likely shared by other cuisines. The diversity of the ingredients can confuse the classifier, and lower the accuracy. Except for British cuisine, the other three cuisines also have large sample sizes, all of which rank in the top five in the list. This fact can also cause issues, since the classifier tends to predict those cuisines more than the others, increasing the number of mistakes. From the confusion matrix, the rows of French, Italian, and Southern US have misclassifications spread over other cuisines.

5.1.2 PCA Methods on Different Datasets

After the natural language processing, we created a binary class-feature matrix, which only takes into account if certain ingredients appear in specific cuisines. We also tried another class-feature weighted matrix. This matrix takes into account the frequency of the specific ingredients that appear in the class, instead of only taking into account if the ingredient appears. By performing the SVM on this matrix, the accuracy increases around 1%, as it takes advantage of how the ingredient is "normally" used in a specific cuisine. These two methods of PCA take around 3 minutes to run, but its accuracy is still higher than 60%.

We tried to run the normal PCA on the sample-ingredient matrix on the server. This PCA method takes more time (more than 6 hours) than the previous method but it improves our accuracy dramatically (from 60% to 74.4%). The reason for this great improvement is that the normal PCA takes into account all the the 39774 samples instead of only the 20 different classes, so there is more raw data and information for the PCA method to preserve as much useful information as it can. However, the tradeoff for higher accuracy is that it takes more time.

5.1.3 PCA Variability

We also considered how much information we wanted to preserve after we performed PCA. In general, the lower the PCA variability, the less features we needed to consider when we train the data. However, based on our observation, 98% of the PCA variability does not necessarily lead to a better result than 95% of the PCA variability, since reserving 98% of the information is more likely to lead to overfitting.

5.1.4 Kernels

We have tried four kinds of kernels: linear, Gaussian, Laplacian and ANOVA. Different kernels have different feature transformation. The Gaussian kernel computed with a support vector is an exponentially decaying function in the input feature space, the maximum value of which is attained at the support vector which decays uniformly in all directions around the support vector, leading to hyper-spherical contours of the kernel function. Linear kernels are a special case of Gaussian kernels. These two give very good performance. We also explored Bessel and ANOVA; however, the training efficiency and accuracy were low. Although these two kernels are based on Gaussian kernels, the extra transformation may not suitable for the data set.

5.2 Decision Trees

The main reason why decision trees work well in this situation is because the data set conforms to the "yes-no" nature of how decision trees are built. Also, it makes sense that using random forests, which can be interpreted as a cross-validation style enhancement to using just a single tree, yielded higher results—by diversifying and partitioning the training set and averaging the predictions, we could have more confidence that the training efficacy of the decision trees was not based on the ordering or locality of the samples.

5.3 Neural Networks

During our testing we noticed no significant difference when applying this technique.

The topology of our network was the key determining factor in both the running time of our training and the amount of resources required to complete it. As the number of input features, hidden nodes, and layers increased, the runtime increased dramatically. As the majority of the operations are matrix multiplications, increasing the number of nodes more severly increases the running time. Further, to keep all of this information in memory at once gets more costly—several times throughout testing the topology had to be changed due to the resource constraints. From our testing, increasing the complexity of the topology did little to improve performance, so this was not a major obstacle.

Given the immense complexity of the feature set, and the natural tendency of neural networks to overfit the training set more harshly than most models, overfitting is a severe problem here. This is evidenced in our results in that we typically see good performance on our training data, typically approaching a fifty percent error rate, but when we then use these models on our testing data the results are much worse. Further, the downsides of using a sigmoid activation function are demonstrated in the improved performance of both the hyperbolic tangent and ReLU activation functions.

5.4 Naïve Bayes Classifiers

Naïve Bayes performed poorly for this application. There are many possible explanations for the high misclassification rates. First of all, the features are not independent, as naïve Bayes assumes. Certain ingredients are likely to appear together. For instance, a recipe with pasta will probably also require olive oil.

One might initially suspect overfitting is to blame. The feature space is enormous; with over 4000 ingredients, there are 2^{4000} possible sets of features. However, there are reasons to doubt that overfitting is the true problem. Firstly, when validated against the training data, the classifier still performed poorly. If overfitting were the issue, then the classifier should have been accurate when used on the training data. Secondly, the data set was extremely large, so overfitting should be less likely.

One way to improve the model's accuracy could be to perform dimensionality reduction. If the number of dependent features is reduced, the naïve Bayes assumption could be more valid, and thus the classifier would be more accurate.

Nevertheless, at least in its current form, the naïve Bayes method was not effective for this application.

5.5 Conclusion

With these findings, it is clear that support vector machines work the best for this specific data set. This method's usage of decreasing the amount of features proved helpful in dealing with a dataset that was very sparse. This study also reinforces the idea that the diversity of machine learning techniques and algorithms imply that some are better than others given certain situations, usually concerning the makeup of the dataset. A future study could be done to apply machine learning techniques to predict which machine learning techniques would work best given a dataset. However, this paper provides a good foundation on which we can make the claim that different machine learning strategies behave and perform differently based off of the problem and the dataset.

6 References

6.1 Support Vector Machines

6.1.1 Packages

- weka: `http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html`
- e1071: `https://cran.r-project.org/web/packages/e1071/e1071.pdf`
- Kernlab: `https://cran.r-project.org/web/packages/kernlab/kernlab.pdf`

6.1.2 Open Source Code

- Started xgboost in R: `https://www.kaggle.com/yilihome/whats-cooking/simple-xgboost-in-r`

6.1.3 Other References

- `https://www.quora.com/Support-Vector-Machines/What-is-the-intuition-behind-Gaussian-kerne`

6.2 Decision Trees

- Random Forest Wikipedia Page: `https://en.wikipedia.org/wiki/Random_forest`
- Stack Overflow - Common Word Removal: `http://stackoverflow.com/questions/25905144/removing-overly-common-words-occur-in-more-than-80-of-the-documents-in-r`

6.3 Neural Networks

- A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Proc. Neural Information and Processing Systems, 2012.
- Deerwester, S., et al, Improving Information Retrieval with Latent Semantic Indexing, Proceedings of the 51st Annual Meeting of the American Society for Information Science 25, 1988, pp. 3640.
- V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proc. 27th International Conference on Machine Learning
- Y. LeCun, I. Kanter, and S.A.Solla: "Second-order properties of error surfaces: learning time and generalization", Advances in Neural Information Processing Systems, vol. 3, pp. 918-924, 1991.

6.4 Naïve Bayes Classifiers

- Guestrin, Carlos. "What's learning, revisited. Overfitting. Bayes optimal classifier." Carnegie Mellon University: 2006. Electronic.

- Manning, Christopher; Raghavan, Prabhakar; and Schutze, Hinrich. *An Introduction to Information Retrieval.* Cambridge: Cambridge University Press, 2009. pp. 225-227. Electronic.

- Minyoung, Kim. *Discriminative Models and Dimensionality Reduction for Regression.* New Brunswick: New Brunswick Rutgers, 2008. pp. 9-11. Electronic.

7 Author Contributions

7.1 Support Vector Machines

- Ge Cheng: Method, kernel, and frameworks research; data preprocessing, code development, testing, results analysis
- Jiaming Xie: Method, kernel, and frameworks research; raw data preprocessing, results analysis, discussion in report
- Weitang Liu: Method, kernel, and frameworks research; results analysis, report write-up

7.2 Decision Trees

- Carl Glahn: Algorithm development (for both feature selection and decision trees), testing, report write-up
- Corbin Gomez: Team organization, report compilation, code development for feature selection using density
- Paul Salessi: data preprocessing, code development, testing, presentation slides compilation

7.3 Neural Networks

- Kenan Nalbant: code development, analysis and testing; framework modifications/optimizations, report write-up
- Alex Parella: code developement, analysis and testing; dimensionality reduction implementation, report write-up

7.4 Naïve Bayes Classifiers

- Michael Rea: initial design, proof of concept, code development, report copy-editing
- Mazar Farran: initial design, proof of concept, report write-up