Inferring Cuisines from Cooking Recipe Descriptions using Language Processing

Simmi Mourya (simmim@seas.upenn.edu)
Mohitrajhu Lingan Kumaraian (moji@seas.upenn.edu)
Rahul Ramesh (rahulram@seas.upenn.edu)

Abstract

In this project, we aim to predict the cuisine of the dish based on the constituents of this dish. The data for the project is from Kaggle (What's cooking? [7]) and consists of ingredients (in text form) and the corresponding cuisine. We implement natural language processing techniques to effectively vectorize the words for performing downstream tasks like classification. We find that TF-IDF with RBF-kernel based SVM yields the higher classification accuracy. We compare how different input representations, classifiers and preprocessing techniques affect the performance of the task. We also perform unsupervised learning and identify similarities and differences of different cuisines and visualize the different cuisines in 2D embedding space. Our work is summarized in 4 Jupyter notebooks (Visualization, Supervised learning, Unsupervised learning and Final Clean Notebook).

1. Motivation

The dataset consists of ingredients and the corresponding cuisine and the aim is to classify an unlabelled recipe into one of the 20 given cuisines. The challenge arises from the fact that the input features are words and cannot be used with a model in the absence of data processing. This requires understanding different representation of words in a fixed dimensional sub-space (like Bag of words or Word2vec). After forming a useful representation for the recipe, the task then devolves to the standard classification problem which can be tackled using popular models like Naive Bayes, Logistic regression, Decision Trees, Random Forests, Gradient Boosted Trees, SVMs and Neural Networks. Since certain cuisines share a significant number of ingredients, we also cluster different recipes with/without dimensionality reduction. This can help with data understanding and visualization and uncover potential patterns in the dataset. The primary purpose of this project is to understand and analyze


```
{
  "id": 40989,
  "cuisine": "southern_us",
  "ingredients": [
    "low-fat mayonnaise",
    "pepper",
    "salt",
    "baking potatoes",
    "eggs",
    "spicy brown mustard"
  ]
},
```


Figure 1. Sample data

a dataset but one can potentially use the built classifier for a simple application that categorizes and catalogues a different recipes. The dataset is interesting because we have the opportunity to explore data visualization techniques, supervised learning, language processing, unsupervised learning and dimensionality reduction.

2. Related Work

Kaggle kernels [4] are a great source of python notebooks that highlight previous solutions to the problem. Kernels convert the ingredients into a vector by using a bag of words model to present the ingredients as a collection of indicator features. Additionally, TF-IDF was used to scale the importance of infrequently occurring words. However, the meaning of words is grounded in our world and works have studied the notion of "Symbol grounding". For example, *Salt* and *Pepper* are fairly similar ingredients when compared to fruits or vegetables. Hence one can potentially leverage the knowledge present in external datasets to build better representations. Hence, we use techniques like Word2Vec [8] to work with better word representations.

Some kernels build some interesting visualizations which understand the distributions of the classes, or the correlation between the different cuisines (kernel example: Cooking is Chemistry [1]). Other kernels have focused on understanding the vocabulary What are Ingredients [2] and re-

Figure 2. Class distribution visualized through a Waffle plot. The number of squares corresponds to the proportion of the class in the entire dataset.

veals some interesting challenges. Some of them include the presence of accented characters, hyphens, misspellings, etc.. . The most popular baseline by far is SVM, with TF-IDF [3, 5]. We also observe the highest accuracy with a Gaussian-kernel SVM. The baseline used minimal feature engineering in order to achieve competitive performance. Another simple yet popular model is a simple logistic regression classifier [6].

Our work attempts to comprehensively analyze and compare different available classifiers. We also use Word2Vec and TF-IDF based models to build better feature representations. Some kernels use very simple versions of word embedding. We also experiment with additional features like number of ingredients, indicator variable for special characters, number of characters. These are a few features that have not been considered in other python notebooks to the best of our knowledge. Finally, the focus of most notebooks has been on the classification task which is unsurprising considering that this is a kaggle contest. We however hope to explore different clustering techniques with an aim to unearth some interesting patterns in the data.

3. Dataset

In this project we are using the dataset provided by Yummly as a part of their kaggle contest "What's cooking?" [7]. This dataset consists of the recipes(features) in text form and the cuisines for each recipes(as in Figure 1). Since the features are in textual form, the major challenge would be to form a mapping from the vocabulary space to the embedding space. The sections below run through some interesting visualizations and patterns.

Figure 3. Word cloud of the vocabulary before pre-processing

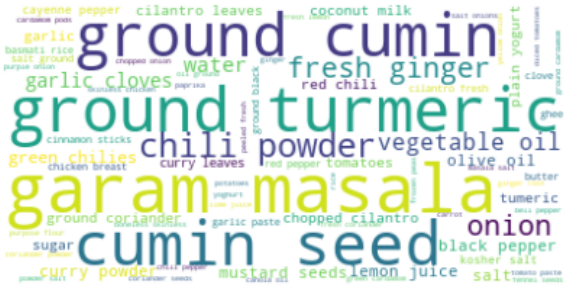

Figure 4. Word cloud for Indian cuisine

Figure 5. Word cloud for Italian cuisine

3.1. Initial Data statistics and visualization

The training dataset initially consists of 39774 datapoints falling under 20 different recipes. The distribution of datapoints into the cuisines is shown with the help of a waffle plot(Figure 2). The initial vocabulary size is 3589. We evaluate how the accuracy of the model varies if we include other n-grams. The initial vocabulary is represented pictorially in Figure 3. We see that some examples of frequently occurring ingredients over all cuisines are olive oil and black pepper.

Although these ingredients are more commonly found in many cuisines, the key ingredients in a recipe vary from cuisine to cuisine (as shown in Figure 4, Figure 5, Figure 6). We also see from these figures that Indian recipes have a number of spices while the Italian cuisine is dominated by ingredients like cheese and olive oil.

We also look at the number of ingredients in each recipe. We visualize this using the violin-plot (similar to box-plot) in Figure 6. The plots indicate that the Indian and Jamaican cuisines have a larger number of ingredients and the Brazilian cuisine has a number of recipes with a small number of ingredients.

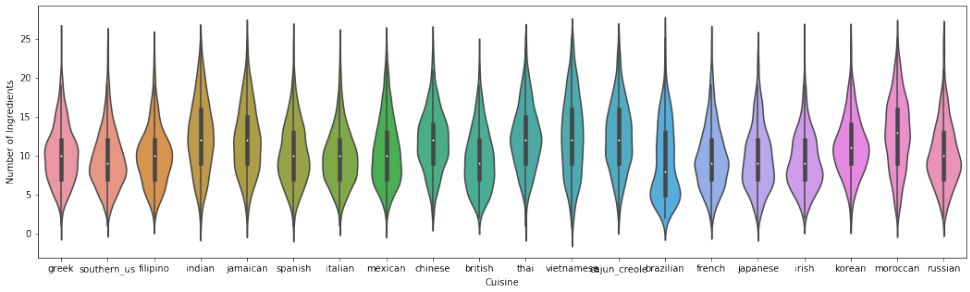

Figure 6. Violin plot, representing the distribution of ingredients against the cuisine

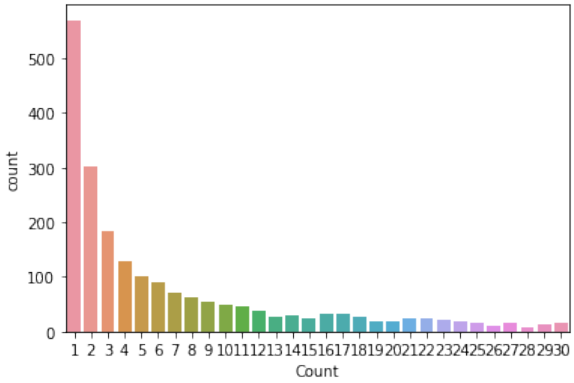

Figure 7. Distribution of the occurrences of words. (the plot is truncated at 50 (however there are quite a few words that occur over 5000 times

Finally, we want to understand the frequency of occurrence of words. Figure 7 plots the number of times a word occurs n-times. The plot indicates that a large number of words occur only once. Such words are not very useful since one cannot train models that generalize well with just 1 sample of the word. More visualizations are present in the notebook (Visualization)

3.2. Data pre-processing

We perform the following pre-processing steps on our data before we vectorize them. They are as follows:

- Convert all upper case characters to lower case.
- Remove all data points with 1 or 2 ingredients.
- Remove words like lb., oz., inch (they are found in some cuisines).
- Remove special characters like [pound symbol], ™, (,), ®, ., -, %,, &.
- Remove accented characters and replace them with an ascii counter-part.
- Remove copyright and trademark symbols.
- Remove words that occurs in less x% of the data (we used x=0.01).
- Change unicode symbol of apostrophe with ' .
- Remove all numbers.

The pre-processing leaves the dataset with only alphabets and spaces as characters. The total number of datapoints is 39559. Since we remove these characters, we construct an additional set of features to see if they are somehow relevant to the task. The features are as follows:

- Binary indicator for accented characters
- Binary indicator for special characters like ™, ®
- Indicator variable for other symbols
- Indicator variable for presence of numbers, % or units of measurement
- Number of ingredients in recipe
- Number of characters in the recipe

4. Problem Formulation

Classification task: The main focus is on the classification task. The first step involves converting the input ingredients, into a vector that can be fed as input into a classifier. Simple transformations like one-hot vectors/bag of words are used to understand the difficulty of the problem. We consequently use Word2Vec and TF-IDF and also try other techniques like stemming and lemmatization.

With a fixed dimensional representation of every data point, we use the cross-entropy loss to train the multi-class classifier. The evaluation is done using K-fold cross-validation or using a held out test set (depending on the time required to train the model). For smaller models, we prefer the more robust cross-validation setup. We use accuracy as the evaluation metric. But, we also used confusion matrix as our additional metric to get an idea of how the classes are getting classified and if we could use any other model which could classify these similar classes.(*eg. as we will see in the experiments part*).

Note that there are a myriad of choices/hyper-parameters and pre-processing techniques to choose from. We attempt to empirically find some of our choices but do not perform fine-grained hyper-parameter grid-searches due to a constraint on time. For example, we explore undersampling/oversampling data points based on the proportion in which they occur in the dataset. Samples from the minority classes are oversampled while those in the majority class are undersampled. Samples from the majority class will otherwise dominate the loss and the classifier will build a bias towards these classes. We try other such modifications and validate our choices based on the training accuracy. After training a classifier, we also look at the incorrectly predicted samples to understand which type of data points are being incorrectly classified.

Unsupervised learning: The representation built for the classifier can also be used to build clusters. Note that clustering is difficult in high-dimensional space and hence dimensionality reduction using PCA/TSNE might be useful. We also train an LDA model to understand the topics of different ingredients. We gain an understanding of different cuisines and their relationship through this exercise.

5. Methods

In this section, we outline the various models and feature generation techniques that we evaluate. We describe our baselines for this task. Our code is written in Python and most machine learning models use the Scikit-learn [10] package.

5.1. Building Feature Representations

We perform the pre-processing steps described in an earlier section. We also empirically evaluate different tokenization schemes (n-grams), stemming and lemmatization. We see that considering only 1-grams works best and don't observer any benefit in using either stemming or lemmatization (using Wordnet [9]). We also discard words that don't occur frequently in the dataset and consider different thresholds in the same. The reasoning for this is to prevent the model from overfitting to infrequently occurring words, resulting is lower generalization. Most of these choices are justified using an empirical evaluation of different choices. Now every recipe is represented as an un-ordered list of words.

The simplest representation we consider is the Bag-of-words representation. This feature representation counts the number of occurrences of a word and populates a vector which is the size of the vocabulary. The size of the vocabulary varies with the type of pre-processing and is around 1000 to 3000. Since certain ingredients like salt and oil occur frequently, it is beneficial to use a inverse document frequency based weighting. We hence consider the TF-IDF representation which is our best performing representation. Both these representations assume that the text is un-ordered which is the case for this problem. We experiment with multiple design choices like using $\log$ for inverse document frequency, smoothing the counts and using binary indicators instead of token counts.

Finally we consider word and document embedding. We don't train a Word2vec [8] from scratch on the given data since the dataset is small in size and most tokens do not occur too many times. Hence we use Google's pretrained Word2vec representation (download link), which generates a 300 dimensional vector representation for each word. We represent each recipe as the average of word2vec representations of all the comprising words. We observe a degradation in performance when using word2vec and we discuss potential reasons in the conclusion section. We notice that words like: ['aleppo', 'tubetti', 'hellmanns', 'kidnei'] do not have corresponding Word2vec representations even though some of them maybe critical to identifying the cuisine.

5.2. Baseline

The simplest model we consider is a random classifier. Since this is an imbalanced classification problem, we consider another baseline of solely predicting the majority class. We then move into slightly more complex models

Another model is the bag of words representation followed by a Naive-Bayes classifier. This model builds a conditional probability estimate for each class (and a prior for each class) and uses the independence assumption to calculate $p(label|data)$. Such an assumption is not necessarily valid in our cases. We also consider logistic regression and decision trees as other baselines. Since logistic regression executes quickly also achieves reasonable accuracy, we use this model to evaluate other modelling choices like stemming, Word2Vec etc.. (this is a heuristic decision).

5.3. More models and unbalanced data

Since the data is potentially unbalanced, we can use methods like under-sampling/oversampling to make sure that a particular class does not dominate the loss. Although this is a good choice when you have a really large dataset, excluding the data potentially leads to loss of information that is valuable, leading to poor generalization of the test data. We also consider more complex models like Random Forests, Ada-boost, Gradient boosted trees, XGBoost, SVMs with different kernels and Neural Networks. For the multi-class classification problem we primarily use classification accuracy as the metric of evaluation but we additionally analyze the confusion matrix to mitigate class imbalances. We evaluate on a held-out test set and for some models (which run fairly quickly), we use K-fold cross-validation to determine the best performing model.

5.4. Unsupervised learning

We consider the TF-IDF representation with the hyper-parameter setting that yields the best classification performance. We perform K-means on this representation. Note that clustering is difficult in high-dimensional space and hence dimensionality reduction using PCA/TSNE is used to reduce data to 2 dimensions. This is easier to visualize. We also form cuisine representations by averaging the PCA/T-SNE representations of the corresponding recipes. We also look at the LDA model to understand the various topics that generate the recipes.

5.5. Tools for Implementation

The implementations are solely in python where we primarily utilize scikit-learn [10] for training our models. We make use of pandas, seaborn and matplotlib in the data analysis and visualization. The word clouds are plotted using https://github.com/amueller/word_cloud.

6. Experiments

6.1. Performance metrics

The experiments primarily use accuracy as the evaluation metric. We also report the confusion matrices in the jupyter notebook. We evaluate on a held-out test set and use K-fold cross-validation for models that run quickly. We do not use F1-score, precision or recall because since the confusion matrices do not indicate predictions biased to any one class. Furthermore, ROC and F1-scores are not typically utilized in multi-class classification problem with many classes.

6.2. Baseline model

The simple majority class classifier is used as the baseline model. This model yields a accuracy of **19.7%**. We also evaluate a model that predicts random labels and observe an accuracy of **5.0077%** (approximately $\frac{1}{20}$)

6.3. Feature Generation

In this section, we evaluate different feature generation techniques. Since the logistic regression model outperforms other simple models, we use it as a reference model. Consequently occurring tables are evaluated with logistic regression using the accuracy metric. Also, the reported accuracies are all on a held-out test set and are hence maybe slightly inaccurate.

1. **Effect of different feature generation techniques**

feature generation model	Accuracy(%)
Bag of words	75.86
TF-IDF	76.91
word2vec	75.36
TF-IDF+word2vec	76.81

Table 1. feature representation models vs Accuracy

2. **Varying n-grams for each feature**

n-gram	Accuracy(%)
1-gram	77.59
2-gram	77.03
3-gram	76.4

Table 2. Model with different n-grams

3. **Variation with smoothing**

Effect of smoothing	Accuracy(%)
no smoothing	76.91
smoothing factor=1	76.91

Table 3. Effect of smoothing

4. **Effect of sublinear function (logarithmic) on inverse document frequency**

IDF manipulation	Accuracy(%)
Without log IDF	76.91
Using log IDF	76.72

Table 4. IDF manipulation

5. **Effect of dropping rarely frequent words with TF-IDF on accuracy**

Word droppping threshold	Accuracy(%)
0.000	75.69
0.0001	76.20
0.003	74.99
0.01	70.91

Table 5. Frequency threshold vs Accuracy

6. **Effect of Stemming and lemmatization**

Process	Accuracy(%)
Stemming porter	76.91
Stemming Lancaster	76.22
Lemmatization	77.36

Table 6. Base-form extraction process vs Accuracy

Model	Train Accuracy (%)	Test Accuracy (%)
Random Classifier	-	5.008
Majority Classifier	-	19.7
Ada Boost	45.36	43.79
Decision Tree	99.98	62.00
Naive bayes	69.77	67.69
Gradient Boosting	97	74.41
Random Forest	99.99	74.97
MLP	98.34	75.78
Extra Trees	99.98	77.45
Logistic	82.07	78.41
XGBoost	94.96	78.59
SVM	99.7	80.24

Table 7. Accuracy of various models

6.4. Supervised Learning: A Summary of Different Classifiers

We experiment with Gradient Boosting Classifier, AdaBoost Classifier, Random Forest classifier, Extra tress classifier, XGBoost Classifier and a simple MLP classifier with 3 hidden layers of sizes 500, 500 and 300 respectively. For a detailed description of tuned hyperparamter values of the experiments mentioned above, refer to the Gradient Boosting section of the *Supervised learning* Colab notebook. Table Table 7 reports the training and test accuracies for the aforementioned experiments.

6.5. Supervised Learning: Decision Tree Classifier

We also experiment with a few decision tree classifiers, train and test accuracies for the experiments are reported in Table Table 11.

Split Strategy	Train Accuracy	Test Accuracy
Best (max depth=5)	37.99	38.40
Best (max depth=15, min samples split=100)	55.90	53.03
Best	99.97	60.60
Best (min samples split=20)	84.524	61.04
Random	99.98	62

Table 8. Overall accuracy of variants of Decision Tree Classifier trained via 5 fold cross validation. The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

6.6. Logistic Regression

After training a plethora of tree based models, we move ahead with Logistic Regression models due to its superior performance. We experiment with a combination of features and employ Logistic Regression classifier under various parameter setting. We start with a simple Logistic Regression classifier with bag-of-words representation. The bag-of-words is done on the pre-processed text. We then try different variants of TF-IDF and do a coarse grid-search over some hyper-parameters. The results for Logistic Regression Classifier are logged in Table 1, Table 2, Table 3, Table 4, Table 5 and Table 6. The best accuracy on the Logistic Regression classifier is reported in Table 9. This is obtained when all the best models for feature generation are used, which can be inferred from the above section on feature generation models as *TF-IDF on 1-gram generated feature words with/without smoothing dropping out words with frequency less than 0.0001 and lemmatizing the words*.

Model	Train Accuracy	Test Accuracy
Logistic Regression	82.07	78.41

Table 9. Accuracy of best Logistic Regression classifier with the following parameters: Without log IDF, smoothing factor =1, word dropping threshold =0.0001

The analysis is furthered by plotting confusion matrices to see which classes are have high True Positive rate. It helped us to investigate which samples/classes are performing poorly and directed us to use class weights for a more balanced classification. Figure 8 represents the Confusion Matrix for the best Logistic Regression experiment. Figure 9 represents the Multi-class extension of Receiver Operator Characteristic graph for the same. From the graph we can observe that ’class 17’ (’Southern-US’ cuisine) has slightly low area under the curve as compared to ’class 3’ (’Chinese’ cuisine) which can be verified from the Confusion matrix as well, since the model makes more errors which result in higher number of false negatives.

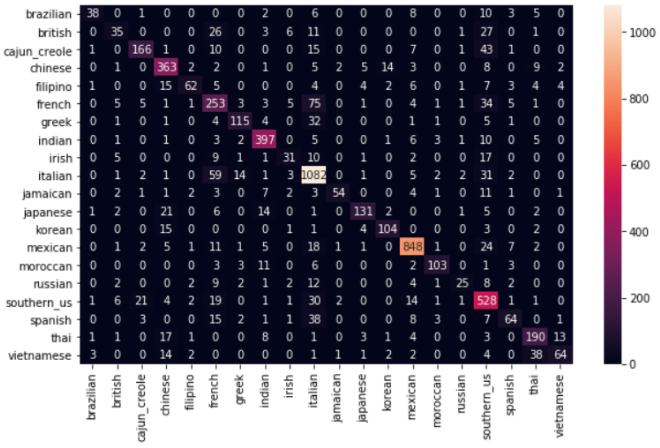

Figure 8. Confusion Matrix: Logistic Regression

Area under ROC for the multi-class problem: The function sklearn.metrics.roc_auc_score is used to compute the score for multi-class classification. The multi-class One-vs-

One scheme compares every unique pairwise combination of classes. [10] Here, we report a prevalence-weighted average and a macro average to further estimate the robustness of the models. Macro-average computes the metric independently for each class and then averages them, therefore weighs them equally. However, a Micro-average collects the class-wise contributions (hence taking care of inherent class-imbalance, if any) and then computes the average. The micro average adequately considers the class imbalance and adjusts the overall precision/recall average accordingly. Table 10 reports the Macro-average and Micro-average scores for Logistic Regression classifier.

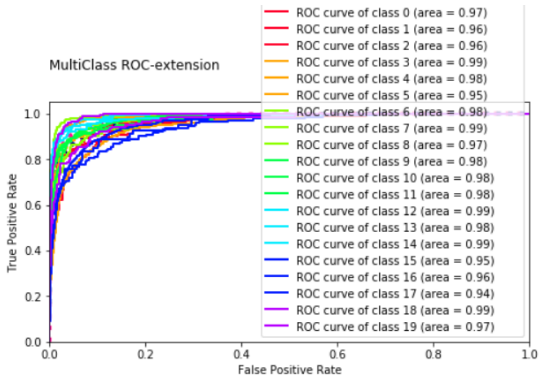

Figure 9. Receiver Operator Characteristic Curve per class: Logistic Regression

Metric	Score
Macro	0.9733
Micro (weighted by prevalence)	0.975452

Table 10. One-vs-One ROC AUC scores for Logistic Regression

6.7. Support Vector Machine

SVMs are conventionally used for binary class classification. For multi-class classification via SVMs, one can either use one-versus-all (usually referred to as OVR or one-versus-rest) or one-versus-one (usually referred to as OVO). The common practice while using OVR method is to build C (where C is the number of classes present in the dataset) OVR classifiers and choose the classifier which chooses the test data with maximum margin. While performing OVO classification one builds a set $\frac{C(C-1)}{2}$ of one-vs-one classifiers and choose the class that is selected by a majority of the classifiers.

Mode and Kernel	Train Accuracy	Test Accuracy
LinearSVC and L2 penalty	86.03	78.66
LinearSVC and L2 penalty, multi-class='crammer singer'	84.25	79.32
LinearSVC and L1 penalty	83.63	79.64
OVO (class balanced) (RBF)	99.99	**80.36**
OVR (RBF)	99.97	80.24
OVR (class balanced) (RBF)	99.98	80.18

Table 11. Overall accuracy on variants of Support vector machines. The modes OVO and OVR are denoted along with the type of kernel used to fit the data.

The Crammer-singer objective in Table 11 uses a joint objective to train a multi-class classification problem.

Figure 10 represents the Confusion Matrix for the best SVM experiment.Figure 11 represents the Multi-class extension of Receiver Operator Characteristic graph for the same.

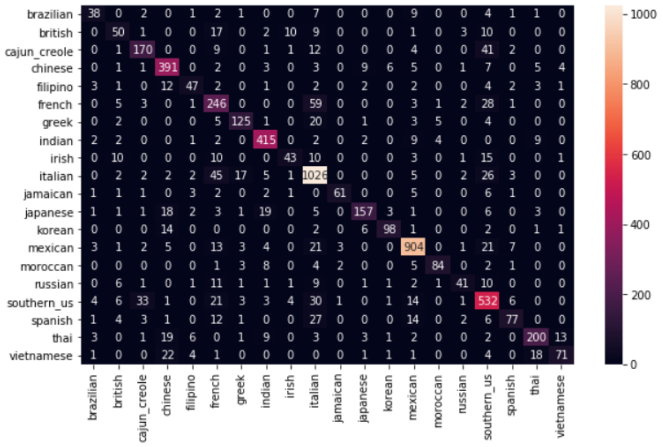

Figure 10. Confusion Matrix: SVM

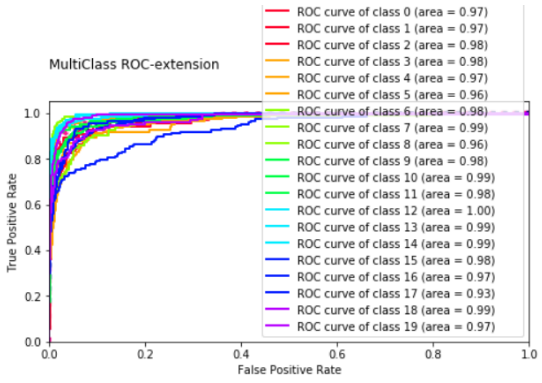

Figure 11. Receiver Operator Characteristic Curve per class: SVM

Table 12 reports the Macro-average and Micro-average

scores for SVM classifier. We observe that the Micro-average score is higher for our best SVM as compared to our best Logistic Regression classifier. The **Final Clean Notebook** linked in the abstract contains our implementation code and visualization for the presented model.

Metric	Score
Macro	0.97723
Micro (weighted by prevalence)	**0.980532**

Table 12. One-vs-One ROC AUC scores for Logistic Regression

6.8. Unsupervised learning

Topic	Common Ingredients
0	pepper oil salt fresh olive ground garlic lemon cloves parsley red wine tomatoes onions extra virgin
1	sugar flour butter salt all purpose eggs milk large baking water cream powder vanilla unsalted egg brown yolks buttermilk whole orange juice
2	cheese pepper chicken cream garlic salt tomatoes shredded onions Parmesan oil seasoning cheddar beans chopped fresh olive bell mozzarella
3	sauce oil soy rice onions sesame chicken water ginger salt green vinegar fresh white red vegetable pork carrots wine
4	pepper cilantro ground oil juice powder chopped cumin onion green red chicken tomatoes chili leaves black bell cloves vegetable coconut

Table 13. LDA topic against the highest probability words from that topic. Topic 1 looks like ingredients from a Salad while topic 2 has dairy-based ingredients. Topic 3 consists of ingredients which are predominantly found in eastern cuisines.

The experiments on unsupervised learning are present in this notebook (notebook). We use the TF-IDF representation using the following parameters in Scikit-learn.

```
TfidfVectorizer(input="content", analyzer="
    word", ngram_range=(1, 1), min_df
    =0.0001, binary=False, use_idf=True,
    smooth_idf=True, sublinear_tf=False)
```

Listing 1. Python example

We run k-means on the above setting with $K = 5$ which we determine by looking at when the reconstruction loss saturates. We operate in a 1983 dimensional space. In Figure 13, we observe that the cuisines cluster into different groups. For example, cluster 1 has Chinese, Japanese, Korean, Thai and Vietnamese.

In order to obtain better visualizations, we perform both TSNE and PCA and reduce all data points to two dimensions. We then represent every cuisine as the average of all embeddings of recipes that belong to that cuisine. We obtain interesting results where similar cuisines like Irish, British, Russian and Southern-US are grouped together and Vietnamese, Japanese, Chinese and Thai are also nearby in embedding space. (all plots present in Figure 12). TSNE yields better embeddings since the algorithm explicitly ensures that local distances are preserved in lower dimensional space.

Finally, we also perform LDA with 5 topics (heuristic choice). This scenario is perfectly suited for LDA since the order of the ingredients is not critical to the recipe (LDA doesn't consider the ordering when generating a document). We see that Topic 4 is mostly comprised of spices while Topic 1 consists of dairy-based ingredients. Table 13 summarizes some high probability words from each topic.

7. Conclusion and Discussion

We explore the *What's Cooking* dataset from Kaggle and achieve an accuracy of 80.36% using the SVM model which is our best performing model (Figure 11, and Table 11). We noticed that SVMs, Neural networks and logistic regression outperform tree based classifiers. This is not particularly surprising since SVMs and neural networks operate better on homogeneous continuous spaced inputs. The best representation is TF-IDF (with hyper-parameters derived from subsection 6.2). A fairly powerful XGBoost classifier is the next best classifier.

Stemming or lemmatization to improve on TF-IDF based representations with simple pre-processing. This could be because the unprocessed tokens have some patterns that correlate with the cuisine. Surprisingly Word2vec also performs fairly poorly. We hypothesize that certain key-words don't have Word2vec representations (397 words don't have representations) and hence these potentially discriminating words are not present in the representation. Furthermore, the Word2vec representation of all ingredients are fairly similar (since Word2vec is trained on all words) and hence the classifier may have less discriminatory power.

One would expect a neural network to outperform an SVM since it is a more powerful model. However, we see that the SVM does better. We believe this could be due to two reasons. The optimization techniques of SVM are superior to that of neural networks in the low data regime. Secondly, we do not use regularization in the form of weight decay/-dropout or batch-normalization (batch-norm has implicit regularization properties). We believe that some fine tuning with neural networks should yield better accuracies. We see that different TF-IDF hyper-parameters do not significantly affect the performance. We believe that smoothing, using log-idf, does not significantly affect representations

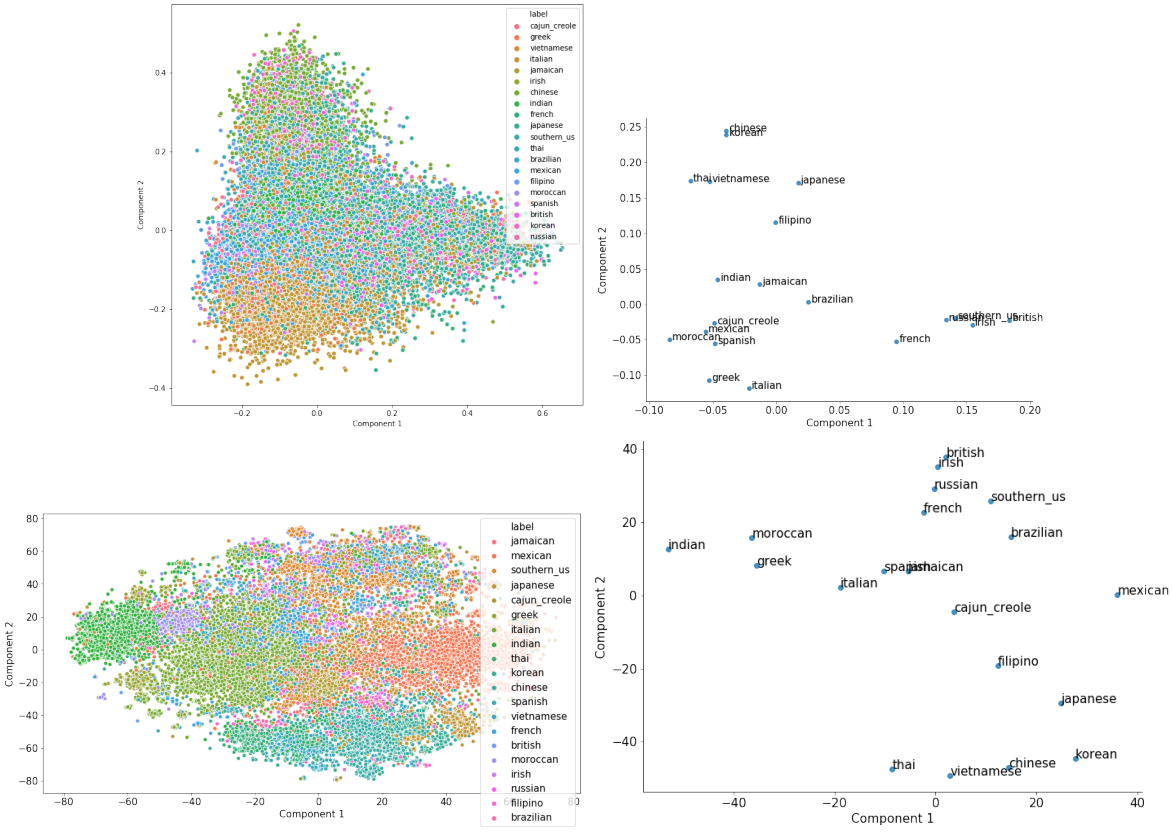

Figure 12. PCA (left) and TSNE (right)

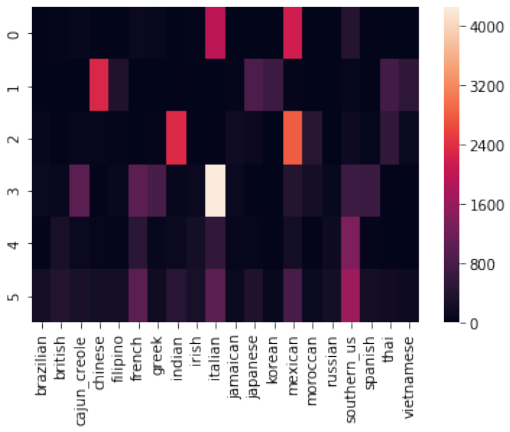

Figure 13. K-means for k=5. The colour intensity represents the number of data points in that particular bin.

for small datasets. However, the choice for the document frequency threshold has some effect on generalization.

For unsupervised learning, we see that the cuisines are neatly grouped into clusters and align with what our intuition with suggest. This suggests that the classification among these family of cuisines might be difficult since some of the cuisines are fairly similar to one another. This also suggests a problem with that data where some cuisines could potentially belong to two classes.

To further improve our performance, a neural network based approach with TF-IDF and Word2vec with effective regularization maybe a good option. We also believe that the dataset has a number of errors (almondmilk as one word, some ingredients are mis-spelled) and hence the Bayes-error of this task maybe fairly high (minimum achievable error). A cleaner dataset might by more useful for generating better representations. A more fine-grained hyper-parameter search will also improve performance since we perform a coarse hyper-parameter search due to a restriction on time and compute. We also analyzed the mis-classified ingredients and found that the number of ingredients in mis-classified samples is far higher that the average data point. We also noticed that multiple southern-us cuisine are mis-classified since the range of recipes of this cuisine is fairly diverse and this cuisines typically has a large number of ingredients.

References

[1] Kaggle Kernel: Cooking is Chemistry Really. https:

//www.kaggle.com/josephgpinto/cooking-is-chemistry-really, 2018.

[2] Kaggle Kernel: What are Ingredients? https://www.kaggle.com/rejasupotaro/what-are-ingredients, 2018.

[3] Kaggle Kernel: TF-IDF with OvR SVM : What's Cooking. https://www.kaggle.com/shivamb/tf-idf-with-ovr-svm-what-s-cooking, 2018.

[4] Kaggle Notebooks. https://www.kaggle.com/c/whats-cooking-kernels-only/notebooks, 2018.

[5] Kaggle. My SVM Test. https://www.kaggle.com/dimanpro/my-svm-test, 2018.

[6] Kaggle. This Model is Bland! Simple Logistic Starter. https://www.kaggle.com/nicapotato/this-model-is-bland-simple-logistic-starter, 2018.

[7] Kaggle. What's Cooking? https://www.kaggle.com/c/whats-cooking-kernels-only/overview, 2018.

[8] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[9] Pedersen, T., Patwardhan, S., and Michelizzi, J. Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pp. 38–41. Association for Computational Linguistics, 2004.

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.