Predicting Cuisines of Recipes

Abstract

This report focuses on the the tasks of using the recipe ingredients for cuisine prediction and recipe completion. We focus on exploring data transformations to maximise the performance of the classification task using a variety of classifiers. Best performance was achieved using TFIDF transformation with SVM, logistic regression and KNN (AUC-ROCS of 0.974, 0.972 & 0.967 respectively). We take forward KNN to explore a potential avenue for recipe completion using ingredient similarity, describing several paths for further development.

1 Introduction

A cuisine is a unique style of cooking distinguished by specific ingredients and techniques typically associated with a culture or geographic area. The ingredient lists of recipes offer great potential for identifying the cuisine of a recipe as many ingredients or ingredient combinations are unique to a specific cuisine. There are limited previous publications that explore recipe cuisine classification using ingredient lists [7, 10, 4] with varying quality. All three of these papers use the Yummly dataset [15] which was produced for a Kaggle challenge. This dataset is larger and more diverse than ours with 50,000 recipes, 6000 unique ingredients and 20 cuisines. These papers achieved the best results using SVM methods and gradient boosting although they were limited in their exploration of data pre-processing or a variety of classifiers. More generally within text classification problems classical classifiers (SVM and logistic regression) have been shown to perform as well as deep learning methods and are also faster to train [8].

There are multiple applications of cuisine classification in academia and industry. Most simply automatic classification of recipes could be useful for curation, for example, on a recipes website. Tangentially, this capability could open up avenues for more dynamic processes such as recipe completion, cuisine transformation or recipe generation. In this report we will focus on exploring the optimal pre-processing and appropriate dimensionality reduction to achieve the best performance with a variety of classification methods. We will also look at some aspects of collaborative filtering for the prediction of ingredients needed to complete a partial recipe which could give chefs new ideas for recipes or allow users to make use of what they already have in their pantry (buying only a few additional ingredients).

2 Exploratory Data Analysis

The data set includes 4236 recipes from 12 cuisines with 709 distinct ingredients, labelled with what type of cuisine (e.g., Italian, Japanese, Chinese, etc.) it is from. It was collected by Facundo Bellosi as part of his MSc at the University of Edinburgh [1]. The recipes are bags-of-word, i.e. a list of an ingredients alphabetically sorted. We work with the data as a binary 4236 x 709 matrix with each row representing a recipe and columns representing ingredients. A 1 indicates that a given ingredient is used in a recipe, and a 0 that it is not. We further use an additional column to represent each of the 12 cuisines. The bags-of-words representation loses important information about the recipes, for example we have no information on the amounts of each ingredient, nor the order in which they are used. Stratified sampling of the original data set with 75% and 25% ratio (i.e. 3177 and 1059 data points) was used to obtain cross validation training set and holdout test set respectively. Exploratory data analysis was performed only on the training set.

Table 1 show the most commonly used ingredients by cuisines as well as the most/least commonly used ingredients across cuisines . Some ingredients seem to be associated with certain geographic areas (e.g., soy sauce in Asian cuisines) while other are common across quite different cuisines (e.g., garlic and onion which are the top two most commonly used ingredients across all recipes in the training set). A total of 23 ingredient had 0 occurrence in the training set. The boxplots in Figure 1 illustrate the distribution of number of ingredients per recipe by cuisine. There seems to be some moderate variability of the median and IQR across cuisines. Tukey's fences (at 1.5× IQR) suggest that only few data points may be considered outliers using this method.

	Top 3 most commonly used ingredients - Ingredient (Percentage)		
Cuisine	**1st**	**2nd**	**3rd**
Chinese	soy sauce (84.15)	garlic (60)	ginger (54.72)
English	onion (54.72)	butter (34.34)	potato (34.34)
French	garlic (53.79)	butter (50.38)	wine (50)
German	onion (66.79)	pepper (40.38)	salt (38.49)
Greek	garlic (70.19)	olive oil (69.06)	onion (56.98)
Indian	onion (72.08)	garlic (70.57)	ginger (62.26)
Italian	garlic (65.91)	olive oil (47.73)	parmesan cheese (45.83)
Japanese	soy sauce (73.58)	rice wine (42.26)	ginger (39.25)
Mexican	onion (55.09)	tortilla (49.06)	garlic (45.66)
Moroccan	onion (75.85)	garlic (66.79)	olive oil (64.91)
Spanish	garlic (79.62)	olive oil (75.09)	onion (67.17)
Thai	garlic (66.67)	fish sauce (51.14)	chicken (50.76)
	Top 1	**Top 2**	**Top 3**
Global most common	garlic (55.59)	onion (51.15)	olive oil (34.14)
	Top 1	**Top 2**	**Top 3**
Global least common	cheshire cheese (0.03)	tequila (0.03)	watermelon (0.03)

Table 1: Most commonly used ingredients by cuisines and most/least commonly used ingredients across all cuisines. 23 ingredients had 0 occurrence in the training set and these were not considered for the least commonly used ingredients herein displayed

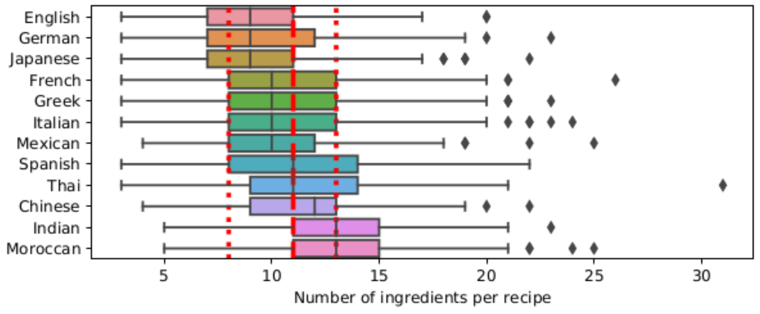


Figure 1: Boxplots showing the distribution of number of ingredients per recipe by cuisine, ordered by median. Whiskers have length 1.5 times IQR (Interquartile Range). Red dotted lines are the 1st and 3rd Quartiles for the entire dataset, red dashed line is the median for the entire dataset. As all values are integers, an outlier diamond might represent multiple datapoints sharing a value.

The histogram in Figure 2 shows the distribution of total occurrences per ingredient. The vast majority of ingredients have a low frequency and may therefore map specific cuisines, whereas very few are used in a large number of recipes.

3 Preprocessing & Dimensionality Reduction

Dimensionality reduction and feature extraction techniques were used on the training set as a preprocessing level. The rationale behind such approaches is manifold: greater interpretability of models developed at a second stage, shorter training times, enhanced generalization by reducing overfitting, learning a manifold of lower dimension which data may live in, and data visualization.

3.1 Methods

Principal component analysis (PCA) was applied as a standard linear dimensionality reduction technique on the training set. No scaling / mean normalization was required prior to PCA given the

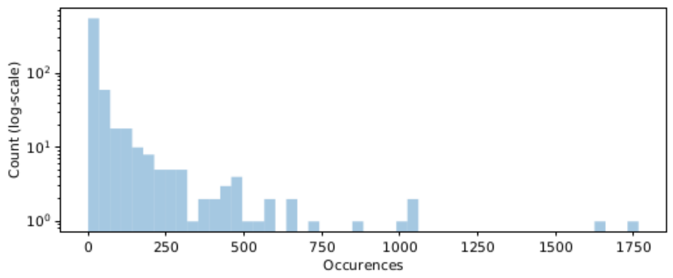


Figure 2: Histogram of the distribution of total occurrences per ingredient. Note that the y-axis is log-scaled so that the bins with fewer ingredients are visible.

binary data. The cumulative explained variance thresholds of 50%, 70%, and 90% were attained upon retention of the first 27, 63, and 174 (out of 709) principal components (as visualised in Appendix C).

Autoencoders are two-part models that maps from the data space $\mathcal{X}$ to a latent space $\mathcal{Z}$ with an encoder and back to the data space with a decoder. The parameters of both parts are optimised to minimise the reconstruction error. This allows for learning to a representation of the data in an unsupervised way, i.e. without using the class labels. Using linear models for the en- and decoder, an autoencoder would be equivalent to PCA so instead we use a two-layer neural networks with Leaky Rectified Linear Units (ReLU) as non-linearity for the encoder to allow for more powerful encoding than in the PCA case. We constrain $\mathcal{Z}$ to have 32 dimensions and use the encoder to project our data to this space after the autoencoder has been fit to the training data. Implementation details for custom methods are in the appendix B

Supervised Learned Embedding is another form of embedding but generated in a supervised way, i.e. taking the class labels into account. Our goal is to learn a matrix W which projects our data to a lower-dimensional space in a meaningful way. We achieve this by training W together with a two-layer neural network with Leaky ReLUs to predict the corresponding class of a recipe using gradient-based optimisation. By using a two-layer network, we do not constrain our lower-dimensional representation to have a linear relation with the classes. This method should retain information about individual ingredients only insofar as they are useful for predicting cuisine which makes the representation potentially very useful for cuisine prediction but not for other down-stream tasks.

TF-IDF stands for term frequency - inverse document frequency. This method allows us to represent the ingredients as a matrix of rational numbers, based on their frequency of occurrence within the recipe and within the entire data-set. This method is an improvement over the Bag-of-Words approach since it not only tells the model whether an ingredient occurs within a recipe but also computes a significance score for each ingredient with equation:

$$\mathbf{tf}(i,r) = \frac{f_r(i)}{size(r)} \quad \text{and} \quad \mathbf{idf}(i,N) = \log\frac{N}{df(i)}$$

where N is the total number of recipes, $f_r(i)$ is the frequency of ingredient in the recipe $size(r)$ is the total number of ingredients in the recipe, and $df(i)$ is the number of recipes in which the ingredient appears. TF-IDF is then computed: **TF-IDF** $= \mathbf{tf}(i,r) \times \mathbf{idf}(i,N)$. Essentially, this means that ingredients common to many recipes should have lower significance than rare ones.

Word2Vec was a further pre-processing selected to explore [12]. This converts words into n-dimensional vectors such that semantically similar words are close to each other. The python library Gensim was used to implement this. The original data-set proved to be too small for Word2Vec to learn any meaningful representations. Instead we re-trained a GloVe embeddings model pre-trained on a much larger data-set with a vocabulary of 1.9 million uncased words [13]. Two-word ingredients were separated and treated as individual words as some of the two-word ingredients such as 'hoisin sauce' were not present in the GloVe model. We were then able to represent each of these new words as vector in 300-dimensional space and these ingredient vectors were then averaged for each recipe to produce the final data transformation in the form of a 3177x300 matrix.

3.2 Visualisation

To visualise the effects of different preprocessing and dimensionality reduction methods, we project each representation of the data into a two-dimensional space using Uniform Manifold Approximation and Projection (UMAP) [11]. The resulting projections can then be visualised as a scatter plots, which is shown in Figure 3.

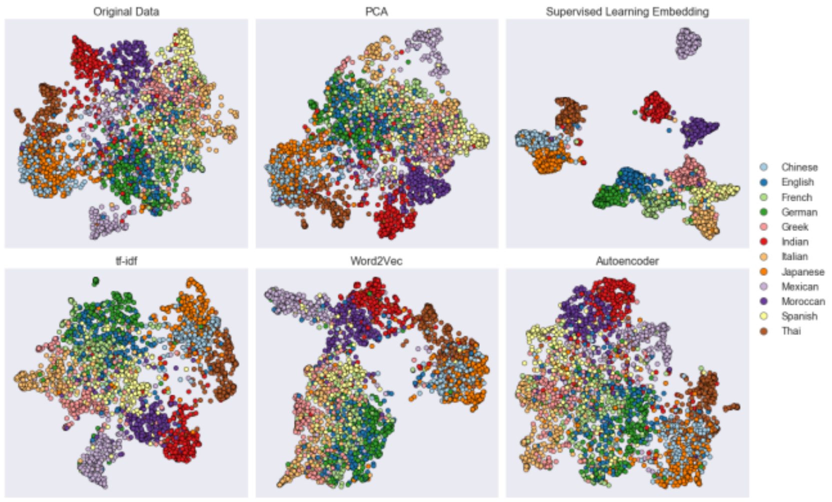


Figure 3: Two dimensional UMAP projections of the recipes in the training split of our data. Each subplot uses a data that was preprocessed in a different way. Colours indicate the corresponding cuisine. We lack the space to include an analysis of the robustness of the resulting projections (w.r.t. subsampling the data and re-running stochastic dimensionality reduction methods). Thus, we decided to tune neither UMAP nor the dimensionality reduction methods. No cherry-picking was done.

The projections reveal several things. Firstly, we can see that the cuisines (represented by point colour) appear in recognisable clusters even in the original data. Based on this, we would expect that our cuisine prediction models should achieve good separation as it indicates that the classes are quite distinct. We can also see that some classes are more distinct than others as they form neat clusters. For example, all plots show a cluster of Japanese, Chinese and Thai (Orange, Light-Blue, Brown, respectively) that is separate from the rest, and Indian and Moroccan (Red, Purple) are close together. Mexican (Lavender) recipes are quite distinct and most isolated in all plots except the Autoencoder one. The supervised learned embedding was the only embedding that used the class labels by being optimised for cuisine prediction, thus information about what makes the dishes of a given cuisine different is not relevant, and as a result we get much tighter clusters than we do with other methods. This means that distance based classifiers (e.g. k-Nearest-Neighbours) should do better when using data that was preprocessed in this way but classifiers that can fit more complex decision boundaries (e.g. RandomForest) might do better on data pre-processing that retains more information. Looking at the UMAP plot for the supervised learned embedding also gives us some indication as to what we might expect in terms of cuisine prediction performance, wherein we would expect that recipes that lie in a cluster dominated by another cuisine will be very hard to correctly predict. Furthermore, our cuisine classifiers will likely confuse cuisines whose clusters neighbour more often than those whose clusters are far apart.

We can also see that there appears to be a cluster of outliers for the Thai dishes (plotted in brown) which is recognisable in the plots for the original data, PCA and Autoencoder. Generally, these three plots appear characteristically similar in terms of which cuisines are close together and which are farther apart. This indicates that our transformations appear to capture most of the information even though the original data had 709 dimensions whilst the PC scores had only 171 dimensions and the latent autoencoder representations had only 32 dimensions. We also observe that the plots for tf-idf and Word2Vec seem to have more global separation between cluster than the other representations, except the supervised learned embedding. Interestingly, despite bringing in additional information by being pre-trained on external data, the Word2Vec plot is quite similar to the remaining plots.

4 Task 1 - Cuisine Prediction

The goal of this task is use the ingredients of a given recipe to classify it as belonging to one of the twelve cuisines in our dataset.

4.1 Classification Algorithms

We evaluate five different classification algorithms: Random Forest (RF), Logistic Regression (Logit), k-Nearest-Neighbours (KNN), Support Vector Machine (SVM), and Deep & Cross Network (DeepCrossNet).

Random Forest is an averaging ensemble of Decision Trees [3] and was selected as they usually achieve excellent performance on tabular data and can learn complex, non-linear decision boundaries while still generalising well to unseen data. **Logistic Regression** is a linear model that uses a logistic function to model the probabilities of a binary outcome whose model coefficients are fit with maximum likelihood estimation. To obtain a class probability rather than a label, we use the fraction of k neighbours that belong to the given class [5]. They generally performs robustly in a wide range of applications; however, without basis functions and feature engineering, it can only fit a linear decision boundary in variable space and cannot account for interactions. **KNN** classifies based on the class of a number of nearest neighbours and should be very sensitive to the representation of the data and thus might highlight effective dimensionality reduction methods. A **SVM** separates classes with a hyperplane in feature space while aiming to maximise the margin between classes. Using kernel functions, SVMs can fit non-linear class boundaries [2]. SVM and Logit are extended to multi-class classification in a "one-vs-rest" fashion. They are effective and popular classifiers, and although they scale poorly to large datasets in terms of computational complexity. We wanted to include a neural network as they might work well on the data at hand. We select **DeepCrossNet** which is a neural network that uses crosslayers [14] in parallel to standard feed-forward layers. Crosslayers take the input to the network x_0 and calculate $x_{n+1} = x_0 x_n^T w_n + b_n + x_n$ where x_n is the output of the n-th crosslayer with weights w_n and biases b_n, respectively. We can calculate $(n+1)$-th degree feature interactions by stacking n crosslayers. The output of both the crosslayers and the feed-forward layers are then concatenated and used as the input to the output layer. The whole model can then be trained with gradient-based optimisation. Additionally, we implement a simple Baseline Classifier which computes per class column-wise means across all training samples. To predict the cuisine of a new recipe, we simply take the distance between the recipe and the mean of each class. We then predict the cuisine with the smallest distance (we consider cosine, euclidean and Manhattan distances), or to obtain class probabilities, we take the softmax of negative distances.

4.2 Metrics

Our EDA has shown that the class distribution of the dataset is perfectly balanced, thus using accuracy is generally appropriate in this case. Although, for classification models that yield class probabilities rather than labels as predictions, this would require a decision procedure to map the probabilities to labels. In a perfectly balanced task such as ours, simply predicting the label that has the highest predicted probability is an appropriate choice. However, this would limit the application of developed tools to the inclusion of further data that may not be balanced.

To mitigate this, we use the Area Under the Receiver Operating Characteristic Curve (AUC-ROC). The advantage of this metric is that it does not depend on the choice of decision threshold. It also has an intuitive interpretation: given a randomly selected case from the positive and negative class each, the AUC-ROC approximates the chance that our classifier has scored the positive case higher than the negative case. The AUC-ROC is a metric for binary problems which we extend to our multiclass setting in a "one-vs-rest" fashion, meaning that we compute the AUC-ROC for each one versus rest problem (e.g. Thai versus non-Thai), and then average these values.

4.3 Methods

The data was split into a cross-validation and holdout dataset in a 75:25 ratio. We consider five different representations of our data and six classification algorithms with six hyperparameter settings each for optimisation (see Appendix D). All pre-processing was implemented as part of the the cross-validation pipeline to prevent information leakage, with the transformations fitted on the training fold only. As different representations of the data may impact the optimal hyperparameter settings we conduct 5-fold CV grid search optimisation on each pair of pre-processing and classifier. This results in $5 \times 6 \times 6 = 180$ variations with 5 fits per variant for a total of 900 model fits. After optimising each of these pairs we take the optimised models and evaluate their performance on the holdout data-set. The performance during the initial cross validation can be used to estimate the overall generalisation performance.

4.4 Results

Evaluation of each of the optimised pre-processing-classifier pairs was performed using the holdout dataset to produce the AUC-ROC scores shown in Figure 4. The highest overall performance was using TFIDF encoding in combination with SVC and Logistic Regression (AUC-ROC of 0.974 and 0.972 respectively), although interestingly the KNeighbours Classifier also performs well after hyperparameter optimisation, achieving the third highest AUC-ROC (0.967). The baseline classifier is the lowest performing classifier for every pre-processing suggesting that the classifiers are successfully learning features from the data. We find that TFIDF encoding is consistently a strong form of data pre-processing, offering the highest performance on all but one classifier (Random Forests). Word2vec and the autoencoder display the worst performance of the pre-processings with AUC-ROCs worse than the original sparse matrix for almost all classifiers.

Hyperparameter optimisation yielded a mild mean improvement in the performance of the classifiers with +1.690% AUC-ROC and +4.240% accuracy. The optimal hyper-parameters we found are shown in Appendix E. Of note KNN had the largest improvement with tuning, as the classifier performance improved significantly with a higher numbers of neighbours. The cross-validation optimisation displayed reliability across the classifiers as shown in Appendix F with box plots of the AUC-ROC scores achieved.

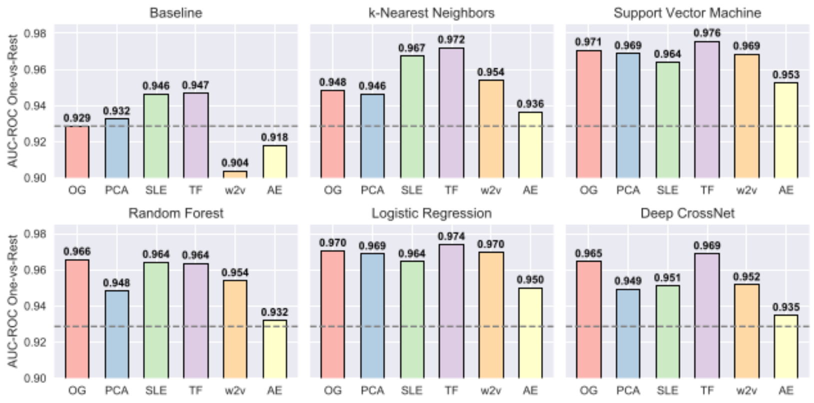


Figure 4: Model performance (one-vs-rest AUC-ROC) on holdout test data per classification algorithm (subplots) per data representation (columns). Grey dashed line indicates baseline classifier performance on original data for comparison. Note that the y-axis range is from 0.9 to 0.98 (rounded mini- and maximum values) to allow easier comparison of the methods. However, all results reflect decent separation of classes (AUC-ROC > 0.9).

4.5 Discussion

The significant uplift in AUC-ROC when using TFIDF validates its standard use in bag-of-words datasets. The presence of very common ingredients that are not providing useful information for classification and the impact of these ingredients on the classifiers is minimised by the inverse document frequency scaling of TFIDF. We saw examples of such ingredients in EDA - for example, onion and garlic occurred in 30-40% of recipes for most cuisines, and the importance of these will be greatly reduced by TFIDF. However, PCA will achieve a similar effect on the input data but does not see the same improvement in performance. This suggests that TFIDF is also improving the results as it allows the less common ingredients, which are likely more unique to specific cuisines, have a greater influence on the classification. There is potential for further processing of the TFIDF transformations - for example, PCA or SLE could be used. However, given the already large number of combinations, sequential pre-processing was outwith the scope of this project.

The poor performance of Word2vec may indicate that splitting 2 word ingredients removed some information from the dataset. This could have been avoided by using is a larger dataset to allow for direct Word2Vec representations without pre-training and this may have improved performance. Supervised learned embedding outperformed the original data for the baseline classifier and the KNN classifier where distances are used directly. This is expected given the results seen from the SLE UMAP plot 3.

5 Supplementary Task - Recipe Completion

5.1 Methods

We used the Surprise Python scikit library [6] to explore recipe completion and ingredient recommendation with collaborative filtering. This library implements various collaborative filtering methods, which can be broadly classified into memory-based methods, such as K-Nearest-Neighbor based methods, and model-based methods, such as SVD-based matrix factorization. They can be further subdivided into user-based and item-based collaborative filtering. Here we use it to explore K-Nearest-Neighbor item-based collaborative filtering.

We removed ingredients that appeared in fewer than 6 recipes from the dataset as they contain negligible information. The data was then reshaped to a long and skinny matrix with three columns containing: user ids (recipe index), items (ingredients), and ratings (1 if ingredient appears in recipe, 0 otherwise). In effect, this transformation represents each recipe as a user who has rated a number of ingredients as either present or absent. This dataset was used to cross-validate multiple KNN models, varying the similarity metric used between pairs of ingredients (Pearson Correlation Coefficient, Mean Squared Difference, Cosine Similarity) and computing their precisions and recalls for the top 20 predicted ingredients above a threshold value. From this search, it was concluded that the Pearson correlation coefficient similarity measure achieved the highest performance. A similar search was also performed on the models using the Pearson correlation coefficient and mean-squared-error as a loss metric. Here, the maximum number of neighbours, k, was set to either 40 or 100. The performances were compared to a naïve model, which randomly predicted whether the ingredient appeared in a recipe based on an assumed normal distribution of the dataset. The highest performing model, KNNWithZScore was selected. Next, this model was trained on the full dataset using the optimal parameters found by the searches. However, it was only able to predict how likely an ingredient was to appear in a given recipe. As such, a function was needed to yield recipe completions given a list of partial ingredients. This was accomplished via two different methods. The first method, when given a list of n ingredients, created a predictions matrix with the number of rows corresponding to the number of recipes and the number of columns equal to the n partial ingredients provided i.e., it calculated the likely-hood of each input ingredient appearing in each recipe. Then, the likely-hoods were summed across recipes yielding a total score for each recipe. The indices of the highest scoring recipes were found and cross-referenced with the recipes in the original dataset. The missing ingredients from the found recipes were then printed. If no recipe containing all the input ingredients was found, the unnecessary ingredients from the input list were also printed alongside the closest-matched recipes. The second method used the same trained model, but instead of scoring all input ingredients against all recipes, it recommended n missing ingredients by calculating the k nearest neighbors to each of the ingredients given, and then finding the n most common ingredients among the neighbors excluding the input ingredients. This method did not "cheat" by looking up the recipes, but instead, relied fully on the KNN model to suggest essentially completely new recipes.

5.2 Results

The results of the 10-fold cross validation of various KNN algorithms with various similarity measures yielded the mean precision and recall scores tabulated in Table 5, these are compared to the baseline performance of a naïve predcitor and outperform it significantly. The results of the mean RMSEs with maximum number of neighbors k=40 and k=100 for various KNN algorithms are also compared to the naïve predictor in Table 2. From these searches, it was concluded that the best performing model was the KNNwithMeans algorithm with a maximum k of 100, using the Pearson correlation coefficient to compute similarities between pairs of ingredients with a mean precision of the 10-fold cross validation of 0.283 and a mean recall of 0.308, and a mean RMSE of 0.141. With regards to the recommendation engine, the first method always predicted the correct missing ingredients by memorising the dataset. The second method suggested a user-specified n number of ingredients, which could go together with the input ingredients. These two methods were packaged in a mini terminal app, which can be used by anybody to recommend either real recipes (method 1) or completely new ones (method 2). The completion engine can be run on Windows and Linux operating systems by following the instructions in the ReadME.md file.

5.3 Discussion

Both the RMSE values and the mean precisions and recalls were much higher for all the KNN-based models than for the Normal Predictor baseline model. The Evaluation of recipe completion model is more complex than evaluating a classifier. Consider that if we obtain a test sample by taking a recipe from our dataset and removing an ingredient, there are, in principle, many sensible ways of completing it. For example, if we remove the protein from a given recipe, many different cuts of meat from different animals - or a vegetarian alternative - could complete the recipe. Thus, a model might complete the recipe differently, yet correctly.

Algorithm	RMSE (k = 40)	RMSE (k=100)
Naïve Predictor	0.19703	0.19706
KNN Baseline	0.14408	0.14507
KNN Basic	0.15528	0.15727
KNN with Means	0.14405	0.14503
KNN with Z-Score	0.14018	0.14146

Table 2: Table showing the mean RMSEs of 10-fold cross validation for various KNN collaborative filtering algorithms using the Pearson correlation coefficient as as similarity measure between pairs of items.

Mean Precisions		Similarity Method		
		Cosine	MSD	Pearson
Algorithm	Naïve Predictor	0.0267	0.0262	0.0268
	KNN Baseline	0.2364	0.2155	0.2651
	KNN Basic	0.1393	0.0244	0.1396
	KNN with Means	0.2362	0.2159	0.2648
	KNN with Z-Score	0.2617	0.2155	0.2832

(a)

Mean Recalls		Similarity Method		
		Cosine	MSD	Pearson
Algorithm	Naïve Predictor	0.0921	0.0908	0.0268
	KNN Baseline	0.3067	0.1918	0.3147
	KNN Basic	0.2907	0.0151	0.1586
	KNN with Means	0.3055	0.1931	0.3131
	KNN with Z-Score	0.2906	0.2035	0.3078

(b)

Figure 5: A table showing the mean precisions and recalls for the various KNN collaborative filtering algorithms using various similarity measures to compute similarities between ingredients.

6 Conclusions

Cuisine Prediction: For this classification task we achieved consistent high performance with SVM and logistic regression classifiers are expected for sparse data with reasonable class separation and clustering as seen in the UMAPs shown in Figure 3. The use of TFIDF was validated as the best pre-processing across classifiers for improving performance in this bag-of-words dataset. Combined we achieve AUC-ROCs of 0.974 and 0.972 for TFIDF transformation with SVC and Logistic Regression respectively.

Recipe Completion: The prediction engine is functional and outperforms the naïve predictor baseline. Qualitatively, the ingredients suggested by the model seem sensible (see Appendix: H). However, more data would likely improve performance and the quantitative results are complicated by the fact that the dataset contains recipes, which represent only a small slice of the culinary universe so even a sensible completion might not appear in the data and could be scored as incorrect. Developing methods of evaluation would be a critical next step here.

References

[1] Facundo Belloni. *Machine learning for cuisine discovery*. UoE, 2012.

[2] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Rishikesh Ghewari and Sunil Raiyani. Predicting cuisine from ingredients. *University of California San Diego*, 2015.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[6] Nicolas Hug. Surprise, a Python library for recommender systems. http://surpriselib.com, 2017.

[7] Shobhna Jayaraman, Tanupriya Choudhury, and Praveen Kumar. Analysis of classification models based on cuisine prediction using machine learning. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, pages 1485–1490. IEEE, 2017.

[8] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification, 2016.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[10] RM Rahul Venkatesh Kumar, M Anand Kumar, and KP Soman. Cuisine prediction based on ingredients using tree boosting algorithms. *Indian Journal of Science and Technology*, 9(45):12, 2016.

[11] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, September 2013. URL http://arxiv.org/abs/1301.3781. arXiv: 1301.3781.

[13] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[14] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[15] Yummly. Understanding cuisines using a new dataset from yummly. URL https://www.yummly.com/insights/understanding-cuisines.

Appendices

A Training Set at a Glance

CUISINE	NO. RECIPES PER CUISINE	NO. INGREDIENTS IN A RECIPE				
		MEAN	MEDIAN	STD	MAD	IQR
CHINESE	265	11.49	12	3.20	2.59	4.0
ENGLISH	265	9.28	9	3.26	2.60	4.0
FRENCH	264	10.53	10	3.67	2.95	5.0
GERMAN	265	9.68	9	3.61	2.95	5.0
GREEK	265	10.83	10	3.61	2.84	5.0
INDIAN	265	12.88	13	3.20	2.55	4.0
ITALIAN	264	10.58	10	3.81	3.01	5.0
JAPANESE	265	9.13	9	3.25	2.60	4.0
MEXICAN	265	10.23	10	3.44	2.69	4.0
MOROCCAN	265	13.16	13	3.55	2.84	4.0
SPANISH	265	11.10	11	3.60	2.88	6.0
THAI	264	11.80	11	3.74	3.02	5.0

Remark on figure 1 Using Tukey's fences (at $1.5\times$ IQR), only few data points may be regarded as outliers. We inspected the most extreme case which is relative to the Thai cuisine and which also happens to be the only recipe under this cuisine to lie further than $1.5\times$ IQR. Here is the list of ingredients for this data point: Broth, Carrot, Chicken, 'Chili Oil', Coconut, 'Coconut Milk Or Cream', Coriander, Cornstarch, Cumin, Garlic, Ginger, 'Green Onion', Lime, Mace, Onion, Paprika, Parsley, Pea, Peanut, 'Peanut Butter', Pepper, Rice, 'Rice Vinegar', Salt, Shrimp, 'Soy Sauce', Sugar, Turmeric, 'Vegetable Oil', Water, Zucchini. Upon a closer look, this does seem a data error.

B Implementation details

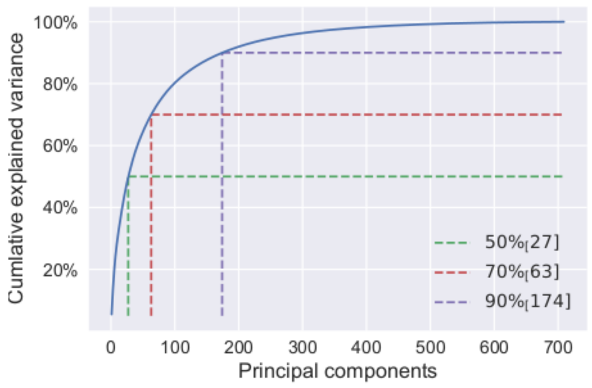
All Leaky ReLUs use a negative slope of 0.01.

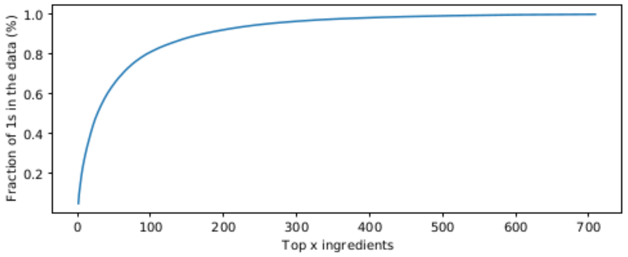
Autoencoder: The autoencoder is trained with a batch size of 128 for 150 epochs with the Adam [9] optimizer using a learning rate of 0.0001, reduced by half after 50 epochs minimising the element-wise binary crossentropy loss between reconstructed and actual recipe. The output is softmax activated - this means that the recipe cannot be perfectly reconstructed as the model can only allocate a mass of 1 for its output. However, it also ensures that the model allocates sufficient mass. Using element-wise sigmoid activations would be more intuitive but as most recipes do not have a particular ingredient this leads to the model diverging and predicting 0 for all ingredients. We also use one-sided label-smoothing to change target values to 0.95 instead of ones. The hidden layer of the encoder has 64 hidden units, the latent space 32 units.

Supervised Learned Embeddings: The network for this method has the embedding layer projecting the input data to a 32-dimensional space, followed by a hidden layer with 64 units, followed by the output layer. It also uses Leaky ReLUs with a batch size of 128 for 150 epochs and has a softmax activation for the class predictions. It is trained to minimise the crossentropy loss using Adam with lr=0.0005, which is halved after 50 epochs.

Deep cross net: The input to the network is optionally rescaled column-wise to the interval [-1,1]. The model uses a weight decay of 1e-5 for regularisation. The model consists of a two hidden-layer neural net with 16 hidden units with Leaky ReLU activations in parallel to the cross layers (which are a hyperparameter). The outputs of both are then concatenated and a output layer maps them to the class space with a softmax activation. It is also trained with Adam(lr=0.00005) for 35 epochs.

C Cumulative Explained Variances of PCA compared to fraction of 1s per top ingredients





D Available Hyper-parameters

Classifier	Parameters [Options]	
kNN	n Neighbours [25, 50, 100]	Weights [uniform, distance]
SVM	Kernel [poly, rbf]	Reg Strength [0.2, 1.0, 5.0]
RF	Max Depth [None, 6, 12]	Max Features [sqrt, 1.0]
Logit	Penalty [l1, l2]	Reg Strength [0.2, 1.0, 5.0]
DCN	Number of Epochs [50, 100, 250]	Scaling [True, False]

E Optimal Hyper-parameters

	kNN	SVM	RF	Logit	DCN
OG	n_neighbors=50, weights='distance'	-	max_features='sqrt'	-	n_epochs=50, scaling=False
PCA	n_neighbors=50, weights='distance'	-	max_depth=12, max_features='sqrt'	C=0.2	n_epochs=100, scaling=False
SLE	n_neighbors=100, weights='distance'	-	max_depth=6, max_features='sqrt'	penalty='l1'	n_epochs=250
TF	n_neighbors=50, weights='distance'	-	max_features='sqrt'	-	n_epochs=50, scaling=False
w2v	n_neighbors=50, weights='distance'	C=5.0	max_features='sqrt'	C=5.0	n_epochs=50
AE	n_neighbors=50	C=5.0	max_depth=12, max_features='sqrt'	C=5.0	n_epochs=250

F Cross Validation Performance

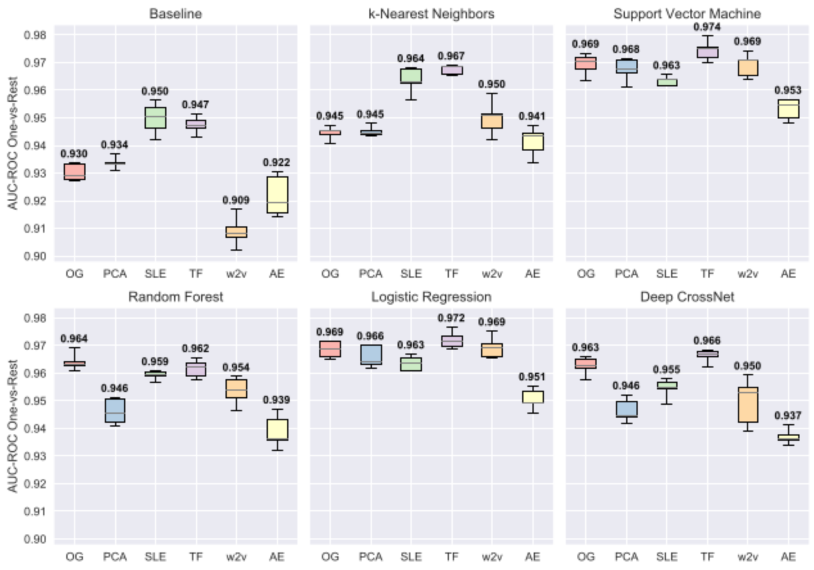


Figure 6: Boxplots of ROC AUC cross-validation scores. Reported values of one-vs-rest ROC AUC. Whiskers indicate maximum and minimums. Grey line indicates median. Mean is shown above each boxplot. OG - Original Data, PCA - Principal Component Analysis, SLE - Supervised Learned Encoding, TF - TFIDF Encoding, w2v - word2vec Encoding, AE - AutoEncoder.

G Additional Ideas for Recipe Completion

These are further ideas that we didn't implement due to lack of time.

Frame it as a classification problem: for a sample of the training data, remove an ingredient from the recipe which is the target and use the stub as the input. We can also remove more than one ingredient and have a target vector with excess probability mass to obtain more data and learn more robust completions

A complication with this approach is that there are 709 ingredients which would be our target classes. If we tried, for example, to use logistic regression for this problem, we would end up fitting over half a million parameters. Neural networks are much more efficient in this case as they can learn a lower dimensional projection first. We consider a DeepCrossNet with a learned input embedding matrix.

Furthermore, we consider an interesting variation of this model. Instead of having an $n_{ingredients}$-dimensional output, we obtain an output of the same dimension as our lower dimensional learned embedding. We then use the inverse of the embedding matrix to project the output of the neural network to ingredient space. The core idea is that the embedding matrix should in a meaningful way capture the similarity of ingredients, which is useful for both the initial dimensionality reduction and for mapping back to the ingredients. A complication is that the embedding matrix might not be invertible, but there might be tricks around that?

Another way of framing the problem is the use an autoencoder. We can train it with full recipes and stubs, yet ask for the original, full recipe as the output in both cases. For a evaluations stubs, the reconstruction should then yield a distribution over possible completions.

Finally, we use truncated SVD to learn a projection of our training data and evaluation stubs jointly. Project and reconstruct stub recipes to get a distribution over ingredients that could complete the recipe (ignoring those that are already part of the recipe). We can also add construct stubs from the training data to increase the size of the dataset we fit the truncated SVD to. We can then project and recover a stub recipe to get a distribution over possible completions.

we take an approach similar to the Netflix price solution https://mlpr.inf.ed.ac.uk/2020/notes/w9b_netflix_prize.html The dataset is decomposed into a tall-and-skinny and a short-and-fat matrix. This is similar to SVD yet the singular values are "consumed" by our two matrices.However, whereas the Netflix price dataset contained user ratings per movie ranging from one to five, we have binary information about whether a recipe contains a specific ingredient. And while the challenge in the Netflix

As a baseline, we predict the most common ingredients for a given cuisine that are not part of the recipe already, using statistics from the training data.

G.1 Metrics

Evaluation of recipe completion model is more complex than evaluating a classifier. Consider that if we obtain a test sample by taking a recipe from out dataset and removing an ingredient, there are, in principle, many sensible ways of completing it. For example, if we remove the protein from a given recipe, many different cuts of meat from different animals - or a vegetarian alternative - could complete the recipe. Thus, a model might complete the recipe differently, yet correctly.

We try to remedy this in two ways: First, our models yield a probability distribution over possible completions. Thus, instead of considering only one, we take the top-k completions. Secondly, to account for other sensible completions, we search the entire current evaluation dataset and the training dataset. If a stub recipe occurs in the training data, it is not undesirable for a model to complete it as it was in the training data. Still, we also report the performance on the current evaluation dataset alone to give the reader an idea of how important memorisation was to the performance of a given model. To allow for this second adjustment, we use accuracy rather than a ranking metric like AUC-PR.

Still, the dataset is quite small compared to the wealth of cultural diversity in the culinary world.

H Recommender

```
Welcome to the Ingredient Recommendation Engine

Please put in all your ingredients separated by commas: lamb, pea, beef stock, carrot

Would you like to use lookup? (yes/no): n

Please put in the number of recommendations you would like to get : 3

To complete these ingredients: ['lamb', 'pea', 'beef stock', 'carrot'], we suggest the following
['onion', 'potato', 'pearl onion']

 Enjoy!



Would you like to try again? (yes/no): yes

Please put in all your ingredients separated by commas: soy sauce, prawn, chile pepper, miso

 Would you like to use lookup? (yes/no): n

Please put in the number of recommendations you would like to get : 3

To complete these ingredients: ['soy sauce', 'prawn', 'chile pepper', 'miso'], we suggest the fo
['sesame oil', 'rice wine', 'ginger']

Enjoy!
```

```
Would you like to try again? (yes/no): yes

Please put in all your ingredients separated by commas: duck,garlic,ginger,honey,pepper,salt

Would you like to use lookup? (yes/no): yes

To complete these ingredients: ['duck', 'garlic', 'ginger', 'honey', 'pepper', 'salt'], we sugge
['soy sauce', 'sugar']
These ingredients come from a Chinese recipe. Enjoy!
```