Predicting Cuisines of Recipes using Ingredients

Rohan Gorantla
s2112695
`rohan.gorantla@ed.ac.uk`

Tamara Lottering
s2124912
`t.lottering@sms.ed.ac.uk`

Akash Bhagran
s2123972
`A.Bhagran@sms.ed.ac.uk`

Aditya Suryanarayan
s2089883
`a.k.suryanarayan@sms.ed.ac.uk`

Abstract

Most cultures across the world are characterised by the unique combination of ingredients and culinary procedures that make up their cuisines. The purpose of this analysis is to examine cuisine classification and ingredient prediction to understand the relationships between different cuisines and their ingredients. Using various exploratory data analysis, pre-processing and machine learning approaches, we build and evaluate the performance of a recommendation system designed to carry out two tasks: (1) predict a cuisine given a set of ingredients (cuisine classification), and (2) complete a recipe given a few ingredients (ingredient prediction).

1 Introduction

Since the beginning of the lockdown period due to the coronavirus pandemic in 2020, there has been increased levels of home baking and cooking as a 'self-care' practise to reduce stress and anxiety. In addition, there is increased public interest in maintaining a healthy lifestyle. Whether it is to fill a void during the pandemic or to maintain good nutrition, the increasing availability of cuisine data has led to data-driven recipe recommendation and generation. Cuisines comprise of a unique combination of ingredients that make up a recipe and are associated with a specific geographical region or culture. Cuisines are thought to be one of the most important factors consumers use when deciding what to eat, making cuisine recommendation systems important in e-commerce.

Previous works[1] approach cuisine classification and ingredient prediction as a Text Classification(TC) problem, by annotating text with categories based on words and their collocation. Others have explored cuisine identification on the basis of using ingredients used in recipes [2, 3]. Others have built recommendation systems that suggest recipes using neighbourhood-based Collaborative Filtering models(CF)[4], by using past ratings of recipes from other users. Support Vector Machines(SVM) and several other machine learning approaches have been employed to effectively perform cuisine classification from the ingredients used in a recipe. The related works however, have not attempted to address the 'explainability' of their implemented machine learning models i.e. predictions of the models, or the reasons behinds these predictions. While their models have been very accurate in classifying cuisines, they do not account for how and why a certain cuisine is predicted for a given set of ingredients. To do this, our analysis uses state-of-the-art machine learning explainability frameworks known as LIME (Locally Interpretable Model agnostic Explanations)[5].

For overall exploratory data analysis we use Multi-dimensionality Scaling(MDS) and t-distributed Stochastic Neighbour Embedding(t-SNE) to better understand the data and its structure. We also try to identify the most important ingredients in each cuisine, using methods from Information Retrieval. Our first task of cuisine classification specifically explores the performance of SVMs, Logistic Regression, and Random Forest classifiers on a set of features defined using univariate statistical tests and dimensionality reduction using Principal Component Analysis(PCA). Our second task examines the use of collaborative filtering to predict ingredients given a partial recipe, where in our context the users are the recipes and the items are ingredients. Collaborative filtering uses a matrix of all items and users to produce recommendations, where each element of the matrix represents how much a user likes an item.

2 Exploratory Data Analysis (EDA)

The data used for the experiments consists of 4,236 recipes from 12 different cuisines: Chinese, English, French, German, Greek, Indian, Italian, Japanese, Mexican, Moroccan, Spanish, and Thai. There are a total of 709 distinct ingredients. The data is spread equally, with 353 recipes in each cuisine. There are no recipes in the dataset that have more than 1 unit of an ingredient, and as such we will assume that only the presence or absence of an ingredient is noted in each recipe, and not the units required.

With this distinction in mind, we see that all the recipes require an average of 11 ingredients. Indian and Morrocan recipes seem to require more ingredients, with an average of approximately 13. Japanese recipes seem to require the least ingredients, with an average of 7. For the particular recipes in our data, the average number of times an ingredient is used overall is 64. This is heavily skewed however, with a large standard deviation of 194 and a low median value of 8. This means our dataset is highly sparse. In order to better understand the data and its structure, we applied standard dimensionality reduction techniques - PCA, MDS and t-SNE - to visualize our dataset in lower dimensions. We also identified the most important ingredients in each cuisine using methods from Information Retrieval. We discuss the findings and results from our EDA here, and provide more information regarding the methods used in the Appendix for the interested reader. All plots used for this discussion can be seen in Figure 1, or in the Appendix where mentioned.

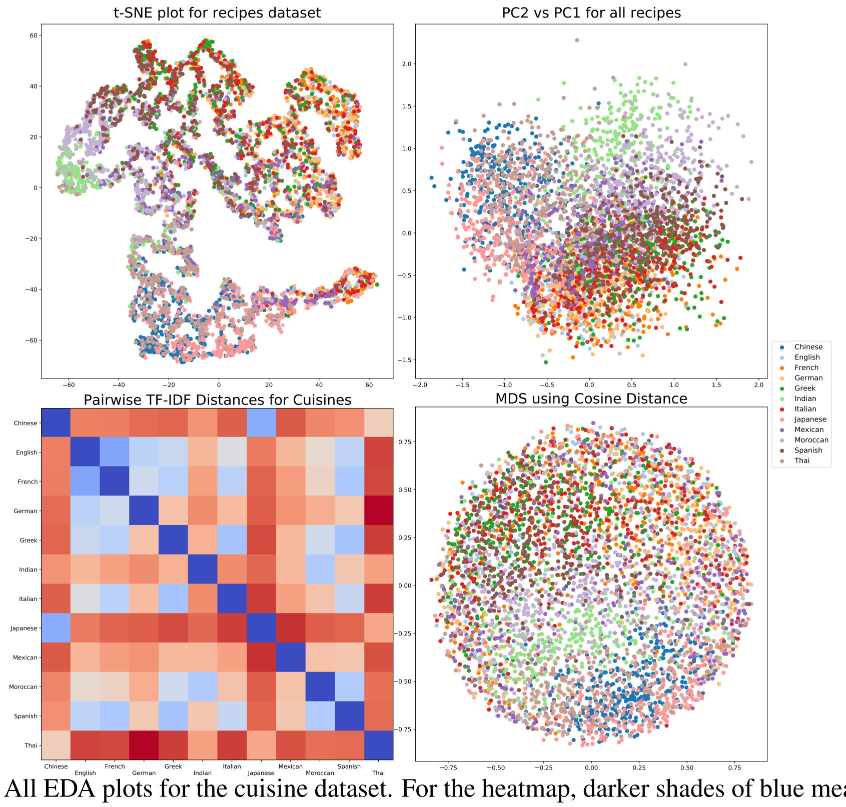


Figure 1: All EDA plots for the cuisine dataset. For the heatmap, darker shades of blue mean cuisines are closer together, darker shades of red mean cuisines are farther away from each other.

We first look at dimensionality reduction using Principal Component Analysis(PCA). We see in the PC scatterplot in Figure 1 that most of the recipes from Asian cuisines are to the left along PC1. European cuisines are to the bottom of PC2, while Indian, Moroccan and some Asian recipes are towards the top. Thus, it seems that the major source of variation between the recipes and cuisines in our dataset is the "base" of the recipe - European recipes have a more fatty base of butter and oils while Asian recipes seem to lean towards a different base that uses soy sauce, sesame oil and ginger. The second source of variation in the dataset seems to be the use of spice - Asian, Moroccan and Indian recipes do use more spice than European recipes.

Multidimensional Scaling (MDS) is a visualization technique that helps in identifying underlying structure in the data. An important consideration for MDS is the distance measure. We use Cosine distance between recipes as opposed to Euclidean distance since we are not concerned with the magnitude of the vectors. Euclidean distance might be better suited for cases where the units of ingredients are available, since length of a the recipe vector would matter and would allow for the separation of two otherwise identical recipes that differ in just units of one ingredient e.g. sugar. In our case, the data is which is one-hot encoded based on whether the ingredient is used in the recipe thus these are identical recipes since they use the same ingredients, and so would be the same vector. While the MDS graph is dense, some clear patterns are seen. All Asian cuisines are clustered together on the lower end, while European cuisines are clustered together in the upper half, with Indian and Moroccan cuisines in the transitory space. Considering just the density of points, we could say that there are some main clusters like European and Asian, and Indian and Moroccan recipes do not seem to fall in either of these from this MDS plot. A view of the same plot split by cuisine can be found in the Appendix. A plot using Euclidean distance is also present in the Appendix, justifying our choice of Cosine distance since we cannot find the same clear structure from the corresponding MDS plot.

A more flexible method of visualizing structure in high-dimensional data is t-SNE. From the MDS plot, it is unclear how many clusters there are in the data. This however, is immediately clear from the t-SNE plot, which shows 3 clusters. We see that all European recipes and Asian (Chinese, Japanese, Thai) recipes are in separate clusters. Indian and Moroccan recipes are clustered together on the top-left. The fact that Indian cuisine is always between Asian and European makes sense even from a historical perspective given the country's position on the global trade route, which is how most of this intermingling of recipes might have occurred. All these clusters are in line with what we have observed so far, although the distribution and position of the clusters seems much more different. The consistent considerable overlap between individual cuisines is also to be expected since we consider here what can be called "major" cuisines, each of which would have taken some inspiration from the other at some point in history. This overlap tells us that fine clustering might not achieve good results on this dataset. Furthermore, information on the method of cooking, or accompaniments to each recipe, might be needed to achieve good separation of cuisines.

Using methods from Information Retrieval, we convert each cuisine into a vector in the ingredient dimension, using TF-IDF. This is then used to find similarities between cuisines using cosine distance between the vectors, and the most important ingredients for each cuisine. The pairwise distances corroborate our previous findings from MDS and t-SNE regarding the closeness of European cuisines to each other, and that of Asian cuisines to each other. Additionally, TF IDF tells us something that is not immediately clear in either MDS or t-SNE: while German cuisine is close to English and French, it is not that close to other European cuisines. Similarly, Greek cuisine seems to be as close to Spanish and Moroccan cuisine as it does to English and French. The distinction between Asian and other cuisines is very distinct, but even here it seems that Chinese and Japanese cuisine are closer to each other than they are to Thai. Indian and Moroccan cuisines again, seem to have a high similarity. For the most important ingredients in Table 1, we have excluded the more common ingredients like salt, pepper, lime, lemon juice, water etc. At least one of the most used ingredients from earlier - onion and garlic - feature in the most important ingredients in all cuisines. Even from this limited set of ingredients, similar themes are seen that are in line with the clusters previously discussed. Indian and Moroccan cuisines have an emphasis on spices (cumin, cinnamon, turmeric, masala etc.), while Asian cuisines require soy sauce/fish sauce and garlic as an important ingredient. This corroborates the scatterplot using PCA as well. The fact that German cuisine is different even from other European cuisines is also seen here, since it lists sauerkraut and caraway seeds as important ingredients in German cooking which are very different from the important ingredients for other European recipes.

3 Data Preparation

One of the main problems we have with this dataset is Data Sparsity. The average number of ingredients per recipe is 16.34, with 89 ingredients appearing in just one recipe and 305 ingredients appearing in less than six recipes. These factors contribute to the high sparsity (98.34%) of the data matrix. Sparse features can lead to over-fitting and increases the time and space complexity of models

Cuisine	Ingr. 1	Ingr. 2	Ingr. 3	Ingr. 4	Ingr. 5
Chinese	soy_sauce	sesame_oil	garlic	ginger	cornstarch
English	onion	butter	potato	garlic	flour
French	garlic	butter	wine	thyme	onion
German	onion	sauerkraut	caraway_seed	flour	bacon
Greek	olive_oil	garlic	oregano	onion	feta_cheese
Indian	onion	garlic	ginger	masala	turmeric
Italian	garlic	parmesan_cheese	olive_oil	pasta	mozzarella
Japanese	soy_sauce	rice_wine	ginger	sesame_oil	garlic
Mexican	tortilla	onion	taco_seasoning	garlic	cumin
Moroccan	onion	garlic	olive_oil	cumin	cinnamon
Spanish	garlic	olive_oil	onion	tomato	saffron
Thai	fish_sauce	coconut_milk_or_cream	curry_paste	garlic	chicken

Table 1: Top 5 most important ingredients in each cuisine, by TFIDF score

[6]. When the dataset is sparse, Kuss [7] demonstrates that goodness-of-fit tests are unreliable. With sparse features, learning algorithms and diagnostic measures can behave in unexpected ways [8] since models fit the noise in the training data if there are too many features. The predictive power of models is affected as a result of this, as models are put into production, they are unable to generalize to newer data due to over-fitting. To overcome these issues we employ feature selection and dimensionality reduction techniques.

For feature selection, we use univariate tests (filter methods) like the chi-square test - a statistical test of independence to determine the dependency of two variables, and mutual information - a measure of the mutual dependence of two variables that quantifies the total amount of information gained about one random variable by observing another random variable [9]. We also use model-based feature selection (wrapper methods) like the Lasso (L1) regularized Logistic Regression model, which selects features based on the coefficient threshold and the regularization parameter. For dimensionality reduction we use PCA.

4 Methodology, Evaluation metrics and Experimentation Strategy

4.1 Cuisine Classification

One of the most essential aspects of our work is classifying cuisine based on the ingredients used in a recipe. Cuisine classification is an exciting problem with applications in recipe recommendation and generation. If we can automatically predict cuisine for a specific recipe given its ingredients, then the performance of recipe recommendation can be improved. Previous works examine the techniques for classifying cuisines based on the ingredients used in the recipe. These techniques include Support Vector Machines (SVM) [10, 1, 11, 3, 2], k-Nearest Neighbours (kNN) [2, 11], Logistic Regression (LR) [2, 1, 11], Gaussian Naive Bayes [3, 1, 11, 2], and Random Forest Classifier (RFC) [1, 2]. The datasets used in these works are larger and have different types of cuisines or ingredients.

Our analysis studies the performance of SVMs, Logistic Regression and Random Forests on different feature sets as discussed in section 3. We have not selected Gaussian Naive Bayes due to its independence assumption between various features i.e. ingredients as we believe the combinations of ingredients play an essential role in a recipe and cuisine prediction. We eliminated kNN as it didn't perform well compared to the other methods we choose, and also was computationally expensive. Moreover, the kNN is not suitable here as our dataset size is comparatively small and highly dimensional thus it has a possibility to suffer from the curse of dimensionality [12].

We assess three models for cuisine classification: SVM, which attempts to identify support vectors to locate a hyperplane separating different classes in the n-dimensional space; Random Forest, an ensemble method that uses bagging and random variable selection to grow a collection of uncorrelated trees that each predict a class which is then voted on by the collective, and Logistic Regression, where the logistic function is used to model class probabilities in a binary case (multinomial cases are considered as one vs. others for each class). Further details regarding these methods can be found in the Appendix. The model hyperparameter tuning was done using GridSearch in Scikit Learn [13] with 5-fold cross validation on the training set.

To assess the performance of our machine learning models, we used accuracy, precision, recall, and F1-score. F1score is the harmonic mean of recall and precision [14]. As far as we know, related works have not attempted to address the 'explainability' of their implemented models i.e. predictions of the models, or the reasons behinds these predictions. While their models have been very accurate in classifying cuisines, they do not account for how and why a certain cuisine is predicted for a given set of ingredients. To explain the reasons behind a particular prediction, our analysis uses state-of-the-art machine learning explainability frameworks known as LIME (Locally Interpretable Model agnostic Explanations)[5]. LIME is a model-agnostic post-hoc explanation technique aimed at approximating any black-box machine learning model with a local, interpretable framework for explaining the reasons behind each individual prediction [5]. A hold-out technique is used to estimate generalization capabilities of the models on unseen datasets. For training, we have used 80% of the data and 20% for the model testing.

4.2 Prediction of Ingredients

A collaborative filtering algorithm in a more general sense seeks to mine for quantitative connections between items which are indirectly provided by user activity. In a more usual context, users could include members of a subscription service such as Netflix. An assumption of collaborative filtering is that users with similar opinions will likely share a preference for certain items. This suggests that a quantitative connection could be formed between the items. The second stage of collaborative filtering seeks to exploit this information to recommend (or predict) a new item to a specific user. A new suggestion is strong if its similar items also exist among the items that the user has recently liked; this employs a 'filtering' mechanism which considers the number of coinciding items for each suggestion. Equation 1 can be found in Sarwar et al.[15] and Hu et al. [16].

$$P_{u,i} = \frac{\sum_{all\ similar\ items,N}(s_{i,N} \cdot R_{u,N})}{\sum_{all\ similar\ items,N}(|s_{i,N}|)} \tag{1}$$

Our data is a grid of 4236 recipes and 705 one-hot encoded ingredients; we use the equation above as usual but, in this context, the users are the recipes and the items are ingredients. The only attribute among recipes is the class of cuisines they belong to; the underlying assumption is that any similarity measurement between ingredients is down to whether or not they are frequently used within a cuisine. The suitability of a certain ingredient i to a particular recipe u is given by $p_{u,i}$. $S_{i,N}$ are the similarity measures between ingredient i and ingredient N which is from the set of the most similar ingredients to the suggestion i. We dot product the recipe vector and similarity measure, and sum by the total similarity scores. For this task, some ingredients such as salt and water were deliberately removed since they are too ubiquitous and provide no further information.

Our experiments utilise cosine and Jaccard similarities to retrieve a similarity score between the ingredients. We consider the approach in [15] which defines similarity as how frequently (but also ideally how much) the two ingredients coincide between all recipes. Simply put cosine similarity, a measure of similarity between two vectors in Euclidean space, is therefore calculated by finding the dot product between two vectors down the list of recipes and divided by their magnitudes multiplied. If the vectors are exact then the cosine similarity is 1.0. We also implement Jaccard similarity. Jaccard similarity is the ratio between the number of all recipes that two ingredients are both present in and the union between the recipes containing the ingredients. (See 8.4 and 8.5 in appendix)

Item-based collaborative filtering is usually implemented as a recommender system; the recommendations could be confirmed by users themselves if they are good suggestions, however, it is more practical to consider retrieval of lost information. Hence, we centre our evaluation on the model's ability to recall. For each recipe in our dataset, we simply remove one ingredient from the table at random and make a number of recommendations for that recipe. If the missing ingredient is located in the list of recommendations, the rank of the ingredient is used and converted to a percentage. We use at least 3 methods to evaluate the performance of the recommender. The first method simply evaluates the rank of the missing ingredient as a raw percentile. L recommended items are ranked in descending order (with a sequence L, L-1, L-2...L-M); an ingredient at the top of the suggestion indicates a score of 100% and 0% if the missing ingredient is not in the list of recommendations. The score is a ratio between the rank and the number of suggestions made, the final evaluation is the mean score of the raw percentiles (RP) across all recipes.

$$\begin{gathered} \overline{RP} = mean(rank_{u,N*}) \\ rank_{u,N*} = \frac{RawRank_{u,N*}}{L} \end{gathered} \tag{2}$$

The N* denotes the ranking specifically of the missing ingredient should it be in the set of recommended ingredients. L is the number of recommendations for recipe u and ingredient i. We also use a weighted rank following a very similar process (though higher values show better performance in our evaluations) to Hu et al [16], equation 3. We will refer to this as weighted rank as it considers the similarity scores in the evaluation.

$$\overline{WR} = \frac{\sum_{u,N} r_{u,N} \, rank_{u,N}}{\sum_{u,N} r_{u,N}} \tag{3}$$

Another evaluation to consider is if the missing ingredient is at least in the top 5 and top 10 recommendations. This is simply evaluated by counting the occurrences in the top 5 and top 10 sets, and then dividing by the total number of recipes in the dataset. Concerning the datasets, the data of 4236 recipes and 705 ingredients is shuffled and split into training, validation and test sets of 60%, 10% and 30% respectively. We consider that 'training' in this context is simply calculating the similarity measurements between ingredients from the training set. The validation set is only 10% with 423 recipes, however, a hyper parameter search over K (K nearest neighbours for each ingredient in the recommender system) was exhaustively conducted; a small set of recipes is then convenient. The test set was untouched, set aside, and only run through the algorithm once for each similarity measure, cosine and Jaccard. The sequence of K values evaluated were [20,40,60,80,100,150,200,250,300,350,400].

5 Results and Discussion

5.1 Classification of Cuisines

In this section, we discuss the results obtained for cuisine classification tasks with different feature sets using LR, SVMs and RFCs. For the experimentation we have used 5 different feature sets as mentioned in section 3- a)consisting all 709 features, b)Features selected by chi-squared statistic, here we have taken 350 features (i.e. $\sim$ 50 percentile as shown in Fig. 2) c) Features selected by mutual information statistic, we have taken 350 features (i.e. $\sim$ 50 percentile as shown in Fig. 2) d) Lasso (L1) regularization-based feature selection with Logistic Regression model selected 353 features, and e) features extracted by PCA with 90% variance as shown in Fig. 2, here we have 350 extracted features. From Table 2, we can learn that LR model is performing better than SVMs and RFC in general across most of the feature sets on all the evaluation metrics. LR is able to generalize well on the test set as compared to SVMs and RFC.

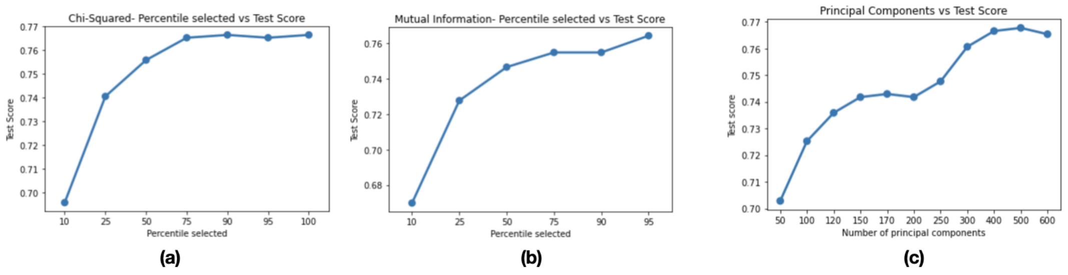


Figure 2: Feature Selection Techniques comparison

When we compare the performance across various feature sets, the feature sets based on L1-based regularisation and PCA are performing similar to the feature set with all 709 features, whereas the feature sets based on univariate statistics chi-square and mutual information are slightly under performing. In between chi-squared and mutual information statistics, chi-squared is performing better on all the evaluation metrics. When features are interdependent and not mutually exclusive, univariate feature selection techniques don't always result in the best model performance. The Lasso based feature selection approach is performing better, since shrinking (where it penalizes the coefficients of the regression variables) and removing coefficients can minimise variance without significantly increasing bias [17]. This is particularly helpful when you have a limited number of observations and a large number of features. One advantage of performing feature selection is that it is able to control overfitting of the models and generalize well, when a model is trained on all the features the difference between training and test accuracy is higher as compared to when it is trained only on reduced feature sets. For further analysis on cuisine-wise performance and explainability we consider LR model with Lasso Regularization feature set as it is performing better and is able

Model and Feature Set	Accuracy	Precision	Recall	F1-Score
Logistic Regression				
All Features	76.65	77.34	76.65	0.768
Chi-Squared	75.59	76.44	75.58	0.758
Mutual Information	74.65	75.54	74.64	0.749
L1-Regularization	76.65	77.58	76.65	0.769
PCA	76.89	77.60	76.88	0.771
Support Vector Machine (SVM)				
All Features	76.06	77.23	76.06	0.764
Chi-Squared	75.24	76.50	75.23	0.756
Mutual Information	74.76	75.90	74.76	0.751
L1-Regularization	75.83	77.13	75.82	0.762
PCA	75.71	77.03	75.70	0.761
Random Forest Classifier (RFC)				
All Features	72.88	73.76	72.87	0.727
Chi-Squared	71.70	72.56	71.69	0.715
Mutual Information	70.87	71.75	70.87	0.707
L1-Regularization	72.17	73.05	72.16	0.720
PCA	71.58	73.06	71.58	0.717

Table 2: Comparative Analysis of various Machine learning models on different feature sets

to generalize well. We are not considering PCA as it doesn't use the original features and the new features generated are not interpretable by LIME algorithm.

From confusion matrix in Fig. 3 we can observe that there is significant overlap between some cuisines. English and French cuisines have the lowest F1-scores with 0.61 and 0.54 respectively, and the model seems to be getting confused when it comes to these recipes since it has a high misclassification of English recipes as French/German/Spanish. The same can be said of multiple pairs in the matrix. Greek, Moroccan, Indian and Mexican recipes seem to be classified with much greater confidence. The misclassification pairs seen here are in line with the structure revealed in our EDA, and so is not surprising. Given how these cuisines mixed over a long period of time, exact classification would ironically be a surprise, and a sign that we might be overfitting.

In terms of explainability using LIME, the results are as expected. For the sake of brevity we restrict ourselves to two random examples. Their predictions from LIME are shown in Fig. 4, and the corresponding instance explanation tables can be found in the Appendix . These are taken directly from the LIME explanation of the prediction of these two instances for the Logistic Regression model. The first is a highly confident prediction of the recipe belonging to Indian cuisine. This is again, not surprising given the ingredients used in the recipe. Interestingly, the most important ingredients for Indian cuisine from our EDA have a higher value in the 'effect' column, which is a measure of how much the predicted probability would have changed if the corresponding variable was absent. In our other example, we see that the classifier is not as confident with the classification of English, and considers the recipe to possibly be from French cuisine as well. The reasons again, are clear from the ingredients - butter, thyme, egg etc. score high in the list for both cuisines. Perhaps the deciding factor was the use of puff-pastry in the recipe, which would have had an impact on the calculation of the 'effect' field. Our EDA and the investigation of important ingredients for each cuisine give a more nuanced understanding of the dataset, which ultimately helps in explaining the classification of an instance when considered together with a tool like LIME.

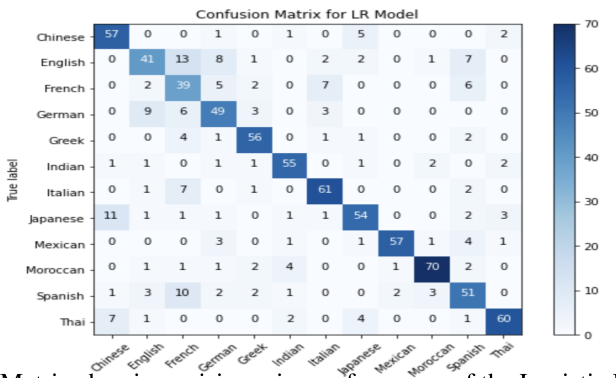
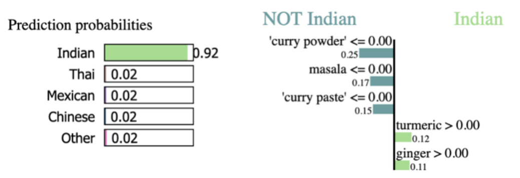
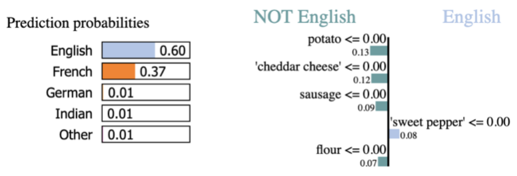


Figure 3: Confusion Matrix showing cuisine-wise performance of the Logistic Regression model on the test set



(a)



(b)

Figure 4: Explainability using LIME for two random samples

Evaluation Type	Raw Percentile	Weighted Percentile	Top 10	Top 5
Cosine (Recall)	39%	59%	65%	38%
Jaccard (Recall)	40%	59%	64%	39%

Table 3: Evaluation of recall on test set

5.2 Prediction of Ingredients

Firstly we find from the validation set the maximising K value to be around 250 for both cosine and Jaccard measures. This seems to be very high, for example, we find in Sarwar et al. [15] that the mean absolute error stops significantly increasing around 30 nearest neighbours of which they choose as their optimal value. However, Sarwar et al. split data 80:20 training and test respectively and it seems they have tuned on a larger dataset; it is possible that smaller data sets such as our validation, carry a higher proportion of noise. This might then require stronger measures of regularisation or adjustment. We present our K search plot below (figure 5). In general, performance clearly increases as the number of neighbours increase. The shape of the plot is expected; at some range of values the performance will stop increasing. However, this is not the case when observing the weighted percentile, which appears to be unaffected by the number of neighbours. This result is strange since both the weighted rank and raw percentiles seek to evaluate an average recommendation across all recipes but have different behaviour.

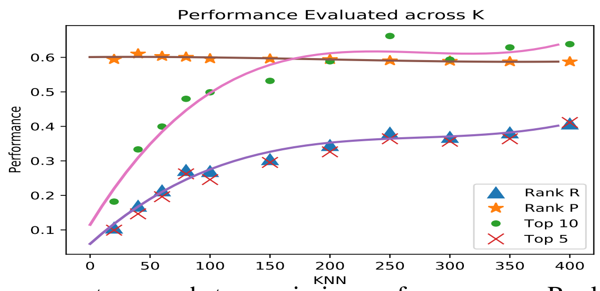


Figure 5: K hyper-parameter search to maximise performance on Rank R (raw, eqn.2), Rank P(weighted, eqn.3), top 10 and top 5 sets.

Performance on the test set appears to be relatively good. Final recall scores [16] achieved by Hu et al. are between 30% and 40%. Our final recall scores (table 3) for finding a missing ingredient in top 10 recommendations are 65% and 64% for Cosine and Jaccard respectively. However, even though Hu et al. use a larger set of binarised data they do consider a different challenge. Additionally, there is no stark difference between using cosine or Jaccard similarities.

6 Conclusion

We demonstrate how a thorough EDA is helpful in understanding the structure of a dataset as well as how standard methods for visualising high-dimensional datasets in lower dimensions give similar, but nuanced results. Furthermore, we show how results from such an EDA bolster the understanding of model explainability gained from tools like LIME. To improve predictions from collaborative filtering models, we suggest quantifying the amount of ingredients used in a recipe instead of binarising the data.

7 Contributions

1. Aditya Suryanarayan (s2089883) - Performed EDA (MDS - exploring Cosine, Jaccard and Euclidean distances, PCA, TF-IDF, fine-tuning the t-SNE plot by testing different perplexity values to get the best plot), synthesized model explanations using LIME and linked them to the EDA performed, general report editing and polishing.
2. Akash Bhagran (s2123972) - Built recipe generation model, sections 5.2 and 4.2, and general report writing, editing and polishing.
3. Rohan Gorantla (s2112695) - Worked on Data Preparation, Cuisine Classification and LIME experimentation's and reporting writing for the same sections (i.e 3,4.1, 5.1, & 8.7).
4. Tamara Lottering (s2124912) - Applied for project submission extension, carried out domain research, performed initial EDA and general report writing, editing and polishing.

References

[1] Tript Sharma, Utkarsh Upadhyay, and Ganesh Bagler. Classification of cuisines from sequentially structured recipes. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 105–108. IEEE, 2020.

[2] 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.

[3] Slobodan Kalajdziski, G Radevski, Ilinka Ivanoska, Kire Trivodaliev, and B Risteska Stojkoska. Cuisine classification using recipe's ingredients. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1074–1079. IEEE, 2018.

[4] Jill Freyne and Shlomo Berkovsky. Intelligent food planning: personalized recipe recommendation. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 321–324, 2010.

[5] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[6] Miha Grčar, Dunja Mladenič, Blaž Fortuna, and Marko Grobelnik. Data sparsity issues in the collaborative filtering framework. In *International workshop on knowledge discovery on the web*, pages 58–76. Springer, 2005.

[7] Oliver Kuss. Global goodness-of-fit tests in logistic regression with sparse data. *Statistics in medicine*, 21(24):3789–3801, 2002.

[8] Osiris Villacampa. Feature selection and classification methods for decision making: a comparative analysis. 2015.

[9] David Freedman, Robert Pisani, Roger Purves, and Ani Adhikari. Statistics, 2007.

[10] Han Su, Ting-Wei Lin, Cheng-Te Li, Man-Kwan Shan, and Janet Chang. Automatic recipe cuisine classification by ingredients. In *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: adjunct publication*, pages 565–570, 2014.

[11] Boqi Li and Mingyu Wang. Cuisine classification from ingredients.

[12] Nikolaos Kouiroukidis and Georgios Evangelidis. The effects of dimensionality curse in high dimensional knn search. In *2011 15th Panhellenic Conference on Informatics*, pages 41–45. IEEE, 2011.

[13] Oliver Kramer. Scikit-learn. In *Machine learning for evolution strategies*, pages 45–53. Springer, 2016.

[14] Yutaka Sasaki et al. The truth of the f-measure. 2007, 2007.

[15] Joseph Konstan Badrul Sarwar, George Karypis and John Riedl. Item-based collaborative filtering recommendation algorithms. *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285 – 295, 2001.

[16] Yifan Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[17] Valeria Fonti and Eduard Belitser. Feature selection using lasso. *VU Amsterdam Research Paper in Business Analytics*, 30:1–25, 2017.

[18] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[19] Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. Feature selection for svms. 2000.

[20] Wun-Hwa Chen, Sheng-Hsun Hsu, and Hwang-Pin Shen. Application of svm and ann for intrusion detection. *Computers & Operations Research*, 32(10):2617–2634, 2005.

[21] Yanchang Zhao and Yonghua Cen. *Data mining applications with R*. Academic Press, 2013.

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

[23] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

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

[25] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[26] Ramón Díaz-Uriarte and Sara Alvarez De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):1–13, 2006.

8 Appendix

8.1 Principal Component Analysis

Principal Component Analysis (PCA) aims to reduce the dimensions of the data by identifying directions in the original space where the data has higher variance. This done so that all directions found are orthogonal to each other. This orthogonality ensures that the variance captured in each new direction is due to "new" information that is not captured in earlier direction. This can also be used for exploratory analysis, where one looks at the spread of points across different directions (or principal components), and tries to identify what themes are captured by each component. This helps in getting a better understanding of the data, and what are the sources of variation between different clusters or labels. For our EDA, we choose the first 2 principal components, and generate a scatter plot of our dataset on these 2 dimensions.

8.2 Multidimensional Scaling (MDS)

Multidimensional Scaling (MDS), is a visualization technique that helps in identifying underlying structure in the data. It attempts to create a lower dimensional embedding such that the distance between points is still respected, IE: it tries to keep distances between points the same when it projects them onto the lower dimensional space. To make plotting easier, data is usually projected to two dimensions, as we have done here. MDS comes in two flavours:

1. Metric - where it uses the exact distances between points
2. Non Metric - where it uses a rank correlation between points instead of the exact distances

We use the metric flavour in our case. We consider each recipe as a vector in the 709-dimensional ingredient space, and calculate the cosine distance as the cosine between any two vectors. We use Cosine distance between recipes. This distance calculation is done in a pairwise manner for all recipes, and the resulting pairwise-distance matrix is fed into the MDS function from the scikit-learn library in Python. The resultant 2D matrix is then used to create the scatter plot in Figure 1.

8.3 t-SNE

t-distributed Stochastic Neighbour Embedding (t-SNE) is an extremely flexible method of creating a lower dimensional embedding of a dataset for the purpose of visualization. Where MDS creates a lower dimensional embedding of a dataset while maintaining the distances between points from the original space, t-SNE creates an embedding but with a focus on maintaining clusters of points. MDS enables us to find clusters since it focuses on keeping distances in the new space as close as possible to that in the original space, and because we can easily use scatterplots in the two new dimensions to find structure. t-SNE on the other hand, has a specific focus on maintaining clusters in the new space and the original space. This has a trade-off however, one of those being that t-SNE does not tell us anything about how far away clusters are, only about their relative positions in the original space and the new lower dimensional space, and helps visualize relative structure in the data. Also, t-SNE is stochastic, not deterministic, and as such requires more parameter tuning and careful examination of the resulting plots to achieve a usable embedding.

8.4 Cosine Similarity

$$cos(\theta) = \frac{R(i) \cdot R(N)}{|R(i)||R(N)|} \tag{4}$$

8.5 Jaccard Similarity

$$J(i, N) = \frac{R(i) \cap R(N)}{R(i) \cup R(N)} \tag{5}$$

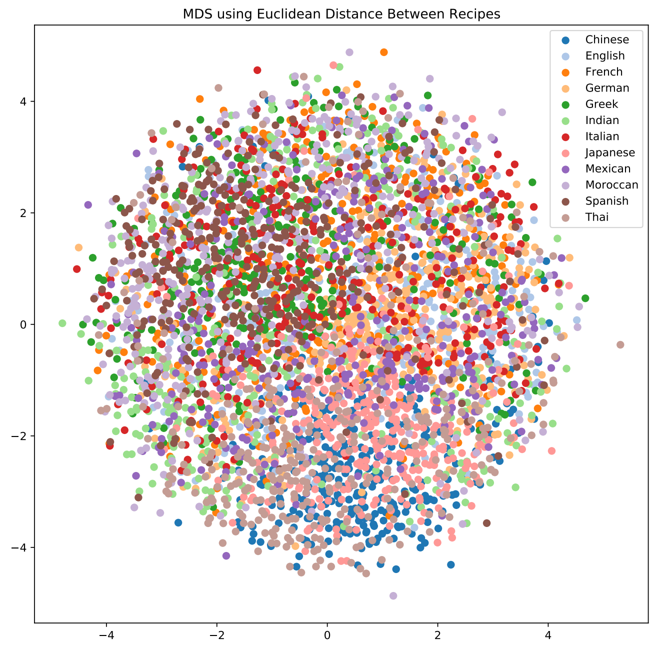


Figure 6: Cuisine-wise MDS plots using Euclidean Distance. Note the lack of structure that was seen in the Cosine MDS plots.

8.6 Information Retrieval Methods - TF IDF

At its base, TF IDF is also a method of embedding data - it generates a vector that can be used to describe a document. In order to generate this vector it looks combines two metrics related to the terms in a document - term frequency (TF) and inverse document frequency (IDF). TF is simply how often a term appears in a document, while IDF is a measure of how many documents this term appears in. Taken together, TF IDF tells us how important (or in other words, how much weightage) to give to a particular term in a document. Since we can consider the score to be a weight for each term, and each term can be considered a dimension, we can convert a document containing terms to a document vector where each dimension represents a term, and the weightage given to each direction is the TF-IDF score of the corresponding term *for this document*. This distinction is important, since a term would have different TF and IDF scores for each document.

In our case, we consider each cuisine as a document, and each ingredient in each recipe from that cuisine as a term in the document. Since we have 709 unique ingredients across all recipes, we thus convert each cuisine into a 709-dimensional vector. A simple pairwise cosine distance of each of these (12) 709-dimensional cuisine vectors gives us how close each of our cuisines are with respect to each other. The weights of each dimension in the cuisine vector gives us the importance of that dimension, and dimensions (ingredients) with the highest weights are the most important to a cuisine.

8.7 Cuisine Classification Models

Support Vector Machines (SVMs): SVMs are a popularly used classification technique, originated as a result of Vapnik's work on structural risk minimization principle, which minimizes the generalization error, i.e. true error on unseen instances [18]. The basic SVM is used to solve two-class problems where the data is separated by a hyperplane identified by a set of support vectors [19]. To divide different classes, SVM creates a hyperplane in multidimensional space. SVM iteratively produces the best hyperplane, which is then used to minimize an error. The main aim of SVM is to find a maximal

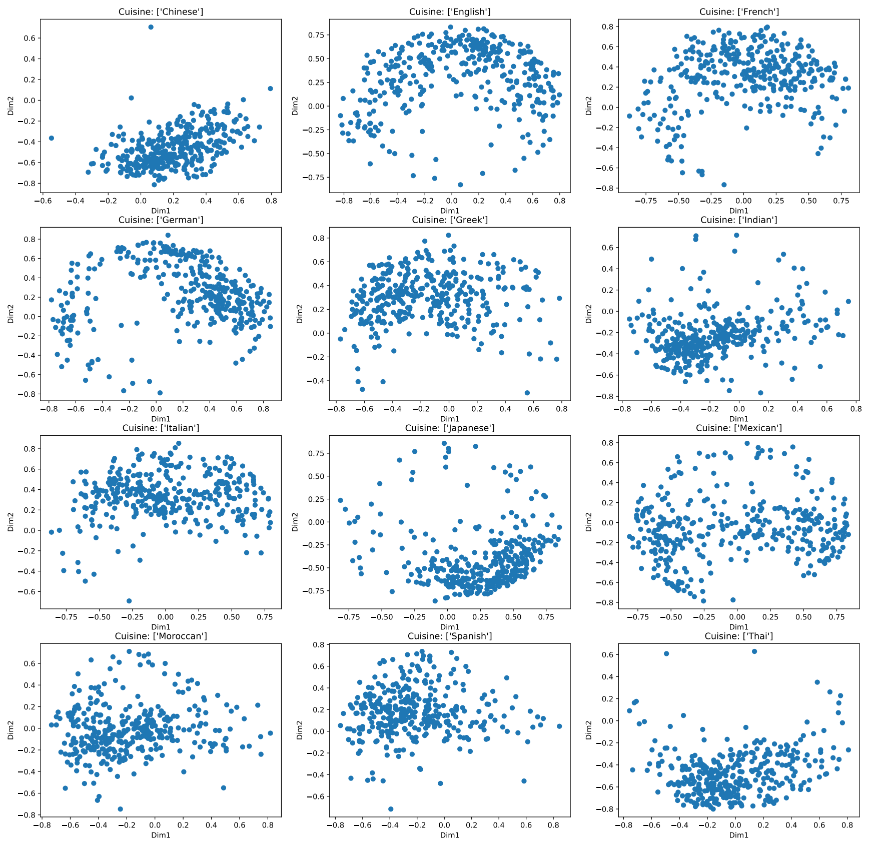


Figure 7: Cuisine-wise MDS plots using Cosine Distance. Note the similar distributions for all European recipes and for all Asian recipes.

marginal hyperplane that divides a dataset into classes as precisely as possible [20]. One of the key properties of SVM is its capability to learn independent of the dimensionality of the feature space, therefore it can generalize well in the presence of a dataset with high dimensionality. The use of SVMs necessitates the transformation of a multi-class classification problem to a binary classification problem. The One-vs-Rest strategy was used in our case.

Logistic Regression (LR): Logistic regression is a widely used statistical modeling technique, and it is probabilistic in nature. It is mainly designed for binary classification tasks and uses logistic function to model a binary dependent variable. Here we use a multinomial version that is capable of handling multiple classes. Multinomial Logistic regression technique is a generalized linear model used to estimate the probabilities for the n classes of a qualitative dependent variable Y, using a set of explanatory variables X [21]

$$Pr\left(Y_{ik}\right) = Pr\left(Y_i = k \mid x_i; \beta_1, \beta_2, \ldots, \beta_n\right) = \frac{\exp\left(\beta_{0k} + x_i\beta_k'\right)}{\sum_{j=1}^{\mathrm{m}} \exp\left(\beta_{0j} + x_i\beta_j'\right)}; k = 1, 2, \ldots, n \tag{6}$$

where β_k is the row vector of regression coefficients of X for the k^{th} category of Y.

Random Forest Classifier (RFC): Leo Breiman [22] developed RFC which employes an ensemble of classification trees where each classification tree is constructed from a bootstrap sample of the data, and the candidate set of variables at each split is a random subset of the variables [23]. For tree construction, RFC utilizes both bagging (bootstrap aggregation), a popular method for mixing unstable learners, and random variable selection [24, 25]. To achieve low-bias trees, each tree is unpruned, at the same time, bagging and random variable selection result in low correlation of the individual trees. The algorithm produces an ensemble with low bias and variance [26].

LIME is a model-agnostic post-hoc explanation technique aimed at approximating any black-box machine learning model with a local, interpretable framework for explaining the reasons behind each individual prediction [5]. LIME works locally, which ensures it is observation specific and can have reasons for each unique observation it encounters. In terms of technique, LIME attempts to match a local model using sample data points that are close to the instance being explained. The following are the explanations made by LIME for each observation x [5]:

$$\xi(x) = argmin_{f \in \mathcal{F}} L\left(g, f, \pi_x\right) + \Phi(f) \tag{7}$$

where $\mathcal{F}$ is the class of potentially interpretable models, $f \in \mathcal{F}$ is explanation considered as a model, $g : \mathbb{R}^d \rightarrow \mathbb{R}$ is the classification model that is being explained, $\Phi(f)$ is the measure of complexity of the explanation $f \in \mathcal{F}$ and $\pi_x(z)$ is the proximity measure of an instance z from x.

8.8 LIME Variable Effects

Ingredient	Effect
puff pastry	0.1262
assorted vegetables	0.0795
mustard prepared	0.0709
thyme	0.0608
eggplant	0.0598
buttermilk	0.0385
turkey	0.0346
creme fraiche	0.0253
butter	0.0103
cranberry	-0.0049
stuffing	-0.0119
cornstarch	-0.0171
egg	-0.0255
parmesan cheese	-0.0572

Table 4: LIME explanation of a random instance of an English recipe

Ingredient	Effect
turmeric	0.1186
ginger	0.1077
mustard seed and powder	0.0582
cumin	0.0419
coriander	0.0237
vinegar	0.0125
chili powder	0.0089
gingersnaps	0.0005
pepperoni	-0.0220

Table 5: LIME explanation of a random instance of an Indian recipe