TastifyNet: Leveraging adversarial examples for generating improved recipes

Abigail Russo
Stanford University
aarusso@stanford.edu

Brynne Hurst
Stanford University
brynnemh@stanford.edu

Trey Weber
Stanford University
tpweber@stanford.edu

Abstract

In the following work, we present TastifyNet: A deep learning-based recipe improvement algorithm. In contrast to existing work which focuses on identifying recipes or generating entirely new ones - we sought to improve existing culinary dishes. After training a neural network model on over 250,000 rated recipes, an adversarial example inspired alteration algorithm recommends improvements.

1 Introduction

As deep learning algorithms become more ubiquitous, practitioners are increasingly recognizing how interaction with such algorithms can improve our own creativity and performance. We wondered whether an algorithm might inspire in another domain: cooking. There are many examples in the literature of deep learning algorithms that can be trained on food/recipe data for various tasks such as ingredient identification, image to recipe translation (1) and recipe generation (2) (3). However, one often may wish to improve an already existing recipe based on some metric, rather than identify recipes or translate between information types. Here, we present TastifyNet, a recipe improvement algorithm comprising of two main components. The first, is a regression-based neural network that maps an ingredient list to a recipe rating (based on user reviews). The next is a recipe alteration algorithm, which seeks to improve an ingredient list by replacing items with functionally similar, but higher rated ingredients.

2 Prior Work

There are many examples in the literature of deep learning algorithms that can be trained on food/recipe data for various tasks. Salvador et. al (1) demonstrate the ability to learn a cross-model embedding of recipes. Compiling a dataset of over 1 million recipes and 800k images, they create automated tools for food and recipe applications, using a joint embedding of recipes and images for image-recipe retrieval. There is quite an extensive list of image-based recipe generators that apply similar techniques (4) (5) (6). TastifyNet uses a different approach: rather than identifying a recipe from its image, we seek to improve an already known list of ingredients based on a mapping between ingredients and rating.

For data pre-processing for our learning model, Pennington et. al (7) show techniques that would allow us to learn vector representations of ingredient lists (e.g., "1 stick butter" to "butter") using Global Vectors for Word Representation (GLoVe). Word embeddings have proven necessary and successful for a variety of applications and performance metrics. For a large vocabulary such as all commonly used cooking ingredients, they allow for a more compact representation of words than alternatives such as one-hot encoding.

To our knowledge, no previous work exists that uses a deep learning model to improve recipe ingredients based on ratings. However, similar applications exist that have proven the ability to generate recipes, although with

CS230: Deep Learning, Winter 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

different techniques than ours (2) (3). Bień et. al used a Transformer with GPT-2 architecture that allowed users to input a list of desired ingredients, and receive a fully generated recipe with instructions, portions, etc. Willis et. al used word2vec embedding of ingredients and an LSTM RNN to then generate new recipes based on what ingredients are available to the user. Both of these works address food shortage issues, where a user has a pre-defined ingredient list from which to cook from. TastifyNet seeks to provide a more nuanced solution to the user, to generate not just a new but also improved recipe.

3 Dataset and Features

3.1 Dataset Sourcing and Statistics

Our web-scraped AllRecipes dataset contains over 60,000 recipes and includes ingredient lists, directions, ratings, and rating count. There is a strong tendency toward a rating of approximately 4.5 (Appendix Figure 3a) and as one would expect, recipes with a fewer number of ratings occur more frequently (Appendix Figure 3b). To account for this, we weight recipes based of the number of ratings received.

We supplemented the Allrecipes dataset with scraped data from Food.com (8), compiling over 700,000 unique user reviews into a dataset of 230,000 recipes. Although the Food dataset has a very large number of five star ratings it also provides significantly more lower rated recipes for a more uniform distribution (Appendix Figure 3a).

3.2 Dataset Processing

To transform our datasets into a format ready for learning relationships between ingredients and ratings, we first needed to parse the ingredients into structured data. For example, if one ingredient for a given recipe is "2 cloves of garlic, finely chopped", we want to separate it into "QTY: 2, UNIT: clove, NAME: garlic, COMMENTS: finely chopped". This can be framed as a structured prediction problem, a sub-field of NLP that involves predicting structured objects based on probability distributions. One way to do this is with Conditional Random Fields (CRF), a statistical modeling technique that works particularly well for pattern recognition. As developing a CRF for this task could itself encompass an entire project, we looked to adapt Ingredient Phrase Tagger (IPT) (9) to our needs, a CRF that was created precisely to parse ingredient phrases into its quantity, unit, name, comment, etc.

A basic explanation of IPT is as follows. Given a set of ingredient phrases and labels:

$$x = \{x^1, x^2, ..., x^n\},\ y = \{y^1, y^2, ..., y^n\}$$

For Example:

$$x^i = [\text{"2", "cloves", "of", "garlic", "finely", "chopped"}]$$

$$y^i = [\text{QTY, UNIT, UNIT, NAME, COMMENT, COMMENT}]$$

training IPT on a large, labeled data set allows it to infer:

$$P(x|y) = P(\text{tag sequence} \mid \text{ingredient phrase})$$

which then allows us to extract the individual ingredient names from each recipe's set of ingredient phrases, based on the highest probability tag sequence.

After training IPT on a provided set of 170,000 labeled recipes (without ratings), we ran it on our Allrecipes dataset to extract each ingredient. The Food dataset was already parsed into its individual ingredients. We then wrote a filtering script in Python to filter and modify any ingredients that our word embedding could not handle directly (e.g., special characters, accents, synonyms, etc). In addition, we removed whitespace and special characters, and then replaced spaces in multi-word ingredients with underscores (e.g., "olive oil" was replaced with "olive_oil"). Finally, we split our dataset into train/dev/test sets with a 80%:10%:10% ratio.

4 Methods

4.1 Ingredient Embeddings

To generate ingredient embeddings to use as inputs to our model, we started by assigning each unique ingredient a numeric identifier j. We then initialized the ingredient embeddings based off of the existing

100-dimensional GloVe embeddings (7). For ingredients made up of only one word (e.g., "butter"), the initial ingredient embedding was the same as the GloVe embedding for that word. For multi-word ingredients (e.g., "olive_oil"), we used the average of the GloVe embeddings for each individual word as the initial embedding. We then used the `Mittens` library (10) to finetune the existing GloVe embeddings, using the raw, unparsed ingredient lists. These embeddings were saved in a matrix E such that the jth column of E was the embedding for the jth ingredient. The input to our model was then constructed as follows.

First, we identified the maximum number of ingredients in a recipe to be 43 and padded recipes with fewer than 43 to create inputs of equal length. Then, given a training example (i.e., a list of ingredient identifiers),

$$x^{(i)} = [j_1, j_2, \cdots, j_{43}],$$

we looked up the corresponding embeddings in E to compute the embedding matrix

$$E^{(i)} = \begin{bmatrix} E_{j_1} & E_{j_2} & \cdots & E_{j_{43}} \end{bmatrix}$$

for each training example. Finally, we flattened $E^{(i)}$ by concatenating the individual columns to produce a 4300×1 dimension feature vector

$$e^{(i)} = [E_{j_1}; E_{j_2}; \cdots; E_{j_{43}}].$$

4.2 Rating Model

We used `Keras` (11) to build a three layer feed-forward neural network with a mean-squared error loss to predict recipe ratings from ingredient lists (as can be seen in Figure 1). To prevent over-fitting, we applied a L2 regularization penalty to our loss function. In addition, to account for the imbalance in number of ratings, we weighted each example by the number of ratings when calculating the loss.

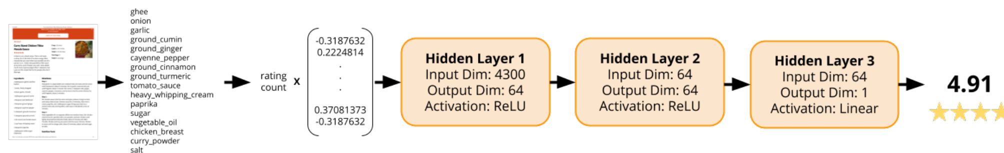


Figure 1: Predictor Model Diagram. The predictor model takes as input a flattened embedding vector, and sends that through three fully-connected feed-forward layers, the last layer being the output layer.

4.3 Recipe Alteration

Next, we developed a method for altering ingredient lists to improve recipe ratings. Inspired by methods to generate adversarial examples (12), we altered the embedding of a recipe, $e^{(i)}$, iteratively as:

$$\eta = \nabla_{e^{(i)}} J(e^{(i)}, \hat{y})$$
$$e^{(i)} := e^{(i)} + \epsilon\eta$$

Where the cost function, J, was simply the rating model's prediction, $\hat{y}$, on the embedding, $e^{(i)}$. That is to say, we iteratively alter the recipe embedding to maximize the model's prediction. ϵ was chosen to balance computational efficiency with appropriately sized changes to the embedding. Updates continued until a stopping condition was reached and an altered recipe, $\tilde{e}^{(i)}$, was generated. A minimum stopping condition was that the ingredient list had changed. That is, for each ingredient after each iteration, we asked whether the altered embedding was more similar to the embedding of a different ingredient. If so, the minimum stopping condition had been reached.

4.4 Evaluation

Given this formulation, it would be circular to evaluate performance by considering how much the rating model's prediction of $e^{(i)}$ and $\tilde{e}^{(i)}$ differ. We attempted several methods for creating an evaluator model that out-performed our rating model. Ultimately, due to time and computational constraints, we settled on using an earlier version of a rating model with different hyperparameter settings (namely, dropout regularization,

a trainable embedding layer, 200 dimensional embeddings) that was trained on GloVe embeddings without fine-tuning and data-cleaning. This was made possible because the ingredients after data-cleaning were a subset of those used to train the earlier version. Thus, even though the two models used different datasets and embeddings, the appropriate input could be 'translated' to the evaluator model using the ingredient to embedding mappings. We can then compare the evaluator's predicted rating for embeddings $e^{(i)}$ and $\tilde{e}^{(i)}$ for a reasonable heuristic of performance.

5 Experiments

5.1 Rating Model Hyperparameter Search

Early trials of our predictor model indicated that we were over-fitting to the training set (see Appendix Figure 4). In an attempt to find the optimal L2 regularization weight λ, we implemented a hyperparameter search using the `RayTune` library (13). We quickly realized that we needed a lower learning-rate to λ ratio in order to prevent over-fitting. However, setting the hyperparameters this way also increased our loss. In an attempt to solve this new issue, we performed a search over different network configurations (e.g., number of hidden layers, hidden layer dimension, etc.) and found little improvement. We also attempted to implement a Gradient-Boosted Decision Tree (GBDT) network. However, we experienced issues converting our dataset into a format compatible with `Keras's` GBDT algorithm, and we were unable to implement a successful GBDT model within the timelines of this project.

In the end, we opted to use the non-over-fitting model as input to the recipe generator because we found that over-fitting was particularly problematic for this application. During the recipe alteration stage, models with high variance would attribute hyperbolic changes in ratings to minute changes in a recipe (e.g., 'onion' $\rightarrow$ 'onions' resulted in a 10 point increase on a scale from 0-5). For these reasons, we opted to choose hyperparameter settings that would prioritize low variance. Specifically, we set the learning rate to 3.719201103501549e-05 and λ to 0.0240873486936871370.

5.2 Recipe Alteration

We also experimented with several hyperparameters corresponding to recipe alteration. In particular: the metric for measuring embedding similarity, constraints on which ingredients could alter, and the stopping condition. First, we compared mean squared error and cosine measures of embedding similarity. We found that rating improvements were similar (Appendix Figure 7a-7b) yet we favored the cosine similarity with which ingredient swaps tended to be more diverse (Appendix Figure 6).

A more consequential hyperparameter we considered was the constraints on which elements of the embedding could be altered (implemented by constraining certain elements of η to zero). First, we considered whether the padding elements of the embedding should be remain unaltered or whether the algorithm should be allowed to add an ingredient to the recipe. In the latter case, evaluator ratings didn't generally improve (Appendix Figure 7c) and added ingredients tended to be unusual (e.g., 'saunf', 'rajma', 'manwich', 'tarama'). Next, we allowed at most one ingredient to change. We determined which ingredient had the largest gradient after the first iteration and zeroed out all other elements of η for all subsequent iterations. With this constraint, the model primarily altered the first ingredient. Intuitively, this is because all recipe lists in our dataset have the first ingredient but not all have the twentieth so the rating model places higher weights on earlier ingredients.

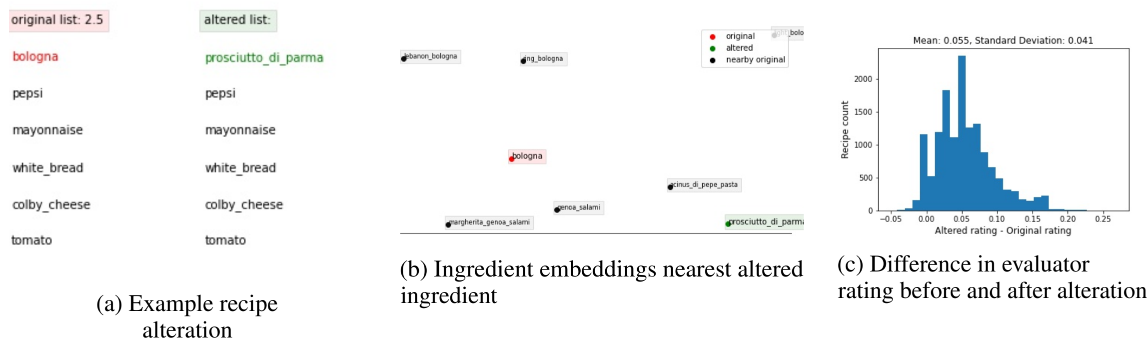
Another important hyperparameter was the stopping condition, i.e., at what point was the embedding considered sufficiently altered. A minimum stopping condition in all cases was that the ingredient list had changed. On it's own, this formulation has a naturally regularizing effect: the altered list remained close to the original list because alterations stop as soon as the list had changed. We wondered whether allowing further alterations might continue to improve the recipe. To this end, we continued iteration until the rating had increased over the original rating by some margin. Interestingly, when this margin was set too large (i.e., the algorithm was asked to improve the recipe by > 2 points), sweets (e.g., 'fruit', 'sherbet', 'chocolate') and liquors were often substituted regardless of the original ingredient list (Appendix Figure 7).

6 Results and Analysis

Overall, we found that altered ratings were consistently larger than original ratings, albeit modestly so (on the order of 0.05 point increase, Figure 2c). While our model certainly has room for improvement, a modest

increase may be expected as the quality of the recipe is influenced by many factors aside from ingredient choice (e.g., cooking directions, difficulty etc).

Importantly, as can be seen in Figure 2a-2b, the alteration algorithm doesn't simply choose the embedding nearest the original ingredient. Additionally, we found that recipe alterations were diverse (Appendix Figure 8). That is, it wasn't the case that soy sauce was always swapped for Worcestershire sauce but rather it was swapped for different kinds of sauces in different recipes. We found that alterations were most successful for ingredient classes that had many viable alternatives (sauces, oils, juices, cheeses) and for recipes that started with a low rating.



(a) Example recipe alteration

(b) Ingredient embeddings nearest altered ingredient

(c) Difference in evaluator rating before and after alteration

Figure 2: Example recipe alteration and performance across test dataset

Inappropriate alterations occurred most often for ingredients that were fundamental to the recipe (e.g., 'eggs', 'flour', 'baking powder'). We found that in such cases, the embedding for the original ingredient was notably isolated causing alterations to require a large shift in embedding space and thus, an inappropriate swap. In such cases, the ingredient swap often involved the addition of alcohol (commonly: 'baking soda' → 'mint chocolate liqueur'). A smaller portion of inappropriate alterations were due to phonetic similarity (e.g., 'Italian breadcrumbs' → 'Italian brandy', 'all purpose flour' → 'all fruit jam'). These issues were largely alleviated by an extensive data cleaning and embedding fine-tuning effort.

7 Future Work

We would like to improve both the predictor model used for input and the predictor model used for evaluation of the recipe generator. We were unable to develop a predictor model that did not overfit and had an acceptable validation loss. The first step would be to overcome the issues experienced with the GBDT model to determine if using GBDT would give us a better performing predictor model. Model performance might also be improved by obtaining more low rated recipes or additional embedding fine-tuning (e.g., with grocery data or nutritional information).

We also wondered if data augmentation strategies might aid performance. As described above, models that overfit had a tendency to report large rating improvements from minute recipe alterations. This is reminiscent of the original observations of adversarial examples (12) so perhaps a similar counter-strategy could be applied. Data could be augmented by adding small amounts of noise to the input embeddings. Noise additions would have to be sufficiently small so that the augmented embedding is still closest to the original embedding (i.e., the 'semantic meaning' of the augmented embedding is retained). Data augmentation might also help with another problem. As described above, our model associates larger weights to ingredients that occur earlier in the recipe causing alterations to be sensitive to list order. Data could be augmented by shuffling the order of ingredients.

Finally, we would like to improve the sensitivity to recipe context (i.e., the other ingredients in the list). For example, we found that 'flour' might be replaced with 'chocolate cake mix' in a chicken recipe. This can also be observed in Appendix Figure 7: each ingredient is improved independently. Incorporating meta-information, such as recipe category, either as an input or another required prediction might help in this respect.

8 Contributions

- **Abigail Russo**: Allrecipe dataset scraping; recipe alteration algorithm and corresponding hyperparameter search.
- **Brynne Hurst**: Fine-tuned the ingredient embeddings, implemented the rating predictor model, and ran hyperparameter search over the predictor model.
- **Trey Weber**: Ingredient list processing, parsing, and cleanup; background research.

References

[1] A. Salvador, N. Hynes, Y. Aytar, J. Marin, F. Ofli, I. Weber, and A. Torralba, "Learning cross-modal embeddings for cooking recipes and food images."

[2] A. Willis, E. Lin, and B. Zhang, "Forage: Optimizing food use with machine learning generated recipes," *CS 229*.

[3] M. Bień, M. Gilski, M. Maciejewska, and W. Taisner, "Cooking recipes generator utilizing a deep learning-based language model," 2020.

[4] K. Zhu, H. Sha, and C. Meng, "Chefnet: Image captioning and recipe matching on food image dataset with deep learning."

[5] S. B. Pune, R. Mahal, V. H, S. Singhal, and M. G. Kousar, "Image-to-recipe translation using multi-model architecture."

[6] M. Kumari and T. Singh, "Food image to cooking instructions conversion through compressed embeddings using deep learning," 2019.

[7] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[8] S. Li, "Food.com recipes and interactions," 2020. [Online]. Available: https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions?select=RAW_interactions.csv

[9] M. Lynch, "Ingredient phrase tagger," 2018. [Online]. Available: https://github.com/mtlynch/ingredient-phrase-tagger

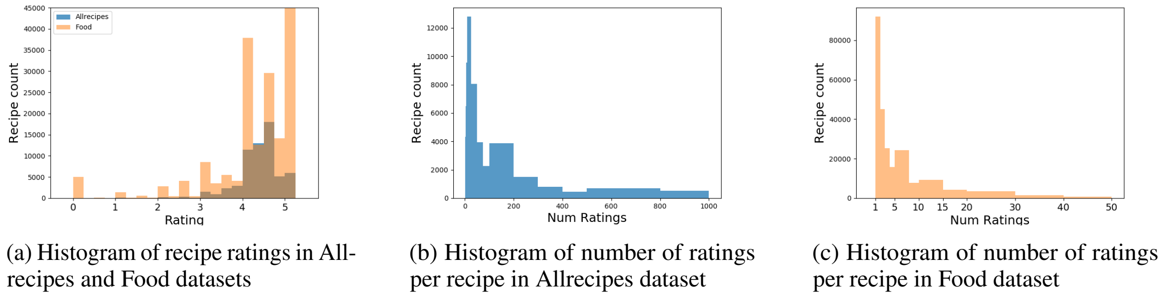
[10] N. Dingwall and C. Potts, "Mittens: An extension of glove for learning domain-specialized representations," in *NAACL-HLT*, 2018.

[11] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv*, vol. 1412.6572, 2014.

[13] "Tune: Scalable hyperparameter tuning," 2021,
Documentation available at https://docs.ray.io/en/master/tune/index.html. [Online]. Available: https://docs.ray.io/en/master/tune/index.html

Appendix

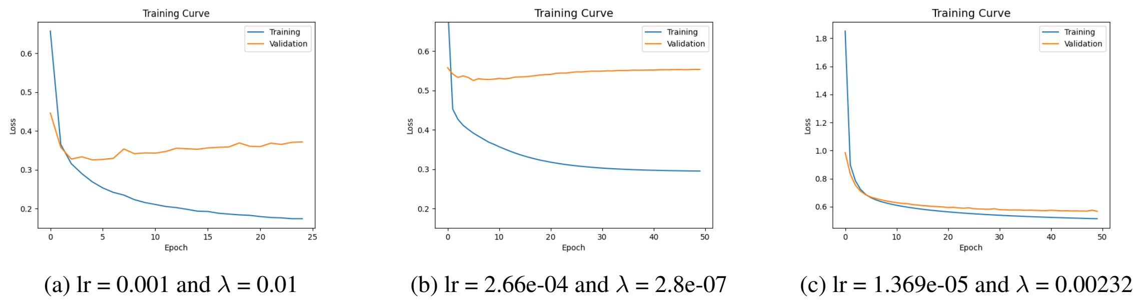


(a) Histogram of recipe ratings in Allrecipes and Food datasets

(b) Histogram of number of ratings per recipe in Allrecipes dataset

(c) Histogram of number of ratings per recipe in Food dataset

Figure 3: Distribution of Allrecipes dataset vs Food dataset

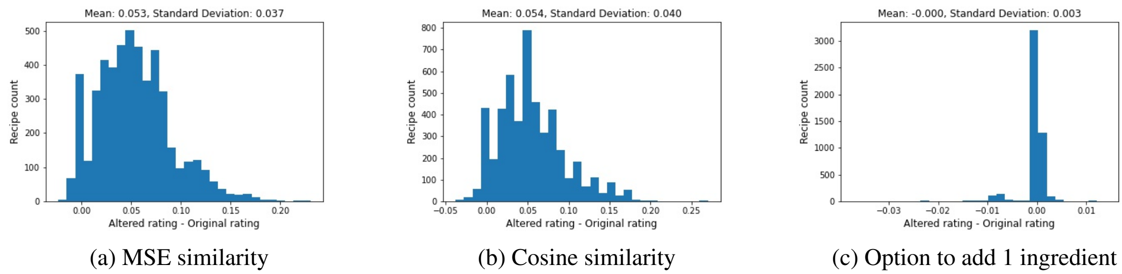


(a) lr = 0.001 and λ = 0.01

(b) lr = 2.66e-04 and λ = 2.8e-07

(c) lr = 1.369e-05 and λ = 0.00232

Figure 4: Predictor model training and validation loss using different values for learning-rate and λ.

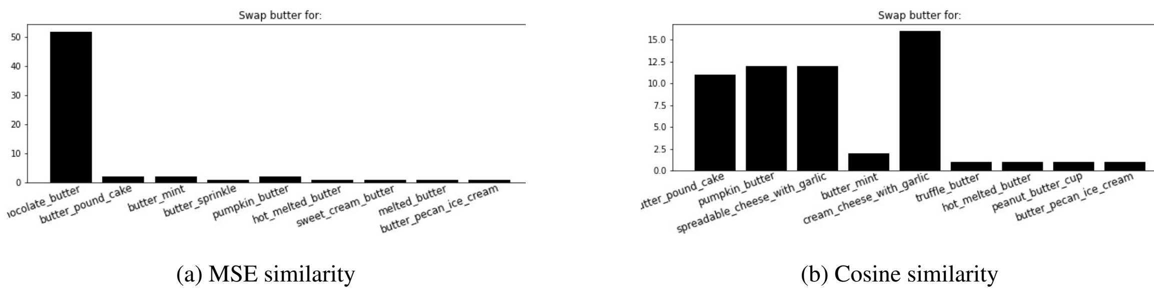


(a) MSE similarity

(b) Cosine similarity

(c) Option to add 1 ingredient

Figure 5: Hyperparameter testing for recipe alteration



(a) MSE similarity

(b) Cosine similarity

Figure 6: Cosine similarity tended to produce more diverse ingredient swaps.

original list: 4.5	altered list:	original list: 4.5	altered list:	original list: 4.5	altered list:
onion	onion	onion	green_jalapeno_pepper	onion	green_chutney
lemon	lemon_juice	lemon	lemon_juice	lemon	raspberry_liqueur
potato	potato	potato	potato	potato	pie_cherry
cod_fish_fillet	smoked_fish_fillet	cod_fish_fillet	red_snapper_fillet	cod_fish_fillet	skirt_steak
black_pepper	black_pepper	black_pepper	black_pepper	black_pepper	red_sprinkle
water	water	water	water	water	water_chestnut
juice_of	juice_of	juice_of	juice_of	juice_of	spanish_olive
parsley	parsley	parsley	parsley	parsley	parsley
salt	salt	salt	salt	salt	salt

(a) Margin: 0.2 pts (b) Margin: 1 pt (c) Margin 2 pt

Figure 7: Testing different stopping conditions. When iteration stop depends on the altered rating increasing, swaps can be inappropriate.

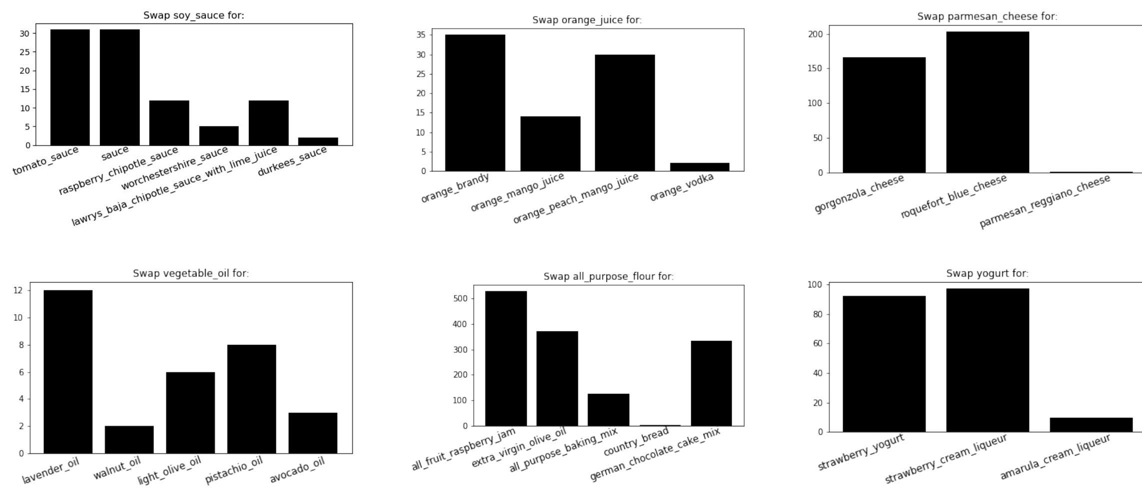


Figure 8: Histograms of altered ingredients conditioned on original ingredients.