Discovering Healthier Recipes using Natural Language Processing and Probabilistic Big Data Structures

Matthew Ormson
mto4125@rit.edu
Rochester Insitiute Of Technology
Rochester , New York, USA

Abstract

With the rise of the internet, sharing recipes has never been easier. Sifting through the billions of recipes online can be tedious and troublesome task, especially if your goal is to compare nutritional content. This paper aims to fix this issue by providing an easy and robust method for consuming and comparing recipes at a large scale. This system can be broken up into 5 distinct sections: data collection, data parsing, ingredient matching, similarity measurement, and recipe comparison. Sections 1 and 2 go over the background and related work. Sections 3 - 7 describe each part of the system and what design choices were made along the way. Finally, section 8 evaluates the impact of using ingredient substitutes in the similarity measurement calculation.

1 Introduction

Recent shifts in the modern lifestyle show that more consumers want to improve their health. However, a large portion of the population struggle to find the right resources and support, especially when it comes to healthy eating. Throughout this struggle, a consumer must manually sift through recipe websites and cookbooks in search of healthy food options that sound tasty. This paper aims to solve this problem by providing the user with an easy way to not only discover similar recipes, but also compare its nutritional content.

There are an unfathomable number of possible recipe combinations. Because the nature of food science is so complex, we focus on discovering recipe similarities between already existing recipes. Although any recipe input can be used, we utilize MIT's Recipe1M+ dataset [11] due to its massive size and easy-to-parse format. We then supplement this data with the publicly-available USDA nutrition dataset in order to calculate the total nutritional value of a recipe.

Different ingredients, although completely different in nutritional content, can taste extremely similar. We often take advantage of this phenomenon when we are looking for ingredient substitutions. Take chicken curry, for example. One recipe could call for *chicken breast, fresh tomatoes, heavy cream, and curry paste* while another recipe could call for *chicken thighs, tomato puree, coconut milk, and curry powder.* Despite technically not having any ingredients in common, both recipes are labeled the same. We scrape substitution data from *The Cook's Thesaurus* and use it in our recipe similarity computation.

A big pitfall with the Recipe1M+ dataset is its non-standardized ingredient list. Instead of providing the measurement of each ingredient, each specified ingredient is given as a string (i.e. *4 grams of sugar*). We utilize Electra[3], a state-of-the-art natural language processing model, to parse the given ingredient strings. We utilize a training dataset provided by The New York Times for training this model.

Matching each ingredient with its nutrition is not as straightforward as it seems. The nutrition labels provided by the USDA contains a lot of descriptors. Take the ingredient *cucumber*, for example. The USDA labels a *cucumber* as *CUCUMBER,WITH PEEL,RAW* or *CUCUMBER,PEELED,RAW*. Although this makes sense to humans, it makes ingredient matching much more difficult, especially when there are ingredients such as *PICKLES,CUCUMBER,SOUR* that make straight text comparison impossible. To solve this, we match each ingredient using the cosine similarity[12] after the label has been preprocessed using Electra. Electra allows us to take semantic meaning into account when matching. We can also use the same technique to match the ingredients with the ingredient substitutions as well.

Each recipe contains a unique set of ingredients. The USDA provides nutritional content for 8790 unique items. Because each ingredient in our dataset is matched to a USDA item, we can compare each recipe using its ingredients' matched USDA item identifier. We use the Jaccard Similarity[7] to compute the similarity between a set of ingredients.

However, because we have over a million recipes, directly comparing each set of ingredients easily becomes an $Nchoose2$ algorithm. This becomes even more infeasible as the number of recipes grow. To solve this issue, we use a MinHash LSH Forest [2] [1], approximate similarity-search algorithm. This allows us to compute K similar recipes efficiently with the trade-off of possibly inaccurate similarity calculations. Its at this point that we can take ingredient substitutions into consideration. In the evaluation section, we go into more detail about introducing ingredient substitutions affects the similarity scores.

Now that we have the ability to find similar recipes, comparing nutrition becomes trivial. All of the data that we capture during this project is stored within an SQL server database. Because everyone's nutrition goals are different,

finding the recipe with the least amount of sugar, for example, becomes a simple SQL query.

2 Related Work

A majority of related work mostly deals with creating recipes from scratch [10], designing recipe recommendation systems [13], or labeling recipe image data [8] [11].

This paper takes a different approach and instead focuses on discovering similar recipes based on the pre-existing recipes.

3 Evaluation Specifications

All tests done in this paper we conducted on a x64-based PC running Microsoft Windows 10 Education. This machine contains an Intel(R) Core(TM) i5-7500 CPU with 3.40GHz, 4 Cores, and 4 Logical Processors. It also contains 8GB of physical DDR4 RAM and 31.9GB of virtual memory. All data was stored using a 1TB SSD. Neural network training was conducted using an NVIDIA GeForce GTX 1060 with 6GB of memory.

In terms of software, we used Python 3.9.7, .NET 6, and C# 10. The .NET Common Language Runtime (CLR) allows our code to be compiled to all major platforms. We also enable ahead-of-time (AOT) compilation to enable fast startup.

4 Data Collection

We utilize Microsoft's SQL Server to store all of our collected data in a unified space. Storing our data using SQL allows us to easier run aggregated statistics and comparative analysis in the *Recipe Comparison* section. And aside from SQL Server being our most fluent database server, SQL server has two unique advantages over its competitors.

Firstly, Microsoft provides a *Bulk Insert* operation that loads data into a SQL Server table extremely fast. Although traditionally used to copy tables between different databases, the *SqlBulkCopy* class allows bulk loading to be done from any data source that implements the *IDataReader* interface. Under the hood, the *SqlBulkCopy* class essentially streams the data as a raw byte stream and writes it to SQL as if it were coming from a file source. Some investigation done by a gentleman on StackOverflow (link) confirms that the *SqlBulkCopy* class essentially "tricks SQL into thinking it's reading a file." This essentially means that we can stream a CSV file as an *IDataReader*, perform any transformations needed in memory, and stream those transformations into an SQL table extremely fast and efficiently. Below is a table of how fast we are able to load different CSV file sizes into SQL server:

Table	Amount Inserted	Time (s)
Recipes	1,029,720	7
Ingredients	9,605,000	236.7
Instructions	10,767,000	116.2
Nutrition	8,000	0.5
Substitutions	14,000	0.5

I would like to note the *SqlBulkCopy* class only performs bulk inserts. Bulk updates are possible, but basically involve performing a bulk insert into a temporary table and then merging the data into the permanent table. Although this is faster than running updates over a transaction, it is still quite slow. A better solution is to bulk insert this data into a separate table, and then create a *View* to easily and quickly access the joined tables. Although this does produce a storage penalty, we found that the trade-off is worth it.

Secondly, Microsoft's SQL Server allows for CLR integration. This means that we can compose SQL functions using managed code written in C#. This code contains all the necessary services needed to execute such as "just-in-time (JIT) compilation, allocating and managing memory, enforcing type safety, exception handling, thread management, and security" (link). This becomes particularly powerful when combined with *persisted computed columns*. Computed columns are what they sound like, columns that are computed from other data within a table. A persisted computed column means that the computed column is stored on disk and only computed on insert and update. Combining CLR integration and persisted computed columns, we can run C# code to automatically preprocess data at insert time, even using regular expressions which is not typically possible to use within SQL server.

This becomes particularly useful when preprocessing 9 million ingredients. This does come with a penalty, however, as seen in the table above. The *Ingredients* table contains a persisted column while the *Instructions* table does not. Despite containing more rows, the *Instructions* table takes half the time to insert data. However, we feel as if the benefits greatly out-weight the performance penalty, as 120 seconds to preprocessing 9 million ingredient descriptions is well-worth it.

4.1 Recipes

Our first attempt at gathering recipes involved scraping the website *AllRecipes*. This approach was working well until we stumbled upon Recipe1M+ [11]. This paper presented MIT provided a dataset with over 1 million recipes meant to train a neural network in classifying food images. Because a lot of these recipes already came from *AllRecipes*, we decided to abandon the *AllRecipes* scraping and use this dataset instead.

4.2 Nutrition

The U.S. Department of Agriculture has made a lot of effort to provide publicly available and transparent food nutrient information. The USDA provides nutrition for not only raw and cooked foods but for commercially available labeled food items. Raw/cooked food products are statistically and scientifically sampled to ensure accurate nutrition measurements. The sampling process and methodology is also transparently provided. We use the *Composition of Foods Raw, Processed, Prepared USDA National Nutrient Database for Standard Reference, Release 28* dataset, released in 2019. This dataset provides 46 nutritional measurements for 8790 unique food items. Nutritional measurements include protein, sugar, fat, carbohydrates, vitamins, etc.

4.3 Ingredient Substitutions

The Cook's Thesaurus is an online encyclopedia that covers food ingredients to kitchen tools foodsubs.com. Most importantly, this online encyclopedia contains a number of suggested substitutions. We utilized a script provided on Github link to scrape the ingredient substitutions listed on this website. This script uses spaCy to categorize ingredient substitutions as "good" or "bad". We modified the output of the "good" substitutions to better import into our SQL table.

4.4 Ingredient Parsing Training Data

The recipes given by the Recipe1M+ dataset has a major flaw. The ingredients provided are simply given as a string of text. For example, a recipe about Mac N' Cheese could contain the following ingredients:

- 6 ounces penne
- 2 cups Beechers Flagship Cheese Sauce (recipe follows)
- 3 cups milk

This becomes a problem when needing to calculate nutrition, match ingredient names, and match ingredient substitution names. This problem is known as named entity recognition. Luckily, the New York Times published a blog post in 2015 about the same exact problem link. In their blog, they explain how they ultimately used conditional random fields in order to classify ingredient strings into 5 different categories: name, quantity, unit, comment, and other. For example, the same ingredients from the Mac N' Cheese recipe above would be categorized as follows:

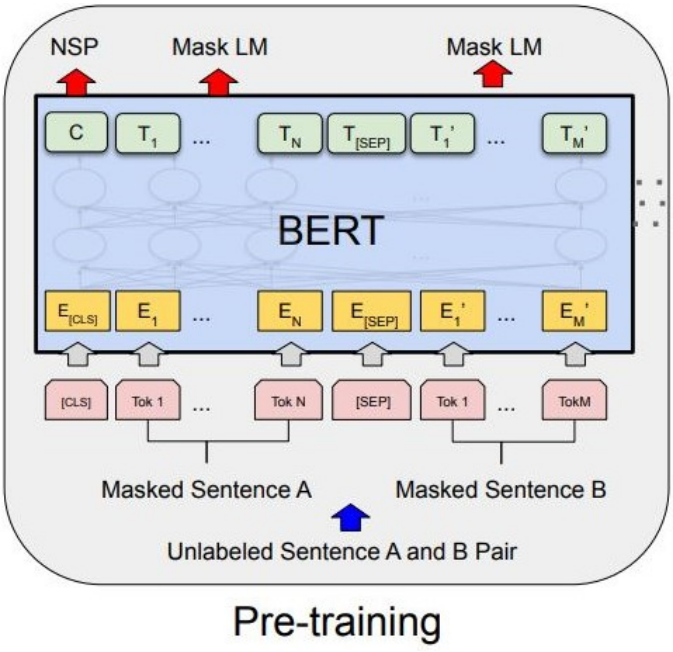
Name	Quantity	Unit	Comment r
penne	6	ounces	
Beechers Flagship Cheese Sauce	2	cups	(recipe follows)
milk	3	cups	

The New York Times was kind enough to provide a dataset with 8,791 ingredients parsed in this format and we utilize this dataset to solve our named entity recognition problem. However, instead of using conditional random fields (which are slow to train due to its sequential nature), we use a more modern deep learning model: Transformers.

5 Ingredient Parsing

5.1 BERT

BERT is a state-of-the-art natural language processing model [4]. These two images, borrowed from Google's paper on BERT, highlights the pre-training and fine-tuning phases of BERT:



Pre-training

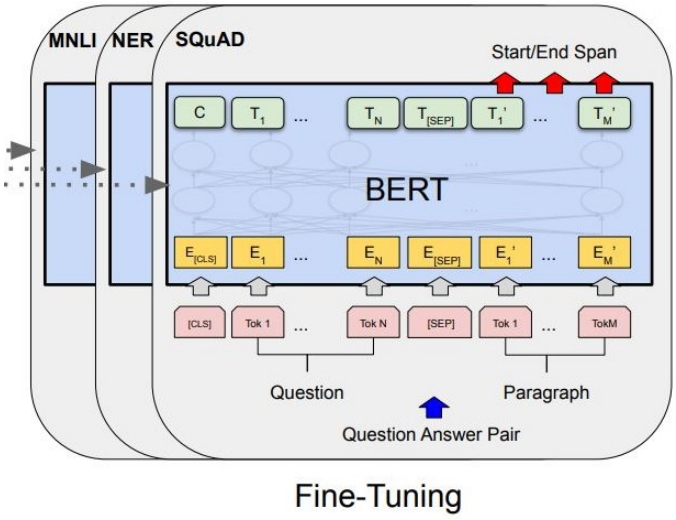
The pre-training phase takes a tokenized sentence as input. Sentences are tokenized using WordPiece emdeddings [14]. 15% of these tokens are chosen at random and replaced with a "masked" token. The model then attempts to predict these "masked" tokens. This technique is known as MLM.

The fine-tuning utilizes the "self-attention" mechanism in the Transformer model in order to perform many tasks such as classification. This is exactly what we use BERT for.

Before training BERT using the New York Times training data, we decided to preprocess the input. These preprocessing steps mimic the NYT's preprocessing steps with slight variations. Preprocessing includes:

Function	Example
Removing unicode fractions	$\backslash x215b \rightarrow 1/8$
Removing abbreviations	$1oz \rightarrow 1ounce$
Removing doubled units	$1cup/236ml \rightarrow 1cup$
Clumping fractions	$1and1/4 \rightarrow 1\$1/4$

We also decided to convert the 5 labels, name; quantity; unit; comment; and other, into part-of-speech (POS) tags [9]. We chose to use a simple tagging method, appending "B-" to



the first unique label in a sequence and "I-" to every sucessive label in that sequence. Take the preprocessed ingredient sentence for example: "1$1/4 cups cooked and pureed fresh butternut squash." Below would be its corresponding POS tags:

1$1/4	B-QUANTITY
cups	B-UNIT
cooked	B-COMMENT
and	I-COMMENT
pureed	I-COMMENT
fresh	I-COMMENT
butternut	B-NAME
squash	I-NAME

We use cross-entropy loss in order to optimize the BERT model against our now preprocessed and labeled data. After training, we ended with a 0.404 validation loss and an 86% accuracy score. Below is a table of our results:

	Comment	Name	Other	Quantity	Unit
Precision	69.5%	81.0%	77.7%	98.5%	93.4%
Recall	81.2%	82.8%	59.4%	99.0%	98.2%
F1 Score	74.9%	81.9%	67.4%	98.8%	95.7%
Support	26162	37253	28847	30808	23668

Although 86% accuracy isn't the best, we really only care about the *Name*, *Quantity*, and *Unit* labels. As seen above, the precision, recall, and F1 scores for these three categories are fairly high.

The downside to using this technique is the extremely large model size: ~400MB model file and ~850MB file. During evaluation, the size that this model took up in memory only allowed my NVIDIA 1060 6GB to prediction 128 ingredients at a time. Having over 9 million ingredients to parse, this process took over 2 days to complete. After realizing that we parsed these 9 million ingredients using non-preprocessed input and that we would have to restart the process, we quickly discovered a much faster and smaller model: Electra.

5.2 Electra

Electra is another machine learning model used in natural language processing. Electra was designed to fix the large computation time needed in models such as BERT that use masked language modeling (MLM). Instead of masking, Electra replaces tokens with "plausible" replacements generated from a "small generator network" [3]. Now, instead of predicting the masked input, Electra predicts whether each token was replaced by the generator network or not. This technique is effective with significantly less computation power and takes up significantly less memory. Now, instead of predicting 9 million ingredients in batches of 128, we were able to predict with Electra in batches of 1024. This allowed me to run these predictions in about half the time. Below is a table of our results using Electra:

	Comment	Name	Other	Quantity	Unit
Precision	69.5%	81.0%	77.7%	98.5%	93.4%
Recall	81.2%	82.8%	59.4%	99.0%	98.2%
F1 Score	74.9%	81.9%	67.4%	98.8%	95.7%
Support	26162	37253	28847	30808	23668

6 Data Matching

For this system to fully work, we must match the ingredient names from the *Recipe1M+ dataset* with the item names from the *USDA nutrition dataset*. We must also match the scraped ingredient substitution results from *The Cook's Thesaurus* with these datasets.

One way to do this would be by comparing the ingredient names letter by letter. Calculating the Levenshtein distance [6] is one approach of doing so.

This becomes a problem, however, when it comes to the *USDA nutrition dataset*. Take the ingredient *cucumber*, for example. If we were to compare this ingredient against the item names in the USDA dataset, we would get these results:

- *CUCUMBER,WITH PEEL,RAW*
- *CUCUMBER,PEELED,RAW*
- *PICKLES,CUCUMBER,SOUR*

The USDA dataset is riddled with semantic information that make it difficult to match. To solve this issue, we tokenize the ingredient names in each dataset using the WordPiece tokenizer we used previously with BERT/Electra. Tokenizing the data allows us to capture the semantic data that we couldn't previously. We can now compare the tokenized ingredient names using cosine similarity.

Cosine similarity is simply the cosine of the angle between to two token vectors [12]. We use the cosine of this angle in order to produce a numerical value between 0 and 1.

Using the example of a cucumber, we can see the cosine similarity values below:

USDA item name	similarity
CUCUMBER,WITH PEEL,RAW	0.40
CUCUMBER,PEELED,RAW	0.62
PICKLES,CUCUMBER,SOUR	0.39

Using this method, we are able to correctly match all of the ingredient names together.

6.1 Substitutions

The USDA nutrition dataset is particularly powerful because it is guarenteed to be a complete set of all food items. Because of this, it only makes sense to match the ingredient substitution data to it as well.

7 Recipe Similarity

7.1 Jaccard Similarity

Because recipe ingredients are inherently *sets*, we cannot directly compare ingredient vectors. Instead, we use the Jaccard similarity coefficient. Developed by Paul Jaccard, the Jaccard similarity between two sets is defined as the "ratio of the intersection over union" [7]. The Jaccard similarity coefficient is also bound between 0 and 1.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

An example can be seen below:

Recipe 1 (r_1): Worlds Best Mac and Cheese

Id	Ingredient Name
8096	penne
1009	Beechers Flagship Cheese Sauce
1009	Cheddar,
1023	Gruyere cheese,
2009	chipotle chili powder
1145	unsalted butter
20080	all-purpose flour
1089	milk
43340	semihard cheese
1042	cheese
2047	salt
2009	chipotle chili powder
2020	garlic powder

Recipe 2 (r_2): Beechers Flagship Cheese Sauce

Id	Ingredient Name
1145	unsalted butter
20080	all-purpose flour
1089	milk
43340	semihard cheese
1042	cheese
2047	salt
2009	chipotle chili powder
2020	garlic powder

$$\begin{aligned} Jaccard(r_1, r_2) &= \frac{8}{13} \\ &= 0.62 \end{aligned} \tag{2}$$

7.2 MinHash

Computing the Jaccard Similarity between every recipe can become expensive, because its time complexity is $O(nlogn)$ [2]. We can solve this by using a teqnique called a MinHash. This technique states that if you hash both sets, the Jaccard similiarity is approximately equal to the probability that each of the set's minimum hashes are equal. This means that we can estimate the Jaccard similarity coefficient in linear time, $O(nk)$ where k is the number of items hashed in each set. Not only can using a MinHash increase our time and space complexity, but it allows us to use techniques such as a MinHash LSH Forest.

7.3 LSH Forest

Local-sensitive hashing is a popular technique for approximating similarity. Items are hashed using a "locality-sensitive" hashing function that exploits collisions such that items that are similar to each other are more likely to collide into the same bucket [1]. The problem with local-sensitive hashing is that it requires heavy fine-tuning and can require very large storage.

LSH Forests aim to provide the same benefits of local-sensitve hashing while eliminating fine-tuning and the large storage requirement. LSH Forests also provide easy access to K number of similar items within a set.

LSH Forests are also designed to work with any hashing function. Combining the MinHash technique with the LSH Forest yields a powerful data structure for querying approximate top K similar recipes. We utilize the MinHash LSH Forest implementation from the *datasketch* python library.

7.4 Recipe Similarity Example

Recipe similar to: Worlds Best Mac and Cheese

Recipe	Approx Jaccard Similarity
Beechers Flagship Cheese Sauce	0.65625
Potatoes Au Gratin	0.4609375
Shrimp and Artichoke Dip	0.3359375
Chili Con Queso Dip	0.3125
Cheesy Cauliflower Gratin	0.265625

8 Recipe Comparison

Now that we have a method for discovering similar recipes, we have all of the tools necessary for recipe comparison. Because all of our data is stored withing Sql Server, comparing recipes can be easily articulated within an SQL query.

For example, imagine you want to find the similar recipes to "Worlds Best Mac and Cheese Beechers" but with more protein. This can be found using following query:

Listing 1. Query Example

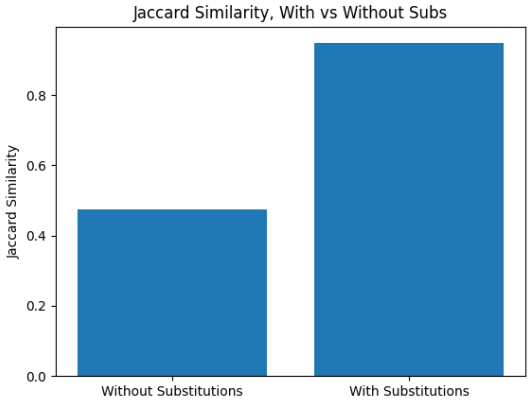
```
SELECT ingredients.RecipeID, ingredients.Name,
    SUM(ProteinCalculated) AS ProteinCalculated
FROM (
  SELECT r.RecipeID, r.Name,
   --Nutrition Conversions
   (CASE WHEN CHARINDEX(LOWER(i.Unit), LOWER(n.HouseholdDesc1)) > 0
    THEN 1 ELSE 0 END * i.Quantity * n.HouseholdWeight1 * n.Protein) +
   (CASE WHEN CHARINDEX(LOWER(i.Unit), LOWER(n.HouseholdDesc1)) > 0
       THEN 1 ELSE 0 END * i.Quantity * n.HouseholdWeight2 * n.Protein) +
   (CASE WHEN CHARINDEX('gram', i.Unit) > 0
       THEN 1 ELSE 0 END * (i.Quantity / 100) *  n.Protein)
       as ProteinCalculated
   FROM Data.Ingredients i
   INNER JOIN Data.Nutrition n ON n.NutritionID = i.NutritionID
   INNER JOIN Data.SimiarRecipes sr ON r.SimilarRecipeID = i.RecipeID
       AND sr.RecipeID = N''
   INNER JOIN Data.Recipes r ON r.RecipeID = sr.SimilarRecipeID
) as ingredients
GROUP BY ingredients.RecipeID, ingredients.Name
```

Unfortunately, the nutrition conversions given by the USDA is not the most elegant, but some easy string comparisons mitigates the issue.

9 Evaluation

9.1 Similarity Confidence

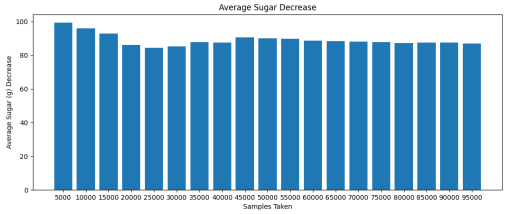
An important part of this system is detecting similar recipes. Because we are using MinHashes, we can quantify how similar (or connected) the recipes in our dataset are from each other. Below is a comparison of how similar our dataset is with using substitutions and without.



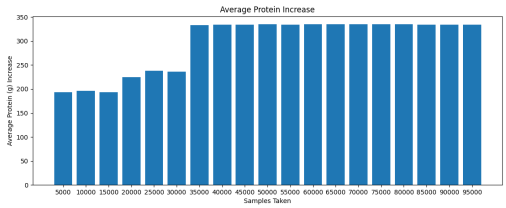
From the graph above, we can see that utilizing substitutions in our similarity computation allows us to find more similar recipes, with an increase of 47%.

9.2 Healthier Discoveries

The main goal of this system is to find healthier *and* similar recipe alternatives. The results shown below are comprised of the top 10 most similar recipes over *all* of the recipes in our dataset. The similarity computation was done without ingredient substitutions. Averages were taken over a random subset of recipes (due to size complexity) and plotted below.

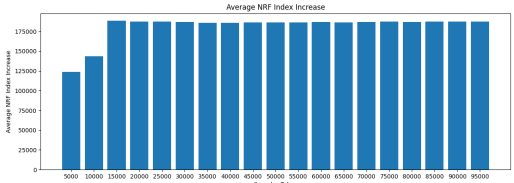


The histogram was taken over above shows that on average, we were able to find similar recipes with 86g less sugar on average.



The histogram above shows that on average, we were able to find similar recipes with 334g more protein on average.

The Nutrient Rich Foods (NRF) Index [5] was created by the USDA to promote nutrient dense food. It highlights increasing protein, fiber, vitamins A, vitamins C, vitamins E, calcium, iron, potassium, and magnesium while decreasing saturated fat, added sugar, and sodium. The "NRF9.3 algorithm" is comprised of the unweighted sum of the "healthy" nutrients minus the unweighted sum of the "unhealthy" ingredients. We plot the greatest NRF9.3 value increase over randomly sampled subsets below:



The histogram above shows that on average, we were able to find similar recipes with a 187386 higher NRF index on average.

10 Conclusion

This paper focuses on recipe discovery and comparison at a large scale. We utilize SQL bulk loading techniques and SQL CLR-computed columns to quickly gigabytes of recipes, nutrition, and ingredient substitution data. We use state-of-the-art machine learning techniques to parse and match this data together, allowing for easy access to every recipes' nutritional content. Our system then employs probabilistic similarity metrics to efficiently determine any number of similar recipes on a large scale, utilizing ingredient substitutions to boost our similarity measurements. Our results find that our system can ultimately discover healthier recipe

alternatives in regards to many different nutritional measurements.

11 Limitations and Possible Improvements

11.1 File IO

In order to gain the SqlBulkCopy performance benefit of Sql Server, data loading and pre-processing was designed in C#. Utilizing Electra and MinHash LSH Trees, however, were not readily available in C# and instead in Python. Therefore, a lot of intercommunication had to be done between the two languages. Transferring data using file IO was the easiest and most cross-platform solution. In the future, we would like to implement Electra using C#'s Tensorflow bindings (this is currently another project we are working on) as well as the LSH Forest.

11.2 LSH Forest

Despite the LSH Forest's incredible performance, it does require a lot of memory in order to perform (around 2.5 GB in our case). A better solution would be to implement the index lookup with Sql Server itself, taking advantage of its already-implemented lookup functionality. Although this would involve a lot of disk IO, our hypothesis is that the indexing function would still perform very fast.

Computing a MinHash could also be improved using newer techniques such as Circulant One Permutation Hashing (C-OPH), although this may be overkill.

12 Source and Data

https://github.com/elslush/RecipeDiscovery

References

[1] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH Forest: Self-Tuning Indexes for Similarity Search. In *Proceedings of the 14th International Conference on World Wide Web* (Chiba, Japan) *(WWW '05)*. Association for Computing Machinery, New York, NY, USA, 651–660. https://doi.org/10.1145/1060745.1060840

[2] A.Z. Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. 21–29. https://doi.org/10.1109/SEQUEN.1997.666900

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. *CoRR* abs/2003.10555 (2020). arXiv:2003.10555 https://arxiv.org/abs/2003.10555

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[5] Adam Drewnowski. 2009. Defining Nutrient Density: Development and Validation of the Nutrient Rich Foods Index. *Journal of the American College of Nutrition* 28, 4 (2009), 421S–426S. https://doi.org/10.1080/07315724.2009.10718106 arXiv:https://doi.org/10.1080/07315724.2009.10718106 PMID: 20368382.

[6] Rishin Haldar and Debajyoti Mukhopadhyay. 2011. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. *CoRR* abs/1101.1232 (2011). arXiv:1101.1232 http://arxiv.org/abs/1101.1232

[7] Paul Jaccard. 1912. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist* 11, 2 (1912), 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x arXiv:https://nph.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x

[8] Shobhna Jayaraman, Tanupriya Choudhury, and Praveen Kumar. 2017. Analysis of classification models based on cuisine prediction using machine learning. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. 1485–1490. https://doi.org/10.1109/SmartTechCon.2017.8358611

[9] Sheldon Klein and Robert F. Simmons. 1963. A Computational Approach to Grammatical Coding of English Words. *J. ACM* 10, 3 (jul 1963), 334–347. https://doi.org/10.1145/321172.321180

[10] Zhenfeng Lei, Anwar ul Haq, Mohsen Dorraki, Defu Zhang, and Derek Abbott. 2020. Composing recipes based on nutrients in food in a machine learning context. *Neurocomputing* 415 (2020), 382–396. https://doi.org/10.1016/j.neucom.2020.08.071

[11] Javier Marín, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. 2021. Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 1 (2021), 187–203. https://doi.org/10.1109/TPAMI.2019.2927476

[12] Amit Singhal and I. Google. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin* 24 (01 2001).

[13] M. B. Vivek, N. Manju, and M. B. Vijay. 2018. Machine Learning Based Food Recipe Recommendation System. In *Proceedings of International Conference on Cognition and Recognition*, D. S. Guru, T. Vasudev, H.K. Chethan, and Y.H. Sharath Kumar (Eds.). Springer Singapore, Singapore, 11–19.

[14] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144