Capstone Project
Recommender System for Recipes
Claudia Cassidy
March 31, 2018

Recommender System for Recipes

Project Overview

The goal of this project is to build a Recommender System for helping home cooks choose recipes. Unlike years ago, when people would look through recipe books and have index cards with a collection of up to 100 recipes, today there are hundreds of thousands of recipes freely available on the internet.

With so many to choose from, it can be difficult to find just the right recipe that uses ingredients you have in your pantry, that takes just the right amount of time to prepare, and that will likely taste good to you and your family. This project aims to use machine learning to help home cooks answer the question: "What's for dinner?".

Recommender Systems are commonly used in online shopping, for example where products are recommended for you based on products you've clicked on or searched for in the past. They are also used to recommend movies, as in Netflix where personalized recommendations are provided to each user. News services recommend articles to online readers.

This report will look at content-based recommender systems in which properties of an item are examined so that other items which are similar to that item are recommended. In this project, we will examine recipe ingredients and categories to identify recipes that are similar to one another. We will also examine recipe ratings and number of reviews.

Content Based Recommender System

We will build a recipe association recommender, which although not personalized, will examine the properties of the recipe that a user is looking at. Based on the recipe's ingredients, categories and title, it will provide recipe recommendations.

One way of providing these recommendations is to go through all the recipes and see what other ingredients have most frequently been part of the recipe along with the given ingredient in the same recipe. We can start by calculating what percentage of recipes with ingredient X (ingredient that the user is currently looking at) also has ingredient Y in the same recipe.

We can count the number of recipes that have ingredient X and Y, our numerator, and divide it by the number of recipes that have ingredient X. If, for example, at least 80% of recipes contain X and Y, then Y is a likely ingredient to add if you're already using X. We will try to build a system that recommends recipes that are similar to a particular recipe and then list the top 10 closest matches.

Analysis

AllRecipes.com is a community based website where people, mostly home cooks, submit their own recipes and have them published on the website. Most of the people who contribute recipes are not professional chefs.

I was able to generate a dataset of over 10,000 recipes by scraping the web pages of AllRecipes.com. An example of the script I wrote to scrape the recipes is in file "arSaladR.py". Recipes were scraped by categories as defined by AllRecipes.com. Using BeautifulSoup I scraped recipes in categories such as chicken, beef, salad, bread, pasta, vegetarian, cookies, beans, fish, pork and more. The data was written to csv files and then combined into a master spreadsheet. The master spreadsheet is file: "dataRecMaster2.csv".

Data Structure

Recipes on AllRecipes.com are structured according to Google's Recipe schema (http://schema.org/Recipe). Once scraped, the recipes were broken down into the following fields:

- **title:** recipe name
- **url:** link to the original recipe on AllRecipes.com
- **category:** one category that this recipe falls in, as defined by AllRecipes.com
- **prep time:** how long it takes to prepare the ingredients before they are cooked
- **cook time:** how long it takes to cook
- **total time:** how long it takes to make this recipe: total of preparation time plus cooking time
- **servings:** number of servings this recipe makes
- **rating:** average overall rating for this recipe
- **review_count:** number of reviews this recipe has received
- **categories:** an array of food categories that this recipe falls in, as defined by AllRecipes.com
- **author:** the person who contributed the recipe
- **ingredients:** array of ingredients

- **instructions:** array of instructions for how to make the recipe
- **fat:** health information: amount of fat per serving
- **calories:** health information: amount of calories per serving
- **carbohydrates:** health information: amount of carbohydrates per serving
- **protein:** health information: amount of protein per serving
- **cholesterol:** health information: amount of cholesterol per serving
- **sodium:** health information: amount of sodium per serving
- **reviews_text:** the text of up to 6 reviews

The fields which I selected to use from the dataset are title, categories, category and ingredients. I also experimented with rating and review_count. These fields seemed to be the most relevant to the goal of identifying recipes which are similar to one another. Ingredients and Categories contain the food words and food categories.

Data Exploration

The master dataset was trimmed down to use the fields: title, category, categories, ingredients, rating and review_cnt. After looking at the data it became obvious that there were terms which were not relevant to recommending foods, such as measurements and descriptive words. I wrote several functions to clean the data and eliminate irrelevant terms. Examples of terms which were removed are tablespoons, pinch, ounces, finely, scrubbed and any numeric term, for example 1/2. Stop words such as "as", "and", "the" and "or" and punctuation were also removed.

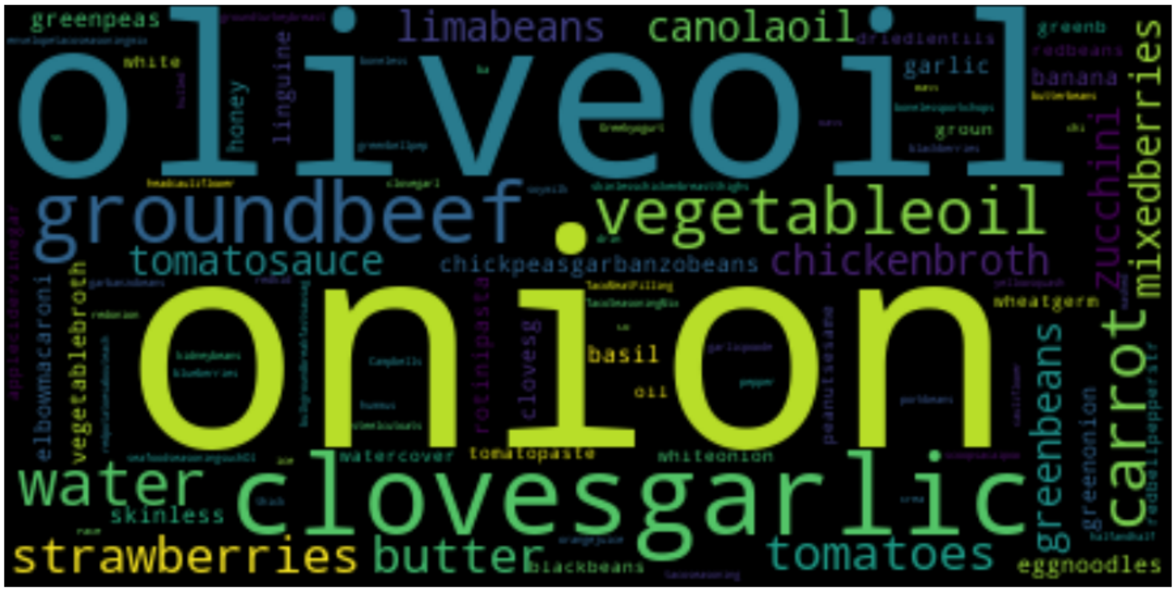
Another issue with the data was that ingredient terms can consist of more than one word. In order to make sure that the model does not treat an ingredient like "ground beef" as two separate words, for example "ground" and "beef", ingredients with multiple words were combined into a single word. For example, "olive oil" was concatenated into "oliveoil" and "beef bouillon cube" was changed to"beefbouillioncube". Every term was then converted to lowercase separated by a space.

Just like the ingredients data, I removed characters from the Categories data which are not letters, for example, punctuation marks. Multiple word terms were combined into single words, for example: "side dish" became "sidedish". All terms were converted to lowercase and separated by spaces.

The Ingredients and Categories fields were combined into one. Categories added more context about the recipe, for example "Salad", "Vegetables".

Ingredient plus Categories Word Cloud

Shown below is a word cloud which illustrates how frequently some ingredients occur among all the recipes in our dataset.



The larger the font size, the more frequently the ingredient occurs in our dataset of recipes. The largest words, such as onion, oliveoil, butter, chickenbroth, clovesgarlic and vegetableoil can even be considered as what chef's call "staples", i.e. ingredients that are so frequently used that it can be assumed that every cook is well stocked with these items in their kitchens. For our purposes, these "high frequency" ingredients are less important because they are so common.

Computing Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each recipe

Once the data was cleaned, the next step was to compute TF-IDF scores for the remaining words. Calculating TF-IDF indicates the importance of each food term to the recipe it belongs to. How many times a word appears in the recipe it belongs to is the TF (term frequency). The higher the TF value, the more important the term is. However, if the term appears in all recipes, then it is not really important for a particular recipe to be identified. For example, most recipes contain "salt". If every recipe contains salt then the term "salt" would not be helpful for identifying individual recipes. To resolve this, we needed a weighting system that decreases the importance of a term like salt when the number of the recipes it appears in increases.

TF-IDF is a weighting mechanism that calculates the importance of each term to each recipe by increasing the importance based on the term frequency while decreasing the importance based on the recipe frequency.

After calculating the TF-IDF scores we had 7,876 different terms which were used to describe 10,542 recipes in the dataset.

The next step was calculating a similarity score. Using the TFIDF matrix I computed a similarity score, the cosine distance between the high TF-IDF words as a vector. I used cosine similarity to calculate a number that denotes the similarity between two recipes.

$$cosine(x, y0) = x \,.\, y^T / (||x||\,.\,||y||)$$

To compute the cosine distance we think of the sets of high TF-IDF words as a vector, with one component for each possible word. The vector has 1 if the word is in the set and 0 if not. Almost all components are 0 in both, and 0's do not impact the value of the dot product. The dot product is the size of the intersection of the two sets of ingredient words, and the lengths of the vectors are the square roots of the numbers of words in each set. That lets us compute the cosine of the angle between the vectors as the dot product divided by the product of the vector lengths.

Next, we needed a function that takes in a recipe title as an input and outputs a list of up to the 10 most similar recipes. To do this, I made a reverse mapping of recipe titles and dataframe indices so that we can identify the index of a recipe given its title.

CountVectorizer

I also tried using a CountVectorizer() in order to compare it to TF-IDF. Using CountVectorizer will not down-weight the presence of an ingredient if it appears in relatively more recipes. I used a CountVectorizer with a pairwise cosine similarity calculation and then compared the results to TF-IDF.

Results of TF-IDF vs CountVectorizer

I wrote a function which takes in the parameters for the title of a recipe and the cosine similarity for either TF-IDF matrix or the CountVectorizer matrix. The function returns a list of up to 10 closest recipes. The following are some examples of the resulting recipe recommendations.

Example 1a: “Baked Slow Cooker Chicken”, using TF-IDF:

Beer Butt Chicken
Bacon Roasted Chicken
Simple Whole Roasted Chicken
Oven Fried Chicken III
Honey Baked Chicken II
Spicy Lemon Pork Saute
Get a Husband Brunswick Stew
Vegetarian Alternative to Ground Beef
Homemade Chicken Cacciatore, Sicilian-Style
Slow Cooked Pork Barbeque
Chicken Chile Spaghetti

Example 1b: “Baked Slow Cooker Chicken”, using CountVectorizer:

Beer Butt Chicken
Bacon Roasted Chicken
High Temperature Eye-of-Round Roast
Slow Cooked Pork Barbeque
Sarge's EZ Pulled Pork BBQ
Vegetarian Alternative to Ground Beef
Bum's Lunch
Slow Cooker Barbeque
Cabin Dinner
Chicken Fried Steak I
Ground Beef and Cabbage
Juicy Butt Steaks

In this example, “Baked Slow Cooker Chicken”, the results were the same for the first two recommended recipes. The TF-IDF results appear to be more related to the given recipe than the CountVectorizer results as it returned more recipes related to chicken and slow cooker than the CountVectorizer had.

Example 2a: Steamed Asian Sesame Veggies, using TF-IDF

Steamed Broccoli
Broccoli in Roast Chicken Drippings
Broccoli With Lemon Almond Butter

Tangy Broccoli
Snow Peas with Water Chestnuts
Roasted Broccoli Salad
Sesame Broccoli
Japanese Spinach with Sweet Sesame Seeds
Stir-Fried Pumpkin
Snow Peas Oriental
Broccoli Polonaise
Broccoli Cheese Casserole with Rice
Chicken Stir-Fry
Stir-Fry Broccoli With Orange Sauce
Sesame Cabbage and Mushrooms

Example 2b: Steamed Asian Sesame Veggies, using CountVectorizer

Broccoli in Roast Chicken Drippings
Tangy Broccoli
Steamed Broccoli
Broccoli Cheese Casserole with Rice
Sesame Broccoli
Grilled Broccoli--My Kids Beg for Broccoli
Broccoli With Lemon Almond Butter
Roasted Broccoli Salad
Broccoli Polonaise
Garlic-Sauteed Asparagus
Stacey's Fabulous Broccoli Fritters
Stir-Fry Broccoli With Orange Sauce
Easy Lemon and Garlic Broccoli
Spicy Bok Choy in Garlic Sauce
Cheesy Cauliflower and Broccoli Gratin

In the second example, given “Steamed Asian Sesame Vegetables”, TF-IDF again returned more results which are related to Vegetables, Asian and Sesame than the CountVectorizer returned.

Both TF-IDF and CountVectorizer produced reasonable recipe recommendations based on the submitted recipe title. Many of the first few recipes returned were the same, but there were variations after about the fourth recipe. Overall, while both produced reasonable results, I found that the TF-IDF matrix returned recipes which were closer to the recipe title submitted. For example, "Steamed Asian Vegetables" results using the TF-IDF matrix returned on the mark recipes like "Stir-Fry Broccoli With Orange Sauce", while the CountVectorizer matrix, we see somewhat off the mark recipes like "Cheesy Cauliflower and Broccoli Gratin".

2 - Recommender System Considering Ratings and Number of Reviews

Since ratings can be different based on the number of reviews, for example should a recipe with only 3 ratings, where the average rating is 5.0, be evaluated as having a truly higher rating than a recipe with 500 ratings, where the average rating is about 4.0? It is more fair to use a weighted rating instead. Here's how it was calculated:

v: number of reviews for the recipe
m: minimum number of votes required to be listed
R: average rating of the recipe
C: mean vote across all recipes

Take the 90th percentile as the cutoff in terms of number of reviews.

The weighted average formula is: $(v/(v+m)*R)+(m/(m+v)*C)$

After applying the weighted average formula, the top 20 recipes based on overall average ratings and number of reviews are:

Top 20 Highest Rated Recipes:

title	category	review_cnt	rating	score
Fluffy Pancakes	Pancakes	9933	4.82	4.790216
Chicken Pot Pie IX	Chicken Breasts	8068	4.81	4.774268
Banana Crumb Muffins	Banana Muffins	9774	4.80	4.770679
Downeast Maine Pumpkin Bread	Breakfast Bread	6877	4.81	4.768479
Janet's Rich Banana Bread	Banana Bread	4944	4.82	4.762791
Clone of a Cinnabon	Yeast Bread	5629	4.81	4.759989
Taco Seasoning I	Mexican	4348	4.82	4.755720
Grandmother's Buttermilk Cornbread	Cornbread	4906	4.81	4.753268
Boilermaker Tailgate Chili	Beef Chili	4663	4.81	4.750584

Guacamole	Guacamole	4222	4.81	4.745004
Apple Pie by Grandma Ople	Fruit Pies	7720	4.77	4.735065
Simple Scones	Scones	1917	4.87	4.730452
Too Much Chocolate Cake	Chocolate	5200	4.78	4.728713
Amish White Bread	White Bread	4277	4.79	4.727752
Chicken Cordon Bleu II	French	4443	4.78	4.720818
Peanut Butter Cup Cookies	Cookies	3063	4.80	4.715047
White Chocolate Raspberry Cheesecake	Raspberry Desserts	2263	4.83	4.715020
Mom's Zucchini Bread	Breakfast Bread	6621	4.75	4.710989
Banana Sour Cream Bread	Breakfast Bread	4118	4.77	4.707661
Caramel Popcorn	Popcorn Candy	1777	4.85	4.705984

The dataset does not include individual user ratings, the rating is the overall average rating of all users who rated the recipe. While these results may represent the overall average ratings of everyone who has reviewed and rated recipes, it is not personalized to individual preferences. We would need individual user preference data to better recommend recipes for individuals. For example, if someone is following a low-carb diet, then almost none of these highest rated recipes would be relevant to that person. This data could be used as a “cold start” where we recommend the overall most popular recipes until we can capture user recipe preferences.

3 - Recommender System with Word2Vec

One of the key ideas in Natural Language Processing (NLP) is converting words into numeric vectors. The vectors are then fed into machine learning models in order to perform predictions. One way of converting words into numeric vectors is to use a "one-hot" method which converts one element of the vector to a 1 value with the rest of the elements being 0. So a sentence like "the cat sat on the mat" would be converted to a 6x5 matrix that looks like:

```
the: 1 0 0 0 0
cat: 0 1 0 0 0
sat: 0 0 1 0 0
 on: 0 0 0 1 0
the: 1 0 0 0 0
mat: 0 0 0 0 1
```

As the dataset grows larger, the one hot encoding method becomes inefficient. It also does not show the similarity among words.

The first component to the Word2Vec methodology is word embedding while maintaining context. One approach to doing this is the Skip-Gram method in which we take an input word and then try to estimate the probability of other words appearing close to that word. The second method is Continuous Bag of Words (CBOW), which does the opposite. It takes some context words as input and then tries to find a single word that has the highest probability of fitting that context.[1]

The Word2Vec models are shallow, two layer neural networks that are trained to reconstruct linguistic contexts of words. Word2Vec produces a vector space of hundreds of dimensions with each unique word assigned a corresponding vector in the space. They are positioned in the vector space such that words that share common contexts are located near to one another in the space.[2]

Word2Vec learns the context of a word by looking at the words which commonly occur around the target word. Words which have similar contexts share meaning under Word2Vec and their reduced vector representations will be similar. In the skip-gram model version of Word2Vec the goal is to take a target word and predict the surrounding context words. In this project we apply this technique to recipe ingredients, learning ingredients that commonly occur together.

We supply our input target words as one-hot vectors to the neural network. Then, via a hidden layer, we train the neural network to increase the probability of valid context words, while decreasing the probability of invalid context words.

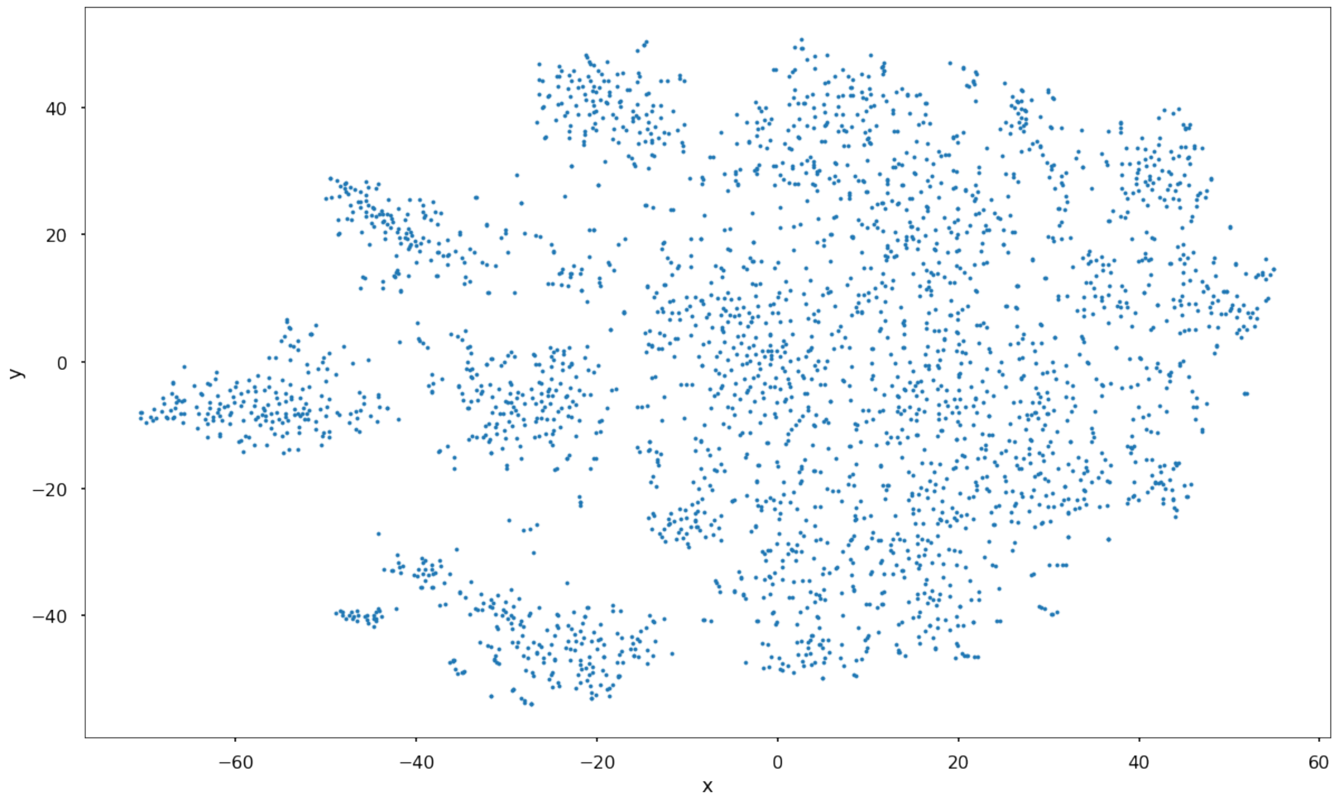
We also need a way to ensure that as the network trains, words which are similar end up having similar embedding vectors. We want to ensure that the trained network will always output a 1 when it is supplied words which are in the same context, but 0 when it is supplied words which are never in the same context. We need a vector similarity score supplied to the output sigmoid layer – with similar vectors outputting a high score and non-similar vectors outputting a low score. As we did previously, we will use the cosine similarity score.

After converting the ingredients and categories into a list of words, I created tokens for each word. This recipe dataset contains 138,366 tokens.

The quality of word embedding increases with higher dimensionality, but after reaching some point, marginal gain will diminish and it can become more computationally expensive to train. Typically, the dimensionality of the vectors is set to be between 100 and 1,000. The dimensionality I used was 300.[4]

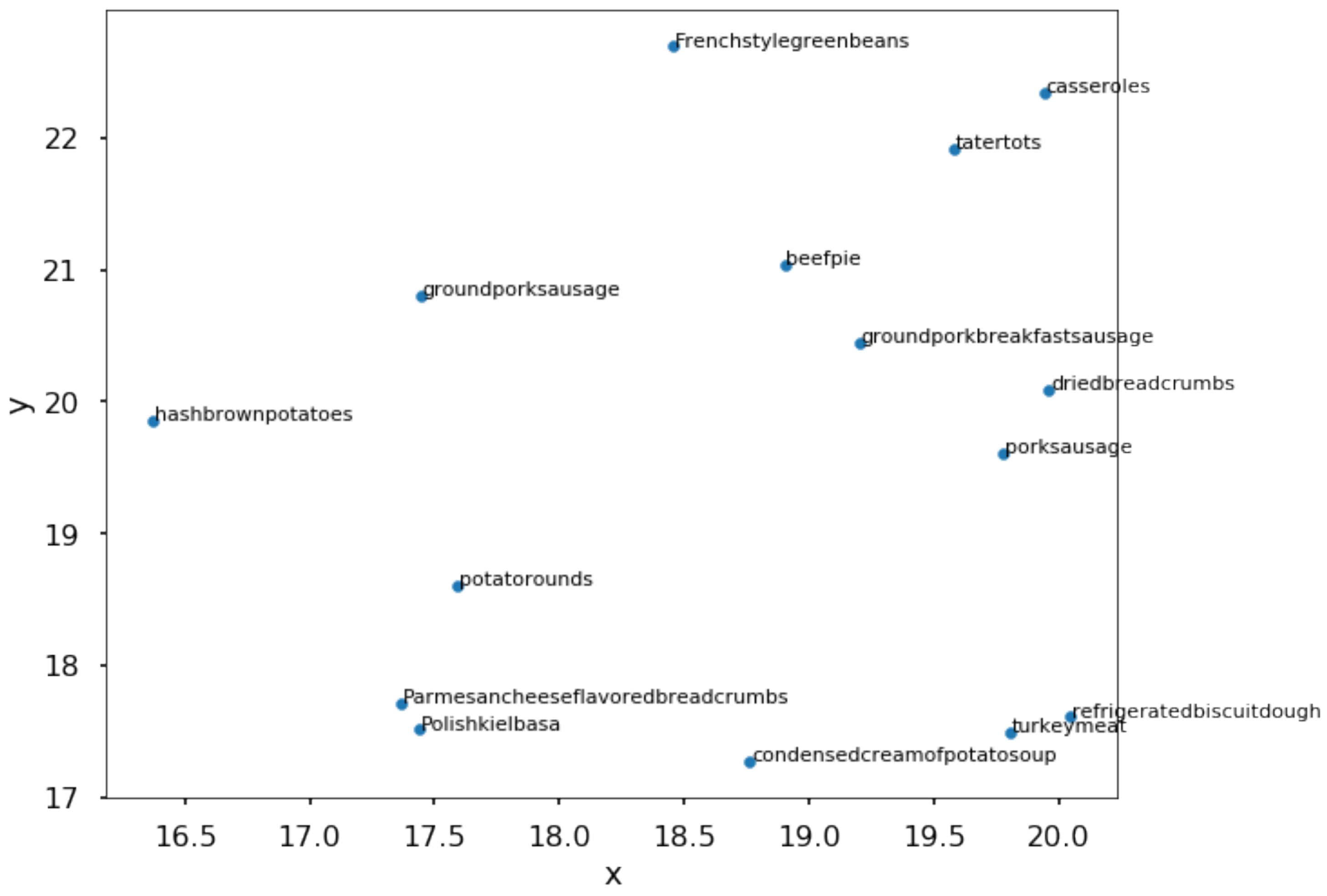
After training the model, I used TSNE (t-distributed stochastic neighbor embedding) to visualize the results. TSNE is a nonlinear dimensionality reduction technique that is useful for embedding high dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Similar objects are modeled by nearby points and dissimilar objects are modeled by distant points.[3]

Here is a scatterplot of the ingredient word embeddings:



The scatterplot shows a 2 dimensional position of each word with a label.

A closeup, zoomed in view of the scatterplot is shown below.



The zoomed in scatterplot shows that similar items are located near each other. For example, potato rounds are near hash brown potatoes and ground pork breakfast sausage is near pork sausage.

Finding items which are most similar to a given ingredient

Once the Word2Vec model is trained, we can submit an ingredient and get a prediction of which other ingredients are most related to the given ingredient. Shown below are several examples where I submitted an ingredient name and got back a list of ingredients that are most similar to the given ingredient.

Example 1: Ingredients which are most related to “beef”:

steaks, 0.8334396481513977
sirloinsteak, 0.7732095718383789
rib, 0.7576472759246826
eyesteak, 0.7522614002227783
filetmignon, 0.7265390157699585
roundsteak, 0.7246712446212769
cubesteaks, 0.7171943187713623
beefconsomme, 0.7168818116188049
ribeyesteaks, 0.7155680656433105
beefstroganoff, 0.7098640203475952

The top related ingredients for this example, beef, are predicted with a similarity of between 70% and 83%. The model determined that “steaks” are similar to “beef” even though the two words are different. If we wrote a general search function without machine learning, the search engine would search only for matching key words and would not show “steak” when searching for “beef”. Word2Vec results are spot on for finding ingredients that are related to “beef”.

Example 2: Ingredients which are most related to “almond milk”:

flaxseedmeal, 0.9310940504074097
flaxseeds, 0.9304770231246948
hempseeds, 0.9259158372879028
almondbutter, 0.9244171380996704
soymilk, 0.9235554933547974
groundflaxseed, 0.9172952771186829
scoopvanillaproteinpowder, 0.9127264022827148
spirulinapowder', 0.9112852811813354
chiaseeds, 0.9052334427833557
vanillaproteinpowder, 0.9029464721679688

The top related ingredients for the second example, almond milk, are predicted with a similarity of between 90% and 93%. The model returned ingredients that are commonly used with almond milk to make smoothies.

Showing analogies: ingredient A is to ingredient B as ? is to ingredient C

We can use cosine similarity to calculate the similarities among ingredients. Mikolov et al[5], the creators of Word2Vec, demonstrated that the model can show relations by showing the closest word vector to: $king - man + woman = queen$

The technique they used for solving analogies was:

$$argmaxx \in V \setminus king, man, womancos(x, king - man + woman)$$

where **V** is the entire vocabulary and **cos** is the cosine similarity between the word vectors. It looks for some word vector x, which is similar to “king", similar to “woman”, and dissimilar to “man”. The best answer is “queen”. [4]

I tried to apply this analogy technique, using cosine similarity to calculate the similarities among ingredients in recipes. The following are some examples, with mixed results.

Example 1: “salt”, “beef”, “chicken”

Result: “salt is related to beef as margarinebutter is related to chicken”

Example 2: “chickenstock”, “soup”, “cookies”

Result: “chickenstock is related to soup as flakedcoconut is related to cookies”

Example 3: “onion”, “pork”, “groundbeef”

Result: “onion is related to pork as hotItaliansausage is related to groundbeef”

The analogies tested above don't quite make sense but are not too far fetched.

Looking up recipes that contain multiple ingredients

One of the objectives of this project is to be able to quickly find recipes given ingredients that the home cook has on hand. I wrote a function that returns recipes which contain multiple ingredients, sorted by average rating. Now that we have a recommender system which identifies related ingredients, we can use the function to return relevant recipes with those combination of ingredients.

Reflection

Several approaches were attempted in order to develop a recommender system for home cooks to choose recipes. A major part of the process was cleaning up the data and making it as understandable as possible for the algorithms to process. For example, handling ingredients that consist of multiple words, such as "olive oil" and combining them into one term: "oliveoil" and eliminating irrelevant words such as measurement terms. The machine learning algorithms think in the language of math and so the words had to be easily translatable into numeric vectors.

The Word2Vec model gave impressive results when it came to predicting ingredients which are similar to a given ingredient. For example: given "beef", the ingredients which were predicted by the model included:

steaks, 83.94%
sirloinsteak, 80.14%
rib, 78.37%
eyesteak, 77.85%
filetmignon, 74.98%
ribeyesteaks, 73.85%
cubesteaks', 72.63%
beefstroganoff, 72.54%
flatironsteak, 71.61%
cornedbeef, 71.09%

A good use case for this would be if a user clicks on a recipe and the main ingredient in that recipe is "beef", then our algorithm could see the similar ingredients listed above and present recipes that contain those ingredients. Word2Vec figured out that "steak" is similar to "beef". Our frontend app would now be able to present useful and relevant recipes for beef, even if none of the words in the recipe contains “beef”, for example, recipes like “Super Deluxe Steak Nachos” or “Grilled Filet Mignon with Gorgonzola Cream Sauce”.

Using TF-IDF, I also was able to find recipes which are similar to a given recipe. So if a user has recently selected a recipe like "Easy Lasagna I", then our recommender system will find recipes that are similar to lasgna but don't necessarily contain the word "lasagna", for example: "Meat Filled Manicotti", "Baked Spaghetti Casserole", "Manicotti" and "Italian Veggie Rolls".

Given that there are hundreds of thousands of recipes available online, it would be very useful to have a recommender system to help people choose the best recipes. It would be an even better system if it can be personalized to each user. I would like to build a mobile app which tracks the recipes each user selects, the search terms they enter and any recipe they save, so that the Recommender System can also make recommendations based on collaborative filtering. With collaborative filtering, the system could show that people who liked the recipes that you like, also liked these other recipes and so there is a high probability that you may like them, too.

While this project uses data scraped from AllRecipes.com, going forward, I would like to write scripts to scrape recipes from additional websites, for example: FoodNetwork.com, Delish.com, BettyCrocker.com, Epicurious.com, and more as long as the recipes are freely available and not behind a firewall. All of the major recipe websites use the same Google Recipe schema, so the basic script used for AllRecipes.com could be customized for each. Once a recipe is in the recommendation list, the user could tap on it and the recipe's URL would open the source webpage on the original website. Having more data would likely yield even better recommendations.

References

[1] http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/

[2] https://en.wikipedia.org/wiki/Word2vec

[3] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

[4] https://www.quora.com/How-does-Mikolovs-word-analogy-for-word-embedding-work-How-can-I-code-such-a-function?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

[5] https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf