Capstone Project
Recommender System for Recipes
Claudia Cassidy
April 15, 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.

Problem Statement

The goal of this project is to create a content-based recommender systems in which properties of a recipe are examined so that other recipes which are similar to that recipe are recommended, i.e. a recipe association recommender. This project examines recipe ingredients, titles and categories in order to predict recipes which are similar to one another.

We will vectorize the ingredient and title words, train and fit several models, and then test by trying to predict recipe categories given ingredients or title. Several supervised models including MultinomialNB and SGDClassifier will be tried as well as the unsupervised Word2vec model. The corresponding Jupiter notebooks for this project are: “CategoryByTitleIngredM.ipynb” in which we try the supervised models, and “recipeSimilar.ipynb” for the word2vec implementation.

Metrics

CountVectorizer and Term Frequency - Inverse Document Frequency (TF-IDF) are used to vectorize recipes and then generate similarity scores between recipes and ingredients. Next, the vectors are fed into several machine learning models and optimized. Hyper parameters for the models are tuned and optimized to get the highest possible test scores.

Semantic word embeddings represent individual ingredients and learn how to weigh every ingredient word in the recipes through the use of TF-IDF information to get an overall representation of ingredients. These representations are evaluated through a semantic similarity task. Applied to recipes, that would mean that ingredients that appear in the same recipes go well (taste good) together. Chefs have combined ingredients through trial and error have written recipes in which certain ingredients are known to taste good when combined to make prepared food, for example: brown sugar + chocolate chips + butter + eggs + flour = chocolate chip cookies; butter + flour + milk = white sauce.

We are interested in the calculated cosine similarity features. We want related pairs to lie close to each other in their representation space, and non-related pairs to lie far apart:

$$g(t_1, t_2) = \begin{cases} \text{pair} & \text{if } d(t_1, t_2) \le \theta \\ \text{non-pair} & \text{if } d(t_1, t_2) > \theta \end{cases}.$$

In this expression t1 and t2 are two short text vector representations of dimensionality v, d : (x, y) ∈ R2v → R+ is a vector distance function of cosine distance, θ is a threshold, and g(·) is the binary prediction of semantic relatedness.

Precision, recall and f1-score of the training and test phases will be reported for different classification algorithms. The precision P measures the fraction of correct predictions:

P = TP / (TP + FP)

where P is the number of true positives and FP is the number of false positives. Precision measures how often the algorithm is correct.

The recall measures the fraction predicted instances that are correct:

R = TP (TP + FN)

where TP is the number of true positives and FN is the number of false negatives.

The recall shows how complete a predicted set is.

The results of the analysis are interpreted with specific examples that can be intuitively understood for common sense validation. The term frequency–inverse document frequency will be used to identify keywords and their importance in the recipes.

Analysis

Data Exploration

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 for this project by scraping the web pages of AllRecipes.com. An example of the script used to scrape the recipes is in file "arSaladR.py". Using BeautifulSoup, recipes were scraped in categories such as chicken, beef, salad, bread, pasta, vegetarian, cookies, beans, fish, pork and more. The data was written to csv and then combined into a master spreadsheet, file: “dataRecipeMaster.csv”.

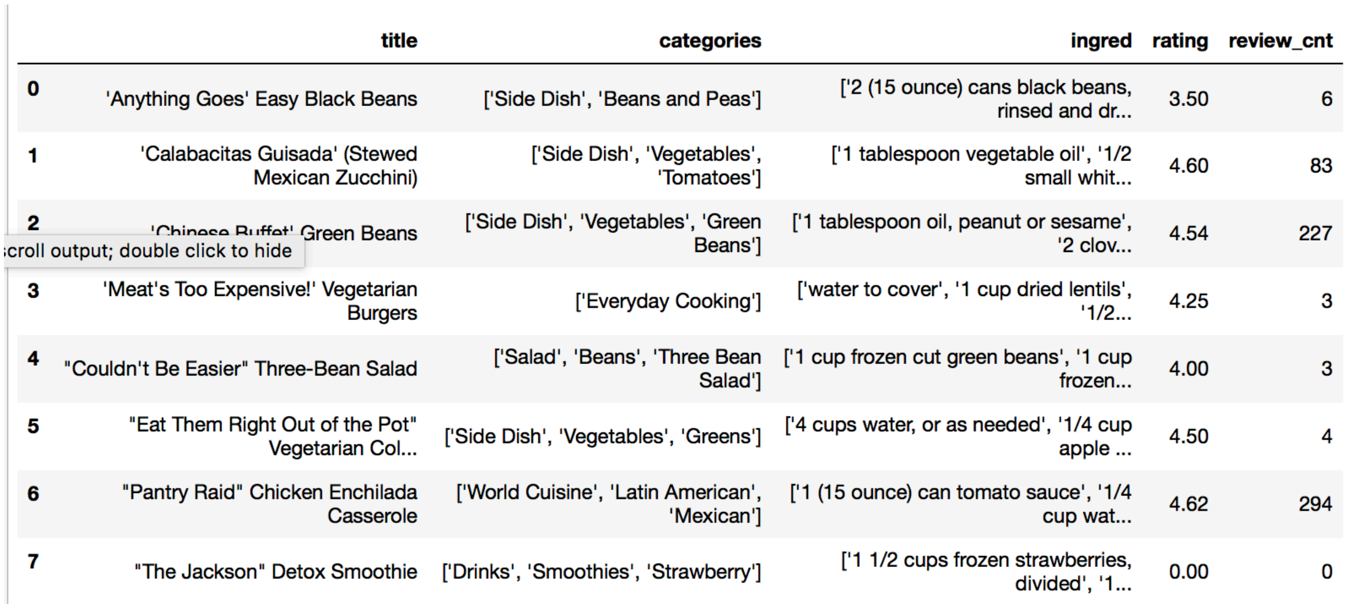
Recipes on AllRecipes.com are structured according to Google's Recipe schema (http://schema.org/Recipe). [5] Once scraped, the recipes were broken down into the following fields:

- **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 were 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. Many ingredients and categories consist of 2 or more words, for example “all purpose flour” or “side dish”. To make sure the vector would understand these to be one single term, multiple words which define a single term were joined to form one term.

Exploratory Visualizations

Shown below are the first 8 rows of the dataset including columns: title, categories, ingredients, rating and review_cnt.


scroll output; double click to hide

	title	categories	ingred	rating	review_cnt
0	'Anything Goes' Easy Black Beans	['Side Dish', 'Beans and Peas']	['2 (15 ounce) cans black beans, rinsed and dr...	3.50	6
1	'Calabacitas Guisada' (Stewed Mexican Zucchini)	['Side Dish', 'Vegetables', 'Tomatoes']	['1 tablespoon vegetable oil', '1/2 small whit...	4.60	83
2	'Chinese Buffet' Green Beans	['Side Dish', 'Vegetables', 'Green Beans']	['1 tablespoon oil, peanut or sesame', '2 clov...	4.54	227
3	'Meat's Too Expensive!' Vegetarian Burgers	['Everyday Cooking']	['water to cover', '1 cup dried lentils', '1/2...	4.25	3
4	"Couldn't Be Easier" Three-Bean Salad	['Salad', 'Beans', 'Three Bean Salad']	['1 cup frozen cut green beans', '1 cup frozen...	4.00	3
5	"Eat Them Right Out of the Pot" Vegetarian Col...	['Side Dish', 'Vegetables', 'Greens']	['4 cups water, or as needed', '1/4 cup apple ...	4.50	4
6	"Pantry Raid" Chicken Enchilada Casserole	['World Cuisine', 'Latin American', 'Mexican']	['1 (15 ounce) can tomato sauce', '1/4 cup wat...	4.62	294
7	"The Jackson" Detox Smoothie	['Drinks', 'Smoothies', 'Strawberry']	['1 1/2 cups frozen strawberries, divided', '1...	0.00	0

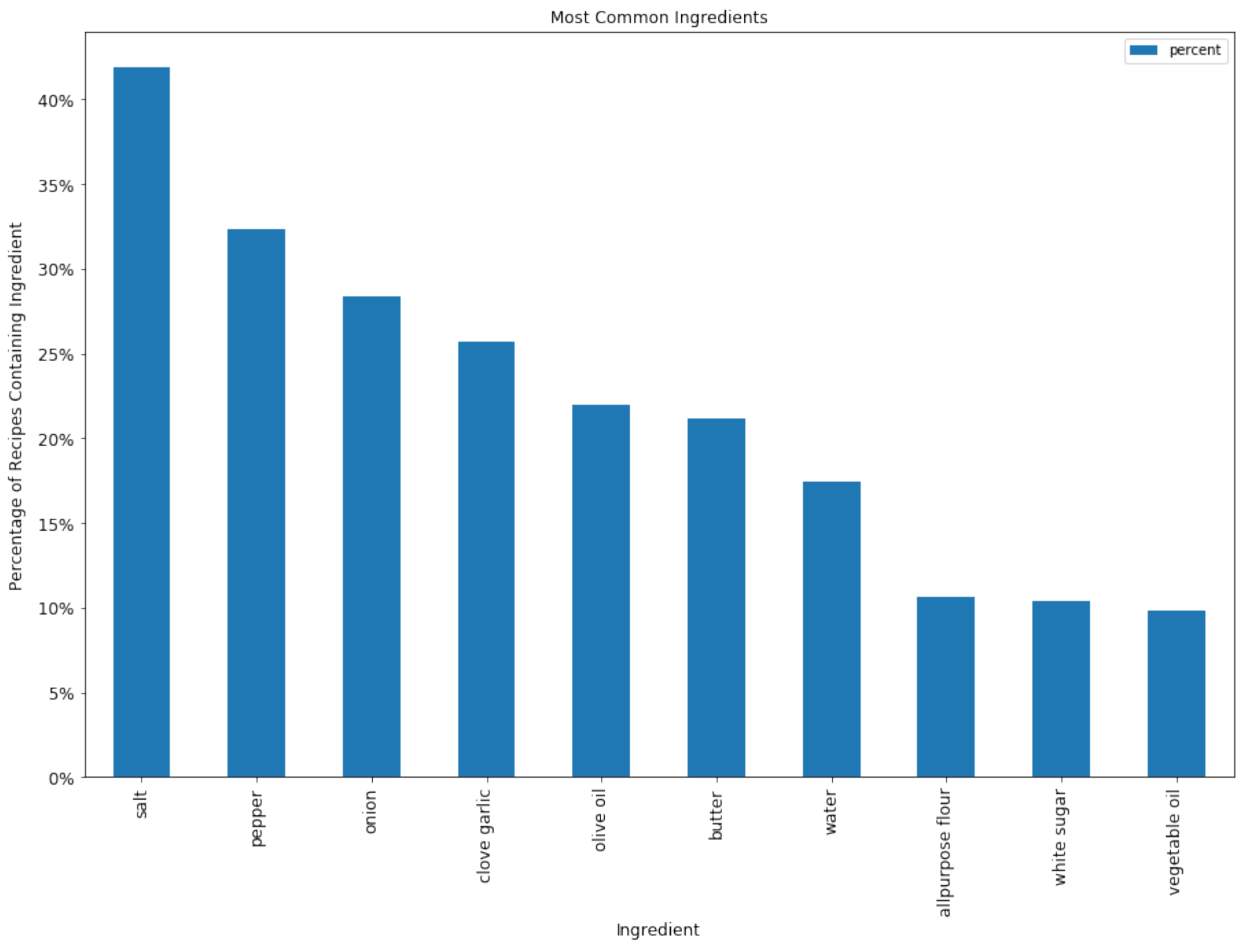
Ingredients for each recipe are stored as a list of individual ingredients. After separating all of the ingredients in all of the recipes into a single list, counting each ingredient, and then sorting by count we can calculate the percentage of times the ingredients occur in our recipe dataset. Shown below, are the top 10 most common ingredients in all of the recipes in the dataset. For example, “salt” occurs in more than 40% of the recipes, followed by “pepper” at 32.4% and “onion” at 28.4%.

Ingredient	Occurrences	Percent in Recipes
salt	4421	41.9%
pepper	3414	32.4%
onion	2993	28.4%

Ingredient	Occurrences	Percent in Recipes
clove garlic	2712	26.0%
olive oil	2316	22.0%
butter	2230	21.2%
water	1839	17.4%
allpurpose flour	1123	10.7%
white sugar	1092	10.4%
vegetable oil	1035	9.8%

The most frequently used ingredients, such as salt, pepper, and onion can be considered to be what chef's call "staples": ingredients that are so frequently used that it can be assumed that every cook has these items in their pantries. These ingredients do not distinguish recipes from one another and our model needs to treat these ingredients as less important because they are so common.

Shown below is a visualization of the top 10 most commonly used ingredients in the recipe dataset:



Popular Recipes Based on Ratings and Number of Reviews

Since ratings can be different based on the number of reviews, for example a recipe with only 3 ratings, where the average rating is 5.0, would be evaluated as having a truly higher rating than a recipe with 500 ratings, where the average rating is about 4.0. A weighted rating was used instead. Here's how it was calculated (see file: "recipeSimilar.ipynb"):

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 in the dataset 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. The ratings are 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 the highest rated recipes would be relevant to that person. This data could be used as a “cold start” for recommending the overall most popular recipes until we can capture user recipe preference data.

Algorithms and Techniques

Text classification is the supervised learning problem of categorizing text documents into one of a given set of classes. Applying this to recipes, we categorize recipes into one of a given set of categories.

Using scikit-learn libraries we several supervised learning algorithms were tried. Supervised learning algorithms require a category label for each recipe in the dataset. The dataset we are using already has a “Categories” column which has been assigned to every recipe.

The dataset was split into two parts: 80% training and 20% test. The data was trained using the training data and tested against the test data.

In order to perform machine learning on text documents, we first needed to turn the text content into numerical feature vectors.

The training method used in this project is based on a bag of words. The bag of words method is refined by modelling text documents as a vector space composed of an n dimensional real space with the real numbers from the term frequency - inverse document frequency(tf-idf) model. This weighting scheme weights the words in a document based on term frequency (ie, word count for the document) relative to the number of documents that contain the term (document frequency). The tf-df is computed with the formula

$$\text{tf-idf} = \text{tf} \times \log(N/\text{df})$$

where tf = term frequency, N = number of documents, and df = document frequency.

CountVectorizer was used to build a dictionary of features and transform documents into feature vectors. Each word was assigned an index value in the training corpus. Next, term frequency and tf-idf were calculated to transform the data into a tf-idf representation.

Naive Bayes (NB) and Support Vector Machine (SVM) classifiers were selected for this project. Starting with Multinomial NB method as a baseline, a Pipeline was built consisting of a CountVectorizer, followed by TfidTransformer and MultinomialNB.

Next, an SGDClassifier was used in the pipeline, replacing MultinomialNB. Finally, the best parameters for the pipeline were determined by using a Grid Search.

Latent Semantic Analysis (LSA) using sci-kit learn libraries was also explored in order to examine cosine similarities. LSA takes the tf-idf vectors and performs dimensionality reduction on them. This can be very useful for text searches and finding synonyms.

Word2Vec is a group of related models that are used to produce word embeddings. These 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. Word vectors are positioned in the vector space such that words that share common contexts are located near to one another in the space.

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. We will attempt to apply this technique to recipe ingredients, learning ingredients that commonly occur together.

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

There are two variants of the Word2Vec paradigm: skip-gram and CBOW. The skip-gram variant takes a target word and tries to predict the surrounding context words. The CBOW (continuous bag of words) variant takes a set of context words and tries to predict a target word. In this case, we will be considering the skip-gram variant.

We also need a way of ensuring 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. Therefore, we need a vector similarity score supplied to the output sigmoid layer – with similar vectors outputting a high score and un-similar vectors outputting a low score. As we did previously, we will use the cosine similarity score.

Benchmark

Several classifiers were trained to try to predict the category of a recipe: to predict the category given several submitted ingredients and to predict the category given a recipe title.

The following benchmarks were used to evaluate the classifiers:

- Accuracy: Represents the difference between the actual and predicted data.
- Precision: A precision representing the number of recipes correctly classified divided by the number retrieved
- Recall: Represent the number of recipes correctly classified divided by the total number of items of that label.
- F1 score: a composite measure of both precision and recall

Using the recipe master dataset, we generate a training and testing datasets. The “categories” column is used as a benchmark to determine if the results generated by the TF-IDF and CountVectorizer are in the same categories as the submitted recipe. It is a multilabel classification: find the categories for a recipe. The threshold to determine if the recommendation is successful is to determine if the model predicts the correct categories for a given recipe title or ingredients.

The fundamental benchmark for the word2vec model is reasonable agreement with the judgment of a human being.

Methodology

Data Preprocessing

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. The data was cleaned and irrelevant terms were removed. Terms which were removed included measurement words such as tablespoons, pinch, ounces, finely, scrubbed and any numeric term, such as 1/4, 1/2. Stop words such as “as”, “and”, “the” and “or” and punctuation were also removed. Every term was then converted to lowercase.

Just like the ingredients data, non-ascii characters were removed from the remaining columns, too. All terms were converted to lowercase and separated by spaces.

Implementation

In order to use machine learning on text documents, it is necessary to translate the text into something computers can understand and process well: numerical feature vectors. The recipes dataset was converted into a list of unique integers and then the integers were converted into vectors. Bags of words are usually high-dimensional sparse dataset, filled with mostly zeros. Next, the dimensionality of the dataset was reduced.

CountVectorizer

The sklearn.feature_extraction.text.CountVectorizer class was used to build a dictionary of features and transform recipes to feature vectors. It converts the recipes to a matrix of token counts and removes stop words. CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, and also to encode new documents using that vocabulary. CountVectorizer returns an encoded vector with a length of the entire vocabulary and an integer count for the number of times each word appears in the dataset. It does not down-weight the presence of an ingredient if it appears in relatively more recipes. The index value of a word in the vocabulary is linked to its frequency in the training dataset.

Term Frequency-Inverse Document Frequency (TF-IDF)

The feature vectors were created using the cleaned dataset by computing a Term Frequency-Inverse Document Frequency (TF-IDF) matrix. Calculating TF-IDF indicates the importance of each food term to the recipe it belongs to. 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. As an example, a term like “salt” which appears in more than 40% of the recipes will be considered less important because it is very common.

At this point, we could use the TFIDF matrix to calculate a similarity score by calculating the cosine distance between the high TF-IDF words as a vector. Cosine similarity is a calculated 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 ingredient. The vector has 1 if the ingredient 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 ingredients, and the lengths of the vectors are the square roots of the numbers of ingredients 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.

To test the vectors, I wrote a function (see file: "recipeSimilar.ipynb") that takes in a recipe title and cosine similarity as inputs and outputs a list of up to the 10 most similar recipes. A reverse mapping of recipe titles and dataframe indices identifies the index of a recipe given its title. The following is an example of the resulting recipes which the function reported as being similar to the submitted recipe. Based on human judgment, the recommended recipes were indeed related to the submitted recipe. Latent Semantic Analysis (LSA) was also tried to generate cosine similarities among recipe terms.

Example: Recipes which are similar to "Steamed Asian Sesame Veggies"

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

Given the recipe title "Steamed Asian Sesame Veggies", the results returned are indeed related to Vegetables, Asian and Sesame, as evaluated by human judgment.

Here are the recipes returned when running the same test recipe that we ran previously:

Example: Recipes which are similar to “Steamed Asian Sesame Veggies”

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

Training the Models

The first classifier used was Multinomial Naive Bayes, as implemented by the sklearn MultinomialNB class as this classifier is more suitable for word counts. MultinomialNB gives a baseline. Next, SGDClassifier was tried. The goal of this implementation was to be able to predict Categories based on a recipe title and/or ingredients.

The word2vec model used was the gensim library implementation using nltk punks and stopwords. The goal of this implementation was to calculate similarities between ingredients and recipe titles in order to find recipes that are similar to a submitted ingredient or title.

Refinement

The data for the categories field had already been labelled by humans. The features used include: title, categories and ingredients. The data was split into two subsets: training and test, where 80% of the original dataset was used for training the models and 20% was used to test.

Term Frequencies (tf) are calculated by dividing the number of occurrences of each word by the total number of words. The weights of words which occur in many recipes are downscaled. TfidTransformer was used to fit the estimator to the data and then transform the count-matrix to tf-idf representation.

The first classifier used was Multinomial Naive Bayes, as implemented by the sklearn MultinomialNB class as this classifier is more suitable for word counts. MultinomialNB gives a baseline. A pipeline was built to make the vectorizer to transformer to classifier easier to work with. It included: CountVectorizer, TfidTransformer, and MultinomialNB.

The next attempt was to use a linear support vector machine, SGDClassifier, which is regarded as one of the best text classification algorithms.[10] In the pipeline, multinomialNB was replaced by SGDClassifier. It included: CountVectorizer, TfidTransformer, and SGDClassifier.

Results

Model Evaluation and Validation

Once the classifiers had been fit and trained to try to predict the categories of a recipe, the next step was to evaluate the predictive accuracy of the models using the test dataset.

Results for the MultinomialNB classifier were low with a mean test score of 21.1%.

A pipeline was then fit with training data and features. The Pipeline included: CountVectorizer, TfidTransformer, MultinomialNB:

This time the predicted mean test score was slightly higher at 21.8%

The next attempt was to use a linear support vector machine, SGDClassifier.

Pipeline: CountVectorizer, TfidTransformer, SGDClassifier:

With SGDClassifier, the test score was much better at 68.5% accuracy.

Precision Score: 54.5%

Recall: 48.4%

Finally, the parameters were tuned using Grid Search in order to find the best parameters for the classifier. The best parameters identified were: clf__alpha = 0.001, tfidf__use_idf = True, vect__ngram_range = (1, 2).

After fitting the tuned classifiers, the best score was 66%, slightly lower than before using grid search.

LSA Model Sample predictions:

title_clean	nanas beef stroganoff	split pea and ham soup ii	eggplant supper soup	turkey tetrazzini	black bean soup iii	sweet and sour pork tenderloin	authentic huevos rancheros	fresh corn and zucchini saute	ham casserole	fajita chili with knorr rice sides	...	mango black bean salad	savory slow cooker squash and apple dish
categories_clean													
world_cuisine european eastern_european russian	1.000000	0.749201	0.793250	0.955709	0.428061	0.980575	0.898610	0.709949	0.990986	0.560566	...	0.246117	0.999561
soupsstews_and_chili soup vegetable_soup	0.749201	1.000000	0.997601	0.521081	0.919296	0.864562	0.382637	0.998351	0.653714	0.968469	...	0.826360	0.768505
soupsstews_and_chili soup vegetable_soup	0.793250	0.997601	1.000000	0.578910	0.889849	0.897272	0.445670	0.991984	0.704527	0.948902	...	0.785398	0.810951
world_cuisine european italian	0.955709	0.521081	0.578910	1.000000	0.143116	0.879417	0.987939	0.471233	0.986522	0.292014	...	-0.050045	0.946565
soupsstews_and_chili soup cold_soups	0.428061	0.919296	0.889849	0.143116	1.000000	0.597011	-0.011859	0.940370	0.303129	0.988360	...	0.981304	0.454663
world_cuisine asian chinese	0.980575	0.864562	0.897272	0.879417	0.597011	1.000000	0.795097	0.834293	0.945459	0.712106	...	0.431446	0.985959
breakfast_and_brunch meat_and_seafood bacon	0.898610	0.382637	0.445670	0.987939	-0.011859	0.795097	1.000000	0.328977	0.949288	0.140398	...	-0.204090	0.885209
side_dish vegetables squash zucchini	0.709949	0.998351	0.991984	0.471233	0.940370	0.834293	0.328977	1.000000	0.609202	0.981172	...	0.857321	0.730513
main_dish pork ham	0.990986	0.653714	0.704527	0.986522	0.303129	0.945459	0.949288	0.609202	1.000000	0.444573	...	0.114050	0.986579

The word2vec model did a good job of finding similar ingredients, as evaluated by human judgment. Some examples are:

"groundbeef":

extraleangroundbeef, 0.7875096201896667
groundsausag', 0.7234644889831543
condensedtomatosoup, 0.7140675783157349
groundbeefchuck, 0.6994967460632324
stuffedmaindishe', 0.6985939741134644
dryzitipasta, 0.6852942705154419
groundround, 0.6849061846733093
meatloaf, 0.683943510055542
porksausage, 0.6809939742088318
extragroundbeef, 0.6803674697875977

"cookies":

semisweetchocolatechips', 0.9692796468734741
desserts', 0.9668233394622803

```
brownies', 0.9458422064781189
chocolatebrownies', 0.9346032738685608
fruitdesserts', 0.9282387495040894
oatmealcookies', 0.9265550374984741
dropcookies', 0.9248393177986145
whitechocolate', 0.9169297218322754
bakecookies', 0.915890634059906
peanutbuttercookies', 0.9148941040039062
```

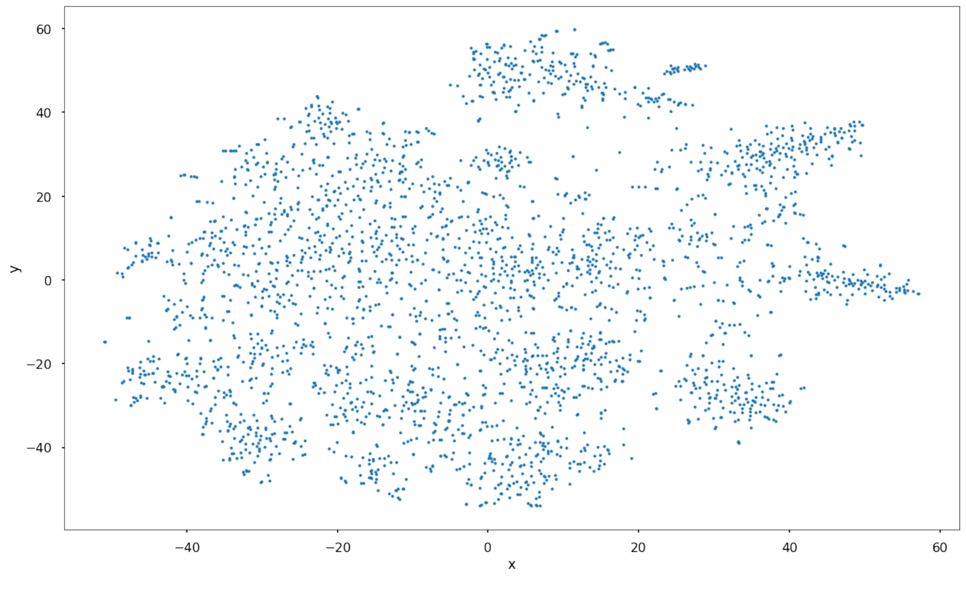
Justification

The SGDClassifier had the best results at 68.5% accuracy, 54.5% precision and 48.4% recall. Our data consists of recipes. It is not a document in which there are sentences with meaningful thoughts and intentions. A more suitable algorithm for this dataset would be Singular Value Decomposition and Latent Semantic Analysis (LSA). In LSA, each component is a linear combination of words with a reduced number of dimensions. Similar recipes point in similar directions and dissimilar recipes have perpendicular (orthogonal) vectors. [3]

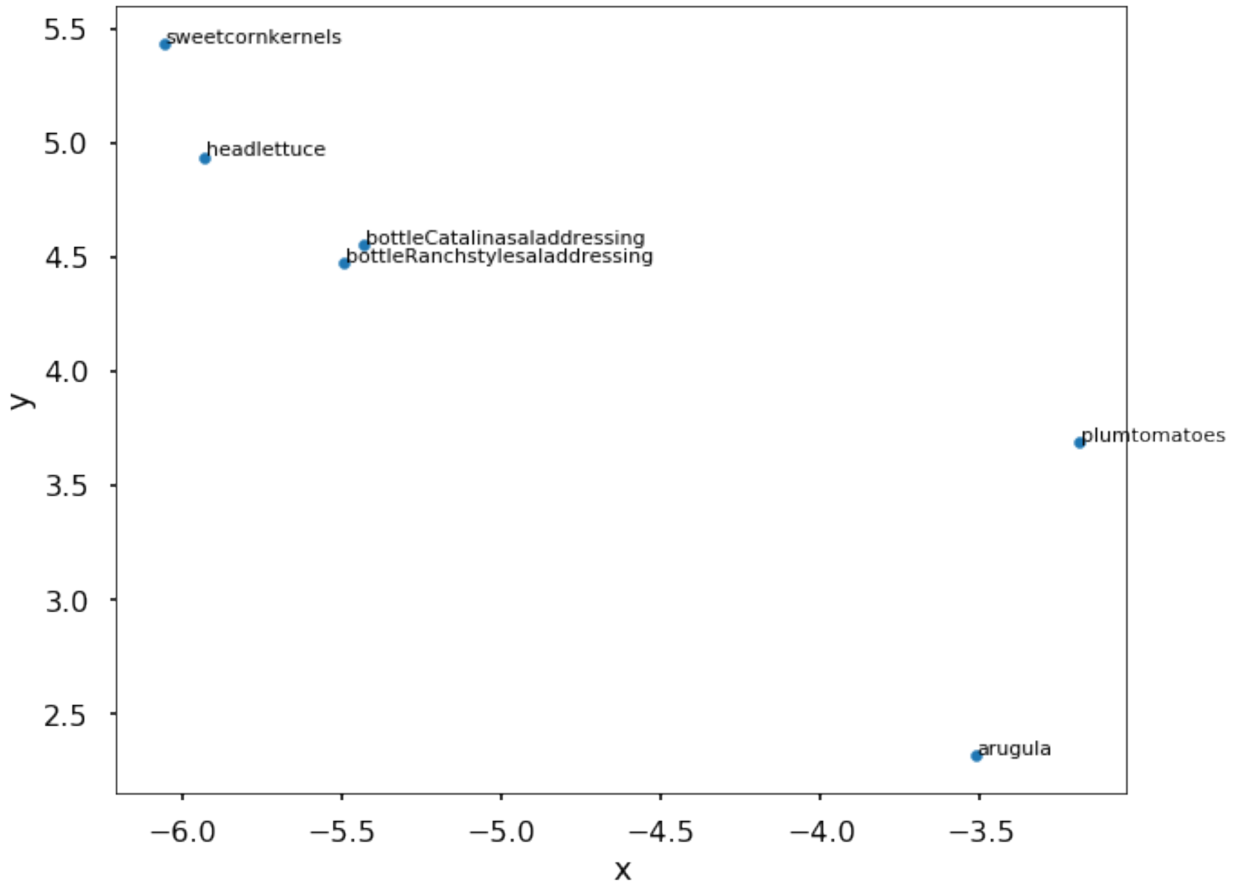
We can use cosine similarity for searching for similar ingredients and titles by determining which recipe has the smallest angle with a given search term. Shown above is a matrix of cosine similarities for categories by titles. As an example, the recipe titled ‘black bean soup iii’ has a cosine similarity of 1.0 with the categories ‘soups, stews and chili’.

Conclusion

Visualization



Shown above is a scatterplot of the word2vec word embeddings. Words which are deemed to be similar are located near each other. Shown below is a zoomed in portion of that same scatterplot. The sample shows salad ingredients such as head of lettuce, salad dressing, plum tomatoes and arugula located near each other.



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. A major obstacle was choosing a dataset that was not a typical body of text with meaningful sentences.

The models used in this project did not score well enough to be used in a production application. However, I found that using a cosine similarity function to call the tf-idf vectors resulted in better matches when looking for similar recipe ingredients and titles. The nature of this dataset is not what the machine learning algorithms which were used to process the text were optimized for.

Using TF-IDF, I 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", the tf-idf vectors found recipes that are similar to lasagne but don't necessarily contain the word "lasagne", for example: "Meat Filled Manicotti", "Baked Spaghetti Casserole", "Manicotti" and "Italian Veggie Rolls”. One of the main objectives of this project was to be able to quickly find recipes given ingredients that the home cook has on hand.

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 include:

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".

Improvement

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.

Another good option which is similar to word2vec might be to try the Global Vectors for Word Representation (GloVe). GloVe is an unsupervised learning algorithm for getting vector representations of words.

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.

Having more data would likely yield even better recommendations.

References

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

[2] Linguistic Regularities in Continuous Space Word Representations, https://www.aclweb.org/anthology/N13-1090

[3] Latent Semantic Analysis (LSA) for Text Classification Tutorial http://mccormickml.com/2016/03/25/lsa-for-text-classification-tutorial/

[4] Working With Text Data, http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

[5] Google's Recipe schema, http://schema.org/Recipe

[6] GloVe: Global Vectors for Word Representation, https://nlp.stanford.edu/projects/glove/

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

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

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

[10] 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

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

[12] t-SNE https://lvdmaaten.github.io/tsne/