Machine Learning Engineer Nanodegree

Capstone Proposal

Claudia Cassidy
March 7, 2018

Proposal: Recipe Recommender System

Domain Background

"What's for dinner?". That is the dreaded question at the end of a work day when families get together after school and work and it is time to prepare dinner. Unless you are well organized, plan meals a week in advance, shop for all the ingredients required, and have readily available recipes, it can be very challenging to prepare a family dinner that is both delicious and nutritious.

You can search through cookbooks and thousands of recipes online, but it takes time to find just the right recipe that can be made quickly, that everyone will like and that you have all the ingredients for. Many cooks repeat the same tried and true recipes each week. It can get boring after a while.

Alternatively, you can order pizza, buy fast food, prepare frozen dinners or make mixes from a box. Food delivery services like Blue Apron have tried to solve this problem by sending a box of ingredients and directions for cooking them. They can be expensive and limited in portion sizes.

When shopping at the grocery store, most people I see do not carry a shopping list. They buy main ingredients, such as meats and vegetables, and stock up on staples like onions, garlic and spices. Sometimes, ingredients go unused and have to be discarded.

Supermarkets often feature a meal of the day and showcase all the ingredients needed to make that meal along with a demo and recipe card. The meals are general, not personalized. From what I've seen, not many people buy the featured meal items.

This proposal is to build a Recommender System which will parse the recipe information, including ingredients, title, description, top reviews, review count, ratings and instructions of recipes in order to make the best predictions about which recipes to recommend for individual users.

Machine learning methods have become a powerful tool to process Recommender System tasks such as news, music, and movies. When there are thousands of choices, Recommender Systems can help people deal with choice overload. Choice overload is a cognitive process in which people have a difficult time making a decision when faced with many options. With thousands of recipes to choose from online, it can be overwhelming to find the best one for you.

Recommender Systems strive to give people personalized recommendations. For example, movies on Netflix and products on Amazon are recommended based on algorithms which predict what a person will probably like based on user preferences, item features and items a user has looked at in the past. Recommender Systems guide users in a personalized way to discover products or services they might be interested in from a large space of possible options.

Recommender systems were first introduced in 1992 with Tapestry, a recommender system that used collaborative filtering [2]. Recommender systems generally fall into three categories: collaborative, content based and hybrid. The collaborative approach may access user profile data such as age and previous purchase history to find what other people like you, like. A content-based approach bases recommendation on the item data, for example, if a person is shopping for a television, then other televisions with similar features to the one the user looked at will be presented. The hybrid approach uses a combination of both user profile and item data.

Netflix, a very popular online movie streaming service, lets people watch movies and tv shows for a monthly subscription fee. In 2006, Netflix released a dataset containing 100 milion movie ratings and held a challenge for anyone to come up with an algorithm that could beat the accuracy of its recommenation system, Cinematch. The winning algorithm was an ensemble approach that combined several results from different algorithms into a large knowledgebase.

Most recently, in 2017 IBM developed "Foodie Fooderson - A Conversational Agent for the Smart Kitchen" [1]. The algorithm tries to recommend more nutritious recipes while using ingredients known to be in the user's refrigeratior. Foodie Fooderson emphasizes recommending recipes based on their nutritional value. It also has a voice interface in order to be hands-free while working in the kitchen.

While our dataset does include nutritional information, unlike Foodie Fooderson, our algorithm will not emphasize nutritional value but will instead match recipes that the user will most probably like, nutritious or not.

Problem Statement

There are two goals for this project:

1 - Make it easier for home cooks to find recipes at the end of the day, spur of the moment, which use the ingredients already at home.

2 - If an item is on sale at the store, for example, if there's a sale on lamb, then recommend recipes for that person which contain that item, so that the person can buy the featured item and any extra ingredients, if needed, to make the recipe at home.

The goal is to build a tool which will make it easier and less expensive for home cooks to find recipes that they will most probably like given the ingredients they have on hand, the amount of time required to prepare the recipe and the similarity to other recipes that they and/or people like them have rated highly in the past and/or saved to favorites list.

This project will try several algorithms including k-Nearest Neighbors, Collaborative Filtering and Word2Vec in order to find the best suited to build a Recommender System for recipes. Once trained, the algorithms should be able to recommend relevant, personalized recipes for each user. The front end will be an iOS app.

Datasets and Inputs

The data will consist of a csv file of recipes scraped from popular cooking websites such as AllRecipes.com. Using BeautifulSoup, we can scrape a sample of at least 1,000 publicly listed recipes.

User Preference Training Data:

The front end will be an iOS app which requires a user to login. The app will allow people to save recipes. By requiring a login we can track search terms submitted by the user to the api in addition to recipes the user has saved. The more the person users the app, the better the user preference data. In summary:

- The recipe predictions will improve if the user has an account on the iOS app and a saved list of favorite recipes. For training purposes, we will create a dataset of user preferences which consists of a list of 1-15 "saved" recipes per user.
- If the user has a low number of saved recipes then we will weigh the features "Ratings", "Number of Reviews" and "Review" contents fields more heavily as they reflect the general population's preferences.
- The iOS app can also have an onboarding processes to ask users what their cuisine preferences are, for example "vegetarian", "low calorie", "italian", "5 ingredients or less".
- The app will have search functionality so we can also save the user's search queries to enhance the

recommendations.
The User Preference dataset will contain userId, saved recipe list, cuisine preferences, and search words entered.

When training the data, we will rely on the fields "Ratings" (1-5), "Number of Reviews", and "Review". Number of Reviews tells us how many times people reported making the recipe. Ratings tell us how much people liked the recipe, with 5 being the highest and 1 the lowest rating. Ratings and Number of Reviews indicate general popularity. There are up to six Reviews for each recipe, which consist of a paragraph of feedback from people who felt strongly enough about a recipe to post their thoughts. "Title" is the name of the recipe and "Description" is a sentence describing the recipe as entered by the recipe's author.

Item-Item distance will be calculated to see which ingredients most often go together, for example meatballs and spaghetti, peanut butter and jelly, chocolate and cake.

Recipe structured data

The major recipe websites follow Google's schema for recipes. This consistent structure will make it easier to import the data in a pre-structured format. https://developers.google.com/search/docs/data-types/recipe (https://developers.google.com/search/docs/data-types/recipe).

The sample recipe dataset includes the following fields:

- name: title of the recipe
- author: the person who wrote the recipe
- description: a one sentence summary of the author's description of the recipe
- ratingValue: 1-5 where 5 is highest rating of how much a person liked the recipe
- reviewCount: number of people who have made this recipe and given it a rating
- reviewBody: text paragraph of people's reviews of the recipe, top 6
- prepTime: amount of time to prepare the ingredients before cooking
- totalTime: total of preparation + cookTime
- recipeYield: how many servings the recipe makes
- nutrition: amount of fat, carbohydrate, calories, cholesterol, servingSize
- recipeIngredients: a list of ingredients
- recipeInstructions: a list of instructions for preparing the recipe
- urlLink: link to recipe on original source website

- image link: link to a photo of the prepared recipe on the original source website

I collected about 1,000 recipes via BeautifulSoup from AllRecipes.com. Here is a link to a subset of the training data containing 284 recipes: https://github.com/cassfinn/Deep-NLP-Recipe-Recommender/blob/master/dataAReveryDay.csv (https://github.com/cassfinn/Deep-NLP-Recipe-Recommender/blob/master/dataAReveryDay.csv)

Solution Statement

As the app learns the user's preferences it will predict more accurate "Recommended for You" recipes.

We will consider algorithms that are designed to work with content data, including TF-IDF, Word2Vec and LDA (Latent Dirichlet Allocation).

There are at least three kinds of data: content, user and interaction data. Different algorithms are suitable for each. We can start by using TF-IDF vectorization to assign a signature to each recipe. This will map every recipe to a multidimensional mathematical space, or vector. On the basis of those vectors we can use a k-nearest neighbors algorithm. We can also incorporate ratings and number of reviews.

Google's Word2Vec algorithm focuses on the meaning of words. It is a neural network implementation that leanrs distributed representations for words. It does not need labels in order to create meaningful representations. Given enough training data, words with similar meanings appear in clusters. Distributed word vectors can be used for word prediction.

Another approach to consider is algorithms based on topic modelling and building a probabilistic language model such as Latent Dirichlet allocation (LDA).

Collaborative filtering in Recommender Systems assumes that people who share an interest in certain things will probably have similar tastes in other things as well. Item-item collaborative filtering draws inferences about the relationship between different items based on which items go together. The more often two items (say, peanut butter and jelly) appear in the same recipe, the closer they are to one another. So, when someone searches for "peanut butter", the algorithm should find other ingredients that are most often in recipes with peanut butter, like "jelly" or "white bread", over things that aren’t, like bologna.

User-item filtering takes a different approach. Instead of calculating the distance between items, we calculate the distance between users based on their ratings (or likes). When coming up with recommendations for a particular user, we look at the users that are closest to them and then suggest items those users also liked but that our user hasn’t interacted with yet. So, if you’ve saved a certain number of recipes the app can look at other users who saved those same recipes and recommend one that they also saved but which you might not have seen yet.

The front end of the project will be an iOS app. We will try several algorithms, tune the parameters and then select the one or a combination that gives the most accurate predictions.

The app will prompt the user to enter at least one ingredient. The model will use the user's list of saved recipes, if it exists, and the ingredient(s) submitted as input data. It will also consider context such as the user's time of day. For example, if it is a week day after 5pm according to the user's device clock, then preference will be given to recipes that are simpler, faster to make, and that have a minimal number of ingredients. The recommender system will return 5-10 recommended recipes.

The user will see a result list of recommended recipe titles. Selecting a recipe will open the recipe details on the host recipe's website (e.g. AllRecipes.com). If the user chooses to Save that recipe, then the recipe data is added to the user's saved recipe list.

For example, if a user saves recipes for "beef stew", "lasagne", "macaroni and cheese", and "chili", then the app can recommend recipes that are similar to those based on cuisine and ingredients in those recipes. If a different user saves recipes for "vegetarian chili", "banana smoothie", "gluten free pizza", then different recipes might be recommended for that user. Given that there are hundreds of recipes for something like "lasagne", the app should find the top 10 that are most similar to recipes the individual user has previously saved, for example vegetable lasagne vs. three meat lasagne.

Example 2: If there is no user preference data, then the search words submitted will be used to filter recipes that have those ingredients along with the highest number of ratings, review count, and recipe reviews content.

Benchmark Model

A benchmark would be to see if the app predicts recipes that are similar. The recommended recipes should make sense given the main ingredient(s) that the user requests. If the user submits "chicken", then the main ingredient in the recipe title and ingredients list of the recipe should contain "chicken".

Given a user dataset containing 10 saved recipes per user, the app should be smart enough to detect if a user prefers to cook more recipes where the ingredients are vegetables vs. meats or if the user prefers recipes that have Italian cuisine. If a user has saved mostly "stir-fry" recipes, then the model should predict "Chicken Stir-Fry" with a higher probability than "Chicken Stew".

k-Nearest Neighbors

k-Nearest Neighbors is a machine learning algorithm which can find clusters of similar users based on common recipe ratings and then make predictions using the average rating of the top-k nearest neighbors. For example, if we have a grid where each row is a recipe and each column was a user, we can find the k items that have the most similar user engagement vectors.[3]

To ensure statistical significance, we will only be training on recipes that have the highest number of reviews. This will filter out the lesser reviewed recipes. We are assuming that the higher the number of reviews, no matter what the rating, the more times people have made the recipe.

The kNN algorithm measures distance to determine the “closeness” of instances. It then classifies an instance by finding its nearest neighbors, and picks the most popular class among the neighbors.

Collaborative Filtering Using Matrix Factorization

Matrix Factorization is a mathematical tool for playing around with matrices. The Matrix Factorization techniques are usually more effective, because they discover the hidden features underlying the interactions between users and items(recipes). We will use singular value decomposition (SVD)—one of the Matrix Factorization models for identifying latent factors.

Word2Vec

Word2Vec is a class of neural network models that are useful for Natural Language Processing tasks. The neural network takes in a large corpus of text, analyzes it, and for each word in the vocabulary, generates a vector of numbers that represent that word. Those vectors of numbers encode information about the meaning of the word in relation to the context in which it appears. The Word2vec Skip-gram model is a shallow neural network with a single hidden layer that takes in a word as input and tries to predict the context of words around it as output. This will be especially useful for processing recipe reviews as they are composed of paragraphs of text, recipe titles, descriptions and even ingredient lists.

What we expect to get at the end of the training phase is a model where each recipe is represented by a vector of weights in a high dimensional space. Similar recipes will have weights that are closer together than recipes that are unrelated.

Benchmark

We will compare the results of the three algorithms and see how well they find recipes that include the specified ingredients and are similar to the user preference recipes. For example, if a sample user's saved recipes contain no beef, but do contain vegetables, then the algorithms should return recipes that contain mostly vegetables.

Evaluation Metrics

Given that a user's saved recipes list contains a sufficient amount of saved recipes and that there are hundreds of recipes to choose from, if a user searches for an ingredient, for example, "ground beef", then the algorithm should find recipes where ground beef is the primary ingredient and the other ingredients and cuisine are similar to recipes the user has previously saved.

The app should let the user enter more than one ingredient, for example "ground beef" and "tomatoes" (assuming the user is at home and that's what's in the pantry). The app should recommend 5 to 10 recipes which use ground beef and tomatoes. Those recommended recipes should be similar to the user's already saved recipes. For example, if the user has 15 recipes saved and 8 of those are casseroles, then the recommender should predict that casserole recipes have a higher preference (weight) for this user then hamburger recipes.

Evaluating Results:

Recipes in each cluster should be similar. Since we will not have a list of user saved recipes, we will test the model by submitting a search, for example "beef" and "stew". The recipe results should return beef stew recipes which are the most popular and have the most number of reviews.

The recipes will be evaluated by prediction accuracy. They should use the same main ingredient or a similar format. For example, if we submit a search for the ingredients "ground beef and rice" then recipes that contain both ingredients, for example "Spanish Rice", should likely be returned. The recipes should contain both ingredients and should also have considered the highest number of reviews and highest ratings. A high number of reviews plus highest ratings means that many people have made the recipe and liked it - it's "tried and true".

Project Design

In a nutshell:

Kid to Mom: "What's for dinner?"
Mom: "I don't know yet, give me 15 seconds to find a recipe".

Mom opens the iOS app and types (because it's faster to type than to take a picture with the phone and wait for image recognition):

```
Primary ingredient:   _ground beef__
Secondary ingredient: _tomatoes__
```

NLP algorithm submits Mom's previously saved recipes along with search ingredient terms. The database of recipes has been trained using the word2vec model to learn vector representations of the words in the ingredients lists.

Of the hundreds of recipes that contain both ground beef and tomatoes, the model predicts which of those recipes have a high probability that Mom will like them. The recipes with the highest probability of Mom liking them are returned to the app and presented to Mom.

When analyzing recipes, the model will have to learn the nature of each word. Is it a measurement (tablespoon), an ingredient (chicken), an action word (stir) or a descriptive word (slowly).

There are two main challenges:
1 - polysemy: words that have several meanings
2 - synonymy: different words that have similar meanings

Given the words "red pepper" in an ingredient, what are the nearby words? When we see the word teaspoon, we can determine the kind of pepper is a spice, when we see "chopped" as part of the ingredient, it is a vegetable. The co-occurrence of words can be used to remove ambiguity.

Training the NLP model:

The model will have to learn which words are measurements and which are food items in the ingredients list. For example:
Recipe for Meat Loaf:
1/2 cup packed brown sugar
1/2 cup ketchup
1 1/2 pounds lean ground beef
3/4 cup milk
2 eggs
1 1/2 teaspoons salt
1/4 teaspoon ground black pepper
1 small onion chopped
1/4 teaspoon ground ginger
3/4 cup finely crushed saltine cracker crumbs

The model will have to learn that black "pepper" is a spice, while red "pepper" can be a vegetable or a spice. Measurement terms should not be considered as ingredients but they can affect the meaning of the ingredient. Spices, such as salt, pepper and ginger, are considered staples and can also be ignored.

References:

[1] The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review https://arxiv.org/pdf/1511.05263.pdf (https://arxiv.org/pdf/1511.05263.pdf)

[2] IBM Watson's Foodie Fooderson: https://migueljimenez.co/academia/2017/Angara-CASCON-2017.pdf (https://migueljimenez.co/academia/2017/Angara-CASCON-2017.pdf)

[3] Collaborative Filtering Using k-Nearset Neighbors (kNN): https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c (https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c)

Bag of Words Meets Bags of Popcorn: https://www.kaggle.com/c/word2vec-nlp-tutorial (https://www.kaggle.com/c/word2vec-nlp-tutorial)

Vector Representations of Words: https://www.tensorflow.org/tutorials/word2vec (https://www.tensorflow.org/tutorials/word2vec)