12-2019

Hybrid Recommender Systems via Spectral Learning and a Random Forest

Alyssa Williams
East Tennessee State University

Hybrid Recommender Systems via Spectral Learning and a Random Forest

---

A thesis

presented to

the faculty of the Department of Mathematics

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Mathematical Sciences

---

by

Alyssa Williams

December 2019

Jeff Knisley, Ph.D., Chair

Bob Gardner, Ph.D.

Michele Joyner, Ph.D.

Keywords: similarity learning, collaborative filtering, nearest neighbor

ABSTRACT

Constructing Recursive Recommender Systems with Similarity Learning

by

Alyssa Williams

We demonstrate spectral learning can be combined with a random forest classifier to produce a hybrid recommender system capable of incorporating meta information. Spectral learning is supervised learning in which data is in the form of one or more networks. Responses are predicted from features obtained from the eigenvector decomposition of matrix representations of the networks. Spectral learning is based on the highest weight eigenvectors of natural Markov chain representations. A random forest is an ensemble technique for supervised learning whose internal predictive model can be interpreted as a nearest neighbor network. A hybrid recommender can be constructed by first deriving a network model from a recommender's similarity matrix then applying spectral learning techniques to produce a new network model. The response learned by the new version of the recommender can be meta information. This leads to a system capable of incorporating meta data into recommendations.

TABLE OF CONTENTS

1 RECOMMENDER SYSTEMS AND COLLABORATIVE FILTERING

Recommender systems are algorithms that recommend items or products to users, and often, data from similar users are used in making such recommendations. Recommenders based on data from similar users are known as collaborative filters. We think of a collaborative filtering as a type of recommender system in which recommendations are based on experiences of other users. This is the type utilized and discussed throughout this thesis. There are recommender systems based solely on a single user's characteristics, called user preference models or content-based models [12]. We are developing recommenders based exclusively (at least initially) on user collaborations. In this thesis the goal is to incorporate user based meta-data to create a hybrid recommender system. Thus, the hybrid approach developed in this thesis is not a true collaborative filter type of recommender system, but is instead an extension of a collaborative filter that incorporates user meta-data.

The use of collaborative filters to create recommender systems is quite popular. In a world where access to data is instantaneous, collaborative filtering can use previously collected data to make personalized recommendations quickly [13]. The amount of information readily available today is increasing and changing constantly. In turn, we need tools that are equally as fast and efficient to process the requested data. Not only is a technique needed that is fast and efficient, but the ability to customize the process for unique results is crucial. A tool that meets all the qualifications is a recommender system [13].

The versatility and customization of recommender systems are just a few of the many reasons they are so desirable. Depending on what data or discipline is being

explored, recommender systems can be created with different methods. Generally, recommender systems can be broken down into two main categories: content-based and collaborative filtering based approaches [13]. We can extend the realm of recommender systems to three main categories: collaborative filtering, content-based filtering, and hybrid recommender systems. One of the well known and ever growing applications of collaborative filtering based recommender systems is social networking. Social networking sites such as Facebook or Instagram allow users to create communities, interact with other individuals and share content [4]. How does Facebook recommend new friends? Or how do users find relevant groups to join on the site? The answer is collaborative filtering. In [4] the use of hybrid collaborative filtering to offer personal recommendations based on a user's profile and prior information patterns is explored in a social network setting. The method [4] views a community as both users and words to describe the different groups or communities. The combination of these two methods yields more accurate personalized results for the users.

To best define similarities and make recommendations for users there are two main approaches: the neighborhood approach and latent factor models. The two methods are discussed in great detail in [7]. Latent factor models transform items and users to the same latent factor space [7].This makes the two different objects (items and users) directly comparable. The latent space characterizes both products and users on factors established from user review [7]. An example of a latent factor model is Singular Value Decomposition. An application of Singular Value Decomposition is discussed in chapter three of this thesis. The example [7] mentions an example for

latent factor modeling. Classifying movies based on genre characteristics is a latent factor model. Neighborhood methods calculate similarities between either items or users. An item based approach evaluates the preference of a user to an item (movies for example) based on ratings of similar items by that specific user (ratings of movies or genres) [7].

1.1 Hybrid Recommender Systems and Issues with Traditional Collaborative Filtering Approaches

Two main types of algorithms are used to make recommendations to users: neighbor models and latent factor models. However, to begin we will discuss research conducted on other hybrid recommender system approaches. We investigate if user-based and item-based approaches always exist independently of each other, or if they can be combined. We also looked at various ways of creating hybrid recommender systems.

Hybrid recommender systems incorporate content and collaborative filtering (or other) processes. These hybrid systems can utilize the different filtering steps in many ways: implementing content filtering before collaborative filtering, the converse, or any combination of these approaches. Another type of hybrid recommender system can not only incorporate the base data set, but also the underlying data as well.

As outlined throughout this chapter, arguably the most crucial step in the collaborative filtering process is establishing similarities. If user-based similarities are utilized where products are recommended to users based on past history, what happens for new users who have no transaction history? Traditionally a vector similarity

measure, discussed in chapter two of this thesis, would be used. The problem encountered above is what [1] describes as a cold-start problem. In [1] a heuristic similarity measure based on minute meanings of co-ratings is researched. The new similarity measure using ratings data in the context of product recommendation. This can assist in cold-start situations [1]. Combining similarity measures like [1] suggests, is one of the many approaches to a hybrid recommender system. Hybrid systems are a great way to overcome any limitations from using a singular collaborative filtering method. For example, if we have information about a new user such as age, gender, location, etc., then such information can be used to address the cold start problem. Such information is called user meta-data, and one of the main goals for our method is the incorporation of meta-data into the recommender.

As [12] mentions, the goal of collaborative filtering is to predict a user's interest in a specific item based on a collection of similar users. With the goal by [12] in mind, we can think of collaborative filtering as a giant network housing information such as users, items, rankings or ratings and background information. This information from existing users and relationships within the network can be applied to new users or new items. Items can be recommended to new users based on historical data from other users who have similar preferences, creating a recommender system. The new user is recommended items that are liked by similar neighbors. Conversely, new items can be suggested to existing users based on user profiles. This type of collaborative filtering based recommender system is created on the basis of user recommendations and is what we utilize in the first few steps of our research algorithm outlined later in the thesis. In [12] this is described as a memory-based approach that can be further

divided into user-based approaches or item-based filters.

However, as [12] thoroughly discusses in both the item-based and user-based approaches only some of the information in the user-item matrix or network is used to predict similarities between items or users. Therefore, the recommendations produced by the collaborative filtering does not use all, or arguably even most, of the information available. For this reason [12] conducted research on fusing the ratings from both similar items and users to produce more accurate results. As one can imagine, these user-item matrices can be sparse. Ratings may not exist for all items and/or some users may not rate products. This is a data sparsity problem associated with using memory-based approaches. This further justifies the approach in [12] of fusing user and item based approaches, a form of a hybrid recommender system. This concept of combining approaches to utilize more information to make better predictions is similar to that of our hybrid approach of introducing meta-information into the recommender system.

In [12] combining both the user-based and item-based approaches is accomplished by fusing all ratings with predictive value for a recommendation to be made. Each rating is regarded as a separate prediction for the unknown test rating of a test-item for a test-user. The confidence of each prediction is estimated by looking at the similarity to the test-user towards the test-item. An overall prediction by taking the average individual rating weighted by the corresponding confidence is taken [12]. Thus, the more similar a rating is to a rest rating, the higher the weight is assigned to make the prediction making it possible to combine the item-base and user-base approaches [12].

So far in this section we've briefly touched on issues that can potentially arise when utilizing a collaborative filtering based recommender system; as well as, solutions as to how to overcome those issues. In [1,12] we reviewed the issues of cold-start problem and missing user-item ratings. We also looked at the hybrid approaches utilized to overcome those issues. In [1] the authors propose a new heuristic similarity measure to improve the cold-start issue and [12] utilizes a combination of both item and user-based approaches to increase prediction accuracy.

Rating prediction and collaborative filtering both work on the assumption that missing ratings are what [9] describes as missing at random. A common way to violate the missing at random condition is for the probability of not observing a rating to depend on the value of that specific rating. In [9] the violation can occur in movie recommendation systems when a watcher is more likely to view movies they assume they will like and enter ratings only for movies watched. This generates bias towards ratings with higher values. This can impact the learning and prediction of the model. The prediction for an item is only based on the available ratings of neighbors who rate the item [9]. states conditioning on the set of users who rated the item introduces bias into the prediction. This presence of non-random missing data can in turn introduce bias into the learned parameters of models [9]. To view the results of the study conducted on the impact of the missing at random assumption on collaborative filter, we encourage the reader to view [9].

It may seem like collaborative filtering is an easy way to incorporate user ratings, item content and historical data in order to make efficiently accurate predictions and overcome common issues with traditional collaborative filtering, which is all true.

However, a different, newer approach to make the predictions more accurate is incorporating meta-data. We can think of meta-data as background information to the data we have for the collaborative filter recommender system. The meta-data cannot be utilized within the recommender system in traditional approaches, but it's information that is useful.

For instance, consider the following scenario: a recommender system is used to recommend movies to a group of users. One user in particular is a male in his mid-thirties who wants to watch a movie on a Thursday night. Typically he watches westerns or horror films. Therefore, the recommender system recommends a new horror film. However, the user selects the Lion King to watch instead. What happened with the collaborative filter recommender system? Why was it so far off for the recommendation? Well, the metadata was not incorporated. This particular Thursday night was a holiday in which the user's young children were out of school the next day so they celebrated the extra family time with a movie.

How do we incorporate meta-information? The hybrid approach this thesis explores is derived from spectral learning. This thesis derives a new method for incorporating meta-data via a recursive spectral learning approach combined with a user-based collaborative filtering approach. In chapter two of this thesis, spectral learning is discussed in more detail.

1.2 User-Based Collaborative Filtering

User-based collaborative filtering only looks for similarities between users to make predictions on item ratings and recommendations and does not incorporate product

similarities as discussed in [12]. This type of filtering works on the base assumption that users who have similar user profiles also have similar tastes in objects. Essentially, this type of collaborative filter works by creating a data base of user and item rankings(like rating a purchase or a recently watched film). The recommender system begins by utilizing a nearest neighbor method to predict how the current user will rate a new film or object by measuring how the neighbors nearest to the current user have already rated that particular item of interest. Thus, this type of collaborative filtering does not necessarily depend on the content, but more so on historical data of user reviews, purchase history, etc. Relying on user behavior allows uncovering complex and unexpected patterns that would be difficult to obtain by just using data attributes [7].

1.3 Item-Based Collaborative Filtering

The item-based collaborative filtering approach analyzes the set of items the specific user rates. The algorithm then determines the similarity between the previously rated items and the targeted item [10]. The k most similar items are then selected while their similarities are computed via a similarity function. The most vital step in the item-based collaborative filtering recommender system is determining the similarity function. There are a multitude of functions one can choose from. Some of the most common are the cosine similarity function and a correlation based similarity function [1]. Regardless of the choice of similarity function, the similarities between items can be calculated. The prediction is calculated by taking the weighted average of the user's ratings on these similar items [10]. This can be done a variety of differ-

ent ways. As opposed to user-based collaborative filtering, item-based collaborative filtering is content dependent. This method also does not require a nearest neighbor method since the similarities between items is calculated versus predicting new items based on similar user ratings. This method requires less user interaction which can be beneficial when new items are introduced [10].

2 SIMILARITY

In this chapter we discuss the idea of similarity, spectral learning and random forests. We begin with a discussion of similarity. Once we define the mathematical concept of similarity, we will discuss how it is utilized in our recommender system for the purpose of this thesis. How do we mathematically define something as similar? To begin to understand the concept of similarity in a mathematical sense, we will need to have a generic definition of a similarity function that will be used to calculate similarities of certain objects or people. The data (items and users) is a collection of vectors in $\mathbb{R}^n$.

Definition 2.1 *A Similarity Score (kernel) $S(u, v)$ satisfies:*

i) $0 \leq S(u, v) \leq 1$

ii) $S(u, v) = 1$ *implies* u *is identical to* v.

iii) S *maps* $\mathbb{R}^n \times \mathbb{R}^n$ *to* $(-\infty, 1]$ *and* $S(u, v) = S(v, u)$

In general terms the above definition states some function S can determine the similarity of two inputs u and v.

2.0.1 Similarity Functions and Measures

A common approach to determine similarities between documents is to count the number of similar words between the two. However, this approach is not the most accurate. The larger the documents, the more words frequently appear inflating the similarity score. Therefore, a similarity score that considers the 'size' of the objects is desired. In a later section, we will mention some of the more common similarity

measures, provide a quick overview, discuss issues with the functions and review potential applications for the measures.

As mentioned in the previous chapter, collaborative filtering methods are popular. While there are many different types and approaches to creating a collaborative filtering based recommender system, they each rely on calculating similarities among users using some measure to recommend items [1]. The definitions of similarity below are listed in [1] and have been used for collaborative filtering recommender systems.

Definition 2.2 *Pearson's Correlation (COR) is defined*

$$sim(u,v) = \frac{\sum_{h=1}^{n'}(r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{h=1}^{n'}(r_{u,i} - \overline{r_u})^2}\sqrt{\sum_{h=1}^{n'}(r_{v,i} - \overline{r_v})^2}}$$

where $r_{u,i}$ is the rating of the item i by user u, $\bar{r_u}$ is the average rating of user u for all of the co-rated items, and n' is the number of items co-rated by both users.

A commonly used similarity measure is Pearson's correlation [1]. Pearson's correlation measures the linear correlation between vectors of ratings. Another of the most commonly used similarity functions is the cosine similarity function [1].

Definition 2.3 *Cosine Similarity is defined*

$$sim(u,v) = \cos\theta = \frac{u \cdot v}{\|u\|\|v\|} = \frac{\sum_{i=1}^{n'} r_{u,i} r_{v,i}}{\sqrt{\sum_{i=1}^{n'} r_{u,i}^2}\sqrt{\sum_{i=1}^{n'} r_{v,i}^2}}$$

where $r_{u,i}$ is the rating of the item i by user u and n' is the number of items co-rated by both users.

The cosine similarity function can detect similarities between items (documents) irregardless of size. Cosine similarity works by measuring the similarity between two vectors (representing the objects/users) of an inner product space. The cosine of the angle between the two vectors is measured. The cosine of 0° is always 1. The cosine for any angle on $(0, \pi]$ is always less than 1. Thus, if the vectors are pointing in the same direction, they are determined to be similar. The larger the angle between the two vectors, the lower their cosine similarity score is; therefore, we can say they are not similar.

Definition 2.4 *Adjusted Cosine Similarity for similarity between items is defined*

$$sim(i_u, i_v) = \frac{\sum_{j=1}^{m'}(r_{u,j} - \bar{r_{u_j}})(r_{v,j} - \overline{r_{v_j}})}{\sqrt{\sum_{j=1}^{m'}(r_{u,j} - \overline{r_{u_j}})^2}\sqrt{\sum_{j=1}^{m'}(r_{v,j} - \overline{r_{v_j}})^2}}$$

where $r_{u,j}$ is the rating of the item i by user u and $\bar{r_u}$ is the average rating of users u for all the items rated by the user, and m' is the number of users who rated both of the items.

Adjusted cosine is used to compute similarities between items where the difference in each user's use of the rating scales are taken into account [1].

Spearman's Rank Correlation calculates similarities between two vectors based on the similarity of ranks of values in those vectors.

Definition 2.5 *Spearman's Rank Correlation is defined*

$$sim(u, v) = 1 - \frac{6\sum_{h=0}^{n'} d_h^2}{n'(n'^2 - 1)}$$

where d_n is the difference in the ranks for item h by the two users and n' is the number of items co-rated by both users.

After similarities are calculated using one of the above measures, or something different, the prediction of a rating of an item i_h by user u_a can be calculated as [1] outlines:

Definition 2.6 *The predicted rating of an item i_h by user u_a is defined to be*

$$p(u_a, i_a) = \bar{r}_{u_a} + \frac{\sum_{h=1}^{m'} sim(u_a, u_h)(r_{u_h,i_h} - \bar{r}_{u_h})}{\sum_{h=1}^{m'} |sim(u_a, u_h)|}$$

where m' is the number of other users who have also rated item i_a.

2.0.2 Kernels

Similarity can also be regarded as a kernel. Kernels are functions with two data element inputs and one numerical output, but also satisfies the property that the function is symmetric and positive-semi definite. For any kernel, k a mapping ϕ must exist, such that for all inputs x, x' in $\mathbb{R}^n$ we have $k\ (x, x') = \phi(x) \cdot \phi(x')$ [2]. Kernels are then used as an algorithm that interacts with a data set. Most algorithms for similarity learning can be written to communicate with the data using dot-products on pairs of examples we know to be similarity [2]. The function can then be used on an entire data set to map similarities.

For a background on similarity functions as kernels, we will rely heavily on [2]. To begin, suppose we have a distribution P over $X\ \times(-1, 1)$. X in this case is an abstract instance space. Our goal is to establish a good classification learning measure. For this scenario we will look at learning algorithms that access the data

by a pairwise similarity function $K(x, x')$. This function takes two points from the data and provides an output between [-1,1] to determine similarity [2]. We can define a symmetric similarity function below from [2]:

Definition 2.7 *A similarity function X, is any pairwise function $K : X \times X \to [-1, 1]$. K is a symmetric similarity function if $K(x, x') = K(x', x)$ for all x, x'.*

We introduce the term "good" in reference to a similarity function by defining good as the ability for a similarity function to produce results in the ability to learn well. As we mentioned before, a similarity function K is a kernel if there exists a function ϕ from the instance space X such that $K(x, x') = \phi(x) \cdot \phi(x')$ [2]. K is an (ϵ, λ)-good kernel function for P if there exists a vector w in the ϕ-space with an error ϵ at margin λ. Essentially, K is an (ϵ, λ)-good kernel function if there exists a vector w such that the probability of the cosine similarity of w with $\phi(x)$ is greater than λ with probability of at least $1 - \epsilon$.

Since we now have a good basic understanding of what similarity and similarity function kernels are, but how do we determine what makes one good? As stated previously we have a similarity function $K(x, x') = \phi(x) \cdot \phi(x')$. [2] states K is a good similarity function for a learning problem if there exists some weighting of the training data for which the expected value with respect to the training data is within δ of the expected value over the test data with probability at least $1 - \epsilon$ for an ϵ and δ [2].

Not all similarity functions are required to be kernels. In [2] a great example of a similarity function that is not a kernel is provided. Suppose we are interested in determining if documents can be classified as similar. The criterion used to establish

similarity is a common author or keyword. If the documents meet this requirement, they are assigned a 1 as being similar or 0 otherwise. In [2] three documents, A, B and C, were considered. $K(A, B) = 1$ since they share an author, $K(B, C) = 1$ since they have a keyword in common, but $K(A, C) = 0$ as they have different authors and no keywords in common. A kernel requires if $\phi(A)$ and $\phi(B)$ are of unit length and $\phi(A) \cdot \phi(B) = 1$, implies $\phi(A) = \phi(B)$ which could not happen if K was a kernel. [2]

2.0.3 Similarity as Matrices

Similarity measures applied to a data set can be represented as a matrix. Utilizing a similarity function, such as the cosine function, we created a similarity matrix S with entries S_{ij} represent the similarity between two items i and j. Thus, S is a pairwise similarity comparison of different rows of the data.

Similarity comparisons can also define a network structure. We can create a weighted adjacency matrix, also referred to as an affinity matrix, to represent the network structure of our data by utilizing the similarity scores from the previously mentioned similarity matrix. The affinity matrix A has entries $A_{ij} = 0$ when i and j are not connected or dissimilar. Entries for $A_{ij} = S(i, j)$ when i and j are not dissimilar where $S(i, j)$ is the similarity score between i and j. Each edge in the network structure is weighted by the similarity score between the two points in the structure; the edges represent the pairwise similarity between the two vertices. Affinity matrices are a basis to conduct spectral learning algorithms discussed later on in this chapter. We also mention how both the similarity and affinity matrices are utilized for the purpose of this research of incorporating meta-data into a collaborative

filtering based recommender system.

2.1 Spectral Learning

Spectral learning algorithms use eigenvector information from similarity matrices to find structure in the underlying data. In this section an in depth overview of [6] will be discussed.The thesis researches the use of spectral clustering methods in a classification application. To begin a method for combining item similarities with classification is used to create a Markov transition process between data items called the *interested reader* model [6].

Definition 2.8 *Spectral Algorithms:Are Algorithms that use information contained in the eigenvectors of a similarity matrix, where a similarity matrix is a similarity function applied to pairs of observations from a set of data.*

2.1.1 Interested Reader Model

The interested reader model contains a collection of documents with varying topics. The user (reader) initially reads a document of a choice topic, presumably one captivating their interest. The user then continues to select other documents to read. The user most likely tries to read documents on the same topic of interest. Thus, the reader attempts to read documents similar to the initial document of interest. The user selecting similar documents creates a mapping between similarities and transition probability. The transition probabilities define a Markov chain between the documents [6]. Clustering in the data (due to distinct topics) would make the

Markov chain have subsets of high intra-set transition probabilities and low inter-set probabilities, called cliques. Each clique corresponds to a document topic [6].

Beginning the process, [6] creates the affinity matrix A with documents whose elements A_{ij} represent the similarity between two documents i and j. Given a row-normalized term-document matrix B, A is now defined to be $B^T B$, which is the cosine similarity matrix we defined earlier in chapter two.

Diagonal matrix D is then defined containing elements correspond to the transition probabilities proportional to the similarity values of the interested reader model documents [6]. The article then defines the normalized matrix N as $N = (A + d_{max} I - D)/d_{max}$ where $d_{max} = \max(D)$. d_{max} is the maximum rowsum of A. Kamvar discusses at this point in the process the transition probabilities are sensitive to the absolute similarity values. If a document is not similar to many other documents, the user may decide to repeatedly read that document as opposed to reading documents of other topics [6].

In [6] it is mentioned that probability transition matrix N for a Markov chain with a number k of strong cliques has k eigenvectors. Similar to our research process outlined in chapter three, clustering is conducted by finding the top k eigenvectors. The algorithm utilized in [6] is similar to that of a Symmetric Divisive Algorithm which is discussed in detail in their thesis. The data B is a term-document matrix where the similarity function provides pairwise cosine similarity scores of zero occurs when i was one of the top k nearest neighbors of j [6]. This helps the spectral methods when items that are not similar are assigned zeroes as they are in separate document classes. In [6] they chose k to be twenty.

After comparing the initial results from the interested reader model using the spectral learning algorithm to k-means on the data sets, [6] discusses modifying the original model. The model is modified to incorporate labeled data. As expected, if the user reads a labeled document, the probability the user will choose another labeled document of the same category is high. The probability the user will choose a labeled document of a different category is low. Transition probabilities to unlabeled documents are still proportional to their similarity to the current source document, even if the current document is not labeled [6]. Like with most similarity functions, the function utilized by [6] establishes the maximum pairwise similarity value to 1 and the minimum to 0. Therefore, it can be established if two labeled documents are in the same class they are maximally similar. Those in separate classes are minimally similar. How did [6] create a Markov matrix that reflects the change to the model? They create a symmetric Markov matrix using supervisory information when present and data similarities otherwise. We refer the reader to [6] for an in depth review of the modified process. The main differences between the spectral classifier and the clustering algorithm is that the transition utilized by [6] incorporates labeling information and a classifier in the spectral space is used as opposed to a clustering method. In chapter three, we will discuss the similarities and differences to the spectral learning approaches utilized by [6].

2.2 Random Forests

One of the most useful and widely used tools within the recommender system process is random forests. Random forest are very versatile but are most commonly

applied to classification and regression tasks [11]. A random forest is an ensemble of decision trees. Random forests work by making predictions based on averaging the prediction of the individual trees [11]. To understand random forests and how we utilize them in our process, we must first get a basic understanding of how they work. We will begin with a discussion regarding trees.

2.2.1 Decision Trees

Decision trees, also known as regression or classification tress, are partition classifiers where the partition is recursively built. [3] Decision trees are essentially a series of yes or no questions directed to the data to help make predictions for classifying the sample data. For example, consider we have two arbitrary covariates (our x points). We can use the decision trees to classify data points into the covariate categories by assigning a 1 if the data point fits into that category and 0 otherwise (our y points) [3]. Now that we see an example of how a tree works, [3] discusses how to build a tree. Let's consider we have one covariate (x) and choose a split point t to divide the real line into two sets: $A_1 = (-\infty, t]$ and $A_2 = (t, \infty)$ where $\overline{Y_1}$ is the mean of the Y_i variables in A_1 and similarly for $\overline{Y_2}$. The data is split to minimize training or classification error. Then we continue to split until sufficient accuracy is met. [3] We explore in further detail how a random forest classifier was utilized in our process for recursive spectral learning for a recommender system in chapter three.

We stated earlier decision trees can be utilized for both classification and regression purposes. The application can differ depending on what our target data (y points) looks like. If we have continuous data points for our Y we choose to utilize our trees

as a regression trees. When dealing with continuous data we choose our testing and training split in such a way to minimize the training error [3]. What if our data is not continuous? When we have discrete data, the classifier application is utilized. In classification applications the split is chosen to minimize a surrogate for classification error [3]. A common choice for the impurity measure is called the Gini index. The split is then chosen to minimize this impurity [3]. Then, we recursively split until sufficient accuracy is achieved.

2.2.2 How Random Forests Work

Now that we understand what trees are and what bagging is, we can finally discuss random forests in greater detail. A random forest is an ensemble of decision trees. Each tree contains randomly chosen subsets of features [3]. We can write the estimator as follows: $m(x) = \frac{1}{M} \sum_j m_j(x)$. In this equation, m_j is a tree estimator based on a sub-sample of size α using p randomly selected features [3]. The trees typically have a set number, k, of observations in the leaves. We will discuss more on leaves and how they were utilized for the purpose of this thesis in a later chapter. For each tree we can estimate the prediction error on the un-used data [3]. Splitting scheme and the maximum specified terminal node size k are two main elements of the random forest [8]. As [8] discusses, terminal node size can impact the prediction accuracy of random forests. There is also a difference between deterministic splitting and random splitting, both of which have an impact on the adaptivness on the random forest, this is explored deeply by [8].

2.2.3 Issues with Random Forests and Bagging

Up to this point decision trees and random forests sound like the perfect tool, right? Well random forests can encounter issues. As [11] showcases, random forests have the potential to be inconsistent with either too much or too little sub sampling causing inconsistencies in prediction from the randomized trees. So, how do we know if a random forest is inconsistent? In [11]two conditions for forests to be considered consistent are established. These conditions were researched utilizing unsupervised random forests where tree constructions do not use label information. The two conditions are diversity and locality as defined below by [11]. The diversity condition essentially states that no single data point should be given too much weight asymptotically. The local condition can be summarized as requiring the estimator to give small weights to sample points outside a certain radius around the centered query [11].

Definition 2.9 *A local average estimator $\hat{N}_n$ with weights $W_{n,i}$ satisfies the locality condition if, for an a greater than 0, the estimator*

$$[\sum_{i=1}^{n} W_{n,1}(X)\|X_i - X\| > a]$$

Definition 2.10 *A local average estimator satisfies the diversity condition, if the estimator*

$$[\sum_{i=1}^{n} W_{n,1}^2(X)] \to 0$$

Where [11] defines a generic local average estimator (similar to how we defined an average estimator previously) as

$$\hat{N}_n(X) = \sum_{i=1}^{n} W_{n,i}(x)Y_i$$

So how can we reduce the prediction error with random forests? One way to reduce the error, simply stated, is by combining trees. Such a method, bagging, is discussed in [3]. Bagging stands for bootstrap aggregation. Bagging works by taking bootstrap samples (sampling with replacement) each time the classifier or regressor is constructed. We continuing to classify by combining the average output. A similar method to bagging is sub-bagging. In this method sub samples are used as opposed to bootstrap samples. In the end this reduces the variance [3]. We refer the reader to [8] and [3] for more in depth information regarding bagging with random forests.

2.2.4 Nearest Neighbors and Random Forests

As stated throughout this chapter, random forests are popular because they are simple tools that can be successfully applied to a wide variety of problems. Nearest neighbor methods are also very popular for being successful for many problems. Much like random forests, nearest neighbor methods can also be used for both regression and classification problems. First, assume we have independent and evenly distributed observations $((x_i,\ y_i), i = 1, ..., n)$ and a random (X, Y) pair where X is the input vector and X is the response or target variable, and $X = (X^1, ..., X^d) \in \mathbb{R}^d$. In [8] $g(x) = E(Y|X = x)$ is the regression function we wish to estimate. For any $x_0 \in \mathbb{R}^d$, we utilize the estimator $\hat{g}(x_0)$ based on $((x_i, y_i), i = 1, ..., n)$ [8]. Thus, the mean square error (MSE) at x_0 is the following:

$$\begin{aligned} MSE\hat{g}(xo) &= E[\hat{g}(x_0) - g(x_0)]^2 \\ &= [E(\hat{g}(x_0) - g(x_0))]^2 + var(\hat{g}(x_0) \\ &= bias^2 + variance \end{aligned}$$

In a k nearest neighbor (k-NN) method, the k is what's referred to as a distance metric. It estimates $g(x_0)$ shown above by looking at the k points that are closest to x_0 where x_0 is the estimator $\hat{g}(x_0)$ which is $\sum_{i=1}^{n} w_i y_i$ with w_i is the weight $1/k$ for the k nearest neighbors and zero otherwise. [8]. This nearest neighbor method assumes the regression function is approximately constant in the neighborhood. The distance measure should be picked to yield the k nearest neighbors that have mean responses close to $g(x_0)$ [8]. If we choose the neighborhood carefully, we can reduce the introduced bias. How do we adaptively choose the distance metric for nearest neighbor methods? Much research has been conducted on the issue. In [8] they discuss the central idea of finding these adaptive nearest neighbor methods to select k-NN's according to an adaptively chosen local metric.

To better understand the connection between random forests and nearest neighbor methods, we need to understand the concept of distance. In [8] monotone distance measure is discussed in full. However, we need to look at distance metrics that satisfy the monotonicity property, positivity condition and triangle inequality. We can see how [8] defines a k-potential nearest neighbor below:

Definition 2.11 *A sample point x_i is a k-potential nearest neighbor (k-PNN) to a target point x_0 if there exists a monotone distance metric under which x_i is among the k closest to x_0 among all the sample points.*

From that definition, we can say any k-PNN is a k-NNN with the properly chosen metric. Since k-PNN contains potential neighbors, the number of k-PNN's is larger than k itself. If we look at random forests with terminal node size k, we can show the voting points for a target point belong to the set of k-PNN's of that target point [8]. If

a sample point is not in the k-PNN set of our target point, then there are more than k sample points in the hyperrectangle defined by the target and sample point [8]. Therefore, [8] states the conclusion that only k-PNN's can become voting points, making random forests a weighted k-PNN method.

3 RECURSIVE SPECTRAL LEARNING

The goal of the research is to demonstrate spectral learning can be combined with random forests to produce a recursive spectral learning technique that can incorporate meta information into recommender systems. Throughout the research process, there was a certain procedure in mind as to how to accomplish this goal. We discuss in detail later in this chapter the entire process that was conducted. To begin a review of the research, we need to dive a little deeper on random forests, leaves and similarity matrices in the sense of their application to our process.

3.1 Getting a New Similarity Matrix from a Random Forest Classifier

A more generalized overview of random forests and similarity is discussed in chapter 2 of this thesis. Here, we will look into how these concepts were applied to step 5 in the research process. Step 5 is getting a new similarity matrix from random forest classifier applied to full data. Random forests, as discussed in chapter 2, are a collection of decision trees. These trees are essentially a series of questions the data answers to be classified into different categories, which are the leaves. The decision tree categorizes the data into different classes based on which leaves the sample data resides.

As illustrated in chapter 2 we can say any k-PNN is a k-NNN with the properly chosen metric. Since k-PNN contains potential neighbors, the number of k-PNN's is larger than k itself. Random forests with leaves of size k, the voting points for a desired target point belong to the set of K-PNN's for that point [8]. Only k-PNN's can become voting points as mentioned in [8] We know a k-PNN is a k-NN method

with the properly chosen metric. Thus, random forests methods are weighted k-NN methods.

We now know random forests are actually a nearest neighbor method. We can also use random forest as a similarity measure by looking at the leaves. Leaves classify the samples in the data into final categories based on the voting points of the decision trees. Therefore, if two sample data points are classified into the same terminal node of the tree, they are similar. This indicates a similarity score of 1. The final pairwise similarity measures become normalized by the total number of trees in the forest as the forest averages the prediction of each individual tree [8]. From these similarities we can form an NxN similarity matrix where the entries s_{ij} represent the pairwise similarity between the two data points i and j. From the leaves this will be 1 if the two samples are in the same terminal node and 0 otherwise.

For the purpose of this research, the random forest was applied to the original data in hopes of getting a new similarity matrix, which is exactly what was done. This step along with the remaining steps in the algorithm, to be discussed further, are what is new about this specific research process. With the unique step of getting a new similarity matrix from applying a random forest to our original data, we can then replace the original similarity matrix used in the process with the new matrix from the leaves of the random forest. We can then test the new similarity matrix by following the entire process. This process can thus be completed infinitely many times.

3.2 Generalized Algorithm

Now that we understand how to extract a new similarity matrix from a random forest that is applied to the full data set, we can look at the entire, generalized algorithm for the research process. To begin, we utilize nearest neighbor methods to create a KNN basic recommender system for our data set. Then, a similarity matrix S is constructed for similarity learning using the cosine similarity function mentioned in chapter 2. Much like the methodology discussed in the Spectral Learning section of this thesis, the eigenvectors are extracted for spectral learning purposes. A random forest with observations equal to the first r eigenvectors, a random forest classifier for our target (y) data and a random forest regressor for the x data is used. Step 5, discussed earlier in this chapter, getting a new similarity matrix from a random forest applied to the full data is completed. The KNN similarity matrix is replaced with a new KNN similarity matrix or some combination of the new and old matrices. We then use the new similarity matrix (or a combination) for similarity learning. Thus, we can cycle through this process indefinitely or until sufficient accuracy is achieved.

3.3 Hybrid Recommender

In this section we develop a hybrid recommender system utilizing the following steps:

1. Given data, construct a KNN basic recommender system.

2. Extract the similarity matrix from the recommender system.

3. Transform the similarity matrix into a natural random walk among users as

nodes.

4. Apply a random forest to r highest weight eigenvectors as observations and a single variable user metadata as the response.

5. Extract a new similarity matrix from the leaves of the trees of the random forest

6. Use the new similarity matrix to replace or modify the original similarity matrix.

7. Test kNN recommender with the modified similarity matrix

Steps 1 through 4 are discussed in sections of this thesis regarding collaborative filtering, similarity, recommender systems and spectral learning. They are extracted from [6]. Step 5 utilizes information from terminal nodes (leaves) of random forests to produce a nearest neighbor method with an associated similarity matrix. Steps 5, 6 and 7 were developed specifically for this new hybrid recommender procedure.

3.4 Research Data

The data used comes from [5]. This data set contains all the transactions taking place between January 12, 2010 and September 12, 2011 for a UK-based and registered non-store online retail. The company sells unique all-occasion gifts. Many customers of the company are wholesalers.

The data contains the following elements:

- Invoice number, a 6-digit integral number uniquely assigned to each transaction.;

- StockCode: item code, a 5-digit integral number uniquely assigned to each distinct product;
- Description: item name; Quantity: quantities of each product item purchased per transaction;
- InvoiceDate: date and time invoice was generated;
- UnitPrice: price per unit of item purchased;
- CustomerID: customer number, a 5-digit integral number uniquely assigned to each customer;
- Country:the name of the country where each customer resides.

3.5 Algorithm Application

We wanted to apply the above developed algorithm to the data set in a way meta information can be incorporated. With the data set described, we wanted to predict products to customers using the quantity purchased as a rating system. Once this initial portion of the algorithm was create we wanted to see if we could predict if a user will be from the United Kingdom.

3.5.1 Data Cleaning

A package, Surprise, in Jupyter notebook was utilized during the application process for the algorithm. The data required some preliminary cleaning prior to implementing the algorithm.

After reading in the data into the notebook we began the cleaning process by dropping duplicates of the StockCode Description combination to ensure only unique item/description pairs were returned. StockCode was then encoded as strings to make it easier to query. The data element CustomerID was converted to integers. The subset of the data containing StockCode, Quantity, CustomerID and Country is cleaned to remove unnecessary information. When grouping by country, it is apparent the majority of the customers are from the UK. The data element Quantity contains zeroes, those were replaces with ones to indicate items were purchased. We then query to get Quantity greater than zero to only get customers where purchase totals were positive. Since the data is cleaned we can utilise a Surprise reader to read in our cleaned data for the recommender system. We will be using the CustomerID, StockCode, and Quantity elements for now. Later on in the algorithm we will add in the Country element as our meta-information. We utilize the Quantity data as a rating system, the more that specific product is purchased, the more liked the product is. The data rating scare is rather large ranging from 1-12540. So, we clipped the data to the 75% quantile to include the majority of the quantities purchased. Now that the preliminary cleaning and reading are complete, we can begin creating the KNN basic recommender system.

3.6 Similarity Learning and Recommender Systems

To begin our algorithm, the first step requires us to create a KNN basic recommender system through collaborative filtering. We need a recommender system where we learn the similarity metric using a random forest regressor bootstrapped

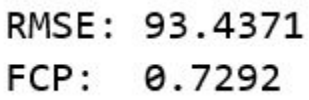

Figure 1: Initial RMSE and FCP Results.

from known similarity data. We utilize the cosine similarity function to establish similarity scores between pairwise data. The data is then stored in a similarity matrix. This similarity matrix will then be utilized for the spectral learning process discussed in a later subsection.

3.6.1 Initial Algorithm Steps Results and Discussion

The data was split into a test set and training set to prepare the data for applying the algorithm. A KNN basic system was used using the cosine similarity function. The system was applied to the training data set to train the algorithm. Next, the KNN system is tested on the testing data. The RMSE and FCP accuracy scores are tested for the prediction from the test data. The target data is the Quantity element. We would like to use quantity as a way to predict or recommend items to customers similar to a customer ranking a specific item.

From Figure 1 we can see the RMSE is initially 93.4371 and FCP was 0.7292. The FCP is the Fraction of Concordant Pairs, a ranking metric to measure how well the recommender system estimate preference. The FCP metric measures what fraction of pairs are in the correct order by testing pairwise accuracy. Roughly 72.92% of all pairs of the data are in the correct order.

Figure 2 represents a 5-fold cross validation showing scores from five different tests

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	103.1298	90.7770	112.3830	79.4029	86.8914	94.5168	11.7809
MAE (testset)	17.7908	17.7196	17.7262	17.6827	17.4159	17.6670	0.1303
Fit time	9.32	9.51	9.40	9.44	9.54	9.44	0.08
Test time	10.15	9.80	10.37	9.72	9.79	9.97	0.25

Figure 2: 5 Fold Cross Validation Results.

of the data. The RMSE ranges from 103 to 86 and an MAE score of approximately 9. MAE is the Mean Absolute Error which measures the distance between two continuous variables. The RMSE score is the Room Mean Squared Error; it is the standard deviation of the residuals or prediction errors. Residuals measure the distance between the regression line and data points. This measure how accurately the model predicts the response.RMSE can be thought of as the standard deviation of the unexplained variance. We now move onto creating the affinity matrix. The affinity matrix is generated as a foundation for the spectral learning portion of the research.

3.6.2 Spectral Learning Application

The affinity matrix A with entries A_{ij} represents the network structure of our data via a weighted adjacency matrix. Entries for i and J in A_ij are 0 when the two points or vertices are disconnected or dissimilar. Entries in A_ij for points i and j are equal to $s(i, j)$ or the similarity score of the two data points. The edges in the network structure are weighted by the similarity score between the two data points representing vertices in the structure. A threshold value was set in the code to ensure two data points are connected by a corresponding edge only if the similarity values are high enough.

Given a row-normalized matrix B, A is now defined to be $B^T B$. This is the cosine similarity matrix previously defined.

Degree matrix D is then defined as described in the spectral learning portion of this thesis. The degree matrix D is a diagonal matrix that is composed of information regarding the degree of each vertex. In D_{ij} entries are 0 when $i \neq j$. When $i = j$ in D_{ij} the entries represent the degree of that vertex, how many edges the vertex possesses. Thus, only the diagonal entries contain values other than 0.

The initial spectral leaning portion of the research process closely aligns with the process outlined by [6]. Much like the process described by [6] and in chapter 2 of this thesis, we created the affinity matrix A with entries A_{ij} representing the network structure of our data via a weighted adjacency matrix. Entries for i and J in A_{ij} are 0 when the two points or vertices are disconnected or dissimilar. Entries in A_{ij} for points i and j are equal to $s(i, j)$ or the similarity score of the two data points. The edges in the network structure are weighted by the similarity score between the two data points representing vertices in the structure.

Given a row-normalized matrix B, we can now define A to be $B^T B$. Degree matrix D is then defined containing elements correspond to the transition probabilities proportional to the similarity values of the items of interest like in [6]. The degree matrix D is a diagonal matrix that is composed of degree information for each vertex. In D_{ij} entries are 0 when $i \neq j$. When $i = j$ in D_{ij} the entries represent the degree of that vertex,the number of edges incident to the vertex. Thus, only the diagonal entries contain values other than 0.

The normalized matrix N as $N = (A + d_{max}I - D)/d_{max}$ where $d_{max} = \max(D)$

and I is the identity matrix. Once we have the normalized matrix we obtain the r largest eigenvectors of using singular value decomposition. For a symmetric matrix with positive eigenvalues, singular value decomposition is the same as diagonalization. Specifically, the Laplacian matrix is a positive semi-definite matrix, so its singular values are also its eigenvalues. Note the formula for defining the matrix N is

$$N = \left(A + d_{max}I - D\right)/d_{max}$$

.

Visually N looks like the below:

$$N = \begin{bmatrix} 1 - d_1/d_{max} & \dots & 1/d_{max} \\ \vdots & \ddots & \vdots \\ 1/d_{max} & \dots & 1 - d_1/d_{max} \end{bmatrix}$$

Notice that the rows and columns of N are non-negative and sum to 1, therefore, N is a stochastic matrix. We interpret the users of the recommender systems as forming a network with Markov transition matrix N. The Laplacian matrix L is a symmetric and positive semi-definite matrix. Singular value decomposition for a symmetric matrix is equivalent to the diagonalization of that matrix. Since L is positive, semi-definite the singular values are also eigenvalues. Thus, in this case L and N are analogous and have the same eigenvalues per demonstration below.

$$\begin{aligned} N = &\quad (A + d_{max}I - D)/d_{max}, \\ = &\quad I - (D - A)/d_{max}, \\ = &\quad I - \frac{1}{d_{max}}L \end{aligned}$$

Where $L = V \sum V^T$ and $V = [V_1|\ldots|V_n]$ where V_1 and V_n are in $\mathbb{R}^n$ is an orthogonal matrix in which eigenvectors are extracted.

$$\sum = \begin{bmatrix} \sigma_1 & \ldots & \sigma_n \\ & \ddots & \\ \ldots & \sigma_n & \end{bmatrix}$$

Where σ_j is the j^{th} singular value of L. Thus, L is analogous to N

The normalized matrix N as $N = (A + d_{max}I - D)/d_{max}$ where $d_{max} = \max(D)$ and I is the identity matrix. Once we have the normalized matrix we obtain the k largest eigenvectors using singular value decomposition. There are 2802 of the largest k eigenvectors.

3.7 Hybrid Recommender Results and Discussion

Once the spectral learning procedure is complete, we can move into the novel portion of our research by generating a random forest classifier and creating a similarity matrix from the information gathered from the leaves of the classifier. Our goal is to predict a new affinity matrix from a random forest. A random forest with observations equal to the first k eigenvectors classifier for our target (y) data is utilized. along with a random forest regressor for the (x) data. The random forest classifier is applied to the full data to produce a similarity matrix based on the classification data produced from the leaves of the random forest. The original KNN similarity matrix is replaced with a new KNN similarity matrix or some combination of the new and old matrices. We then use the new similarity matrix (or a combination) for similarity learning. Thus, we can repeat this cycle infinitely many times. We are now interested in predicting if the customer is from the UK. The leaves will produce a UK based

similarity representation.

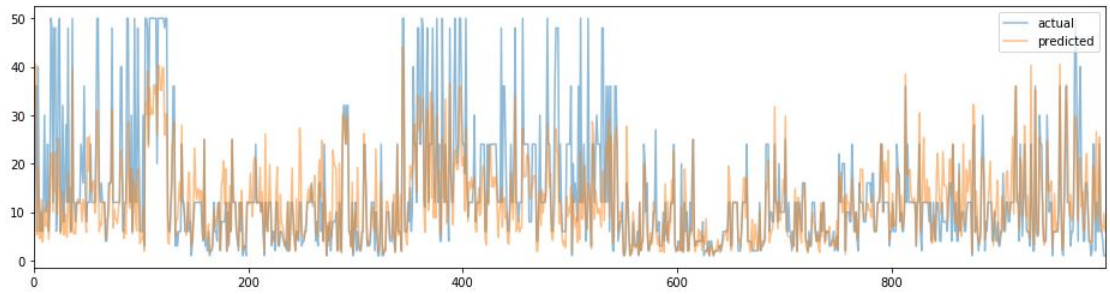

Figure 3: Actuals vs Predicted.

Our rfc (Random Forest Classifier) score is 0.9458525 which is good. Once we get the similarity matrix, we begin the process over starting by using the new similarity data to generate a recommender system. The FMSE and FCP scores of the system using the new similarity matrix are calculated. The RMSE is 95.5654 and FCP is 0.7127. The RMSE score increased while the FCP decreased slightly.

We can see from Figures 3 and 4, the new recommender system actual values versus predicted vales graphs are quite good.

This is just one example using the algorithm. We could recurse through the process multiple times until sufficient results are achieved. In this particular situation, we were interested in incorporating customer country as the meta-information; however, any background data could be incorporated. From the RMSE, FCP, RFC, the actual versus predicted scores and the figures graphing the actual values versus the predicted values, we can see this hybrid recommender system approach is a great way to include meta information and create an accurate system that generates sound predictions.

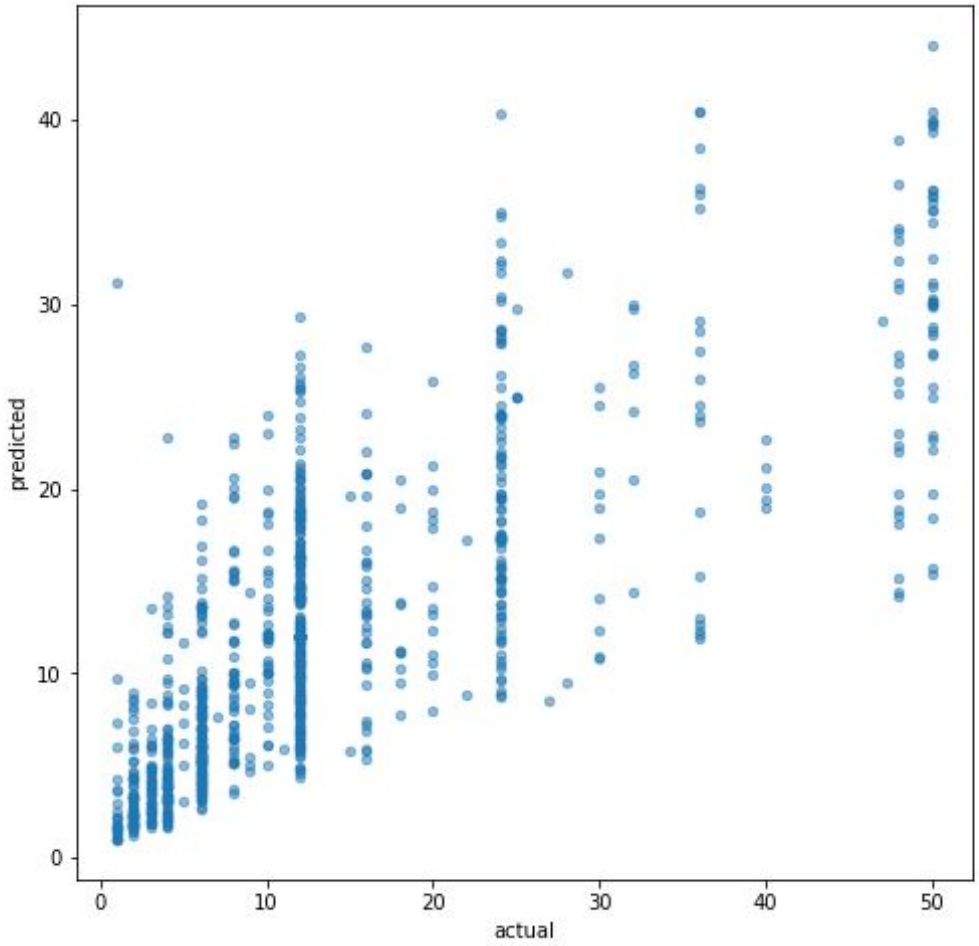

Figure 4: Actuals vs Predicted.

	actual	predicted
actual	1.000000	0.716922
predicted	0.716922	1.000000

Figure 5: Actuals vs Predicted.

4 FUTURE DIRECTIONS

Learning more information and reducing uncertainty are the ultimate goals for any data analysis and machine learning tool. This certainly applies to collaborative filters, random forests and recommender systems. Recommender systems that have the ability to incorporate meta information can be useful in a multitude of disciplines: Image processing and computer vision Computational biology Data mining and information retrieval Statistical data analysis Machine learning and pattern recognition

In the business world one must always be anticipating and preparing for what's happening next. While predicting future events isn't exactly attainable (at least not yet) we can rely on machine learning tools to better prepare businesses or customers for various scenarios. As demonstrated throughout this thesis, having the ability to incorporate meta information and utilize a recursive method improves the systems used to produce even better results. We could now use data that we have to understand and make predictions for customers or clients in a way that never seemed possible until now. Creating a hybrid system to incorporate meta-information can provide insight to users that simply relying on user profile similarities to predict items cannot. Similarity functions can be combined so that the recursive method could be used to produce ever improving similarity metrics, which in turn corresponds to ever improving recommender systems.

In the business world an approach for incorporating meta information using a collaborative filter and retrieving similarity information from the leaves of a random forest as outlined above. However, that approach is somewhat naive. We can only incorporate meta information for a given context (in this case we included customer

country), which may not always be useful. A more state of the art concept is utilizing a tensor based approached. This method is good but difficult to explain to clients in the business world. A better approach involves utilizing soft tensor methods. Soft tensors work by building a collaborative filter, get network, spectral learning, classifier objective is "context" and predict a context alternate model for the collaborative filter. This method is exactly what we have accomplished in this thesis.

Not only is the ability to incorporate meta-information into the reccomender system impressive and useful for businesses and other applications, but also the ability to recurse and generate a self-improving method is quite useful. With this recursive outlined created in this thesis, one could reproduce the algorithm an indefinite number of times until sufficient accuracy is achieved. One could also recurse the algorithm multiple times using different bits of meta-information to look at the impact of context each element has on the recommender system.

The algorithm and concepts presented in this thesis can be expanded upon to generate other types of hybrid recommender systems. We incorporated user-based collaborative filtering methods along with incorporating meta-information, but multiple collaborative filtering techniques could be combined to utilize even more information readily available in data sets. The newer aspect of this algorithm is using spectral learning and the leaves from a random forest classifier to generate a new similarity matrix from a context type of data element. Future directions could potentially explore other methods of including meta-information into recommender systems to achieve another level of personalized results for user predictions.

BIBLIOGRAPHY

[1] Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[2] Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008.

[3] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

[4] Wen-Yen Chen, Dong Zhang, and Edward Y Chang. Combinational collaborative filtering for personalized community recommendation. pages 115–123, 2008.

[5] London South Bank University Daqing Chen, School of Engineering. Online retail data set.

[6] Kamvar Kamvar, Sepandar Sepandar, Klein Klein, Dan Dan, Manning Manning, and Christopher Christopher. Spectral learning. 2003.

[7] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. pages 426–434, 2008.

[8] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[9] Benjamin Marlin, Richard S Zemel, Sam Roweis, and Malcolm Slaney. *arXiv preprint arXiv:1206.5267*, 2012.

[10] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. Item-based collaborative filtering recommendation algorithms. *Www*, 1:285–295, 2001.

[11] Cheng Tang, Damien Garreau, and Ulrike von Luxburg. When do random forests fail? In *Advances in Neural Information Processing Systems*, pages 2983–2993, 2018.

[12] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. pages 501–508, 2006.

[13] Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. A survey of collaborative filtering based social recommender systems. *Computer Communications*, 41:1–10, 2014.

APPENDICES

A Python Code Implementation

Text of Appendix 1.

```
#!/usr/bin/env python
# coding: utf-8

get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import scipy.sparse as sparse
from scipy.sparse.linalg import spsolve
from scipy.sparse import linalg as slinalg
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
import networkx as nx
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import NearestNeighbors,
   KNeighborsClassifier, KNeighborsRegressor
from sklearn.neighbors import RadiusNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from scipy import linalg
from scipy import linalg as li
np.set_printoptions(precision = 5, suppress = True,
   linewidth = 100)
pd.set_option('display.max_columns',None)
import warnings
from matplotlib import MatplotlibDeprecationWarning
warnings.simplefilter('ignore',
   MatplotlibDeprecationWarning)

website_url = 'http://archive.ics.uci.edu/ml/machine-
   learning-databases/00352/Online%20Retail.xlsx'
retail_data = pd.read_excel(website_url)

retail_data.Country.unique()

retail_data.info()
```

```
cleaned_retail = retail_data.loc[pd.isnull(retail_data.
    CustomerID) == False]

item_lookup = cleaned_retail[['StockCode', 'Description'
    ]].drop_duplicates()
item_lookup['StockCode'] = item_lookup.StockCode.astype(
    str)

cleaned_retail['CustomerID'] = cleaned_retail.CustomerID.
    astype(int)
cleaned_retail = cleaned_retail[['StockCode', 'Quantity',
    'CustomerID','Country']]

CountryLookup = cleaned_retail.groupby(['CustomerID']).max
    ()

len(CountryLookup[ CountryLookup.Country == 'United␣
    Kingdom'])/len(CountryLookup)

grouped_cleaned = cleaned_retail.groupby(['CustomerID', '
    StockCode']).sum().reset_index()
grouped_cleaned.Quantity.loc[grouped_cleaned.Quantity ==
    0] = 1

grouped_purchased = grouped_cleaned.query('Quantity␣>␣0')

from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

from surprise import Reader
reader = Reader(rating_scale=(1, 12540))
data = Dataset.load_from_df(grouped_purchased, reader)

Target = pd.Series(grouped_purchased.Quantity, name = "
    Quantity")

from surprise import KNNBasic
from surprise import KNNBaseline
from surprise import accuracy

trainset = data.build_full_trainset()
testset  = trainset.build_testset()
kNN = KNNBasic(k = 50, sim_options = {'name': 'cosine', '
    user_based': True})
kNN.fit(trainset)
predictions = kNN.test(testset)
accuracy.rmse(predictions)
```

```
accuracy.fcp(predictions);

from surprise.model_selection import cross_validate
cross_validate(kNN, data, measures=['RMSE', 'MAE'], cv=5,
   verbose=True);

from surprise import Dataset, KNNBasic
from surprise.model_selection import GridSearchCV
param_grid = {'k': [30, 40, 50, 60, 70],
              'sim_options': {'name': ['cosine'],
                              'min_support': [5],
                              'user_based': [True]}
              }
gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse',
   'mae'], cv=3, n_jobs = 5 )
gs.fit(data)
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])

A = kNN.sim
A -= np.eye(len(A))
A.shape

A[:10,:10]

D = np.diag( A.sum(axis = 1) )
D[:10,:10]

dmax = D.max(axis = None)
dmax

N = (A + dmax*np.eye(len(A)) - D)/dmax
N[:10,:10]

N.sum(axis = 1)

V, Sigma, Vt = linalg.svd(N)
plt.plot(Sigma)

VarRatios = np.cumsum(Sigma**2)/sum(Sigma**2)
VarRatios

k = np.argwhere( VarRatios > 0.95 ).flatten()[0]
k

X = V[:,:k]
X
```

```
y = np.zeros(len(A))
for uid in trainset.all_users():
    customerID = trainset.to_raw_uid(uid)
    if( CountryLookup.loc[customerID].Country == 'United␣
       Kingdom'):
        y[uid] = 1
sum(y)

from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100, max_depth = 4
    )
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.2)
rfc.fit(X_train,y_train)
rfc.score(X_test,y_test)

rfTrees = rfc.estimators_
m = X.shape[0]
n = len(rfTrees)
print('0123456789'*5)
Sr = -n*sparse.eye(m)
cnt = 0
nCoeff = 0
for tree in rfTrees:
    if(cnt == 50):
        print('')
        print('0123456789'*5)
    cnt += 1
    print('.', end = '')
    Leaves = tree.apply(X)
    for m in set(Leaves):
        F = sparse.coo_matrix( (Leaves == m).reshape
           ((-1,1)).astype(int))
        Sr += F @ F.T
Sr /= n
Sr.todense()

A = kNN.sim
A -= np.eye(len(A))
A.shape

predictions = kNN.test(testset)
accuracy.rmse(predictions)
accuracy.fcp(predictions);

F @ F.T

Leaves.shape
```

```
Leaves == 3
F=sparse

Leaves

cross_validate(kNN, data, measures=['RMSE', 'MAE'], cv=5,
   verbose=True);
```

VITA

ALYSSA WILLIAMS

Education:	B.S. Mathematics, East Tennessee State University, Johnson City, Tennessee 2016 M.S. Mathematical Sciences, East Tennessee State University Johnson City, Tennessee 2019
Professional Experience:	Trade Compliance Specialist, Eastman Chemical Company, Kingsport, Tennessee, 2017–Present