Utilizing Context for Novel Point of Interest Recommendation

by

Jason Matthew Morawski

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Recommender systems are a modern solution for suggesting new items to users. One of their uses is for novel point of interest recommendation, recommending locations to a user which they have not visited. This can be applied to a location-based social network, which contains information about their users' travel history and social connections. Within this context, there are various challenges, such as data sparsity, that limit recommendation effectiveness. We propose an algorithm for personalized novel point of interest recommendation to overcome these challenges. Our solution leverages social, temporal, and spatial context, together with collaborative filtering and a classification algorithm.

Preface

Chapter 3 of this thesis has been submitted to ACM Transactions on Interactive Intelligent Systems as Jason Morawski, Torin Stepan, Scott Dick and James Miller, 2017. Novel Point of Interest Recommendation with Location-Based Social Networks. Portions of chapters 1 and 2 were also present it that paper. For the paper, I have conducted all of the research and experimentation. The contributions of Torin Stepan relate to his initial development of experimentation software which the current work has expanded upon. My supervisors, Scott Dick and James Miller provided editorial feedback and guidance for this work.

Acknowledgements

I would like to thank my supervisors, Scott Dick and James Miller for their efforts in guiding me through my graduate program. I would also like to thank my graduate course professors, including Petr Musilek, Witold Pedrycz, and Lukasz Kurgan.

I would also like to thank my friends and family. It is only through your support that I was able to proceed to graduate studies. Most of all I would like to thank Katherine for her continued support during my academic endeavours. Thank you.

Table of Contents

1. Introduction 1
1.1 Motivation 1
1.2 Contributions 2
1.3 Thesis Organization 3
2. Recommender Systems 4
2.1 Definitions 4
2.2 Practical Application 5
2.3 Collaborative Filtering 7
2.4 Incorporating Context 10
2.5 Geospatial Prediction 13
3. Collaborative Context Recommender 15
3.1 Related Work 15
3.2 Analysis 17
3.2.1 Implicit Feedback 17
3.2.2 Sparsity and Imbalanced Data 18
3.2.3 Small disjuncts 21
3.2.4 Cold-Start 21
3.2.5 Scalability and Complexity 22
3.2.6 Short-Term Effect 23
3.3 Designing the Collaborative Context Recommender 24
3.3.1 Temporal Component 25
3.3.2 Spatial Component 26
3.3.3 Social Component 29
3.3.4 Collaborative Component 30
3.3.5 Classification Algorithm 33
3.3.6 Classification Sampling 36
3.3.7 Classification Parameters 37
3.3.8 Classification Prediction 39
3.4 Experimental Methodology 39
3.4.1 Datasets 39
3.4.2 Performance Measures 43

3.4.3 Experimental Design 47
3.5 Experimental Results 48
3.5.1 Individual Components 48
3.5.2 Parameter Exploration 52
3.5.3 Comparison 56
3.6 Conclusions 66
4. Summary and Future Work 67
References 69

List of Tables

Table I: Classification Training Attributes 34

Table II: Classification Algorithm Comparison 35

Table III. Classification Attributes 36

Table IV. Random Forest Performance 38

Table V. Dataset Statistics 40

Table VI: Temporal Performance 48

Table VII: Spatial Recommendation 49

Table VIII: Social Recommendation 50

Table IX: Collaborative Recommendation 50

Table X. Classification Component - Removing Time Input 51

Table XI. Correctly Predicted items based on Time Window (Days) 52

Table XII: Parameter Exploration of Social Constant 53

Table XIII. Exploration of Neighborhood Size 54

Table XIV. Random Forest Attributes Performance 55

Table XV. Classification Location Set Size Exploration 56

Table XVI. LFBCA Gowalla Accuracy Comparison 60

Table XVII. LFBCA Gowalla Coverage Comparison 61

Table XVIII. LFBCA Gowalla Statistics 61

Table XIX. LURA GSCorr Comparison 63

Table XX. LURA GSCorr Statistics 63

Table XXI. LURA Gowalla Comparison 64

Table XXII. LURA Gowalla Statistics 64

Table XXIII. LRT GSCorr Subset Comparison 65

Table XXIV. LRT GSCorr Statistics 66

List of Figures

Figure I: Recommender Design 24

Figure II: LFBCA Gowalla User Activity 58

Figure III: LFBCA Gowalla F-Measure Comparison 59

List of Abbreviations

CF	- Collaborative Filtering
KNN	- K-Nearest Neighbours
LBSN	- Location-based Social Network
LFBCA	- Location-Friendship Bookmark Coloring Algorithm
LURA	- Learn-Update-Recommend-Aggregate
LRT	- Location Recommendation framework with Temporal effects
POI	- Point of Interest

1. Introduction

1.1 Motivation

More people than ever are using social networking services that link them to the locations they visit. Facebook, a popular social network, saw over 1 billion active users daily in 2016 (Facebook, Inc, 2016), and its users have used the Facebook Places service to check-in millions of times at just a handful of airports (socialbakers, 2016). More than 87 million people used Google Maps each month in 2015, and these numbers are increasing (The Nielsen Company, 2015). Foursquare claims to have more than 50 million users active each month, and tracks the location of 65 million businesses. (Glueck, 2016). Foursquare also claims to have more than 8 million check-ins each day via their Swarm app, and exceed 9 billion total check-ins world-wide (Foursquare, 2016b). AlterGeo, a primarily Russian location-based social network claims to have more than 100 million daily users. Their company sells ads targeted to users based upon their current location (Altergeo, 2016). All of this points to the growing impact of Location-Based Social Networks (LSBNs). LBSNs function in part as a traditional social network, providing communication between affiliated users in a social graph. Additionally, they track check-ins – locations a user visits and declares to the LSBN at that time via their mobile device. As an LBSN is theoretically unlimited by geographic distance, it can record a user's check-ins throughout a city or region. This in turn means that the LBSN can aid users in discovering *new places to go*, based on their own history of check-ins and the histories of other users. This is the novel point-of-interest recommendation problem (Yu & Chen, 2015), and a value-added service that may help increase participation in the LBSN, forming a virtuous circle that allows even more targeted and accurate recommendations to be made.

Novel Point-Of-Interest (POI) recommendation seeks to provide new locations which will interest a user at the moment they are made (Yu & Chen, 2015); this problem cannot be effectively solved solely with a collaborative filter due to some of the inherent challenges (Adomavicius & Tuzhilin, 2005). A user's location and interests (Ye, Yin, Lee, & Lee, 2011), even the time of day (Yuan, Cong, Ma, Sun, & Thalmann, 2013), profoundly influence what points of interest a user can be interested in visiting. Even with historical user location data from a location-based social network, it is difficult to provide effective POI recommendations due to issues such as data sparsity and the cold-start problem (Burke, 2002), (Ye, Yin, Lee, & Lee, 2011).

1.2 Contributions

Our solution is to use a hybrid recommender which leverages multiple components to provide the best recommendations. We propose the Collaborative Context (CoCo) algorithm for novel POI recommendation. Our algorithm uses a hybrid design to incorporate social, temporal and spatial components, unifying them with a collaborative filter via a random-forest meta-classifier. The final result is a set of locations that are expected to be of greatest interest to the current user at that moment in time.

The primary contributions of this thesis can be summarized as follows:

- We introduce the CoCo algorithm, which provides a novel solution to the novel POI recommendation problem. Specifically, the use of a random forest meta-

classifier to unify multiple contextual components for the novel POI recommendation.

- We outperform existing novel POI recommender algorithms on all of the benchmark datasets available, and demonstrate that our algorithm's effectiveness is not dependent upon a particular dataset or experimental setup.

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 covers background concepts and relevant definitions for recommendation algorithms. We give additional information on methods of collaborative filtering. It concludes with how other works have incorporated context into recommenders, and the use of recommenders for geospatial prediction.

Chapter 3 showcases our solution for novel POI recommendation for a LBSN, and compares against related works. This includes analysis of the challenges for novel POI recommendation. We provide the details and rationale of our design. We detail our experimental methodology, and provide a comparison against existing algorithms.

Chapter 4 contains a summary of the paper and provides pointers to potential future work.

2. Recommender Systems

2.1 Definitions

Novel POI recommendation is essentially a question of where would users like to go. All that is known is a set of users, U, locations, L, check-ins, C, and the set of all friendship connections, F. These terms are common throughout works on location recommendation (Cheng, Yang, King, & Lyu, 2012), (Cho, Myers, & Leskovec, 2011), (Gao, Tang, & Liu, 2012), (Wang, Terrovitis, & Mamoulis, 2013). Within this context, the following definitions are important.

Check-in: A timestamped user-location pair. This indicates the user was at the location at the given time. (Cramer, Rost, & Holmquist, 2011)

Co-visit: When two users check-in to the same location at roughly the same time. This may also be referred to as a co-occurrence. (Crandall, et al., 2010)

Sparsity: The fraction of a matrix for which elements have a value of zero. (Duff, 1977)

Check-in sparsity: The overall sparsity for the user-location matrix, as shown in Equation (1). This is comparable to sparsity based on ratings with basic recommender systems. (Anand & Bharadwaj, 2011)

$$Checkin\ Sparsity = \frac{|U| * |L| - |C|}{|U| * |L|} \tag{1}$$

Friendship sparsity: The sparsity of the friendship network. This can also be described as the ratio of user pairs without a friendship connection to the total number of possible user-user pairs. (Ugander, Karrer, Backstrom, & Marlow, 2011) This sparsity is derived from the user-user matrix, as shown in Equation (2).

$$Friendship\ Sparsity\ = \frac{|U| * |U| - |U| - 2 * |F|}{|U| * |U| - |U|} \quad (2)$$

Top-n list: The set of n items recommended for a specific user by a recommender. With novel POI recommendation the items are locations which the user has not previously visited. This is typically an ordered list of the items which are expected to be of the greatest interest (Deshpande & Karypis, 2004).

Rating: A value assigned to a user-item pair indicating the user's preference for the item, typically on a scale with a defined maximum. (Nichols, 1998).

Explicit Rating: The rating value a user has assigned to an item manually. (Hu, Koren, & Volinsky, 2008)

Implicit Rating: A rating which is inferred for a user-item pair based upon the user's behaviour. (Oard & Kim, 1998)

2.2 Practical Application

Recommender systems are discovery services, which are now used in a great many different industries. A bewildering array of possible selections for their customers is a hallmark of industries deploying recommenders; an automated discovery service thus

becomes essential. The challenge for a recommender is to discover and suggest items that the user *actually* becomes interested in. A recommender system is given information about user preferences, and uses this to determine how much a user would prefer other items in the catalogue; the ones expected to be preferred the most are then suggested to the user (Adomavicius & Tuzhilin, 2005).

Various methods exist for implementing recommender systems. Collaborative methods are based upon comparing the current user to others previously encountered. The items the most similar users preferred are then suggested to the current user. A less personalized method is demographic recommendation, in which the user is matched to a demographic profile, and the recommendations are based upon the profile. Content-based methods make use of attributes associated with individual items, and use this to recommend items that are similar to those the user likes. Utility-based recommenders also make use of item attributes, but typically require the user to indicate their own preferences for each attribute. These preferences are used to construct a personalized utility function, which is used to make recommendations. Knowledge-based methods join item attributes with domain-specific knowledge involving user needs, and a model of how items satisfy a particular need (Burke, 2002), (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013).

The specific design of a recommender system is going to be dependent upon the domain in which it operates. Netflix famously offered a million-dollar prize for creating a more accurate movie recommendation algorithm. However, they chose not to make use of the winning design, in part due to a change in their recommendation goals. This includes placing importance upon having diversity among the recommended items

(Amatriain & Basilico, 2012). Online retailer Amazon indicated its primary concerns were dealing with a large item catalog, and allowing for fast response times. Amazon found that it was most effective for them to compute the similarity of items in advance, and store the results in a similar-items table. When they need to recommend items to a specific user, they can just lookup that user's purchases in the similar-items table (Linden, Smith, & York, 2003).

2.3 Collaborative Filtering

We begin with the assumption that users that historically favor many of the same items (i.e. *similar* users) will continue to do so in the future. If this is accurate, then items that users similar to the current one have favored, but which the current one has not viewed, are more likely to be interesting than ones simply chosen at random. Collaborative filtering algorithms operationalize this idea by forming a user-item matrix, which records the rating each registered user assigns to each possible item (if it exists; the user-item matrix is typically very sparse). This information is used to predict unknown ratings, and by extension, which items are of interest to a specific user. Collaborative filtering algorithms can be broadly separated into three categories, memory-based, model-based, and hybrids (Su & Khoshgoftaar, 2009).

A memory-based algorithm directly uses the user-item matrix to make predictions. These algorithms can be classified as either item-based or user-based. With user-based collaborative filtering, the intent is to make predictions based upon the actions of the most similar users. For this method, a formula for similarity between users must be defined. Then, for a specific user, the k nearest neighbours (most similar users) are determined. The specific predictions will vary based upon the specific similarity

function, as well as the value being used for k (Jannach, Zanker, Felfernig, & Friedrich, 2010). The general form for prediction based on user-based collaborative filtering is shown in Equation (3):

$$Prediction\ (u_j, i_k) = \bar{r}(u_j, I_j) + \frac{\sum_{u_l \in U_j} sim\ (u_j, u_l) * (r(u_l, i_k) - \bar{r}(u_l, I_l))}{\sum_{u_l \in U_j} sim\ (u_j, u_l)} \quad (3)$$

where u_j is user j, i_k is item k, $r(u_j, i_k)$ denotes the rating user j gave item k, I_k is the set of all items rated by user k, sim(u_j, u_l) is the similarity value between users j and l. The previous equation is based upon having ratings for items. Traditionally these would be explicit ratings, where a user assigns a value based upon how they feel about the item. Again, in order to use the equation in an environment where there are no explicit ratings, such as a LBSN dataset, then implicit ratings based upon user behaviour are necessary (Konstan, et al., 1997). An item-based algorithm follows a similar design, requiring an item-item similarity measure to determine each item's k nearest neighbours. Prediction for a user-item pair depends upon the ratings that a user has given that item's neighbours (Sarwar, Karypis, Konstan, & Riedl, 2001), (Koenigstein & Koren, 2013). (Huang & Gartner, 2014) compares the results of using various similarity measures for novel POI recommendation. In their work, the least effective method for collaborative filtering was using a simple user similarity measure, which did not utilize any information regarding check-in frequency. Their results improve upon constructing a more complex definition for similarity. Ultimately their best results occurred by adding spatio-temporal context to their calculations.

A model-based algorithm makes predictions from a model, which is learned from the dataset. The model is trained on some fraction of the data with the intent of learning the patterns which can be used to make predictions. As a result many of the techniques can ultimately be described as taking a probabilistic approach to prediction. A general model-based prediction formula has been suggested in (Breese, Heckerman, & Kadie, 1998) or (Hernando, Bobadilla, & Ortega, 2016), and is shown in Equation (4):

$$Prediction\left(u_j, i_k\right) = \sum_{n=0}^{m} Pr\left(r(u_j, i_k) = n \,|r\left(u_j, i_p\right), p \in I_j\right) * n \tag{4}$$

where u_j is user j, i_k is item k, m is the maximum rating for an item, I_j is the set of all items rated by user j, and Pr is a probability function based on the model. Essentially the prediction for a user-item pair is the summation of the probability of each possible rating multiplied by the respective rating.

A number of machine-learning algorithms have been employed to construct model-based collaborative filters. These include Bayesian networks, neural networks, and clustering models. A Bayesian network is an acyclic graph with nodes representing variables, and the edges representing the probabilistic dependencies of the variables (Korb & Nicholson, 2010). Bayesian algorithms are often augmented with decision trees or logistic regression to deal with multi-class variables and missing data (Su & Khoshgoftaar, 2009). Neural networks are a connected graph of simple processing nodes, designed to simulate the functioning of biological neurons; the specific topology and algorithms used vary. A variety of neural network architectures have been proposed, which can be roughly divided into feed-forward and recurrent networks (i.e.

with or without feedback connections). Feed-forward neural networks for collaborative filtering were examined in e.g. (Billsus & Pazzani, 1998), (Mannan, Sarwar, & Elahi, 2014). Clustering models identify groupings of objects based on their feature-space representations. In the context of a recommender the objects could be either users or items. To make a prediction for a specific user-item pair, the model looks at the relative membership of the user to each cluster, and the predicted rating of each cluster for that item (Nilashi, Jannach, bin Ibrahim, & Ithnin, 2015).

A hybrid recommender system combines multiple recommendation algorithms. A number of approaches for combining the individual outputs into a consensus exist (Burke, 2002), (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013). Simple approaches include weighted sums, or simple selection of one of the components. More complex approaches include fusing the component outputs; combining distinct feature sets from different components into a single feature vector; cascades where the output of one component is the input to another in series; the more general feature augmentation, where component outputs are a subset of the features to the next recommender; and meta-recommenders that take the model learned by a component as an input. A hybrid recommender system may use multiple combination methods to incorporate multiple recommendation algorithms.

2.4 Incorporating Context

Recommenders that incorporate context are often hybrid algorithms; such designs been utilized in many kinds of recommenders such as search applications and music recommendation (Adomavicius & Tuzhilin, 2011). Approaches for incorporating context vary by domain, as the availability and utility of different pieces of information

is often significantly different. For example, in an anonymized medical dataset you would be unable to directly use external data specific to individuals. For novel POI recommendation in an LBSN, we believe the most relevant contextual information would be the user's location, temporal patterns in their movements, and social influences. We defer a discussion of location to Section 2.5, and discuss temporal and social context in the current section.

Various works focus on improving upon using collaborative filtering for location prediction. (Ye, Yin, Lee, & Lee, 2011) show various ways of using collaborative filtering for novel POI recommendation. Their results have a comparison of performance for several methods of using collaborative filtering on two datasets. A notable result is that the plain user-based collaborative filtering generally outperforms friend-based collaborative filtering. They improve performance by tuning linear combinations of the components. Their best result is the linear combination of user-based collaborative filtering, friend-based collaborative filtering, and a geographic distance value.

Temporal context in the present work refers to the date and time associated with events from the user's history. One method of incorporating temporal data in recommendations is using it to reweight the ratings we input to a collaborative filter. This method has been used to improve the performance of an item based collaborative filter in (Ding & Li, 2005). However, their results showed that time weighting was most effective when the weighting algorithm parameters were learned and adjusted for each user, adding computational complexity. Another approach is to focus on the periodic aspects of temporal information. This has been done in works such as (Yuan,

Cong, Ma, Sun, & Thalmann, 2013). In their paper, they split the day into hour-length slots and examined how the hour of the day impacts individual user behaviour as well as the popularity of each location. They found that handling time in this manner improved the performance of the algorithm. A similar approach was employed in (Cho, Myers, & Leskovec, 2011), with a focus on the day of the week rather than the hour of the day. A final approach for using temporal data is to look at the sequence in which events occur; the specific times of events are ignored in favor of their ordering. Markov chains have been used to build recommenders using this concept in e.g. (Rendle, Freudenthaler, & Schmidt-Thieme, 2010). This technique is inherently useful when limited temporal information is available. However, if a dataset has accurate time records such a technique essentially ignores a valid variable and its predictive value. Works such as (Liu, Liu, Liu, Qu, & Xiong, 2016) have explored other methods of sequential pattern learning, which outperformed Markov chains in POI recommendation. The LRT algorithm (Gao, Tang, Hu, & Liu, 2013) is another take on leveraging temporal data. They divide the user-location matrix into multiple sub-matrices corresponding to specific time intervals. They then make use of matrix factorization to establish the top-N recommendations for each sub-matrix. Finally, temporal aggregation combines sub-matrix preferences into a final preference model; experimental investigation showed that a voting method was the most effective. The top-N items finally recommended to the user are the N items occurring most frequently among the sub-matrix recommendations.

Various authors have explored social context for recommendations. This usually means that items that have drawn the interest of the current users' contacts in a social graph ("friends") will be treated as more relevant than ones from other users. In (Ye, Yin, & Lee, 2010) a collaborative filter was restricted to just the user's friends. The

recommendation quality remained roughly the same, but the method was computationally faster as only a much smaller subset of the user-item matrix was processed. Other works have found that social context enhances accuracy. One example is the use of social relationships to determine latent social factors influencing individual behaviour (Shen & Jin, 2012). Their algorithm utilizes the mixed membership stochastic block model (Airoldi & David M. Blei, 2008). The block model is used to factorize the social network, and associate each user with multiple groups, which are used as latent social factors. This is combined with other latent factors derived from a matrix factorization over the user-item matrix. They show an improvement in accuracy over other models, such as item-based collaborative filtering and a matrix factorization model. Social context has been implemented to improve other matrix factorization based recommendation algorithms, as in (Yang, Zhang, Yu, & Wang, 2013). Their work builds upon probabilistic matrix factorization, which does not make use of social information. Their results show that the performance is improved by incorporating the social influence from friends, essentially assuming that a user is similar to their friends. Furthermore, they demonstrate additional performance improvement by weighting the social influence of friends by a similarity measure, rather than treating all friends equally. They specifically found an improvement when using the Pearson Correlation Coefficient as a similarity measure.

2.5 Geospatial Prediction

An active LBSN will see check-ins across multiple cities, countries, and even continents. A number of studies including (Cho, Myers, & Leskovec, 2011) and (Noulas, Scellato, Lathia, & Mascolo, 2012) have observed that users are more likely to visit locations that are geographically close. This suggests that the locations of a user's past

check-ins can be useful for predicting a future location. The novel POI recommendation problem is one application of this finding; the related *location prediction* problem is to predict a user's movements over a period of time (Leca, Nicolaescu, & Rîncu, 2015). In particular, repeated visits to specific locations form the majority of location predictions, whereas a novel POI is by definition a location that has not been previously visited (Wang, et al., 2015).

Various approaches for incorporating geographic data have been explored; the simplest method is to simply make predictions based upon geographic proximity, ignoring all other factors which may indicate user behaviour. Another option might be to look at consecutive check-ins and estimate the user's "trajectory" as a basis for location prediction; however that solution has been found to be ineffective in (Ye, Zhu, & Cheng, 2013). When other methods of predicting location are available, then a weighted combination of them is another possibility. In a non-personalized approach, an 'average' profile could be constructed from available training data. The average profile can be used to construct a fitted curve which captures the probability that a user would travel a specific distance. The probability curve can then be applied as a weight when predicting travel to any new location. A problem with this technique is that different users may have different travel patterns. It can be enhanced by tailoring the weightings based upon a user's specific travel patterns. Papers such as (Cheng, Yang, Lyu, & King, 2013) and (Monreale, Pinelli, Trasarti, & Giannotti, 2009) focus on mining user location history to predict the next location, but make no use of available social information. Some works for novel POI recommendation make significant use of spatial data, in the form of GPS data returned from a PDA or mobile phone, such as (Park, Hong, & Cho, 2007). They discuss other similar designs which make use of real-

time spatial data. The viability of such techniques is obviously dependent upon the recommender having access to real-time data.

3. Collaborative Context Recommender

3.1 Related Work

The LURA algorithm (Lu, Wang, Mamoulis, Tu, & Cheung, 2015) aggregates multiple recommender systems to form its predictions. The component recommenders cover social, spatial, and temporal data and many use collaborative filtering. In total, LURA has 11 component recommenders. They have various user-based collaborative filtering components, starting with a basic user-based collaborative filter. They also have a friend-based collaborative filter, which computes similarity based upon common friends. This is built upon for the friend-location collaborative filter, which also considers the how the users visit the same locations. Their geo-distance collaborative filter calculates similarity between users based upon their geographic distance. The final user-based collaborative filter they call category based. It considers some additional metadata about locations, their category, and calculates similarity between users based upon their category history. There are two item-based collaborative filters, the first is just item-based, and the second is augmented to be time weighted. They have three probabilistic components, including a power-law model, kernel density model, and spatial kernel density model. The final component recommender is based upon implicit matrix factorization. Based upon their results, their strongest component is user-based collaborative filtering. They ultimately have two sets of results which use different aggregation strategies. The two strategies are score-based aggregation and rank-based aggregation. When looking at both strategies, they show that they can

outperform simple collaborative filtering by roughly 11.8% on a Foursquare dataset, and 8.5% on their Gowalla dataset. However, their best aggregation method depends upon the dataset. Rank-based aggregation was more effective on Foursquare, while score-based was more effective with Gowalla. Using their rank-based aggregation on Gowalla only shows, on average, 3% higher recall and 4% higher precision than user-based collaborative filtering. Ultimately they use a different algorithm for each dataset, suggesting less overall robustness.

The LFBCA algorithm (Wang, Terrovitis, & Mamoulis, 2013) looks at friendship connections as well as establishing similarity between users who have visited the same location. They also utilize a distance measure to omit distant locations. Their method is based on a 'bookmark' graph coloring algorithm. Their algorithm is based on constructing and then augmenting a graph of users via graph coloring. For a given user, the graph is a union of their friends and any users who have visited the same locations. Weights are assigned to both the edges occurring from friendship connections, as well as the location based similarity edges. They combine the two types of edges such that they have a single transition probability value associated between a pair of users. They appear to have some performance issues on a rapidly growing dataset, where there is a significantly larger amount of new information occurring. This can be seen by looking at their results on Gowalla, where there is drop across multiple performance metrics before an eventual recovery. This seems to correspond to the portion of the dataset where the number of check-ins in the training data is significantly lower than the number of check-ins in the testing data. Their results demonstrate perfect coverage, indicating they are able to make a recommendation for all users in their tests. However, this does not mean that recommendations are useful,

as by their own utility metric, for each snapshot, less than one quarter of users had any correct recommendations. They compare their results against several alternative algorithms, including user-based and location-based collaborative filters. They consistently have higher precision and recall than most of the algorithms, although there are a few instances where a random walk with restart algorithm has better results for these metrics. The bookmark coloring algorithm is based on work in (Berkhin, 2006). The bookmark coloring algorithm is a model for 'coloring' a graph. When a node receives a 'color', it keeps a percentage, and distributes the remainder equally to neighboring nodes, such that a node can receive 'color' from several neighbours at once. This process repeats until only an arbitrarily small amount of 'free' color is left to distribute. The algorithm results in 'color' propagating through the graph, outward from the first colored node. More importantly, it causes nodes to have more 'color' when they are fewer edges away from the first node, and when they have more paths to it. Once completed, the amount of color on each node is essentially a measure of how close it is to the original node. This is used to quickly evaluate random walks traversing the graph, as long as they start at the first node. This is because the amount of 'color' on a node directly corresponds to the probability of a random walk ending on that node.

3.2 Analysis

3.2.1 Implicit Feedback

For many recommenders, users provide their ratings, indicating their preference for or against an item. However, currently-available LBSN datasets include only implicit ratings. This means that we only know that a user has visited a location. This is

different than having an explicit rating made by users for the locations which they visit. Implicit feedback is less common within recommendation research (Jawaheer, Weller, & Kostkova, 2014). This is likely because an implicit rating is expected to be of lesser value than an explicit rating (Nichols, 1998). Within the context of recommenders this presents significant challenges. Compared to explicit ratings, implicit ones are more subject to noise, and we cannot reliably separate missing feedback from negative feedback (Hu, Koren, & Volinsky, 2008). There are various solutions for dealing with implicit ratings from an LBSN. This includes having a binary rating based upon whether the user has visited the location, or using a frequency based approach where the rating is based upon the number of check-ins a user has at the location (Zheng, Xie, & Ma, 2010).

3.2.2 Sparsity and Imbalanced Data

Novel POI recommendation suffers from both sparsity and imbalanced data. An LBSN is typical used by at least tens of thousands of users, and contains an order of magnitude more locations. When working with a real dataset, the number of user-location pairs where a check-in has occurred will be dwarfed by the number of pairings where there has never been a check-in. This results in a user-location matrix with a very high level of sparsity. There is also imbalance among the actual check-ins, as some users and locations will have a disproportionately larger amount of check-ins than the others.

Dealing with heavily imbalanced data is inherently challenging. Naïve methods which work in other situations may be ineffective. The typical example of this would be an overall accuracy maximization solution. If a user visits one of one hundred locations,

then 99% accuracy can be achieved in predicting they visit no locations. This result is plainly not useful, despite demonstrating 'high accuracy'. This well-known issue has been demonstrated in works such as (Monard & Batista, 2002).

Sampling techniques are one approach to mitigating imbalanced data. Oversampling approaches reduce the class imbalance by increasing the samples of the minority class. The simplest method is to perform random sampling with replacement over the minority class. This would be as simple as recounting check-ins at random. The problem is that this is believed to cause overfitting (Kotsiantis, Kanellopoulos, & Pintelas, 2006). An alternative would be use some form of weighted algorithm, adjusting the resampling rate for specific users or locations. A more complex solution is to use the Synthetic Minority Over-sampling Technique (SMOTE) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002), or its variants. With SMOTE, artificial data points are created for the minority class. The new data points are created at a random point on the line connecting neighboring minority data points. Plainly, SMOTE was designed for feature spaces where each (orthogonal) dimension forms at least an interval scale. It is difficult to construct synthetic data points for context based location prediction. Checking in to two locations does not necessarily indicate that a user has interest in checking in to a location between them (it might, for example, be an empty field!) Each user has their own behaviour, which is in part characterized by the order of locations for check-ins, as well as the time between successive check-ins. The use of synthetic check-ins inherently alters aspects of the observable user behaviour. Other factors such as the frequency of users' co-visits with friends add additional complexity.

In contrast, undersampling balances the class frequencies by removing elements from the majority class. Undersampling of the majority class has been used effectively to deal with severe class imbalance (Drummond & Holte, 2003).The simplest option is to randomly select which majority points are included. This has the advantage of being fast and not inherently introducing a bias. The alternative is to make an informed decision about which instances of the majority class are included. One option would be to ensure that instances of the majority class are well-distributed amongst the feature space. For a LBSN this could mean only including the instances which are a specific geographic distance away from the existing instances. Another option would be to remove instances of the majority class which are nearest to instances of the minority class. This has been done using Tomek Links (Tomek, 1976) with the intent of reducing noise and borderline instances (Kubat & Matwin, 1997). In order to intelligently remove instances of the majority class, the topology of the feature space needs to be considered. This would mean accounting for the urban clustering of check-ins within an LBSN (Bawa-Cavia, 2011).

The other major approach to correcting imbalanced data is cost-sensitive classification. With a basic classification algorithm, the goal is to simply minimize the number of classification errors. No particular importance is placed upon the types of errors occurring. However, false-negative and false-positive errors can have drastically different consequences in some applications (Glas, Lijmer, Prins, Bonsel, & Bossuyt, 2003), leading us to specify different costs to each (Ling & Sheng, 2011). Cost sensitive classification refers to training an algorithm to minimize the total error cost rather than the number of errors. While this can be accomplished by modifying individual algorithms, the well-known MetaCost algorithm retrofits cost-sensitive classification

onto existing classifiers (Domingos, 1999). Note however, that the misclassification costs usually need to be specified *a priori*; and there is usually little guidance on what those costs should be, or an appropriate ratio between them.

3.2.3 Small disjuncts

The existence of imbalanced data has been associated with another problem, small disjuncts. Small disjuncts are regions in feature space which only cover a small number of training examples (Holte, Acker, & Porter, 1989). Small disjuncts emerge when performing classification of data. The classifier can easily learn large homogenous regions of the feature space, which is likely to occur in where the majority class is dominant. However, in regions where the minority class is common, there may still be many instances of the majority class, even if it is just noise. This results in a classifier producing multiple small, disjunct regions (He & Garcia, 2009). The work in (Jo & Japkowicz, 2004) suggests that poor performance on an imbalanced dataset may be caused more by the existence of a small disjuncts problem when dealing with multi-dimensional data. Their suggestions for dealing with small disjuncts are either pruning away sufficiently small regions from the classifier, or performing cluster based oversampling. However, as we have already discussed, oversampling is inherently challenging within the context of novel POI recommendation.

3.2.4 Cold-Start

As with other recommenders, the cold-start problem also afflicts novel POI recommendation for a LBSNs. A new user by definition has not checked-in at many distinct locations, and this makes it difficult to accurately determine their preferred next destination. This is closely related to the problem of imbalanced data, as a user

with few ratings is contributing a mostly empty row to the user-item matrix. The cold-start problem is known to be particularly problematic for collaborative filtering. Numerous solutions have been proposed to alleviate this problem. With typical recommender systems, there is a suggestion to switch from a user-based collaborative filter to an item-based collaborative filter (Sarwar, Karypis, Konstan, & Riedl, 2001). This solution was proposed to deal with datasets where there was a large number of users. Obviously, this solution is less effective on datasets with a large number of items. Looking at various sample LBSN datasets, the number of locations may be an order of magnitude larger than the number of users, negating the usefulness of this method. The use of additional context is another way to handle the cold-start. The addition of both social and geographic factors have been shown to help with the cold-start problem with collaborative filtering (Ye, Yin, Lee, & Lee, 2011). Another option is to utilize a different kind of algorithm for recommendation, one which is less susceptible to the cold-start problem (such as a content filter or a hybrid content-collaborative filter, e.g. (Morawski, Stepan, Dick, & Miller, 2017)).

3.2.5 Scalability and Complexity

Novel POI recommendation on an LBSN may encounter another issue faced by recommender systems dealing with a large or even expanding dataset. Scalability has been identified as a significant issue in several works, including (Sarwar, Karypis, Konstan, & Riedl, 2000) (Papagelis, Rousidis, Plexousakis, & Theoharopoulos, 2005). Again, for generic recommender systems, switching from a user-based collaborative filter to an item-based collaborative filter has been proposed to mitigate scalability problems (Sarwar, Karypis, Konstan, & Riedl, 2001). The usefulness of that method is dependent upon the user-item balance of the dataset being evaluated. Another solution

is to reduce the dimensionality of the user-item matrix, such as through singular value decomposition (Sarwar, Karypis, Konstan, & Riedl, 2000). The use of singular value decomposition has been criticized for efficiency concerns, and alternatives such as the use of an incremental approach to collaborative filtering have been suggested (Papagelis, Rousidis, Plexousakis, & Theoharopoulos, 2005). Some approaches pre-filter user-item data to reduce its size. One such method is to restrict similarity in collaborative filtering to only be between friends, not all users (Ye, Yin, & Lee, 2010). This method inherently sacrifices potentially useful data, and is particularly problematic for users with few connections in that particular social graph.

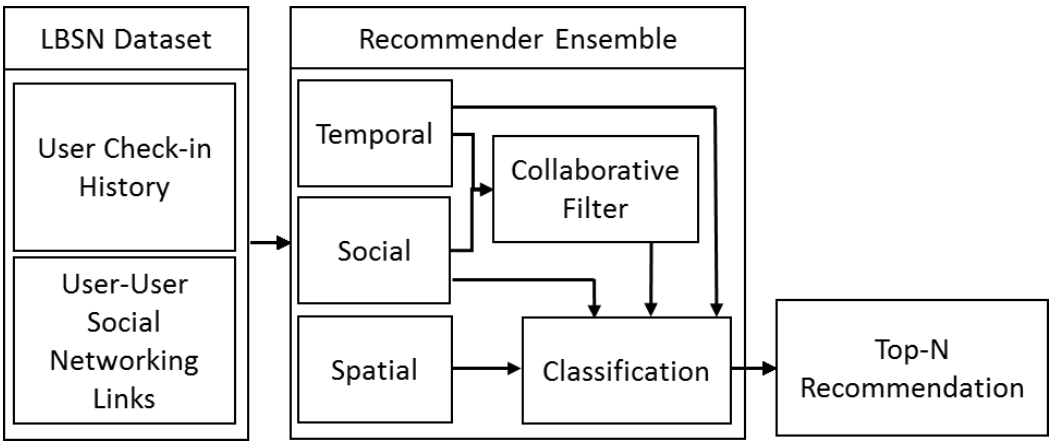
3.2.6 Short-Term Effect

There is evidence that in a LBSN, check-ins have a 'short-term' effect, meaning that the older a user's check-in, the less impact it has upon their next location (Gao, Tang, & Liu, 2012). This may be caused by user preferences and behaviour changing over time as has been seen with the Netflix data (Koren, 2010). For these reasons, it will likely be useful, when analyzing a user's check-in history, to apply a weighting to check-ins based upon their absolute or relative age, as in (Ding & Li, 2005). One approach might be to apply weighting based solely upon on the order of the check-ins, rather than the actual times. Such a method would not differentiate between subsequent check-ins occurring with significant temporal separation and check-ins made in rapid succession. This could be an issue as it ignores some knowledge embedded within the system. An alternative is to consider the amount of time occurring between each check-in in a user's history. In both cases, an adaptive weighting is expected to be the most accurate, but also the most computationally intensive.

3.3 Designing the Collaborative Context Recommender

In order to overcome issues discussed in Section 3.2, our recommendation system consists of multiple components. This includes relatively simple spatial, temporal and social components, as well as the more complex collaborative filter and classification components. The former are used to provide the additional context necessary to overcome the challenges. These intermediate results are passed to the collaborative and classification components. The final recommendations are generated by the classification component. The overall design can be seen in Figure I.

Figure I: Recommender Design



Sections 3.3.1, 3.3.2, and 3.3.3 discuss the detailed design of the Temporal, Spatial and Social components, respectively, while Section 3.3.4 discusses the Collaborative Filter. The final classifier is discussed in Sections 4.5 (selection of the classifier), Section 4.6 (sampling an LBSN dataset), Section 4.7 (parameter exploration) and Section 4.8 (novel POI recommendation with this design).

3.3.1 Temporal Component

From the literature, a solely temporal approach is plainly not going to be an effective novel POI recommendation system. However, a temporal component can provide useful information about a user's behaviour. Specifically, this component is intended to account for the 'short-term' effect that is described in (Gao, Tang, & Liu, 2012).

The temporal component is used to reweight locations based upon how recently they were visited. It is expected that if a user's friend visited two different restaurants a year apart, they would find the more recent restaurant a more relevant recommendation (Cho, Myers, & Leskovec, 2011). For simplicity, we use a linear weighting starting from the start of the dataset, as shown in Equation (5). Such an equation meets the requirements of monotonicity as suggested in (Ding & Li, 2005). With this equation, it is not necessary to learn additional parameters which may be specific to an individual dataset. For each user, a total check-in weight value is calculated as the summation of the temporal weights for all locations the user has visited as per Equation (6).

$$Temporal\ Weight\ (u_i, l_k) = \frac{CheckinTime(u_i, l_k) - StartTime}{CurrentTime - StartTime} \tag{5}$$

$$TotalWeight\ (u_i) = \sum_{k=1}^{n} Temporal\ Weight\ (u_i, l_k) \tag{6}$$

The total weight is relevant for handling users of differing activity levels. If two users visited a location at the same time last week, they would both receive the same temporal weight for the location. However, that location's relevance may be different if the users vary in the number of locations they visit. If a user has been to hundred different places, a location is less significant than if they only have ten distinct locations, because it accounts for smaller fraction of their recent check-ins.

3.3.2 Spatial Component

The spatial component is used to reweight locations based upon their physical proximity to the user's last known location. Spatial weighting is important because it is expected that users are less likely to visit more distant to locations (Cheng, Yang, King, & Lyu, 2012), (Cho, Myers, & Leskovec, 2011). All other things being held equal, you are more likely to visit the coffee shop down the street than across the country. In some applications such spatial relationships could be gathered using real-time GPS data (Park, Hong, & Cho, 2007). Unfortunately, we do not have access to a user's real-time location in any of the public LBSN datasets; the only positional data available is the user's check-ins. We therefore use the user's last check-in location as a proxy for *current location*. For each user u_i, we calculate the average distance $\overline{D}_{all}$, between their check-in locations l_k as per Equation (7). The average distance is used to capture an individual user's preferred travel distance. For instance, someone who walks through the downtown would likely not travel as far within their own city as a person that is regularly driving. As there is no additional data on each user's methods of travel, we must depend upon the recorded check-ins for predicting their behaviour. The average distance is used in a Gaussian function to calculate a weight for a given distance, following (Cho, Myers, & Leskovec, 2011). The Gaussian function returns a value

between zero and one for *spatial weighting*, as shown in Equation (8). The function is then used whenever a *spatial weighting* is required for a new location, by passing in the distance from the user's last known location. If a user does not have enough historical check-ins to calculate $\overline{D}_{all}$, the mean of $\overline{D}_{all}$ across all users is used as the default.

$$\overline{D}_{all} = \frac{\sum_{k=1}^{n-1}(Distance(l_k, l_{k+1}))}{n-1} \tag{7}$$

$$SpatialWeight(u_i, l_k) = e^{-\frac{Distance(lastCheckin(u_i), l_k)}{2*(0.5*\overline{D}_{all})^2}} \tag{8}$$

The entire process is also repeated with a special subset of the user's visited locations. Instead of looking at all consecutive check-ins, the subset is restricted to pairs which have occurred on the same day. This provides additional information based on the user's behavioural pattern which can be exploited to recommend more spatially relevant locations. This second set of weights is also available to later components for additional context.

As the locations in a LBSN have their position recorded with respect to latitude and longitude on the Earth, we use the haversine formula (Robusto, 1957) to determine the actual distance between two locations on a sphere.

$$hav(\frac{d}{r}) = hav(lat_k - lat_j) + \cos(lat_j)\cos(lat_k)\, hav(long_k - long_j) \tag{9}$$

In this equation lat_i and $long_i$ correspond to the latitude and longitude, respectively, for location i. The distance between the points is denoted by d, and the radius of the sphere is r. Note that $hav(\theta)$ denotes the haversine function, which is shown in Equation (10).

$$hav(\theta) = sin^2\left(\frac{\theta}{2}\right) \tag{10}$$

This equation is rearranged for actual distance in Equation (11). Note that we must substitute a value for r. We use the value of 6371.009 km, corresponding to Earth's mean radius (Moritz, 1980). This value is roughly 7 km smaller than the Earth's equatorial radius (Williams D. R., 2016).

$$d = 2 * 6371.009 * arcsin\sqrt{sin^2\left(\frac{lat_k - lat_j}{2}\right) + \cos(lat_j)\cos(lat_k)\, sin^2\left(\frac{long_k - long_j}{2}\right)} \tag{11}$$

There are some methods which could be used to produce a more accurate measure of distance. In urban environments a user will rarely be able to travel along a direct line to their destination. Road topology will limit the paths available to user. This will be a factor for both pedestrians walking on sidewalks, as well as motorists on the streets. Accounting for the physical limitations would allow a more accurate calculation of a user's effective proximity to locations. This could be further enhanced with considerations regarding speed limits, road conditions, or public transportation routes (Chen, Lu, & Gu, 2009). As it stands, existing LBSN datasets do not contain this information.

3.3.3 Social Component

The social component is used to reweight locations to account for social context. It is expected that users will preferentially visit locations which have been visited by their friends. In an experiment, it has been shown that "… the user's friends consistently provided better recommendations than RS [Recommender Systems]" (Sinha & Swearingen, 2001). More recent literature has also argued the relevance of opinions from users with friendship connections. Such user pairs are more likely than average to share locations among their check-ins (Cheng, Yang, King, & Lyu, 2012). It has also been shown in various LBSNs that a user's first check-in to a location is disproportionately likely to be preceded by a check-in from the user's friendship community (Wang, Terrovitis, & Mamoulis, 2013). Work in (Cho, Myers, & Leskovec, 2011) showed that users are particularly impacted by their friends' check-ins when visiting more distant locations. However, a user is unlikely to be influenced by all of their friends equally. It follows that for each user, the "strength" of their friendships must be determined. We define the strength of the friendship based upon their check-

in history following (Crandall, et al., 2010), which shows a correlation between friendship connections and co-visits. For this reason, we use a similarity measure based upon co-visits. This is calculated between the user, u_i, and their friend, u_f, as the number of co-visits divided by the user's total number of check-ins, as shown in Equation (12).

$$Friendship(u_i, u_f) = \frac{Covisits(u_i, u_f)}{Total\ Checkins\ (u_i)} \tag{12}$$

Ideally, the LBSN dataset would track whether two users visit a location together. In practice, only individual check-in times are recorded, which does not tell whether a user has just arrived, is in the middle of a visit, or is just about to leave. Some services even allow the user to retrospectively record a check-in for a previous day (Foursquare, 2016c). For these reasons, the time window for declaring a co-visit is set to a full day. Note that the friendship weighting is not symmetric; if two users have a different number of total check-ins, their friendship values towards each other will not be equal. The asymmetry of friendships allows for a leader-follower dynamic between users, where recommendations between the users may effectively only be one-way, following (Eagle, Pentland, & Lazer, 2009).

3.3.4 Collaborative Component

The basic idea of collaborative filtering is that similar users like similar items. You can therefore recommend new items to a user based on what similar users liked. With user based collaborative filtering a common method is to use a nearest neighbour algorithm.

The algorithm looks at the N most similar users and uses their preference for an item to determine the target user's preference for it. Each user is effectively weighted by their similarity to the target (Schafer, Frankowski, Herlocker, & Sen, 2007).The collaborative component takes input from the context components. Ultimately, we want to recommend locations visited by the neighbourhood of most similar users. This requires determining the similarity between users. The first part of this is to calculate the similarity of the check-in history. We calculate the check-in overlap for two users as the summation of the product of their respective temporal location weights for each location they have visited. This is described in Equation (13). This equation is used because it places the strongest weight on locations which both users visited recently. By taking the product, more importance is placed upon visits which occur near the same time. This is useful as check-ins with close enough check-in times are co-visits, which are a good indicator of similarity between users (Pham, Hu, & Shahabi, 2011). Another effect is there is significantly less weighting applied to the oldest check-ins. This is useful because of evidence that in a location based social network, the check-ins have a short-term effect, as discussed in Section 3.2.6. This is used to calculate the check-in similarity as per Equation (14).

$$Checkin\ Overlap(u_i, u_j) = \sum_{k=1}^{n} Temporal\ Weight\ (u_i, l_k) * Temporal\ Weight\ (u_j, l_k) \quad (13)$$

$$Checkin\ Similarity(u_i, u_j) = \frac{Checkin\ Overlap(u_i, u_j)}{TotalWeight\ (u_i) + TotalWeight\ (u_j) - Checkin\ Overlap(u_i, u_j)} \quad (14)$$

Location based social networks provide users with more information about what their friends are doing rather than arbitrary non-friend users. For this reason, we weight friends and non-friends differently for similarity. We calculate a social similarity measure between users, based upon their associated friendship value, as shown in Equation (15). It uses a social constant, S_C, to allow for similarity between users who are not friends. This constant determines the weighting applied to the friendship strength. As non-friends have a friendship value of zero, their social similarity will be equal to S_C. We restrict the constant to the range [0, 1]. If S_C is set to zero, then the collaborative filter will only consider users who are friends when calculating similarity, similar to the design in (Ye, Yin, & Lee, 2010). This equation places more importance upon users who are friends when S_C is less than one.

$$SocialSimilarity(u_i, u_j) = S_C + (1 - S_C) * Friendship(u_i, u_j) \quad (15)$$

The similarity between two users is shown in Equation (16) as the product of their check-in similarity and their social similarity. The highest similarity occurs when both of the components are large. By taking the product, there is no similarity if either component is zero. This equation for similarity is used so that the collaborative filter can leverage both the temporal and social components.

$$Similarity(u_i, u_j) = CheckinSimilarity(u_i, u_j) * SocialSimilarity(u_i, u_j) \quad (16)$$

The similarity values are used to determine the user's neighborhood, UN_i. The neighbourhood, UN_i is the set of the most similar users to user u_i. We consider a top subset of the similar users in a k-Nearest Neighbour design. The exploration of k, our neighbourhood size, is shown in Section 3.5.2. We use the check-in history of location l_k to establish a set visitors, V_k, for that location. The intersection of UN_i and V_k is used to determine the prediction value for user u_i visiting location l_k. This means only neighbours who have been to a location contribute weightings for subsequent calculations recommending that location. The final value for each location the target user has not visited is the sum of the similarities to all of the neighbours that have visited the location.

$$Collaborative(u_i, l_k) = \sum Similarity(u_i, u_j), \quad u_j \in V_k \bigcap UN_i \tag{17}$$

The collaborative component can be used to make a prediction for every user-location pair, however it predicts zero for locations that have not been visited by any members of the neighborhood. The set of top locations, L_i, can be used on its own for location recommendation, but can also be passed to another component. The collaborative component can be classified as feature combination hybrid algorithm (Burke, 2002) due to the manner in which it incorporates the earlier components.

3.3.5 Classification Algorithm

The last stage of our algorithm is a meta-classifier that accepts the outputs of the four prior components, as well as a few features directly drawn from the dataset, and produces our final predicted ratings; the top-N highest rated locations are then recommended to the user. This allows us to take advantage of the known effectiveness

of collaborative filtering, while also mitigating its weaknesses against cold starts and sparsity (Adomavicius & Tuzhilin, 2005), (Lu, Wang, Mamoulis, Tu, & Cheung, 2015).

Plainly, our first step in building the meta-classifier is to select a classification algorithm from the many options available. In order to do so, we evaluate several well-known algorithms on a subset of one of the LBSN datasets. The classification task is to determine *novelty*: is a particular check-in at a novel location for that user? The feature set provided to the classifiers is presented in Table I. The original Brightkite dataset is filtered down to only include the 1000 most active users and locations, leaving roughly half a million check-ins. The algorithms were trained on the first half of the dataset and tested against the second half, with the division made chronologically. As shown in Table II, the random forest algorithm is the most effective with respect to both time and accuracy.

Table I: Classification Training Attributes

Attributes	Type
User Identification Number	Integer
Location Identification Number	Integer
Time	Integer
Time from last Novel Check-in	Integer
Latitude	Real
Longitude	Real
Novelty	Class (Target)

Table II: Classification Algorithm Comparison

	Random Forest	JRip	SMO	RBFClassifier
Running Time (s)	25.79	45.05	7433.48	55.68
Accuracy (%)	95.4019	95.3639	95.3798	95.3781

It is an implementation of a Random Forest (Breiman, 2001). Similar algorithms have famously seen practical application in the motion tracking Kinect accessory for Microsoft's Xbox 360 and Xbox One gaming consoles (Shotton, et al., 2013). They have also been applied to modeling gene selection in bioinformatics (Díaz-Uriarte & De Andres, 2006), and chemical compound classification (Svetnik, et al., 2003). The basic concept is that an ensemble of decision trees is created and trained on the training data. Each tree will have a random set of attributes or features to work with at each node. The training causes the tree to create branching decision points based on the value of an attribute. A completed tree makes future predictions by comparing new data to the existing tree structure. For the forest to make a prediction, each tree contributes a vote towards the final result. The implementation we use is the Weka (Hall, et al., 2009) random forest classifier.

The random forest handles many features. This includes user identification number, number of friends, number of different locations visited, distance from the user's previous check-in, and the total number of distinct visitors for the location. The outputs of the collaborative and context components are also features. The classification target is visitation. In training, this has a value of one if the user has had any check-ins at the location and zero if they did not check-in. The classifier attempts to predict the visitation value when receiving new data. Unlike the binary novelty class target we

used when comparing the different classification algorithms, the visitation value is a real number. This means the predicted visitation can either be rounded, or the fractional value can be used. This was done because having a fractional value is useful for determining a ranking among the predictions of this classifier.

Table III. Classification Attributes

Attributes	Type
User Identification Number	Integer
Number of Friends	Integer
Number of distinct locations visited by user	Integer
Number of distinct visitors for location	Integer
Haversine distance from user's last location	Real
Social Prediction	Real
Spatial Near Prediction	Real
Spatial Far Prediction	Real
Temporal Prediction	Real
Collaborative Prediction	Real
Visitation	Real (Target)

3.3.6 Classification Sampling

In addition to check-ins, the classifier needs some examples of locations which the user is not inclined to visit. This requires using user-location pairs for which no check-in has occurred. The naïve solution is to use a full user-location matrix and train on every single user-location combination. A small dataset such as gScorr, with roughly 2.2

million check-ins, would have over 1 billion entries in this matrix. This large number of pairs is costly with respect to both time and memory. Instead, we use a form of stratified undersampling (Cochran, 1953). Note that the work in (Jo & Japkowicz, 2004) suggests that poor performance on an imbalanced dataset may be caused more by the small disjuncts problem when dealing with multi-dimensional data. Our use of stratified undersampling is supported by recommendations in this situation that sampling should focus on removing the majority class (Weiss, 2004). Specifically, for each actual check-in made by a user, a random location is selected which has not visited by the user. This user-location pair is used to construct a "negative" check-in, which will have a zero visitation value. These negative check-ins can then be included in the training data. By using random selection we are not inherently introducing bias with our selection. In addition, random selection is fast and does not need to be tuned to a specific dataset; implying also that as the dataset evolves over time, there will be no need to modify the sampling technique.

3.3.7 Classification Parameters

There are various parameters which can be adjusted when setting up a random forest. The significant parameters are the number of trees in the forest, and the number of attributes considered at each node. The default Weka parameters have a forest which is constructed with 100 trees, which may grow to an unlimited depth. By default each tree will use 4 random features at each node. Increasing the number of trees appears to increase the algorithm's performance, but is overshadowed by an increase in memory usage and execution time. The results for varying the number of trees for the random forest is shown in Table IV. Random Forest Performance.

Table IV. Random Forest Performance

Number of Trees	10	50	100	1000
Execution Time (minutes)	1.991833	10.70833	21.196	196.5752
Accuracy (%)	76.1036	77.3278	77.6423	77.8704

The results in Table IV come from a training dataset with the attributes in Table I. Like Table II, the class variable for predictions is novelty. Again the dataset is based upon the Brightkite LBSN check-ins. However, the method selecting the check-ins is different. Only check-ins which occurred during the first year of the dataset are included. This means that there are approximately 1.2 million check-ins available to the experiment. Again these check-ins are divided evenly into training and test data chronologically. Note that increasing the number of trees causes a roughly linear increase in execution time, but there are diminishing returns for accuracy. Based on these results we set the number of trees in our random forest to 100, as we find it to be an adequate compromise between execution time and accuracy. We include the results of varying the number of features per node during parameter exploration in Section 3.5.2.

A final adjustment was made to the classification component. Both users and the LBSN as whole can change their behaviour over time. This is closely related to the short-term effect discussed in Section 3.2.6. It was suspected that the simple temporal component was not enough to account for this. An experiment was done with the application of a time window during the training of the classification component. With the time window, check-ins are only used for training if they occurred within a specified amount of time from the current time. We include the results of varying the size of the

time window in Section 3.5.1. Other components, such as the collaborative filter, are not restricted to looking at only the most recent check-ins.

3.3.8 Classification Prediction

The classification component is ultimately a random forest. Once the forest has been trained it can output predictions, given a set of inputs. Ideally, a prediction could be every user-location pair to determine the top locations. However, with the size of a LBSN this could require millions of locations to be evaluated for each user. And each earlier component must provide its feedback for that user-location pair. This is incredibly costly in terms of execution time. Instead, for prediction, a set of probable locations is constructed. This set starts with the top predictions of the collaborative component. It is expanded by adding the top locations of the user's friends, as determined by the social component. This is done due to the effectiveness of the social component on its own. The forest then evaluates the set of locations, making its own predictions for each location. The number of locations requested from these components we call set size, and is a constant which we will denote as Z_C. The final predictions allow for the set of probable locations to be sorted, and the top N can then be given as recommendations. Typically this is done with $N = 10$, but this can be varied to allow for better comparison with other algorithms.

3.4 Experimental Methodology

3.4.1 Datasets

We use four datasets for our experiments; general dataset statistics are shown in Table V. All datasets contain a list of time stamped check-ins. Each check-in therefore

denotes a particular user that has been to a particular latitude-longitude at a specific time. The datasets also include users' social network connections. A connection between users indicates that they are to be considered friends. The friendship connections are undirected. This means that if user u_1 is friends with user u_2, then user u_2 must be friends with user u_1. The datasets covered all feature a high level of check-in sparsity, as well as friendship sparsity. Check-in sparsity means that most users have not visited most locations. Friendship sparsity indicates that most users have few friends, relative to the total number of users.

Table V. Dataset Statistics

Dataset	Start date	End date	Users	Locations	Check-ins	Social links
GSCorr (Foursquare)	January 1, 2011	December 31, 2011	11 326	96 002	2 199 782	94 328
GSCorr subset (Foursquare)	January 1, 2011	March 31, 2011	5 269	26 381	288 079	10 208
Gowalla	February 4, 2009	October 23, 2010	196 591	1 280 969	6 442 890	950 327
Gowalla subset	February 4, 2009	October 23, 2010	74 725	767 936	5 829 873	950 327
Brightkite	March 21, 2008	October 18, 2010	58 228	772 966	4 491 143	214 078

The datasets we use have been collected by other researchers. Most data was originally collected directly from the corresponding LBSN, via the application program interface, or API. In cases where the original researchers found the direct access to the LBSN inadequate, the datasets were augmented from an additional source, such as Twitter.

The twitter API allows software to programmatically access twitter data online, using specialized commands (Twitter, Inc, 2016).

The GSCorr dataset is from the Foursquare LBSN. Foursquare is a social networking system currently split over multiple apps (Foursquare, 2016d). As a social network it allows users to connect to each other, marking other users as friends. Users can check-in on their phone when they visit known locations. Foursquare also allows users to search for places to go, showing how many check-ins the locations have received. Typically, the check-in information is only visible to the user and their friends, but check-ins can be shared over Twitter or Facebook, rendering them more public (Foursquare, 2016a). The check-in records were gathered from Twitter with the public API by (Gao & Liu, 2014). The social links were collected directly from Foursquare. Users average roughly 194 check-ins and locations receive an average of 22.9 check-ins. The user-location matrix has close to 99.80% sparsity. The sparsity of social links is just over 99.85%. The GSCorr subset is a temporal subset of the GSCorr dataset. Users make about 54.7 check-ins on average, and the average location is visited 10.9 times. The generally check-in sparsity is close to the large GSCorr dataset, again just below 99.80%. The friendship sparsity is increased, just under 99.93%.

The Gowalla dataset is from the former LBSN, Gowalla. Gowalla was a LBSN accessible through a website or mobile app (Crunchbase, 2016). Gowalla had many features relating to travel, such as virtual passports to track the places you visit. It allowed users to plan and share trips, which were a set of specific locations to visit (Gowalla Incorporated, 2011). Ultimately Gowalla was acquired by Facebook (Williams J. , 2011). The Gowalla datasets were originally collected by (Cho, Myers, & Leskovec,

2011). The Gowalla dataset was collected from the Gowalla LBSN using their public API. The LFBCA version of the Gowalla dataset is somewhat larger. The average user makes fewer than 32.8 check-ins, and on average locations are visited just over 5 times. The check-in sparsity is under 99.997%. The social map demonstrates more than 99.995% sparsity across users. The Gowalla subset, used by LURA, is filtered to only have users which made at least 10 check-ins, and locations which were visited at least twice. In this dataset, the average user makes 78.0 check-ins on average and locations receive 5.8 visits on average. The user location check-in matrix sparsity is approximately 99.99%, and friendship connections exhibit roughly 99.97% sparsity.

The Brightkite dataset was also collected by (Cho, Myers, & Leskovec, 2011). It was collected from the Brightkite LBSN via the public API. Like other LBSNs, it allowed users to check-in to locations via mobile apps, and included the ability to connect with friends. Brightkite was active from 2007 until April 2009, when it was acquired by Limbo (Crunchbase, 2015). In this dataset, the average user makes 77.1 check-ins on average and locations receive 5.8 visits on average. The user location check-in matrix has sparsity above 99.99%. The friendship connections demonstrate over 99.98% sparsity. In order to keep the other datasets entirely as test data, this dataset is the dataset exclusively used for parameter exploration. Work in (Cho, Myers, & Leskovec, 2011) and (Scellato, Noulas, Lambiotte, & Mascolo, 2011) demonstrates that user behaviour and other patterns are similar across multiple LBSN datasets, meaning that the parameter exploration results should be usable for other LBSN datasets.

3.4.2 Performance Measures

We use multiple measures of performance for evaluating our algorithm. Precision and recall are commonly used by many sources (Herlocker, Konstan, Terveen, & Riedl, 2004). We use standard definitions for recall and precision. Recall, R, corresponds to the fraction of correctly predicted locations, L_C, out of the total number of actual visits in the test set, L_V. This is shown in the following equation:

$$R = \frac{L_C}{L_V} \tag{18}$$

Precision P, is the fraction of correct location predictions, L_C, out of the total predictions made, L_P. This corresponds to the following equation:

$$P = \frac{L_C}{L_P} \tag{19}$$

An important derived metric is the F_1 metric or F measure. This metric uses both precision and recall to give a single value. It is useful because it can act as a summary of performance, and can be calculated from previously published results.

$$F_1 = \frac{2 * P * R}{P + R} \tag{20}$$

Another metric is Mean Average Precision (MAP) (Kaggle.com, 2015). This metric depends on the order of the items being recommended to users. MAP is calculated by taking the arithmetic mean of the average precision for each user being evaluated. Average precision is given by the following equation:

$$AP(n) = \frac{\sum_{k=1}^{n} P(k)}{n} \tag{21}$$

This looks at the top-n items recommended, where n indicates the number of recommendations being made for the user, and *P(k)* is the precision for the kth item.

This means that MAP takes the relative ranking of the recommended locations into consideration. All other factors held constant, a correct prediction in the second position will receive a higher score than one in the third.

We have included Utility from the LFBCA paper (Wang, Terrovitis, & Mamoulis, 2013). It measures the fraction of users for which one of the recommendations is correct. This is equivalent to the fraction of users for which precision is above zero, and is shown in Equation (22). This metric is not common in the existing literature, having been defined in the LFBCA paper. We include it for completeness of comparison, and because it provides insight into how many users find the recommendations helpful.

$$Utility = \frac{\sum_{i=1}^{n} |P(u_i) > 0|}{n} \tag{22}$$

We include a metric for coverage. The use of coverage is based on considering which fraction of users for which any recommendation can be made. (Note that Coverage and Utility each measure a different level of usefulness to the end user.) We calculate prediction coverage on a test set as the fraction of locations visited, L_V, for which we have a non-zero prediction for the test user-location pair, as shown in the following equation:

$$User\ Coverage = \frac{\sum_{k=1}^{n} |Predictions(u_i) > 0|}{n} \tag{23}$$

For comparison of performance measures, we make use of the Wilcoxon Signed-Rank Test for statistical significance. We use this test because the results are paired over the datasets. As has been stated in (Demšar, 2006), the Wilcoxon test is preferable to

a paired t-test because it does not assume the data is normally distributed and should be less impacted by outliers. The test provides us with p-values which indicate statistical significance. We will use the traditional value of $p<0.05$ as the type I error rate (Shani & Gunawardana, 2011).

Following is a summary of the mechanics of the Wilcoxon Signed Rank Test. Note that a paired set of values for comparison are required. First, we calculate the difference between each pair of values.

$$x_{Di} = x_{Ai} - x_{Bi} \tag{24}$$

We rank the differences by their absolute magnitude, assigning 1 to the smallest difference, and iterating through all of the pairs. The rank at index i is multiplied by the sign of x_{Di}. This means that the rank is negative whenever $x_{Bi} > x_{Ai}$. The ranks are then denoted by R_i. Two intermediate statistics are calculated, W^+ and W^-. These are the sum of the ranks with the matching sign.

$$W^+ = \sum R_i \, , [R_i > 0] \tag{25}$$

$$W^- = -\sum R_i \, , [R_i < 0] \tag{26}$$

The real test statistic is W, calculated as shown in Equation (27).

$$W = \min(W^+, W^-) \tag{27}$$

This test statistic can be checked against a table of critical values to determine statistical significance. (Zaiontz, 2016) However, when there is a small number of comparisons, the p-value can be calculated exactly. This requires enumerating through

the possible sets of ranks. The p value is calculated by counting the number rank combinations that result in an equal or smaller test statistic, and dividing by all possible combinations. If we take N to be the total number of paired samples, there are 2^N possible combinations.

$$p = \frac{\sum |W(R_i) \leq W|}{2^N} \tag{28}$$

For effect size, we make use of Cliff's delta, d (Cliff, 1996). This measures the magnitude of the performance difference between two groups. As shown in Equation (29), this is the probability that a measure in set A is superior to a measure in set B, minus the probability that the reverse is true. Empirically this is calculated over all of the measures in each set.

$$d = \mathrm{P}\left(x_{Ai} > x_{Bj}\right) - P(x_{Ai} < x_{Bj}) \tag{29}$$

We also include a paired Cliff's delta, d_p, shown in Equation (30). The paired version only compares the results which are paired due to the experimental setup.

$$d_p = \mathrm{P}(x_{Ai} > x_{Bi}) - \mathrm{P}(x_{Ai} < x_{Bi}) \tag{30}$$

The values for Cliff's delta range from 1 to -1. The extremes indicate total superiority of the first and second set respectively.

We make use of the R software environment for calculating the previously mentioned statistical measures. For the Wilcoxon-Signed Rank test we use 'wilcox.test' in the core 'stats' package (R Core Team, 2015). For calculating the Cliff's delta values for effect size, we use 'dmes' in the package 'orddom' (Rogmann, 2013).

3.4.3 Experimental Design

For developing our algorithm and performing parameter exploration we make use of the Brightkite training dataset described in Section 3.4.1. We make use of a training methodology similar to the one demonstrated in (Wang, Terrovitis, & Mamoulis, 2013). The dataset is divided into temporal snapshots, identified by a number of days elapsed from the start of the dataset. Check-ins prior to the specified day are training data, and novel check-ins over the next sixty days are used as test data. This method of dividing the data is used for multiple reasons. First, it maintains the sequential order of each user's check-ins, which maintains any inherent behaviour of the users. Dividing the dataset into snapshots allows for multiple points of reference to evaluate how the algorithm handles the dataset as it evolves over time. The users who make novel check-ins during the test period are used for evaluating recommender performance. Unless otherwise stated, performance is based upon making a top-10 recommendation for each of the test users. As this training dataset is the only dataset originating from the Brightkite LBSN, we ensure that our parameter exploration is completely independent from the data used when making comparisons to other algorithms. As a result, we have not tuned our parameters individually for each testing dataset.

For comparison against existing algorithms, we recreate their experimental setup. This includes using the same dataset, as well as any necessary filtering. Consequently, the algorithm will still undergo a comparable training phase to learn the behaviours of the datasets specific users. The only change is that to reduce the variance which may be caused by the random nature of the classification component, our performance results are averaged over ten independent trials. In addition to including the same performance metrics as the existing algorithms, we provide measures for statistical significance and effect size.

3.5 Experimental Results

3.5.1 Individual Components

In this subsection, we cover the performance of the individual components on the training dataset. The performance is evaluated across multiple snapshots of the dataset. As discussed in Section 3.4.3, the evaluation is based upon the components ability to make a top-10 recommendation for each user who made a novel check-in during the subsequent sixty day testing period.

For the temporal component, purely temporal recommendation consists of recommending only the most recently visited items across all users. Table VI: Temporal Performance shows this temporal recommender is completely ineffective for a majority of the experiment, and does not demonstrate high precision or recall when it can make recommendations.

Table VI: Temporal Performance

Day	Precision	Recall
90	6.59848E-05	6.63328E-05
120	2.82646E-05	3.09224E-05
150	0	0
180	0	0
210	0	0
240	0	0
270	0	0
300	0	0

The spatial component recommends the locations closest to the user's previous check-in. The results of using solely spatial recommendation are shown in Table VII: Spatial

Recommendation. The performance of the spatial component is significantly better than the temporal method. The performance metrics are higher and at no point was the component unable to provide recommendations.

Table VII: Spatial Recommendation

Day	Precision	Recall
90	0.006631475	0.006666446
120	0.004494064	0.004916664
150	0.003570408	0.00392773
180	0.00234287	0.002752488
210	0.002899126	0.002875374
240	0.003741801	0.003475107
270	0.003553438	0.003427086
300	0.00343018	0.003534641

For the social component, we perform collaborative filtering with the set of nearest neighbours restricted to the current user's friends. This is similar to the design in (Ye, Yin, & Lee, 2010). The results of this method are shown in Table VIII: Social Recommendation. The performance metrics remain consistently higher than the earlier alternatives.

Table VIII: Social Recommendation

Day	Precision	Recall
90	0.017024084	0.01711386
120	0.015065008	0.016481648
150	0.01179663	0.012977219
180	0.007929714	0.009316113
210	0.009769658	0.009689617
240	0.011642815	0.010812981
270	0.011980939	0.011554925
300	0.012287214	0.0126614

The collaborative filter component incorporates the earlier components, as discussed in Section 3.3.4. The results of this component are shown in Table IX: Collaborative Recommendation. Once again the performance of this method exceeds that of the previous methods.

Table IX: Collaborative Recommendation

Day	Precision	Recall
90	0.019465523	0.019568174
120	0.018456755	0.020192337
150	0.01459583	0.016056559
180	0.010610498	0.012465594
210	0.011914218	0.011816606
240	0.013565891	0.012598992
270	0.014717495	0.014194175
300	0.014411878	0.014850767

The low performance of the temporal component on its own suggested the performance results of the classification component could be improved by adjusting its use. The results of removing the temporal component input as is shown in Table X. Classification Component - Removing Time Input. This corresponds to removing "Temporal Prediction" from the Classification Attributes listed in Table III. Removing this input is completely detrimental and was not done in the remaining experiments. The results suggests it is beneficial for the classification component to use temporal information.

Table X. Classification Component - Removing Time Input

Day	Original	No Time
90	626	538
120	537	402
150	399	278
180	333	212
210	459	299
240	706	451
270	730	446
300	791	475

As discussed at the end of Section 3.3.7, we develop the concept of a time window for the classification component. The results of using various sizes of windows are shown in Table XI. Again, with this design rather than training the classification component on all of the check-in data, only check-ins which have occurred within a specified amount of time are used. The table shows the performance increasing as the window gets smaller, peaking at a single day. Decreasing the size of the time window to half a day demonstrated a significant drop in performance. Because of these results, the classification component makes use of the time window with the length of one day.

Table XI. Correctly Predicted items based on Time Window (Days)

Day	0.5	1	2	3	7	14	21	28	Full
90	516	728	719	703	679	659	667	627	626
120	533	718	628	658	651	643	615	562	537
150	413	572	553	541	525	500	451	448	399
180	352	523	501	478	486	445	457	446	333
210	492	704	696	664	652	594	583	626	459
240	781	1104	1008	1038	959	943	919	840	706
270	857	1195	1117	1110	1093	1053	998	904	730
300	921	1318	1253	1242	1187	1112	1169	980	791

3.5.2 Parameter Exploration

This section covers the parameter exploration performed on our training dataset. The experimental methodology again follows the design outlined in Section 3.4.3. For expedience, parameter exploration was performed on the snapshot with 300 training days. The parameter exploration is performed sequentially as presented.

We use parameter exploration to determine the ideal value for S_C. Parameter exploration was performed by taking the results of the collaborative component, and the final results of the classification component. The results of the parameter exploration are shown in Table XII. The collaborative component does not appear to be very sensitive to changes in the value of S_C, so long as it is above zero. Based on the results, the best value for S_C is 0.40, based upon the final predictions. As such we only use S_C = 0.40 for our algorithm for all subsequent testing. Note that all parameter

exploration is done on the Training dataset to preserve out-of-sample testing for the remaining datasets.

Table XII: Parameter Exploration of Social Constant

	Collaborative			Final		
S_C	Precision	Recall	F measure	Precision	Recall	F measure
0	0.0036	0.0037	0.0037	0.0087	0.0090	0.0089
0.05	0.0142	0.0146	0.0144	0.0167	0.0173	0.0170
0.1	0.0143	0.0147	0.0145	0.0166	0.0171	0.0169
0.15	0.0143	0.0147	0.0145	0.0167	0.0172	0.0169
0.2	0.0144	0.0148	0.0146	0.0168	0.0173	0.0171
0.25	0.0144	0.0148	0.0146	0.0166	0.0171	0.0169
0.3	0.0144	0.0149	0.0147	0.0167	0.0172	0.0170
0.35	0.0145	0.0149	0.0147	0.0169	0.0174	0.0171
0.4	0.0145	0.0149	0.0147	0.0171	0.0176	0.0174
0.45	0.0145	0.0149	0.0147	0.0163	0.0168	0.0166
0.5	0.0145	0.0149	0.0147	0.0167	0.0172	0.0170
0.55	0.0145	0.0149	0.0147	0.0165	0.0170	0.0167
0.6	0.0145	0.0149	0.0147	0.0165	0.0170	0.0168
0.65	0.0145	0.0149	0.0147	0.0166	0.0172	0.0169
0.7	0.0145	0.0149	0.0147	0.0163	0.0168	0.0165
0.75	0.0145	0.0149	0.0147	0.0164	0.0169	0.0166
0.8	0.0145	0.0149	0.0147	0.0164	0.0169	0.0167
0.85	0.0144	0.0149	0.0146	0.0165	0.0170	0.0168
0.9	0.0144	0.0149	0.0146	0.0168	0.0173	0.0170
0.95	0.0144	0.0148	0.0146	0.0168	0.0173	0.0171

Exploration of the neighborhood size, k, for the collaborative component is shown in Table XIII. The best results occur when k = 125. Only for small neighborhood size is

there a significant change in performance. The values for the final predictions do not smoothly decrease from this maximum, and a smaller local maximum exists at $k = 40$.

Table XIII. Exploration of Neighborhood Size

	Collaborative			Final		
k	Precision	Recall	F-Measure	Precision	Recall	F-Measure
10	0.011593	0.011946	0.011767	0.014924	0.015378	0.015148
20	0.013599	0.014013	0.013803	0.015525	0.015998	0.015758
30	0.014399	0.014838	0.014615	0.016741	0.017251	0.016992
40	0.014847	0.015299	0.01507	0.017138	0.01766	0.017395
50	0.015186	0.015649	0.015414	0.016485	0.016987	0.016733
75	0.01549	0.015962	0.015722	0.017151	0.017673	0.017408
100	0.015528	0.016002	0.015761	0.017471	0.018003	0.017733
125	0.015647	0.016124	0.015882	0.017509	0.018042	0.017772
150	0.015596	0.016071	0.01583	0.017484	0.018016	0.017746
200	0.015535	0.016008	0.015768	0.017304	0.017831	0.017564
250	0.015394	0.015863	0.015625	0.017061	0.017581	0.017317
300	0.015369	0.015837	0.015599	0.017228	0.017752	0.017486

The results for varying the number attributes for each node in the random Forest's trees is shown in Table XIV. Based on the data, using a single attribute gives the best results. There is a smaller local maximum which occurs when using seven attributes, but the performance is lower across all metrics than when using one or two attributes.

Table XIV. Random Forest Attributes Performance

Attributes	Precision	Recall	MAP
1	0.017471	0.018003	0.067234
2	0.017362	0.017891	0.064443
3	0.016658	0.017165	0.060731
4	0.016831	0.017343	0.060949
5	0.016402	0.016902	0.057328
6	0.016255	0.01675	0.05867
7	0.016703	0.017212	0.056591
8	0.01564	0.016117	0.053465
9	0.015704	0.016183	0.052646
10	0.01564	0.016117	0.053066

The results of exploring set size, Z_C, the number of locations passed from the collaborative and social components for consideration by the classification component, is shown in Table XV. The best results occur when the size is set to 30. The next highest values for precision and recall occur with a size of 70, while the second highest MAP is with a size of 10. Overall there is not an obvious correlation to set size and the variation in performance is relatively small.

Table XV. Classification Location Set Size Exploration

Size	Precision	Recall	MAP
10	0.017348	0.017876	0.062741
15	0.017284	0.017810	0.061186
20	0.017349	0.017878	0.062125
25	0.017325	0.017853	0.061955
30	0.017410	0.017940	0.062825
40	0.017295	0.017822	0.061194
50	0.017368	0.017897	0.061490
75	0.017407	0.017937	0.062597
100	0.017352	0.017881	0.062100

3.5.3 Comparison

In this section, the CoCo recommendation algorithm we created is compared against other existing novel POI recommenders. We will illustrate why the algorithms we compare against represent the state-of-the-art competitors. To do this we will explain why various alternative novel POI recommenders are not included in the detailed comparison. Many potential competitors use datasets which we were unable to replicate often due to gathering the dataset independently. An example of this is the FMFMGM algorithm suggested by (Cheng, Yang, King, & Lyu, 2012), which uses a variant the Gowalla dataset, that we could not reproduce, with fewer users and locations. Their dataset covers 50% more time and the division of training data is made randomly, not chronologically.

Many recommenders only look at subsets of the existing LBSN datasets. One example is (Yang, Zhang, Yu, & Wang, 2013), which covers two datasets derived from the

Foursquare LBSN. Their datasets are each geographically limited to a single city, and the larger of the two has only 2601 users and 2392 locations. Another example is in (Yuan, Cong, Ma, Sun, & Thalmann, 2013) which uses Gowalla and Foursquare datasets limited to specific geographic regions and consequently are an order of magnitude smaller than the datasets we use, with respect to the number of users, locations and check-ins. Again the selection of training and testing data is made randomly and may not preserve the inherent temporal behaviour of the dataset. That being said, when comparing we have higher precision and lower recall. Our lower recall would likely be caused by their filtering of the dataset.

Some other papers may not strongly establish their performance results. In (Liu, Fu, Yao, & Xiong, 2013) they use a Foursquare dataset with roughly half the number of locations and one third as many check-ins as our full Foursquare dataset. They initially report comparable precision but significantly lower recall than our own results. Their final results are relative performance against a random recommender. Their results show an almost 19 times improvement over their random recommender, when recommending 10 items. By their definition, a random top-10 recommender on our Foursquare dataset has a recall and precision of 0.0001 and 0.344 respectively. Following their equations we have more than 648 times relative improvement.

We found that the LFBCA algorithm demonstrated comprehensive performance results for their algorithm (Wang, Terrovitis, & Mamoulis, 2013). They use a large LSBN dataset and make use of multiple performance metrics to show their superiority of existing algorithms. Comparison against LFBCA is done on a Gowalla dataset. This dataset is selected as we were able acquire the same dataset described in their paper,

in order to directly compare results. They also have results on a Brightkite dataset. We do not compare against the algorithm on the Brightkite dataset as we were unable to acquire or reproduce a Brightkite dataset with similar totals of user, locations and check-ins. Their results come from treating all entries before a specific day as training data, and using the next 60 days as test data. In addition, during the test period, predictions are only made for users who made at least one new check-in. We have recreated the same process using our algorithm. Figure II illustrates some relevant characteristics of the dataset over the time. The data for Figure II comes from the 60 day test period following each snapshot. It is worth noting that both the number of active users, and the number of novel check-ins are monotonically increasing across all snapshots of the dataset.

Figure II: LFBCA Gowalla User Activity

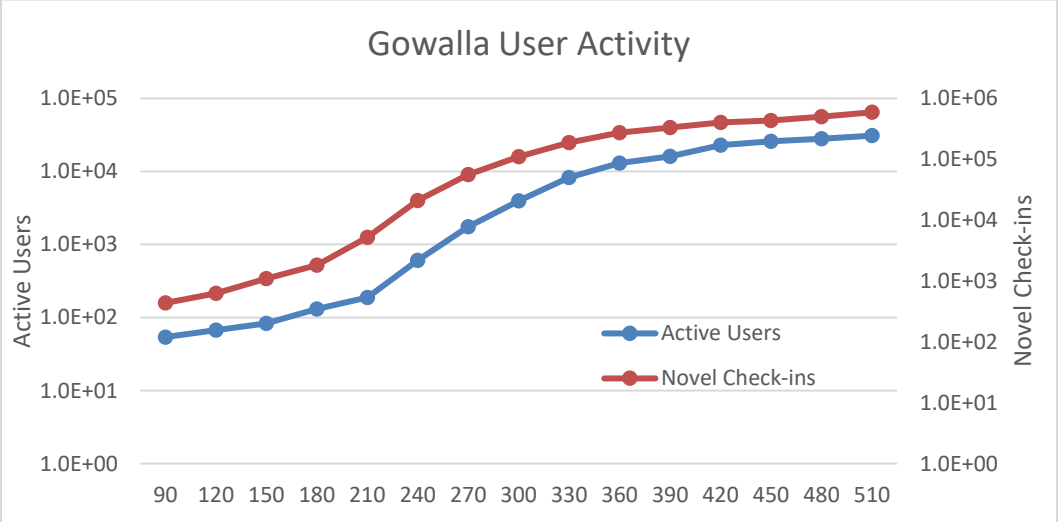


Figure III: LFBCA Gowalla F-Measure Comparison

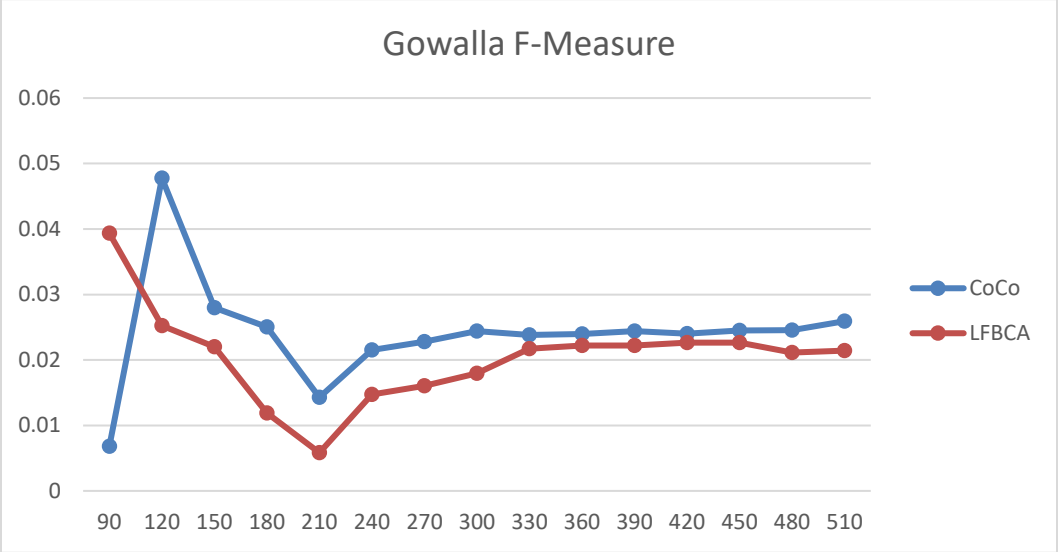


Figure III compares our performance to the LFBCA algorithm. Based on the results, our algorithm is outperformed on the first snapshot, with only 90 days of training data. After the initial result, our design outperforms LFBCA with respect to the F-measure. Full comparison of performance metrics by day are shown in Table XVI and Table XVII. Results of statistical tests for significance and effect size estimation are in Table XVIII. It is worth noting the characteristics of the first snapshot, when 90 days have elapsed. Comparing with Figure III it can be observed that this occurs when the dataset is the least active. At this time there are 54 active users, and together they will make 436 novel check-ins. Based on the difference in recall, the LFBCA was able to correctly predict roughly 15 more locations than CoCo at this time. Due to this being the first snapshot, and the low number of active users, this would appear to be our algorithm suffering more from the cold-start problem.

Table XVI. LFBCA Gowalla Accuracy Comparison

	CoCo			LFBCA		
Day	Precision	Recall	F measure	Precision	Recall	F measure
90	0.00617	0.00765	0.00683	0.037	0.042	0.039342
120	0.04627	0.04936	0.04777	0.023	0.028	0.025255
150	0.03253	0.02452	0.02796	0.022	0.022	0.022
180	0.03003	0.02151	0.02506	0.011	0.013	0.011917
210	0.02698	0.00971	0.01428	0.007	0.005	0.005833
240	0.04811	0.01385	0.02151	0.019	0.012	0.01471
270	0.04804	0.01495	0.02280	0.021	0.013	0.016059
300	0.04620	0.01657	0.02439	0.025	0.014	0.017949
330	0.03901	0.01715	0.02382	0.03	0.017	0.021702
360	0.03731	0.01768	0.02399	0.029	0.018	0.022213
390	0.03750	0.01809	0.02441	0.029	0.018	0.022213
420	0.03310	0.01887	0.02403	0.028	0.019	0.022638
450	0.03282	0.01956	0.02451	0.028	0.019	0.022638
480	0.03428	0.01913	0.02456	0.028	0.017	0.021156
510	0.03778	0.01974	0.02593	0.029	0.017	0.021435

Our method shows a statistical improvement over LFBCA on the Gowalla dataset. Using the Wilcoxon signed-rank test, we have a significant p-value of 0.008362 for both precision and f-measure. The change in recall is also significant with a p-value of 0.02557. Across all snapshots, our precision and recall are an average 71.1% and 16.3% higher respectively. Our F measure is on average 32.8% higher.

Table XVII. LFBCA Gowalla Coverage Comparison

	CoCo			LFBCA	
Day	MAP	Utility	User Coverage	Utility	User Coverage
90	0.005489418	0.030864198	0.6851852	0.16	1.0
120	0.112377285	0.1840796	0.7761194	0.13	1.0
150	0.08681671	0.16064257	0.8313253	0.12	1.0
180	0.10521704	0.162849867	0.8244275	0.08	1.0
210	0.08380374	0.153439153	0.8042328	0.05	1.0
240	0.13959388	0.244663383	0.84400654	0.10	1.0
270	0.13589366	0.27385024	0.89737743	0.13	1.0
300	0.131132777	0.274752053	0.93292993	0.16	1.0
330	0.112407722	0.25403924	0.96410006	0.19	1.0
360	0.11000608	0.2472066	0.9739539	0.19	1.0
390	0.110951857	0.255324	0.9812058	0.20	1.0
420	0.099323978	0.233099013	0.9865083	0.20	1.0
450	0.09709301	0.233450783	0.9927595	0.20	1.0
480	0.100383928	0.239068407	0.99379414	0.20	1.0
510	0.114454813	0.262711143	0.99466777	0.22	1.0

Table XVIII. LFBCA Gowalla Statistics

	Precision	Recall	F measure	Utility	Coverage
p-value	0.008362	0.02557	0.008362	0.0053711	0.00006104
d	0.7511	0.1378	0.5111	0.6	-1
d_p	0.8667	0.6	0.8667	0.8667	-1

With the exception of the first snapshot, our method's utility remains higher than LFBCA, with an average improvement of 53%. The associated p-value is 0.0053711. Our method shows statistically lower coverage with a p-value of 0.000061. Note that

although a simple average for our coverage is 89.9%, when the values are weighted by the number of active users, this rises to 98.4%. The disparity between utility and user coverage suggests that despite our method not being able to recommend locations to every single user, more users would find our recommendations useful. The effect size values for coverage indicate maximum preference for the LFBCA results. For other metrics, using just Cliff's delta shows a small effect size in our favor for recall, with larger values for precision, f-measure and utility. With the paired Cliff's delta, d_p, the effect size values are larger, and the value for recall is again lower than the other measures.

We found the LURA algorithm to be another excellent algorithm for comparison (Lu, Wang, Mamoulis, Tu, & Cheung, 2015). They provide performance comparison against several other existing methods to show the effectiveness of their method. For comparison they cover two of the most common performance metrics, precision and recall. Comparison against LURA is performed on the GSCorr–LURA and Gowalla-LURA datasets, described in Section 3.4.1. The comparison is made over these datasets as these are the datasets for which they published results, and we have been able to acquire. We follow their test design, using all data before a specified date for training, and the following 60 days for testing. Their results are based on finding the recall and precision when recommending a varying number of locations, N.

Table XIX. LURA GSCorr Comparison

	CoCo				LURA		
N	Precision	Recall	F measure	MAP	Precision	Recall	F measure
5	0.06482	0.04237	0.05125	0.06714	0.06	0.04	0.048
10	0.04923	0.06437	0.05579	0.05222	0.046	0.06	0.05207547
15	0.04126	0.08092	0.05465	0.04445	0.04	0.08	0.05333333
20	0.03655	0.09558	0.05288	0.03945	0.035	0.09	0.0504
25	0.03337	0.10908	0.05111	0.03511	0.03	0.1	0.0461538

Table XX. LURA GSCorr Statistics

	Precision	Recall	F measure
p-value	0.0625	0.0625	0.0625
d	0.2	0.2	0.76
d_p	1	0.6	1

The comparison on GSCorr occurs with training ending on day 300. The Wilcoxon p-value for precision, recall and F-measure are consistent at 0.0625. Although this is not a significant p-value, it is the smallest p-value value possible with the number of data points. Overall we have 13.2% higher precision, 12.4% higher recall, and 12.9% higher F measure. Our algorithm also delivers 100% user coverage on this dataset. Cliff's delta, or the unpaired effect size, is lower for precision and recall than f-measure. Looking at the paired effect size, value for recall is lower than both precision and f-measure, which achieved the maximum value of 1.

Table XXI. LURA Gowalla Comparison

	CoCo				LURA		
N	Precision	Recall	F measure	MAP	Precision	Recall	F measure
5	0.04444	0.01209	0.01901	0.09996	0.04	0.011	0.01725490
10	0.03901	0.02122	0.02749	0.11252	0.033	0.02	0.02490566
15	0.03559	0.02905	0.03199	0.11862	0.029	0.025	0.02685185
20	0.03278	0.03567	0.03416	0.11977	0.026	0.03	0.02785714
25	0.03067	0.04171	0.03535	0.12159	0.025	0.035	0.02916667

Table XXII. LURA Gowalla Statistics

	Precision	Recall	F measure
p-value	0.0625	0.0625	0.0625
d	0.52	0.28	0.52
d_p	1	1	1

The comparison on Gowalla occurs with training ending on day 420. The p-value for precision, recall, and F-measure are all 0.0625. The results show that we perform better than LURA on all measures. Our precision is always at least 11.1% higher, and recall is at least 6.12% higher. Our F measure is similarly at least 10.1% greater. On average we have 20.1% higher precision, 14.1% higher recall, and 16.7% higher F measure. Additionally, our method demonstrates 99.7% user coverage on this dataset. With regards to effect size, with the paired Cliff's delta, d_p, we show complete superiority with a value of 1 for all performance measures. With the unpaired Cliff's delta the effect size is larger for precision and f-measure than for recall.

A final comparison is made against LRT (Gao, Tang, Hu, & Liu, 2013). Their results showed superiority over collaborative filtering and matrix factorization methods. We use their algorithm for comparison to help demonstrate robustness as their experimental methodology differs significantly from other novel POI algorithms. The comparison against LRT is only possible on a single dataset, GSCorr Subset. We follow their evaluation format. Each row in Table XXIII has a testing percentage, T%, and a recommendation size, N. The testing percentage indicates the fraction of locations for each user to be included in the test set. Performance values for each row are averaged over 5 separate runs due to the randomized format of the data. We only include the results of the LRT Voting strategy as it has their highest results for both precision and recall. Our results show superiority versus their algorithm. Our algorithm had an average user coverage of 99.9%. The results for statistical significance and effect size are shown in Table XXIV. Note that although the p-value is not significant, it is the smallest possible p-value for this number of examples. It is worth noting that here we consistently achieve the maximum possible value for effect size, 1.

Table XXIII. LRT GSCorr Subset Comparison

		CoCo				LRT		
T%	N	Precision	Recall	F measure	MAP	Precision	Recall	F measure
20	5	0.09976	0.07337	0.08455	0.2840	0.0147	0.0171	0.0158094
20	10	0.06834	0.10060	0.08139	0.2923	0.0134	0.0311	0.0187299
40	5	0.16121	0.06230	0.08987	0.4226	0.032	0.0179	0.0229579
40	10	0.11049	0.08547	0.09638	0.4254	0.03	0.0335	0.0316535

Table XXIV. LRT GSCorr Statistics

	Precision	Recall	F measure
p-value	0.125	0.125	0.125
d	1	1	1
d_p	1	1	1

3.6 Conclusions

In this chapter, we discuss the CoCo algorithm for novel point of interest recommendation. Our method utilizes social, temporal and spatial analysis to provide context within a location based social network. Temporal data is used to apply greater weighting to the more recent, more relevant locations. Spatial data is used to weight locations by their geographical proximity to the user. Social data is used to provide additional weighting to the locations visited by a user's friends. This information is fed to both collaborative filtering and classification components. The classification component trains a random forest on the output of the other components to generate a final result. The classification component's final predictions are ordered by rank to produce a top-N list. We utilize sampling techniques to improve performance on the heavily imbalanced LBSN datasets.

Against existing alternatives on identical datasets, our algorithm shows statistically significant improvements in precision and recall when there are enough points to have statistical significance. We have the lowest possible p-values in cases when it is not statistically possible to have significant results. We have large, and often maximum,

effect size in paired comparison across multiple metrics. Our design demonstrates high coverage across the sparse datasets. We include several metrics to demonstrate the effectiveness of our design.

4. Summary and Future Work

In this thesis, we have discussed recommendation algorithms for the purpose of novel point-of-interest recommendation on location-based social networks. Various challenges exist that inhibit the performance of recommenders in this domain. We propose a design for mitigating the problems. Our algorithm considers social, spatial and temporal contextual data, which are utilized by both a collaborative filter and classification algorithm. Experimentally, we have an overall increase in predictive accuracy, compared to existing recommendation algorithms on the same datasets.

Future work in this area could include working with location tags or categorization which can indicate similarity between locations. If the data is being pulled from a source such as Twitter, where users leave public messages, their messages could be analyzed to construct an additional preference vector. Such preferences could be mapped to location tags. Additional work could be done to allow for datasets that make use of explicit user ratings. Spatial prediction could be enhanced by referencing street maps to more accurately determine effective distance of locations. This could be further enhanced by including related factors such as historical traffic data, road conditions, and construction work. Within certain domains, social friendship may be augmented by looking at the communication between users, such as retweets, mentions, and replies on Twitter. The use of temporal context could be expanded in several ways.

This would include focusing on specific temporal factors that relate to human behaviour such as time of day and day of the week. Additional weather-based context could be added, to see how temperature, precipitation, and wind affect user behaviour. Exploration of alternate algorithms or parameters for the classification component could also be used to enhance performance.

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*(6), 734-749.

Adomavicius, G., & Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook* (pp. 217-253). Springer.

Airoldi, E. M., & David M. Blei, S. E. (2008). Mixed membership stochastic blockmodels. *" Journal of Machine Learning Research, 9*(Sep), 1981-2014.

Altergeo. (2016, September 7). *Products*. Retrieved from Altergeo: http://platform.altergeo.ru/index.php

Amatriain, X., & Basilico, J. (2012, April 6). *Netflix Recommendations: Beyond the 5 stars (Part 1)*. Retrieved from The Netflix Tech Blog: http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

Anand, D., & Bharadwaj, K. K. (2011). Utilizing various sparsity measures for enhancing accuracy of collaborative recommender systems based on local and global similarities. *Expert systems with applications, 38*(5), 5101-5109.

Bawa-Cavia, A. (2011). Sensing the urban: using location-based social network data in urban analysis . *Pervasive PURBA Workshop*.

Berkhin, P. (2006). Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics, 3*(1), 41-62.

Billsus, D., & Pazzani, M. J. (1998). Learning Collaborative Information Filters. *Icml*, *98*, pp. 46-54.

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems. 46*, pp. 109-132. Elsevier.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (pp. 43-52). Morgan Kaufmann Publishers Inc.

Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5-32.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction, 12*(4), 331-370.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research 16*, 321-357.

Chen, P., Lu, Z., & Gu, J. (2009). Vehicle travel time prediction algorithm based on historical data and shared location. *2009 Fifth International Joint Conference on INC, IMS and IDC* (pp. 1632-1637). IEEE.

Cheng, C., Yang, H., King, I., & Lyu, M. R. (2012, July). Fused Matrix Factorization with Geographical and Social Influence in Location-Based Social Networks. *Aaai, 12*, 17-23.

Cheng, C., Yang, H., Lyu, M. R., & King, I. (2013). Where You Like to Go Next: Successive Point-of-Interest Recommendation. *IJCAI, 13*, pp. 2605-2611.

Cho, E., Myers, S. A., & Leskovec, J. (2011). Friendship and mobility: user movement in location-based social networks. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '11)* (pp. 1082-1090). New York, NY: ACM. doi:http://dx.doi.org/10.1145/2020408.2020579

Cliff, N. (1996). Answering Ordinal Questions with Ordinal Data Using Ordinal Statistics. In *Multivariate Behavioral Research* (pp. 331-350). Routledge. doi:10.1207/s15327906mbr3103_4

Cochran, W. G. (1953). *Sampling techniques.* John Wiley & Sons.

Cramer, H., Rost, M., & Holmquist, L. E. (2011). Performing a check-in: emerging practices, norms and'conflicts' in location-sharing using foursquare. *Proceedings of the 13th international conference on human computer interaction with mobile* (pp. 57-66). ACM.

Crandall, D. J., Backstrom, L., Cosley, D., Suri, S., Huttenlocher, D., & Kleinberg, J. (2010). Inferring social ties from geographic coincidences. *Proceedings of the National Academy of Sciences, 107*(52), 22436-22441.

Crunchbase. (2015, September 1). *Organization Brightkite: Overview*. Retrieved from Crunchbase: https://www.crunchbase.com/organization/brightkite#/entity

Crunchbase. (2016, August 11). *Organization Gowalla: Overview*. Retrieved from CrunchBase: https://www.crunchbase.com/organization/gowalla#/entity

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research, 7*, 1-30.

Deshpande, M., & Karypis, G. (2004). Item-based top-n recommendation algorithms.". *ACM Transactions on Information Systems (TOIS), 22*(1), 143-177.

Díaz-Uriarte, R., & De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics, 7*, 1.

Ding, Y., & Li, X. (2005). Time weight collaborative filtering. *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 485-492). ACM.

Domingos, P. (1999). Metacost: A general method for making classifiers cost-sensitive. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 155-164). ACM.

Drummond, C., & Holte, R. C. (2003). C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. *Workshop on learning from imbalanced datasets II*, *11*.

Duff, I. S. (1977). A survey of sparse matrix research. *Proceedings of the IEEE, 65*(4), 500-535.

Eagle, N., Pentland, A. S., & Lazer, D. (2009). Inferring friendship network structure by using mobile phone data. *Proceedings of the national academy of sciences, 106*(36), 15274-15278.

Facebook, Inc. (2016, July 27). *Facebook Reports Second Quarter 2016 Results*. Retrieved from Facebook Investor Relations: https://investor.fb.com/investor-news/press-release-details/2016/Facebook-Reports-Second-Quarter-2016-Results/default.aspx

Foursquare. (2016a, April 11). *Privacy 101*. Retrieved from foursquare.com: https://foursquare.com/privacy

Foursquare. (2016b, April 16). *Since Foursquare launched in 2009*. Retrieved from The Foursquare Blog: http://blog.foursquare.com/post/142900756695/since-foursquare-launched-in-2009-there-have-been

Foursquare. (2016c, September 6). *What if I forgot to or couldn't check in somewhere?* Retrieved from support.foursquare.com: https://support.foursquare.com/hc/en-us/articles/211542067-What-if-I-forgot-to-or-couldn-t-check-in-somewhere-

Foursquare. (2016d, June 20). *Why are Foursquare and Swarm separate apps?* Retrieved from support.foursquare.com: https://support.foursquare.com/hc/en-us/articles/202630254-Why-are-Foursquare-and-Swarm-separate-apps-

Gao, H., & Liu, H. (2014). *Location-Based Social Network Data Repository*. Retrieved from http://www.public.asu.edu/~hgao16/dataset.html

Gao, H., Tang, J., & Liu, H. (2012). Exploring Social-Historical Ties on Location-Based Social Networks. *ICWSM.*

Gao, H., Tang, J., Hu, X., & Liu, H. (2013). Exploring temporal effects for location recommendation on location-based social networks. *Proceedings of the 7th ACM conference on Recommender systems (RecSys'13)* (pp. 93-100). New York, NY: ACM. doi:http://dx.doi.org/10.1145/2507157.2507182

Glas, A. S., Lijmer, J. G., Prins, M. H., Bonsel, G. J., & Bossuyt, P. M. (2003). The diagnostic odds ratio: a single indicator of test performance. *Journal of clinical epidemiology, 56*(11), 1129-1135.

Glueck, J. (2016, January 14). *Foursquare Direct: Foursquare Ushers In A New Era*. Retrieved from Medium: https://medium.com/foursquare-direct/foursquare-ushers-in-a-new-era-f52edb39af6#.7rc896u3n

Gowalla Incorporated. (2011, February 1). *Gowalla*. Retrieved from Gowalla: http://gowalla.com/

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, a. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations, 11*(1).

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering, 21*(9), 1263-1284.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS), 31*(3), 5-53. doi:10.1207/s15327906mbr3103_4

Hernando, A., Bobadilla, J., & Ortega, F. (2016). A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowledge-Based Systems, 97*, 188-202.

Holte, R. C., Acker, L., & Porter, B. W. (1989). Concept Learning and the Problem of Small Disjuncts. *IJCAI*, *89*, pp. 813-818.

Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (pp. 263-272). Ieee.

Huang, H., & Gartner, G. (2014). Using trajectories for collaborative filtering-based POI recommendation. *International Journal of Data Mining, Modelling and Management, 6*(4), 333-346.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: an introduction*. Cambridge University Press.

Jawaheer, G., Weller, P., & Kostkova, P. (2014). Modeling User Preferences in Recommender Systems: A Classification Framework for Explicit and Implicit User Feedback. *ACM Trans. Interact. Intell. Syst., 4*(2), 8:1-8:26. doi:http://dx.doi.org/10.1145/2512208

Jo, T., & Japkowicz, N. (2004). Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter, 6*(1), 40-49.

Kaggle.com. (2015, December 3). *Mean Average Precision*. Retrieved from Kaggle.com: https://www.kaggle.com/wiki/MeanAveragePrecision

Koenigstein, N., & Koren, Y. (2013). Towards scalable and accurate item-oriented recommendations. *Proceedings of the 7th ACM conference on Recommender systems* (pp. 419-422). ACM.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM, 40*(3), 77-87.

Korb, K. B., & Nicholson, A. E. (2010). *Bayesian artificial intelligence.* CRC press.

Koren, Y. (2010). Collaborative filtering with temporal dynamics. *Communications of the ACM, 53*(4), 89-97.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering , 30*(1), 25-36.

Kubat, M., & Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. *ICML*, *97*, pp. 179-186.

Leca, C.-L., Nicolaescu, I., & Rîncu, C.-I. (2015). Significant Location Detection & Prediction in Cellular Networks using Artificial Neural Networks. *Computer Science and Information Technology, 3*(3), 81-89.

Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing, 7*(1), 76-80.

Ling, C. X., & Sheng, V. S. (2011). Cost-sensitive learning. In *Encyclopedia of Machine Learning* (pp. 231-235). Springer.

Liu, B., Fu, Y., Yao, Z., & Xiong, H. (2013). Learning geographical preferences for point-of-interest recommendation. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1043-1051). ACM.

Liu, Y., Liu, C., Liu, B., Qu, M., & Xiong, H. (2016). Unified Point-of-Interest Recommendation with Temporal Interval Assessment. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1015-1024). San Francisco, California, USA: ACM.

Lu, Z., Wang, H., Mamoulis, N., Tu, W., & Cheung, D. W. (2015). Personalized location recommendation by aggregating multiple recommenders in diversity. *Proceedings of the Workshop on Location-Aware Recommendations (LocalRec2015) co-located with the 9th ACM Conference on Recommender Systems, (RecSys 2015)* (pp. 28-35). New York, NY: ACM.

Mannan, N. B., Sarwar, S. M., & Elahi, N. (2014). A new user similarity computation method for collaborative filtering using artificial neural network. *International Conference on Engineering Applications of Neural Networks* (pp. 145-154). Springer.

Monard, M. C., & Batista, G. E. (2002). Learning with Skewed Class Distrihutions. *Advances in Logic, Artificial Intelligence, and Robotics: LAPTEC 2002 , 85*, 173-182.

Monreale, A., Pinelli, F., Trasarti, R., & Giannotti, F. (2009). Wherenext: a location predictor on trajectory pattern mining. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 637-646). ACM.

Morawski, J., Stepan, T., Dick, S., & Miller, J. (2017). A Fuzzy Recommender System for Public Library Catalogs. *International Journal of Intelligent Systems*.

Moritz, H. (1980). Geodetic reference system 1980. *Journal of Geodesy, 54*(3), 395-405.

Nichols, D. (1998). Implicit rating and filtering. *Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering* (pp. 31-36.). Budapest: ERCIM.

Nilashi, M., Jannach, D., bin Ibrahim, O., & Ithnin, N. (2015). Clustering-and regression-based multi-criteria collaborative filtering with incremental updates. *Information Sciences, 293*, 235-250.

Noulas, A., Scellato, S., Lathia, N., & Mascolo, C. (2012). Mining user mobility features for next place prediction in location-based services. *2012 IEEE 12th International Conference on Data Mining* (pp. 1038-1043). IEEE.

Oard, D. W., & Kim, J. (1998). Implicit feedback for recommender systems. *Proceedings of the AAAI workshop on recommender system*, (pp. 81-83).

Papagelis, M., Rousidis, I., Plexousakis, D., & Theoharopoulos, E. (2005). Incremental collaborative filtering for highly-scalable recommendation algorithms. *International Symposium on Methodologies for Intelligent Systems* (pp. 553-561). Springer .

Park, M.-H., Hong, J.-H., & Cho, S.-B. (2007). Location-based recommendation system using bayesian user's preference model in mobile devices. *International Conference on Ubiquitous Intelligence and Computing* (pp. 1130-1139). Springer .

Pham, H., Hu, L., & Shahabi, C. (2011). GEOSO-a geo-social model: from real-world co-occurrences to social connections. *International Workshop on Databases in Networked Information Systems* (pp. 203-222). Springer .

R Core Team. (2015). R: A language and environment for statistical computing. Vienna, Austria. Retrieved from https://www.R-project.org/

Rendle, S., Freudenthaler, C., & Schmidt-Thieme, L. (2010). Factorizing personalized markov chains for next-basket recommendation. *Proceedings of the 19th international conference on World wide web* (pp. 811-820). ACM.

Robusto, C. C. (1957). The cosine-haversine formula. *The American Mathematical Monthly, 64*(1), 38-40.

Rogmann, J. J. (2013). orddom: Ordinal Dominance Statistics. University of Hamburg, Department of Psychology, Germany. Retrieved from https://CRAN.R-project.org/package=orddom

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). *Application of dimensionality reduction in recommender system-a case study.* DTIC Document, Minnesota University, Department of Computer Science, Minneapolis.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web* (pp. 285-295). ACM.

Scellato, S., Noulas, A., Lambiotte, R., & Mascolo, C. (2011). Socio-Spatial Properties of Online Location-Based Social Networks. *Proceeding of the 5th International AAAI Conference on Weblogs and Social Media.*

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative Filtering Recommender Systems. In *The Adaptive Web* (pp. 291-324). Springer.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. *Proceedings of the 25th annual international ACM SIGIR conference on Researchand development in information retrieval* (pp. 253-260). ACM.

Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook* (pp. 257-291). Springer US.

Shen, Y., & Jin, R. (2012). Learning personal+ social latent factor model for social recommendation. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1303-1311). ACM.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., . . . Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM, 56*, 116-124.

Sinha, R. R., & Swearingen, K. (2001). Comparing Recommendations Made by Online Systems and Friends. *DELOS workshop: personalisation and recommender systems in digital libraries*, *106*.

socialbakers. (2016, August 5). *10 Most Checked-In Facebook Places*. Retrieved from socialbakers: https://www.socialbakers.com/blog/479-10-most-checked-in-facebook-places

Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence 2009*, 4.

Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., & Feuston, B. P. (2003). Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of chemical information and computer sciences, 43*(6), 1947-1958.

The Nielsen Company. (2015, December 17). *Tops of 2015: Digital*. Retrieved from Nielsen: http://www.nielsen.com/us/en/insights/news/2015/tops-of-2015-digital.html

Tomek, I. (1976). Two modifications of CNN. *IEEE Trans. Systems, Man and Cybernetics, 6*, 769-772.

Twitter, Inc. (2016, August 18). *REST APIs*. Retrieved from Twitter Developer: https://dev.twitter.com/overview/documentation

Ugander, J., Karrer, B., Backstrom, L., & Marlow, C. (2011). The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*.

Wang, H., Terrovitis, M., & Mamoulis, N. (2013). Location Recommendation in Location-based Social Networks using User Check-in Data. *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2013)* (pp. 374-383). New York, NY: ACM. doi:http://dx.doi.org/10.1145/2525314.2525357

Wang, Y., Yuan, N. J., Lian, D., Xu, L., Xie, X., Chen, E., & Rui, Y. (2015). Regularity and conformity: Location prediction using heterogeneous mobility data. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1275-1284). ACM.

Weiss, G. M. (2004). Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter, 6*(1), 7-19.

Williams, D. R. (2016, May 19). *Earth Fact Sheet*. Retrieved from NASA Space Science Data Coordinated Archive: http://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

Williams, J. (2011, December 5). *Going to Facebook*. Retrieved from Gowalla Blog: http://blog.gowalla.com/post/13782997303/gowalla-going-to-facebook

Yang, D., Zhang, D., Yu, Z., & Wang, Z. (2013). A sentiment-enhanced personalized location recommendation system. *Proceedings of the 24th ACM Conference on Hypertext and Social Media* (pp. 119-128). ACM.

Ye, J., Zhu, Z., & Cheng, H. (2013). What's your next move: User activity prediction in location-based social networks. *Proceedings of the SIAM International Conference on Data Mining*. SIAM.

Ye, M., Yin, P., & Lee, W.-C. (2010). Location recommendation for location-based social networks. *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems* (pp. 458-461). ACM.

Ye, M., Yin, P., Lee, W.-C., & Lee, D.-L. (2011). Exploiting geographical influence for collaborative point-of-interest recommendation. *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 325-334). ACM.

Yu, Y., & Chen, X. (2015). A survey of point-of-interest recommendation in location-based social networks. *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.*

Yuan, Q., Cong, G., Ma, Z., Sun, A., & Thalmann, N. M. (2013). Time-aware point-of-interest recommendation. *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (pp. 363-372). ACM.

Zaiontz, C. (2016, August 7). *Wilcoxon Signed-Ranks Table*. Retrieved from Real Statistics Using Excel: http://www.real-statistics.com/statistics-tables/wilcoxon-signed-ranks-table/

Zheng, Y., Xie, X., & Ma, W.-Y. (2010). GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull., 33*(2), 32-39.