Imperial College of Science, Technology and Medicine
Department of Computing

Practical Challenges of Learning and Representation for Large Graphs

Benjamin Paul Chamberlain

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College, September 19, 2018

I hereby declare that this thesis, unless indicated by reference or in the acknowledgements, is my own original work and that no element of it has previously been submitted to any university to be considered for a degree or other qualification.

Abstract

An ever increasing amount of the humanity's information is being stored in large graphs. The world wide web, digital social networks, e-commerce platforms and chat networks now contain digital traces of the majority of living humans. Many of the most valuable companies ever created are dedicated to organising, managing and extracting useful information from large digital graphs. Machine learning has been shown to be an important tool for automating this task. Discovering scalable machine learning systems, to extract useful information from graphs, is a problem of great practical significance.

Interesting graphs, such as the web graph, often contain more information than can be stored on a single computer, and so working with the raw data presents considerable challenges. Graph representations are often employed that encapsulate key properties of the underlying data and enable certain tasks to be performed efficiently, at the expense of others. Representations are chosen to balance time complexity, space complexity and predictive performance on a downstream task, such as labelling vertices with attributes.

We are concerned with problems of extracting and inferring information from large graphs and applying the results in deployed commercial systems. The research revolves around two large-scale machine learning projects (1) A system for searching and organising data from social media graphs (2) A system to profile customers through their interactions with products on an e-commerce platform.

We use representations of graphs to allow algorithms to be run faster, cheaper and more accurately. Doing so allows us to satisfy systems constraints that could not be achieved by operating directly on the raw data. We demonstrate how careful choices of representation can be used to improve machine learning performance on several real-world tasks. We do this under challenging industrial constraints such as real-time serving, runtime costs or maintainability.

Acknowledgements

I would like to acknowledge the many people who supported me in the completion of this work. Above all I would like to thank Dr. Marc Deisenroth for accepting me as his first PhD student at Imperial College. Marc has repeatedly acted in ways that are far beyond what is expected of a PhD advisor. I will forever be grateful for his counsel, his diligence and his rigour. Some of which, I hope, he has instilled in me. I would like to thank my collaborators, Marc, Josh, Clive, Angelo, Bryan, Roberto, Duncan and James for your many fine ideas, enlightening conversation and for making deadline nights much more bearable. I am grateful for the attention to detail and the selfless willingness to help of the people who reviewed this thesis, Hugh Salimbeni, Genevieve Barr, Elaine Bettaney, Marc Deisenroth, Michael Reed, Sesh Kumar and James Clough. There are also a large number of unnamed colleagues, friends and family who endured absences and chronic lapses in humour and social skills. Thank you for sticking with me during this time.

This work was generously funded by an Industrial Fellowship of the Royal Commission for the Exhibition of 1851 and aided by the support of Starcount Insights and ASOS.com.

Finally, I would like to thank Annabel for her ceaseless patience, support and understanding.

‘One accurate measurement is worth more than a thousand expert opinions’

Admiral Grace Hopper

‘The most profound technologies are those that disappear. They weave themselves into the fabric of our everyday life until they are indistinguishable from it’

Mark Weiser

‘I have not failed, I have just found 10,000 ways that did not work’

Thomas Edison

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Eleven years before the dawn of the twenty-first century, a document was left on the desk of Mike Sendall, who was working as an engineer at a large European physics laboratory. It was innocuously titled, 'Information Management: A proposal', and Sendall's early review was mixed. He wrote "vague but exciting" on the cover. Sendall was a senior software engineer at CERN, and the paper was written by Tim Berners-Lee. Within it were the ideas that grew into the world wide web, a graph stretching the globe that would enable the information age of mankind.

Seven years after Berners-Lee's seminal paper, Larry Page and Sergey Brin began studying ways in which to organise information on the nascent web. Conventional search engines ranked pages by counting the occurrences of key words on a site, but the rankings were brittle and easily manipulated. Page and Brin realised that a far more robust *search engine* could be built by considering the global structure of the webgraph. Their PageRank algorithm (Page et al., 1999) is the canonical example of the power of learning algorithms for graph-structured data.

While the early web was mainly used to consume information from a small number of publishers, the second epoch, often called web 2.0, heralded the arrival of mass publication. The most notable children of web 2.0 were the great social networks and e-commerce platforms that now hold significant influence over how humans consume both digital and physical products. Vast datasets have been captured to profile users and supply personalised recommendations of products, friends and news articles. The datasets are heterogeneous in nature, but principal within them are giant graphs describing the relationship between users of social networks or between customers and products of e-commerce platforms.

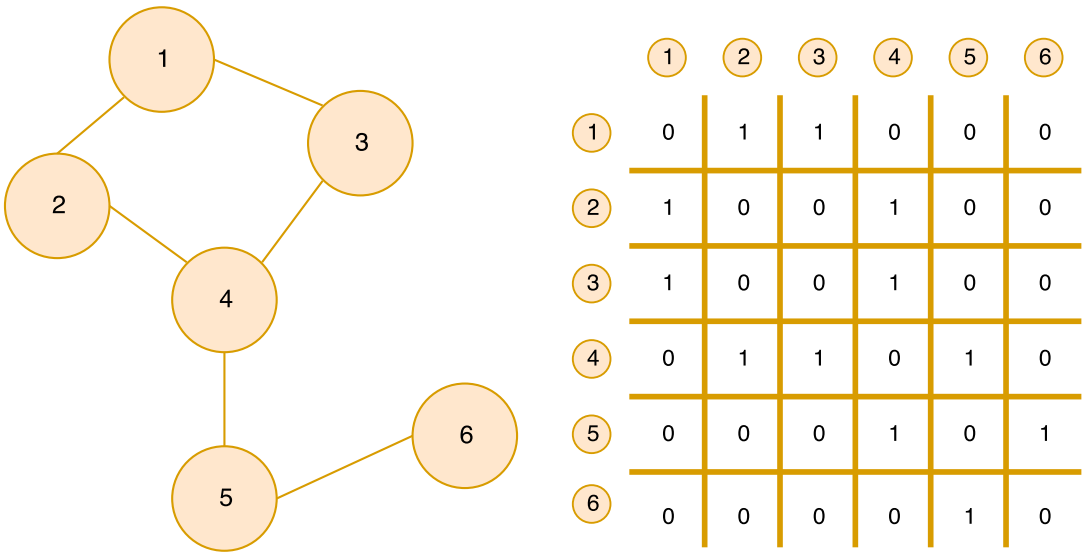


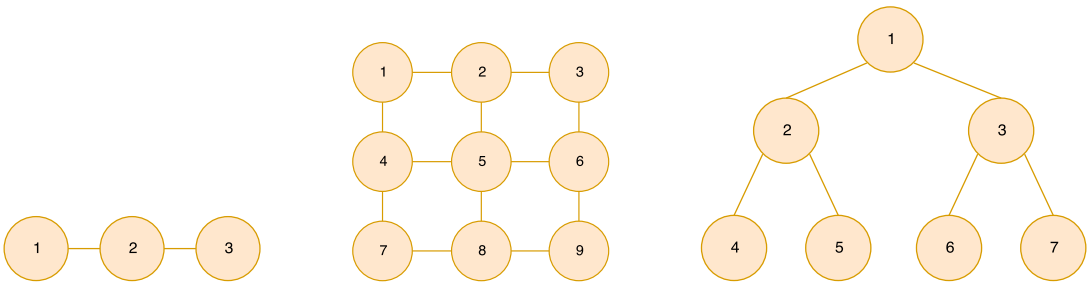
Figure 1.1: An undirected unweighted graph and the equivalent binary symmetric matrix representation

The size of such graphs means that working directly with the raw datasets presents considerable challenges. For this reason representations are often employed that encapsulate key properties of the underlying data. Learning also differs from the usual statistics paradigm as vertices in a graph are related in complex ways instead of being independent and identically distributed. This thesis is concerned with solving new problems of learning and representation that have emerged as humans attempt to make sense of the giant digital graphs spun into web 2.0. We are interested in research that can be deployed in the short to medium term. Scalability is an important factor, but there are additional factors that are rarely the subject of machine learning research. In addition to extracting accurate information, we are interested in evaluating the cost of running systems against the benefit they deliver. We are also constrained by the need to be maintainable and end user requirements such as latency.

The remainder of this chapter contains the preliminary concepts, a summary of the published work and a precis of the remaining chapters.

1.1 Graph-structured Data

Graphs are a general data structure defined by a collection of vertices and a collection of edges between vertices $G(V, E)$ where $E = (V_i, V_j)$. Edges come in two forms: directed—where the ordering of (V_i, V_j) conveys information and undirected—where it does not. Often we deal with



(a) A list (b) An image (c) A tree

Figure 1.2: Graph representations of common data structures. We show graphs for small lists 1.2a, images 1.2b and trees 1.2c

attributed graphs where there is additional meta-data describing either the vertices or edges. The most common form of attribute is an edge weight and graphs with edge weights are often called *weighted graphs*. Many forms of data can be considered to be special cases of graphs. A list is a graph in which each vertex connects to its two neighbours. An image is a graph where each pixel is a vertex that is connected to its four neighbours. A tree is a graph in which a vertex is connected only to its parents and its children. Small examples of each of these are shown in Figure 1.2.

The graphs shown in Figure 1.2 are simple *regular graphs*. Graphs of human behaviour are generated by random processes and are irregular. The simplest type of random graph is the Erdős-Renyí graph (Erdős and Renyí, 1960). In the Erdős-Renyí model, an edge between any two vertices is generated independently with constant probability. As each edge is independent, Erdős-Renyí graphs are sometimes called *Poisson graphs* or *completely random graphs*. Studies of large scale graphs of human behaviour have revealed that they are not completely random and contain structure (Albert and Barabási, 2002). One of the most surprising results of recent network science is that the graphs that describe our social interactions, transport systems, financial transactions, communications infrastructure and internal biological functions all seem to share similar properties (Newman, 2003). Collectively these graphs are often known as *complex networks* in the literature (Albert and Barabási, 2002).

1.2 Properties of Complex Networks

Many of the properties of complex networks are described in Newman (2003). Three of the most important are the *small world effect*, the presence of communities or clusters and the distribution of vertex degrees.

The Small World Effect The shortest path between two vertices is the minimum number of vertices that must be touched on a journey between them. In real-world networks, the average shortest path is surprisingly low compared to completely random graphs of the same size. The effect was made famous by the work of Travers and Milgram (1967) who sent letters randomly to Americans and asked them to forward the message to a friend who they felt might know a target person. They were surprised by how few friends were needed to connect the initial recipient to the target. This work coined the phrase "six degrees of separation".

Communities or Clustering Complex networks tend to have sets of vertices that have many more interconnections than can be expected at random. This is because, unlike in the Erdős-Renyí model, the edge generation process is not independent. This is clearly true in social networks; friends of friends are likely to know each other.

Degree Distribution The *degree* of a vertex is the number of edges incident on it, or simply the number of direct connections in undirected networks. The distributions of the degrees of the vertices within a complex network often follow a *power law* distribution (Newman, 2005; Mitzenmacher, 2004). Power laws permit very large ranges of observed values compared to distributions such as Gaussians. Well known examples of two such distributions are human height and human wealth. The tallest humans are only twice the median height, while the wealthiest among us command fortunes of $\approx 10^9$ times the median. Power law distributions are said to be fat tailed due to the high probability associated with extreme values. They are also *scale free*. The wealth distribution looks the same if measured in pence, pounds or millions of pounds. This is not true for Gaussians, which possess an inbuilt scale defined by the variance. A popular model that generates power law degree distributions is *preferential attachment* (Mitzenmacher, 2004). Preferential attachment is a sequential model that assigns edges to vertices with a probability that is proportional to their degree. For this reason, it is colloquially known as a *rich get richer* model.

1.3 Social and Co-purchasing Networks

This thesis is concerned primarily with digital social networks and co-purchasing networks. Both are complex networks in the sense that they exhibit small world properties, clustering and have power law degree distributions.

1.3.1 Digital Social Networks

Social media data provides a record of global human interactions at an unprecedented scale. The size of social networks (e.g., Facebook or Twitter), with hundreds of millions of users and billions of connections, makes Digital Social Network (DSN) analysis hard, requiring vast computing resources distributed across multiple data centres to extract information. Owing to its simplicity, size and openness, Twitter is the most popular DSN for scientific research. For this reason the bulk of Chapter 5 and Chapter 6 use data mined from Twitter.

Twitter Data

Twitter allows users to generate data by *tweeting* a stream of 140 character (or less) messages. To consume content, users *follow* each other. Following is a one-way interaction, and for this reason Twitter is regarded as an *interest network* (Gupta et al., 2013). By default, Twitter is entirely public. In the context of Twitter, a vertex V is a Twitter account and a directed edge $E = (V_i, V_j)$ from V_i to V_j exists if V_i Follows V_j. We capitalize Follows throughout this thesis to indicate when the Twitter specific meaning of this word is being used. The Twitter graph has a large number of vertex attributes (e.g., name, description) and edge attributes (e.g., creation time, retweet count).

1.3.2 Co-purchasing Graphs

A co-purchasing graph contains two classes of vertices: products and customers. Edges only exist that connect customers to products and so these graphs are *bipartite*. Co-purchasing graphs can have very large numbers of vertex and edge attributes. Vertex attributes can include all meta-data known about the product or customer and this can vary with time. Edge meta-data includes the time and price of a purchase.

1.4 Representation Learning in Graphs

There is no single *best* representation for graph-structured data. Typically a representation is selected that offers a sweet spot in the trilemma balancing time complexity, space complexity and performance on a downstream task such as classifying attributes of a vertex.

Any matrix has a graph representation and vice-versa. Non-zero entries specify an edge between a vertex indexed by a column value and a vertex indexed by the row value. Symmetric matrices represent undirected graphs and binary matrices represent unweighted graphs. The matrix representation of a small graph is shown in Figure 1.1.

Small graphs are often represented by an adjacency matrix or the closely related Laplacian matrix. This is attractive as it opens up the graph domain to the machinery of linear algebra. Many of the most interesting properties of graphs follow from a spectral analysis of the Laplacian matrix (Spielman, 2007). However, for large, sparse graphs, the adjacency or Laplacian matrices are not practical representations as they have $O(|V|^2)$ space complexity. The Facebook social graph is a well known example of a large, sparse graph. It contains upwards of two billion accounts, but a typical user will only have a few hundred friends, equating to a sparsity of $\approx 1 \times 10^{-7}$. Storing a billion binary values to represent a users' missing edges is inefficient. Instead of an adjacency matrix, an edge list $E = (V_i, V_j)$ can be stored with $O(|V|)$ space complexity. An edge list representation makes storing large, sparse graphs practical, but at the expense of increasing the complexity of analysis. Trade-offs like this are at the heart of the problem of learning good representations of graphs.

Both the adjacency matrix and edge list representations are lossless. In many cases we do not need all of the information in the graph to be reconstructible from the representation. Representations that lose information so that the original data can not be completely reconstructed are called *lossy* representations. They can dramatically improve the time and space complexity of graph algorithms. The choice of lossy representation is governed by the properties of the graphs that need to be preserved.

We deal with two forms of lossy representations, global and local models. Global models represents the large-scale structure of a graph and are useful for tasks such as comparing the similarity of two graphs. We apply global methods in Chapter 5 to allow rapid querying, using a representation that preserves the large scale community / clustering structure. Local models learn representations based on information contained in small regions of a graph and are used to characterise vertices or small sub-graphs. They are useful for learning attributes of nodes and edges or for finding clusters or communities. In Chapters 6 and 7 we apply local graph representations to predict the ages of Twitter users and the future value of customers respectively. In Chapter 8 we extend the local representations to better incorporate the important properties of complex networks.

1.5 Collaborating Companies

This thesis is kindly funded by an Industrial Fellowship awarded by the Royal Commission for the Exhibition of 1851. Accordingly, it deals with research that is applied to solve industrial problems related to large social or e-commerce graphs. The composite elements are either deployed, or in the process of being deployed by two British companies: Starcount Insights and ASOS.com.

Chapters 5 and 6 were completed while I was employed by Starcount Insights. Starcount is a social media analytics firm based in London that fuses data from eleven major social networks to help companies understand their customer base and improve their marketing. Chapter 7 was written in collaboration with ASOS.com. ASOS is a global e-commerce company specialising in fashion and beauty for young adults. The business is entirely online, and products are sold through eight country-specific websites and mobile apps. At the time of writing, there were 15 million active customers and the product catalogue contained more than 85,000 items. Products are shipped to 240 countries and territories and the annual revenue for 2017 was £1.9B, making ASOS one of Europe's largest pure play online retailers.

1.6 Publications

This thesis relates to work that is also publicly available in the following shorter articles:

1. Chamberlain, B.P., Cardoso, A., Liu, C.H., Pagliari, R. and Deisenroth, M.P., 2017. Customer Lifetime Value Prediction Using Embeddings. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Pages 1753–1762*. I led the team developing the CLTV model, developed the concept and implementation for the customer embeddings and wrote the majority of the paper.

2. Chamberlain, B.P., Humby, C. and Deisenroth, M.P., 2017. Probabilistic Inference of Twitter Users' Age based on What They Follow. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Pages 191–203. Springer, Cham. I developed the concept and implementation and wrote the majority of the paper.

3. Liu, C.H., Chamberlain, B.P., Little, D.A. and Cardoso, A., 2017. Generalising Random Forest Parameter Optimisation to Include Stability and Cost. In Joint European Conference on

Machine Learning and Knowledge Discovery in Databases. Pages 102–113. Springer, Cham. I led the team developing the model and worked on the mathematical proofs and wrote the introduction.

4. Chamberlain, B.P., Levy-Kramer, J., Humby, C. and Deisenroth, M.P., 2016. Real-Time Community Detection in Large Social Networks on a Laptop. PLoS ONE 13(1): e0188702. I developed the concept and implementation and wrote the majority of the paper.

5. Chamberlain, B.P., Clough, J. and Deisenroth, M.P., 2017. Neural Embeddings of Graphs in Hyperbolic Space. The 13th International Workshop on Mining and Learning with Graphs hosted jointly with the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. I developed the concept and implementation and wrote the majority of the paper.

6. Predicting User Return Time. Grob G.L., Liu, C.H., Little, D.A. and Chamberlain, B.P., 2018. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham. I was the principal investigator and wrote the introduction.

7. Aletras, N. and Chamberlain, B.P., 2018. Predicting Occupation and Income from Neural Embeddings of the Twitter Network. HT '18 Proceedings of the 29th on Hypertext and Social Media. Pages 20–24. I developed the concept and implementation for the Twitter embeddings and wrote the sections pertaining to their use.

8. Liu, C.H., Chamberlain, B.P., 2018. Online Controlled Experiments for Personalised e-Commerce Strategies. I was the principal investigator and wrote the introduction.

1.7 Thesis Summary

This thesis is composed of eight chapters. The first four are introductory and the remaining four contain the original research elements.

Chapter 2 introduces the machine learning models and techniques that are used throughout the research elements.

Chapter 3 covers the underlying mathematical spaces of the graph representations used in the research elements. The relationships between different types of spaces and the constraints this places

on the representations are also explained here.

Chapter 4 provides a thorough introduction, including some history, for two forms of graph representations that we make frequent use of. These are minhash signatures and neural embeddings.

Chapter 5 presents an efficient method for querying social networks that imitates the function of market research operations such as focus groups, but using a fraction of the time and cost. The method embeds sub-graphs of large networks in a way that supports rapid exploration of local regions of the network using only commodity hardware. The key idea is that the aggregate actions of large numbers of users provide a robust measure of similarity in a large network and that this similarity structure can be compressed into a form where local querying is efficient.

Chapter 6 addresses the problem of learning the attributes of vertices in social networks using the structure of the graph. This is performed at massive scale using a Bayesian framework that consistently handles the inherent uncertainty in the data. We also show that attribute learning is improved by using a representation of vertices based on neural graph embeddings.

Chapter 7 takes the idea of a neural graph embeddings and advances the concept to make it useful for forecasting events occurring in disjoint periods of time. Embeddings of customers are learned from a network of customer product interactions and this representation is able to improve performance on the task of forecasting future spending behaviour.

Chapter 8 extends the concept of graph embeddings in a different direction, by shifting the underlying geometry. Hyperbolic geometry has been shown to provide a useful continuous analogue to the structure of complex networks. It is demonstrated that using hyperbolic geometry provides more compact representations that outperform Euclidean geometry when applied to vertex attribute prediction tasks.

Chapter 2

Background Machine Learning Methods

A broad array of Machine Learning (ML) methods are applied in this work. These include: logistic regression, Bayesian networks, random forests, support vector machines, feed forward neural networks, Gaussian processes and Bayesian optimization. Many excellent books are available offering detailed and insightful descriptions of these techniques e.g. (Friedman et al., 2001a; Murphy, 2014; Bishop, 2016). In this chapter, we provide introductory material describing Logistic Regression (LR), Support Vector Machines (SVM), Gaussian Processes (GP)s, Random Forests (RF)s and Bayesian Optimization (BO). These techniques are used in Chapters 7, 6 and 8. Some introduction to feed forward neural networks as they apply to embeddings models is given in Chapter 4.

Here, and throughout this thesis, bold font indicates column vectors while capitalisation denotes either a set or a matrix with any ambiguities resolved in text. We use the notation $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^n$ to refer to a training set, $X \in \mathbb{R}^{n \times k}$ for the design matrix and $\mathbf{y} \in \mathbb{R}^n$ for the corresponding labels that are the goal of the prediction problem. We use calligraphic font to indicate random variables. $\mathcal{Z}$ is a scalar random variable, $\underline{\mathcal{Z}}$ is a vector random variable and $\underline{\underline{\mathcal{Z}}}$ indicates a matrix.

2.1 Logistic Regression

Logistic regression is a linear technique for classification (Zou and Hastie, 2005). For binary labels $y_i \in \{0, 1\}$, the probability of the positive class is given by

$$P(\mathcal{Y} = 1 | \underline{\mathcal{X}} = \mathbf{x}) = \frac{1}{1 + e^{-z}} = \sigma(z) \tag{2.1}$$

$$z = \mathbf{w}^T \mathbf{x} + b \tag{2.2}$$

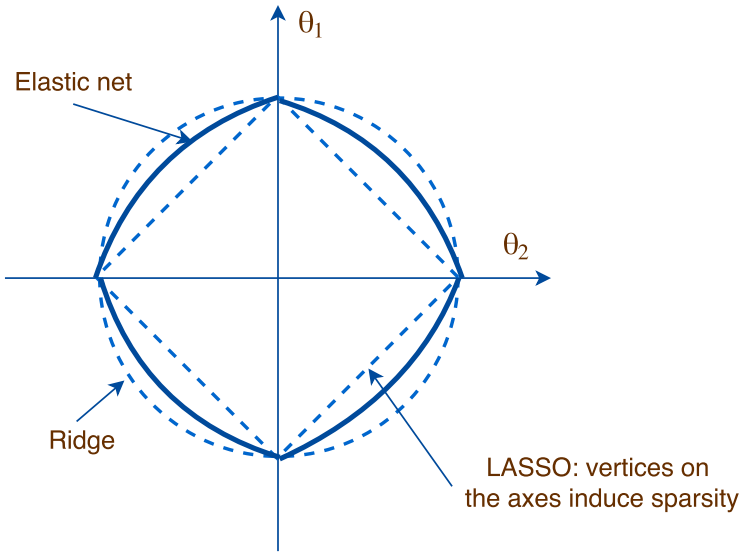


Figure 2.1: The regularisation constraints induced by applying ridge, LASSO and elastic net regularisation

where $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$ are the model parameters that are learned by *maximum likelihood estimation* over $\mathcal{D}$ and the *negative log likelihood* is given by

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \ln(\sigma(z_i)) + (1 - y_i) \log(1 - \sigma(z_i)) \,. \tag{2.3}$$

The negative log likelihood is convex and so the optimal parameters

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{\theta}) \tag{2.4}$$

can be uniquely found using gradient descent.

2.1.1 Regularisation

In practice, directly optimizing Equation (2.4) often leads to over-fitting and a model that fails to generalise to unseen data. Regularisation is used to tackle this problem by adding additional terms to Equation (2.4) that penalise large more complex models. Three types of regularisation are typically employed: (1) L1 or *LASSO*, (2) L2 or *ridge* and (3) a linear combination of both, which is known as

an *elastic net*. The elastic net loss function is

$$L(\boldsymbol{w}) = -\sum_{i=1}^{n} y_i \ln(\sigma(z_i)) + (1 - y_i) \log(1 - \sigma(z_i)) + \lambda_1 \sum_{j=1}^{k} |\theta_j| + \lambda_2 \sum_{j=1}^{k} \theta_j^2 \tag{2.5}$$

where λ_1 and λ_2 are hyperparameters that control the amount of regularisation. Ridge and LASSO are special cases of Equation (2.5) when λ_1 or λ_2 equal zero respectively. Ridge regression will tend to product smaller parameter values while LASSO encourages some θ_i to go to zero and so induces sparsity. A known limitation of LASSO is that it will tend to select only one variable from any group of highly correlated variables (Bishop, 2016). An elastic net alleviates this problem while still promoting sparsity.

2.2 Support Vector Machines

Support Vector Machines (SVM)s are supervised models for classification and regression (Joachims, 1998). We use SVMs to predict the occupations and incomes of Twitter users in Chapter 6. We include only a brief introduction and refer the interested reader to the books by Schölkopf and Smola (2002) or Bishop (2016).

SVMs are related to logistic regression, but the loss function is modified to learn a *large margin* decision boundary so that the distance between the boundary and any training datum is maximised. There are two components: (1) a linear classifier (2) a feature space transformation Φ. A two class linear model can be formulated as

$$z(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b \tag{2.6}$$

where for $y_i \in \{-1, 1\}$ non-probabilistic predictions are given by

$$\hat{y_i} = \operatorname{sgn}(z(\mathbf{x})) \tag{2.7}$$

Here we only consider data that is linearly separable in the feature space. The general case is handled by adding a *slack* variable as described in e.g. Bishop (2016). The maximum margin choice of parameters is given by

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \left(\frac{1}{\|\mathbf{w}\|} \min_i (y_i(\mathbf{w}^T \Phi(\mathbf{x}_i) + b) \right). \tag{2.8}$$

The optimization problem can be simplified by rescaling so that

$$y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) = 1 \tag{2.9}$$

for the training point closest to the decision boundary. There will always be at least two such points and these are called *support vectors*. This leads to the constrained optimization objective

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) - 1) \tag{2.10}$$

where α_i are Lagrange multipliers satisfying $\alpha_i \geq 0$. The associated *dual representation* is

$$\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{2.11}$$

where the kernel function is defined as $k(\mathbf{x}, \mathbf{x}') = \langle\Phi(\mathbf{x}), \Phi(\mathbf{x}')\rangle$ and predictions are given by

$$\hat{y}(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right), \tag{2.12}$$

where only the support vectors provide non-zero contributions to the sum. The data is projected into a *feature space* where an optimal linear decision boundary is found. As the mapping $\Phi : X \mapsto \mathbb{H}$ from the data space to the feature space can be nonlinear, SVMs are nonlinear in the data space. The data vectors only enter the SVM equations as inner products and so the *kernel trick* can be used to perform implicit inner product calculations in the data space without having to project every data point into a feature space that is usually high-dimensional. There are many choices of SVM kernel and Mercer's theorem defines valid kernels. The theorem states that k is a kernel if for some Φ, $k(\mathbf{x}, \mathbf{x}')$ is positive semi-definite defined by

$$\int k(\mathbf{x}, \mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0 \tag{2.13}$$

for all square-integrable functions $g(\mathbf{x})$. Or equivalently that the *Gram matrix*

$$\begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots \\ k(x_2, x_1) & & \\ \vdots & & \ddots \end{bmatrix}$$

is positive semi-definite for any collection $\{x_1, x_2, ..., x_n\}$. In Chapter 6 we use the Radial Basis

Function (RBF) kernel, which is given by

$$k(\mathbf{x}, \mathbf{x}') = \alpha \exp\left(-\frac{1}{2l^2}\|\mathbf{x} - \mathbf{x}'\|^2\right) \tag{2.14}$$

Here α and l are parameters determining the amplitude of the kernel and the *length scale* that determines how points are correlated.

2.3 Gaussian Processes

A Gaussian Process (GP) is a flexible tool for nonlinear regression and classification (Rasmussen and Williams, 2006). They are the default ML method for probabilistic, nonlinear regression, providing sufficient computational budget is available. Their major weakness is that the required budget can be large as the runtime for training is $O(n^3)$ and prediction is $O(n^2)$ where n is the number of training points. There is an extensive literature on GPs and here we supply only the introductory material required to explain Bayesian optimization in Section 2.5 and our use of this model in the second half of Chapter 6. A more complete exposition is provided by Rasmussen and Williams (2006).

A GP is a collection of random variables such that any finite subset have a jointly Gaussian distribution. A GP is fully defined by a mean and covariance *function* instead of a mean vector and covariance matrix, and we write

$$f \sim GP(m, k) \tag{2.15}$$

such that the function f is distributed as a GP with mean function m and covariance function k satisfying

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{2.16}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \tag{2.17}$$

The goal is to learn a function that can predict the output values $p(f_*)$ of future test inputs $\mathbf{x}_*$. This is tractable even though a GP is an infinite dimensional object because calculations are only made over the union of the training and test sets, which are always finite. A GP is a prior distribution on

such functions. We introduce

$$\mu_i = m(\mathbf{x}_i) \tag{2.18}$$

$$\Sigma_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.19}$$

to distinguish between a process and a distribution. In prediction we wish to estimate the posterior distribution of $\mathbf{f}_*$ for the test data given the input X and output $\mathbf{f}$ of the training data. The joint distribution is given by

$$p\left(\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\Sigma & \Sigma_* \\ \Sigma_*^T & \Sigma_{**}\end{bmatrix}\right) \tag{2.20}$$

where $\boldsymbol{\mu}_i = m(\mathbf{x}_i)$ for $i \in 1, 2..., n$, $\boldsymbol{\mu_*}$ is the corresponding quantity for the test points, Σ is the training set covariance, Σ_* the test-training covariance and Σ_{**} the test covariances. As Equation (2.20) is a joint Gaussian, applying the rules for Gaussian conditioning immediately leads to

$$\mathbf{f}_*|\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}_* + \Sigma_*^T\Sigma^{-1}(\mathbf{f} - \boldsymbol{\mu}), \Sigma_{**} - \Sigma_*^T\Sigma^{-1}\Sigma_*) \tag{2.21}$$

and a posterior process

$$f|\mathcal{D} \sim GP(m_{\mathcal{D}}, k_{\mathcal{D}}) \tag{2.22}$$

with

$$m_{\mathcal{D}}(x) = m(x) + \mathbf{k}(X, x)^T\Sigma^{-1}(\mathbf{f} - \mathbf{m}) \tag{2.23}$$

$$k_{\mathcal{D}}(x, x') = k(x, x') - \mathbf{k}(X, x)^T\Sigma^{-1}\mathbf{k}(X, x') \tag{2.24}$$

where $\mathbf{k}(X, x)$ is a vector of covariances between x and every training point X. In practice, f is often not directly observed and instead only a noisy value y is directly observable where the values of y are i.i.d. according to

$$y = f + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.25}$$

In this case we have

$$p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\Sigma + \sigma^2 I & \Sigma_* \\ \Sigma_*^T & \Sigma_{**}\end{bmatrix}\right) \tag{2.26}$$

where the off-diagonal terms do not change because of the independence assumption. Equation (2.22) is modified so that y is drawn from a GP of the form

$$y|\mathcal{D} \sim GP(m_{\mathcal{D}}, k'_{\mathcal{D}}) \tag{2.27}$$

$$k'_{\mathcal{D}}(x, x') = k(x, x') - \mathbf{k}(X, x)^T(\Sigma + \sigma^2 I)^{-1}\mathbf{k}(X, x') \tag{2.28}$$

So far we have just described GP regression. In Chapter 6 we also use a Gaussian Process Classifier (GPC). Inference for classification using GPs is more complex than regression as the assumption of a Gaussian likelihood is invalid when the outputs are discrete values. As a result the posterior is no longer a GP and can not be computed in closed form. In practice, this problem is solved by approximating the posterior with a GP (Hensman et al., 2015).

To adapt a GP to the task of classification the output must be mapped to a set of discrete values. For binary classification, GP outputs $f(\mathbf{x}) \in \mathbb{R}$ are often mapped through a logistic sigmoid

$$\sigma(f) = \frac{1}{1 + e^{-f(\mathbf{x})}}, \tag{2.29}$$

where $f \sim GP$. The distribution of a new test case from a GP is given by

$$p(f_*|X, \mathbf{x}, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|X, \mathbf{x}_*, \mathbf{f})p(\mathbf{f}|X, \mathbf{y})d\mathbf{f}. \tag{2.30}$$

For classification problems

$$p(\mathbf{f}|X, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X)}{p(\mathbf{y}|X)} \tag{2.31}$$

is not a Gaussian because $p(\mathbf{y}|\mathbf{f})$ is not Gaussian and Equation (2.30) is intractable. The probabilistic class prediction

$$\pi_* = \int \sigma(f_*)p(f_*|X, \mathbf{y}, \mathbf{x}_*)df_* \tag{2.32}$$

has no closed form solution. Common solutions are to approximate the joint posterior with a Gaussian using the Laplace approximation or Expectation Propagation (Minka, 2001).

2.3.1 Training

Training of a GP, which we define as being distinct from inference, is a two stage process. Firstly, families of functions must be specified for m and k. Secondly, the hyperparameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_m, \boldsymbol{\theta}_k\}$ associated with the function families are optimized with respect to the training data. The object of optimization is the *log-marginal likelihood*. Assuming a Gaussian likelihood the log-marginal likelihood can be computed analytically and is given by

$$L = \log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\log|\Sigma| - \frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{y} - \boldsymbol{\mu}) - \frac{n}{2}\log(2\pi) \tag{2.33}$$

where $\boldsymbol{\mu}$ depends on $\boldsymbol{\theta}_m$ and Σ depends on $\boldsymbol{\theta}_k$. The three terms of Equation (2.33) can be interpreted as (1) a term penalising model complexity, (2) a term that measures the quality of the fit to the training data and (3) a log normalization term. Locally optimal values of the hyperparameters are obtained by differentiating Equation (2.33) to give

$$\frac{\partial L}{\partial \theta_m} = (\mathbf{y} - \boldsymbol{\mu})^T\Sigma^{-1}\frac{\partial \boldsymbol{\mu}}{\partial \theta_m} \tag{2.34}$$

$$\frac{\partial L}{\partial \theta_k} = \frac{1}{2}\operatorname{Tr}\left(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}\right) + \frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^T\frac{\partial \Sigma}{\partial \theta_k}\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}(\mathbf{y} - \boldsymbol{\mu}) \tag{2.35}$$

and applying a gradient based numerical optimizer.

2.3.2 Hyperparameters

The hyperparameters of a GP are those that specify the form of the mean and covariance functions. The mean function is often set to zero and the covariance function or *kernel* controls the nature of the functions that can be inferred. There are many valid choices of kernels, but they must be able to generate positive definite covariance matrices.

A commonly used kernel is the exponentiated quadratic kernel (also known as the RBF, Gaussian kernel or the squared exponential) given in Equation (2.14). In Chapter 6 we use Automatic Relevance Determination (ARD) (Neal, 1995) with an RBF kernel. Instead of using a single length scale, ARD uses a separate learned length scale for each input dimension. Equation (2.14) becomes

$$k(\mathbf{x}, \mathbf{x}') = \alpha \exp\left(\sum_{i=1}^{k} -\frac{(x_i - x_i')^2}{2l_i^2}\right) \tag{2.36}$$

where α determines the amplitude of the latent function $\mathbf{f}$ and l_i are the *length scales*, which determine how points are correlated. The length scales are learned separately for each input dimension. A large l_i induces smoothness, implying that the output is less sensitive to changes in x_i. For this reason, a useful heuristic is to interpret the values of l_i as feature importances, where the features with low length scales are regarded as more important. However, this approach should be used with caution as features with high length scales can still contribute significantly to the predictive power of the model.

The exponential quadratic kernel is infinitely differentiable and produces a smooth function prior with infinitely differentiable sample paths. The Matérn kernel is another popular GP kernel, particularly for Bayesian optimization, that is finitely differentiable and is able to produce rough functions. The Matérn kernel is defined

$$k_\nu(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{2\sqrt{\nu}\|\mathbf{x} - \mathbf{x}'\|}{l} \right)^\nu H_\nu \left(\frac{2\sqrt{\nu}\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \tag{2.37}$$

where $\Gamma(.)$ and $H_z(.)$ are the Gamma and order ν Bessel functions respectively. ν controls the degree of smoothness of the GP and the kernel is $\lceil \nu \rceil - 1$ times differentiable. In the limiting case $\nu \to \infty$ the Matérn kernel becomes the squared exponential. Equation (2.37) is greatly simplified for half integer values of ν and in practice $\nu = 3/2$ is often used leading to

$$k_{3/2}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \exp \left(-\frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|}{l} \right). \tag{2.38}$$

2.4 Random Forests

Random Forests (RF)s are ensembles of decision trees (Breiman et al., 1984) that can be used to solve classification and regression problems (Criminisi, 2011). They are often used for practical applications because they can be trained in parallel, easily consume heterogeneous data types and achieve state of the art predictive performance for many tasks (Fernández-Delgado et al., 2014; Tamaddoni et al., 2016; Chamberlain et al., 2017a). We make extensive use of RFs in Chapter 7. Figure 2.2 depicts the process of generating predictions from a trained regression tree (the process is identical for classification trees). In the figure, the features are the age, number of items purchased and the tenure of an e-commerce customer. To make a prediction a series of binary functions are evaluated starting at the root of the tree. Each binary function determines the next function to be evaluated

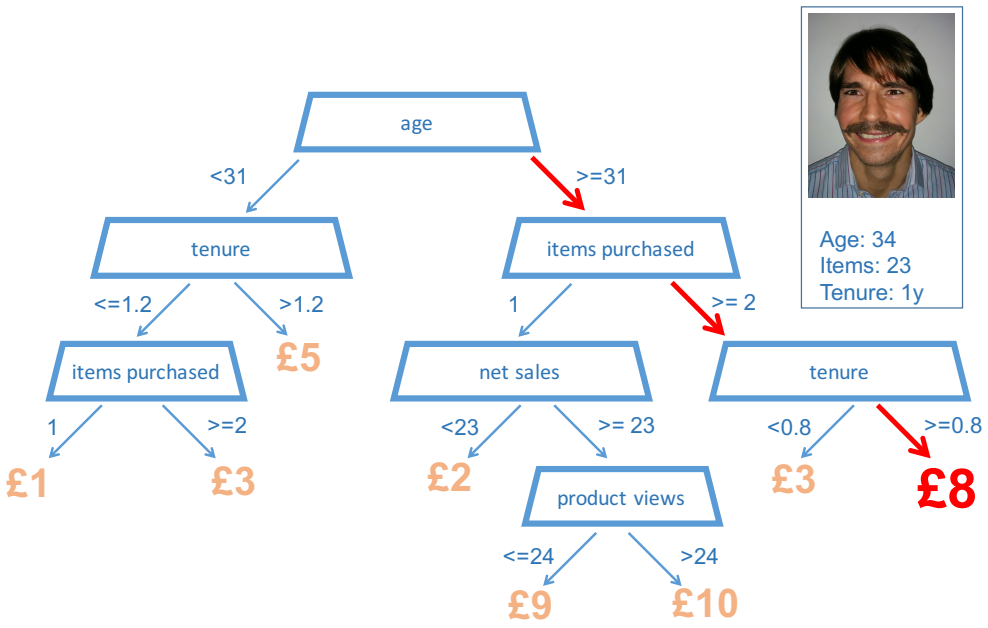


Figure 2.2: An illustrative regression decision tree for predicting how much a customer will spend on an e-commerce site given their age, tenure and the number of items that they have purchased. The data point starts at the root and after each splitting function is sent either right or left until it reaches a leaf where a predictive value is assigned.

until a leaf node is reached where a predictive value is stored. Prediction in RF models is achieved by aggregating the prediction of each composite tree. For classification problems there are several ways in which this is commonly done. Trees may store either a point estimate or a distribution in each leaf. For point estimates, the mode is often used. If probabilistic outputs are required, the mean estimate is usually taken of the form

$$p(c|\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|\mathbf{x}_i) \tag{2.39}$$

where T is the number of trees and $p_t(c|\mathbf{x_i})$ is the histogram distribution in the leaf of the t^{th} tree reached by the i^{th} datum. For regression problems the mean value of all trees is typically taken.

2.4.1 Training

Each constituent tree of a RF is trained independently. A tree is trained by learning a hierarchical organized collection of binary splitting functions in a greedy manner initiating at the root. At the j^{th} vertex, a subset of the training data $\mathcal{D}_j$ arrives and is split into $\mathcal{D}_j^{(L)}$ routed to the left child vertex and $\mathcal{D}_j^{(R)}$ routed to the right child by a binary splitting function f_j. The f_j are usually restricted to

simple functions, most commonly axis aligned splits of a single training feature, in which case

$$f(\mathbf{x}_i) = \mathbb{1}(\eta(\mathbf{x}_i) > \theta) \tag{2.40}$$

where η is a projection operator that selects a single dimension of a training point $\mathbf{x}_i$ and θ is a learned threshold. The splitting function at vertex j of the tree routes the data incident upon it $\mathcal{D}_j$ either to the right or left, which we denote $\mathcal{D}_j^{(R)}$ and $\mathcal{D}_j^{(L)}$ respectively. For classification problems, f_j is usually chosen to maximise the information gain

$$H(\mathbf{y}_j) - \sum_{i \in \{L,R\}} \frac{|\mathbf{y}_j^i|}{|\mathbf{y}_j|} H(\mathbf{y}_j^i) \tag{2.41}$$

where $\mathbf{y}_j$ are the labels at the j^{th} vertex, $\mathbf{y}_j^i$ are the labels sent to each child and $H(\mathbf{y})$ is the entropy given by

$$H(\mathbf{y}) = -\sum_{y \in \mathbf{y}} p(y) \log p(y) \tag{2.42}$$

and $p(y)$ is given by the empirical histogram of class labels in $\mathbf{y}$. The result is that the splitting function that maximises the purity of labels at each child vertex will be selected. For regression problems the mean squared error is maximised

$$\sum_{y \in \mathbf{y}_j} (y - \bar{y}_j)^2 - \sum_{y \in \mathbf{y}_j^{(L)}} (y - \bar{y}_j)^2 - \sum_{y \in \mathbf{y}_j^{(R)}} (y - \bar{y}_j)^2 \tag{2.43}$$

which corresponds to the splitting function that minimises the label variance of the child vertices.

When training RFs two forms of randomness can be used: (1) Sampling a subset of the training data to be used for each tree (2) Sampling a subset of the features over which to choose the splitting function at each node. The former is useful for parallelising training for large datasets as a sample of the training data can be sent to each processor independently. The latter is useful when working with large numbers of features as the search space of the optimization procedure that selects f_j is reduced.

2.4.2 Hyperparameters

The parameters of a random forest are the choice of splitting functions for each tree. Additionally there are a number of hyperparamters that control the nature of a RF. The most important are:

1. The number of trees
2. The maximum tree depth
3. The amount and type of randomness
4. The family of splitting functions
5. The training objective

For a RF to be effective, the hyperparameter values must be carefully selected (Huang and Boutros, 2016). The three most significant in our work are (1) the number of trees in the forest, (2) the maximum depth of each tree and (3) the type and amount of randomness. The generalised prediction performance of a RF increases monotonically with the number of trees, while the training time grows linearly. The maximum tree depth can be controlled by either setting it explicitly or by controlling the minimum number of data points at each leaf during training. Deeper trees are more complex and the computational cost increases exponentially with tree depth. The optimal depth for error reduction depends on the other RF hyperparamaters and the data; too much depth results in overfitting. However, if the forest is too shallow the model will underfit resulting in a high error rate. We restrict the form of randomness in the RF to sampling the training data for each tree. A reduction in the amount of training data sampled leads to shorter training times, reducing costs. However, reducing the amount of training data also reduces the generalisability of the model as the estimator sees less training examples, increasing predictive errors.

2.5 Bayesian Optimization

Bayesian Optimization (BO) is a nonlinear optimization framework that is often used for hyperparameter tuning in machine learning. We provide a brief overview here and a thorough introduction is available in Brochu et al. (2010). BO finds optimal parameter settings for unknown functions faster than competing methods such as random / grid search or gradient descent (Snoek et al., 2012) and it is particularly useful when the function is expensive to evaluate and is highly non-convex. In the context of this thesis, we use BO to search for hyperparameters that minimise the loss functions of machine learning models, particularly random forests, for given training and test datasets. We use the running example of predicting ASOS customer churn probabilities using a random forest to illustrate Bayesian optimization. In this case, the unknown function is a mapping from random forest

hyperparameters to the error rate on a held-out labelled test set (the loss function). Each evaluation of the loss function is expensive as it requires a complete model retraining on millions of data points (customers) and the loss function is highly non-convex.

The key idea is to perform a search over hyperparameter space that balances *exploration* (trying new regions of the space that we know little about) with *exploitation* (choosing parts of the space that are likely to contain the optimal hyperparameters). The loss function L is unknown and so it is modelled with a surrogate function. Normally this is achieved by placing a GP prior distribution on the mapping from hyperparameters to the loss and using the posterior mean of the GP as an estimate of the objective. The posterior distribution of the loss is given by Bayes law as

$$P(L|\mathcal{D}) \propto P(\mathcal{D}|L)P(L) \tag{2.44}$$

where L is the model loss as a function of the hyperparameters and $\mathcal{D}$ is the set of observations of L. We use a zero mean Gaussian Process (GP) for the prior on functions $P(L)$. The posterior distribution of any point of the surrogate function is then given by Equation (2.21). The next observation of L to add to $\mathcal{D}$ is determined by maximising an *acquisition function* that selects successive hyperparameter settings from the surrogate function by balancing high variance regions of the prior (good for exploration) with low mean regions (good for exploitation). The hyperparameters that minimise the posterior mean after a predefined number of query iterations are selected. Many different acquisition functions have been used and three of the most popular are (1) Lower Confidence Bound (LCB) (2) Probability of Improvement (PI), (3) Expected Improvement (EI). The LCB is a relatively simple measure that selects points with the lowest posterior lower confidence bound given by

$$LCB(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\sigma(\mathbf{x}) \tag{2.45}$$

where κ is a user-defined free parameter balancing exploration and exploitation that is often scheduled (Srinivas et al., 2010), $\mu(.)$ is the posterior mean and $\sigma(.)$ is the predictive uncertainty interval. The PI method chooses a sample point that maximises the probability of being better than the current best point $\mathbf{x}^+$. This probability is given by

$$PI(\mathbf{x}) = P\left(L(\mathbf{x}) \leq L(\mathbf{x}^+) - \xi\right) \tag{2.46}$$

$$= \Phi\left(\frac{\mu(\mathbf{x}) - L(\mathbf{x}^+) + \xi}{\sigma(\mathbf{x})}\right) \tag{2.47}$$

where $\Phi(.)$ is the CDF of the standard normal and ξ takes the role of the exploration-exploitation

parameter, which is sometimes sequentially reduced in the spirit of simulated annealing (Kushner, 1964). In practice it is difficult to choose a good value of ξ and values that are too low will result in extensive local exploration at the expense of more promising areas of the global distribution (Jones, 2001). In PI the search procedure selects sample points based on the probability that they are better than all previous sample points, but does not take account of how much better they are. EI resolves this issue by considering the size of the improvement

$$I(x) = \max\left(0, L(\mathbf{x}^+) - L(\mathbf{x})\right). \tag{2.48}$$

New samples are taken at the point that sequentially maximises the expected improvement

$$EI(\mathbf{x}) = \mathbb{E}(I) = \begin{cases} \left(L(\mathbf{x}^+) - \mu(\mathbf{x})\right)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \sigma(\mathbf{x}) > 0 \\ 0, & \sigma(\mathbf{x}) = 0 \end{cases} \tag{2.49}$$

$$Z = \frac{\mu(\mathbf{x}) + L(\mathbf{x}^+)}{\sigma(\mathbf{x})} \tag{2.50}$$

where $\phi(.)$ and $\Phi(.)$ are the PDF and CDF of the standard normal respectively and a full derivation of Equation (2.50) is available in Mockus et al. (1978). There are several noteworthy precursors to BO where elements of the procedure first appeared. One of the earliest was *kriging*, a process for interpolating a random field using a linear predictor (Krige, 1952). In kriging, errors are modelled with a GP. While krieging is closely related to BO, the fitting process is markedly different and does not rely on maximum likelihood. An approach similar to modern BO, but using Wiener instead of Gaussian Processes, was developed by Kushner (1964) who introduced a parameter that emulated the exploration-exploitation trade-off for one-dimensional problems. Later work by Mockus (1974) described multidimensional BO over linear combinations of Wiener fields and contains the firmament of the *acquisition function*.

Chapter 3

Mathematical Spaces

It is often useful to apply geometric principles to problems in machine learning. In this context, data exists in mathematical spaces and classifications or regressions are performed by measuring distances or similarities in the space of the data. In general, graphs, or the elements composing them, do not live in mathematical spaces. However, in this thesis we show that embedding graphs, or elements of graphs into mathematical spaces is a powerful tool for learning from graph-structured data. In this chapter we present the introductory definitions and properties of the mathematical spaces that are used later.

A mathematical space is a set with some additional structure. The elements x in a general set X are called points and can be any form of object. Here, and throughout this thesis, bold font indicates vectors while capitalisation denotes either a set or a matrix with any ambiguities resolved in text.

3.1 Vector Spaces

A general vector space (sometimes called a linear space) is a non-empty set V of points that is closed under addition and scalar multiplication. There are ten axioms that must be satisfied by all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$ and all $c, d \in \mathbb{R}$ for a space to be a vector space.

1. Closure under vector addition: $\mathbf{x} + \mathbf{y} \in V$
2. Commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
3. Distributivity: $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$

4. Neutral element with respect to vector addition: $\exists \mathbf{0} \in V : \forall \mathbf{x} \in V \, \mathbf{x} + \mathbf{0} = \mathbf{x}$
5. Inverse element with respect to addition $\forall \mathbf{x} \in V \, \exists - \mathbf{x} \in V : \mathbf{x} + (-\mathbf{x}) = \mathbf{0}$
6. Closure under scalar multiplication $c\mathbf{x} \in V$
7. Distributivity: $c(\mathbf{x} + \mathbf{y}) = c\mathbf{x} + c\mathbf{y}$
8. Distributivity: $(c + d)\mathbf{x} = c\mathbf{x} + d\mathbf{x}$
9. Distributivity: $(cd)\mathbf{x} = c(d\mathbf{x})$
10. Neutral element with respect to scalar multiplication: $1\mathbf{x} = \mathbf{x}$

where c and d are general scalars. In Chapters 6 and 7 we represent the vertices of graphs as elements of a vector space. Vectors are sequentially updated using the properties of closure under vector addition and scalar multiplication.

3.2 Metric Spaces

A metric defines a distance between a pair of points in a set. Let X be a non-empty set. A metric on X is a function $d : X \times X \to \mathbb{R}$ and X is a metric space if for all $x, y, z \in X$ was have:

1. Nonnegativity: $0 \leq d(x, y) < \infty$,
2. Uniqueness: $d(x, y) = 0$ if and only if $x = y$,
3. Symmetry: $d(x, y) = d(y, x)$
4. The Triangle Inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

Metric spaces are not necessarily vector spaces. A general metric space is just a set with a distance between each element. The elements do not need to be vectors. For example, consider the set X = {chair, apple}. If we define a metric d such that d(apple, apple) = d(chair, chair) = 0 and d(chair, apple) = d(apple, chair) = 1 then X is a metric space. In Chapter 8 we represent vertices as elements of a metric space that is not a vector space.

3.3 Norms and Normed Vector Spaces

While a metric provides a notion of the distance between points in a space, a norm provides the notion of the length of an individual vector. A norm can only be defined on a vector space, while a metric can be defined on arbitrary sets.

For a vector space V over the real field $\mathbb{R}$ a norm on V is a function $\|.\| : V \to \mathbb{R}$ such that for all vectors $\mathbf{x}, \mathbf{y} \in V$ and all scalars $c \in \mathbb{R}$ we have:

1. Nonnegativity: $\|\mathbf{x}\| \geq 0$
2. Homogeneity: $\|c\mathbf{x}\| = |c|\|\mathbf{x}\|$
3. Triangle Inequality: $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$
4. Uniqueness: $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$

A vector space together with a norm is called a *normed vector space*. As the distance between two vectors can be defined by $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ a normed vector space is always a metric space. However, given a metric $d(\mathbf{x}, \mathbf{y})$ it is not always true that $d(\mathbf{x}, 0)$ is a norm. Therefore a norm induces a metric, but the converse is not true.

3.4 Inner Product Spaces

An inner product space is a vector space V with an additional inner product structure $\langle ., . \rangle : V \times V \mapsto \mathbb{R}$ such that for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$

1. Linearity: $\langle c\mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = c\langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$.
2. Symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$.
3. Positive Definite: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ with equality if and only if $\mathbf{x} = \mathbf{0}$.

The inner product provides rigorous notions of concepts like lengths and angles between vectors in a vector space. The angle between two vectors is given by

$$\cos\theta = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{3.1}$$

and two vectors are orthogonal if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. An inner product induces a norm

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \tag{3.2}$$

which in turn induces a metric

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle} \tag{3.3}$$

The linearity and symmetry properties impose an additional bilinearity property. Using the bilinear property of the inner product we have that

$$d(\mathbf{x}, \mathbf{y})^2 = \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \tag{3.4}$$

$$= \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle \tag{3.5}$$

$$\therefore \langle \mathbf{x}, \mathbf{y} \rangle = \frac{1}{2}(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - d(\mathbf{x}, \mathbf{y})^2) \tag{3.6}$$

3.5 Summary

An inner product always induces a norm and a norm always induces a metric. However, the converse is not true and so mathematical spaces have a hierarchical ordering that is shown in Figure 3.1. In Chapter 6 and Chapter 7 we represent graphs by embedding them into inner product spaces, which are also metric and vector spaces. In Chapter 5 and Chapter 8 we learn representations of graphs

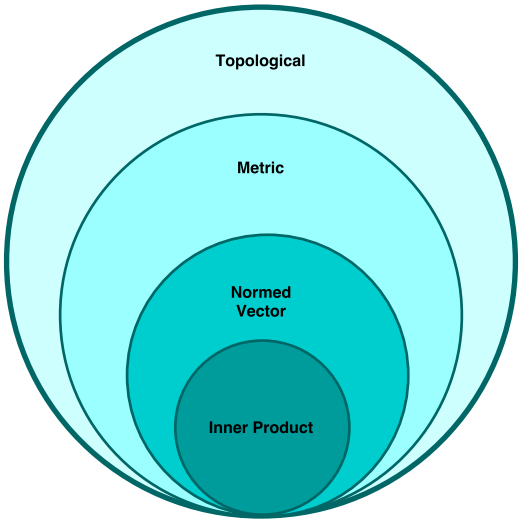


Figure 3.1: The hierarchical organization of mathematical spaces

by embedding them into metric spaces. These representations are not elements of a vector space as they do not obey all of the axioms defined in Section 3.1.

Chapter 4

Learning Representations of Graphs

Graphs where the vertices represent users, items or products, describe the relationships between categorical variables. To learn attributes of vertices in a graph they must be represented in a form that is admissible to a Machine Learning (ML) system. This is not a problem that is unique to the graph domain. Many datasets of great interest to the ML community are either categorical or contain categorical variables. Words take this form, as do moves in chess, objects in images and many more widely studied forms. An index is typically used as a space efficient representation of each class from a set of categories. Figure 4.1 shows an index based representation of a set of 20 shoes. A representation like this is problematic for most ML systems because algorithms implicitly interpret the difference between index values as a similarity.

To remove spurious similarities, a one-hot or one-of-k encoding is often utilised. This representation is partially illustrated for the same set of shoes in Figure 4.2. All pairs of representations are orthogonal, having zero cosine similarity and are separated by equal distances. While the one-hot encoding solves the problem of spurious class similarity, it has several major drawbacks. In the ex-

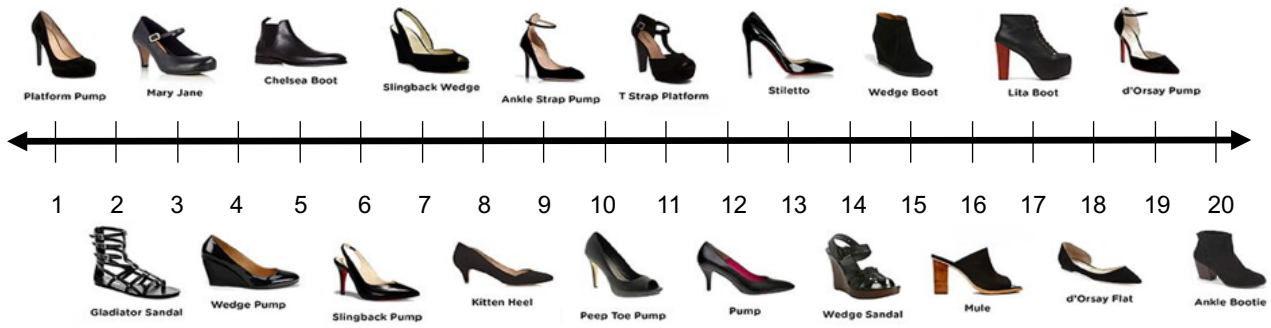


Figure 4.1: A naive representation of a set of shoes based on image indices

Figure 4.2: A one-hot encoding of a set of shoes

ample shown in Figure 4.2, the representation is 20 dimensional, but many datasets contain far more classes; a one-hot representation of English words has roughly 10^5 dimensions for instance. Very high-dimensional data is difficult to store and challenging to model. Many standard ML libraries add additional dummy columns for each one-hot dimension (Pedregosa et al., 2012), requiring $O(nd)$ space instead of $O(n)$, where d is the number of dimensions and n is the number of training examples. Equally problematic is the *curse of dimensionality*; the often observed phenomena that the amount of data required by a learning system grows exponentially with dimensionality and our 3D concepts of similarity break down (Domingos, 2012).

The one-hot representation uses high dimensions to enforce an egalitarianism of classes, while an index representation makes spurious similarity assertions in a single dimension. A natural solution to the problems faced by both is to find representations that uses fewer dimensions than there are classes and encapsulates data driven class similarities. Figure 4.3 shows a one-dimensional representation of shoes (only four are shown) where similar shoes are assigned similar valued representations. By encoding object similarity, it is possible to have compact representations that alleviate the curse of dimensionality and outperform on downstream predictive tasks.

In the one-dimensional case shown in Figure 4.3 the distance (which is the inverse of the similarity) between two representations is given by $D = v_i - v_j$ where v_i is the representation of a shoe. One dimensional representations are only useful for simple data and to describe complex graphs, more dimensions are needed. In this thesis, we work with three distinct compact representations of graphs, each equipped with different notions of distance. The representation used in Chapter 5 is a minhash signature, which is introduced in Section 4.1. In Chapter 6 and Chapter 7 we use neural

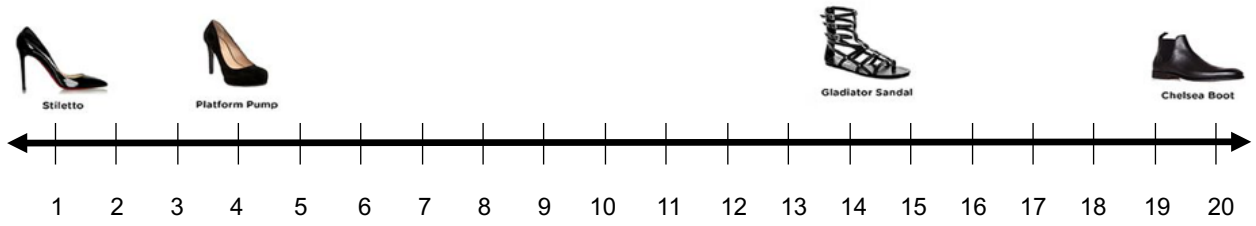


Figure 4.3: A one-dimensional embedding of shoes that places similar shoes close to each other (only four shown for clarity)

embeddings in Euclidean space and these are introduced in Section 4.3. Finally, in Chapter 8 we develop neural embeddings in hyperbolic space.

4.1 Minhashing

Minhash signatures are fixed-length arrays that efficiently encode the estimated similarity between sets (Broder et al., 2000). They are called *signatures*, instead of vectors, because they are not members of a mathematical vector space (see Section 3). A minhash representation of a vertex v can be learned by hashing the set of vertices connected to v, often called the *neighbours* of v.

The notion of distance encoded by minhash signatures is not the usual Euclidean distance, but the Jaccard distance. The Jaccard distance D and corresponding similarity J are given by

$$J(A,B) = 1 - D(A,B) = \frac{|A \cap B|}{|A \cup B|}\,, \tag{4.1}$$

where A and B are general sets. An unbiased estimate of the Jaccard distance between two signatures can be calculated in $O(K)$, where K is the length of the signatures. However, the operation differs from the normal vector distance and is given by

$$\hat{J}(A,B) = \frac{I}{K} \tag{4.2}$$

where we define

$$I = \sum_{k=1}^{K} \delta(h_k(A), h_k(B))\,, \tag{4.3}$$

$$\delta(h_k(A), h_k(B)) = \begin{cases} 1 & \text{if } h_k(A) = h_k(B) \\ 0 & \text{if } h_k(A) \neq h_k(B) \end{cases}. \tag{4.4}$$

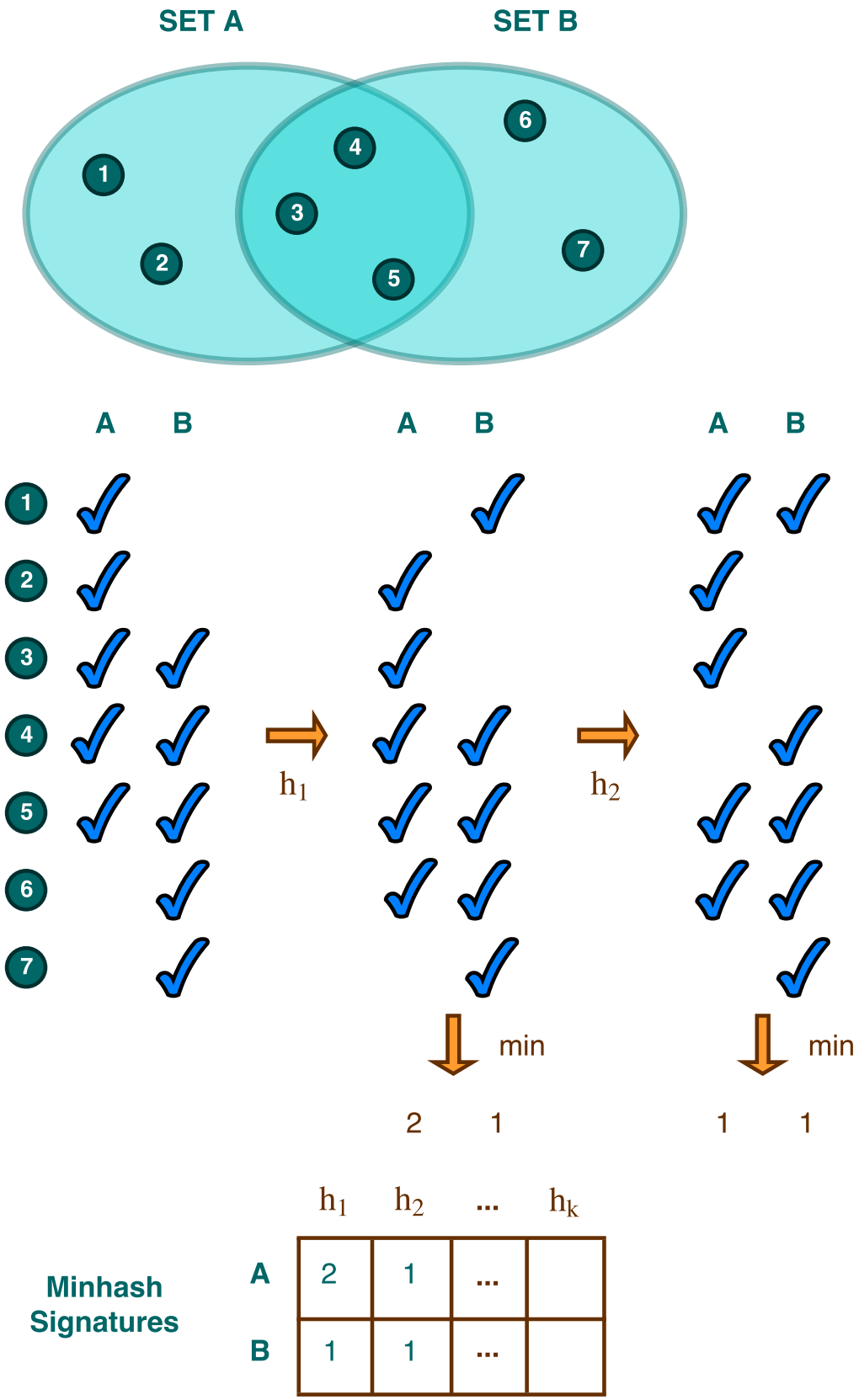


Figure 4.4: Representing sets A and B as minhash signatures. First every element of the universe is assigned an index. A series of independent hash functions are used to efficiently simulate a permutation of the indices. For each permutation the minhash value of each set is the lowest index value contained in the set. A minhash signature is made up of the minhash values from k independent hash functions.

and $h_k(A)$ is the minhash value generated by applying the k^{th} hash function to set A. Figure 4.4 shows the process of generating minhash signatures for two sets A and B. In this case the universe χ contains seven elements with five belonging to both A and B. Each element is assigned an initial index $i \in \{1, 2, ...|\chi|\}$. To generate a minhash value, the set of indices are pseudo-permuted using a hash function. A hash function of the form

$$h(i) = (ai + b)\%|\chi| \tag{4.5}$$

is often used where a and b are large random numbers that are sampled iid for each hash function. The minhash value for a set A would then be given by

$$\min(h(A)) \tag{4.6}$$

$$h(A) = \{h(i) : i \in A\} \tag{4.7}$$

In Figure 4.4, tick marks represent the indices that are present in each set after pseudo-permutation by a hash function. For each hash function, the minhash value is simply the minimum permuted value in the set. A minhash singature $\mathbf{h}$ is built up by taking the minhash values of K independent hash functions. The estimate $\hat{J}(A, B)$ of the Jaccard similarity $J(A, B)$ is attained by exploiting that

$$p(h_k(A) = h_k(B)) = J(A, B) \quad \forall k = 1, \ldots, K\,. \tag{4.8}$$

where p is a probability function. Therefore, we obtain an estimate of $J(A, B)$ by estimating the probability $p(h_k(A) = h_k(B))$ using the Monte-Carlo estimate in Equation (4.2). As each h_k is independent, $I \sim Bin(J(A, B), K)$. The estimator is fully efficient, i.e. the variance is given by the Cramér-Rao lower bound (Rao, 1992)

$$\text{var}(\hat{J}) = \frac{J(1 - J)}{K}\,. \tag{4.9}$$

Theoretical guarantees for minhashing require that min-wise independent permutations of the sets can be efficiently generated. In practice, this is not true and the hash functions only approximate the minwise independent permutations due to the presence of collisions. Using hash functions to simulate permutations leads to slightly weaker, but still practically useful guarantees (Broder et al., 2000).

Since the pioneering work on minhashing by Broder (1997), which dealt with hashing binary vec-

tors, several extension have been developed. Minhashes of counts (integer vectors) were invented by Charikar (2002) and later continuous variables by Philbin (2008). Efficient algorithms for generating the hashes were developed by Manasse and Mcsherry (2010). Equation (4.9) shows that there is a trade off between the length of the signatures (which is proportional to runtime and space requirements) and the variance of the estimator. Two important innovations that improve upon this trade-off are b-Bit minhashing (Li and König, 2009) and Odd Sketches (Mitzenmacher et al., 2014). Li and König (2009) show that it is possible to improve on the size-variance trade off by using longer signatures, but only keeping the lowest b-bits of each element (instead of all 32 or 64). Their work delivers large improvements for very similar sets (more than half of the total elements are shared) and for sets that are large relative to the number of elements in the sample space. Mitzenmacher et al. (2014) developed b-bit minhashing by showing that for approximately identical sets (Jaccard similarities ≈ 1) there was a more optimal estimation scheme.

4.2 Vector Space Models

Minhashing is a powerful technique to compactly represent vertices of graphs, but as the signatures are not in a vector space, the operations that can usefully be performed on them are limited. General embedding (or vector space) methods learn a lower-dimensional continuous space in which to represent high-dimensional complex data (Roweis and Saul, 2000; Belkin and Niyogi, 2001). The geometry of the lower-dimensional space encodes the similarity between vectors. Common similarity metrics include the angle or distance between two vectors. The embeddings are usually learned by first postulating a vector space and then optimizing an objective function of the vectors in that space. Objective functions require a trade-off between computational speed and accuracy and include criteria such as the error in reconstructing the original data from the embeddings or the difference between pairwise distances in the embedding and full spaces (Lee and Seung, 1999). It is generally agreed that there is no general optimal embedding for a dataset and that an embedding is only optimal for a given task. One of the major trade-offs is the preservation of global or local structure (Charikar and Makarychev, 2010; V. de Silva and J.B. Tenenbaum, 2003).

Vector space representations provide three principal benefits over sparse schemes: (1) They encapsulate similarity, (2) they are compact, (3) they perform better as inputs to ML models (Salton et al., 1975). These benefits all apply to graph structured data where the native representation is the ad-

jacency matrix; a typically large, sparse matrix of connection weights. For graphs small enough to apply $O(|V^3|)$ time complexity algorithms, embeddings can be learned directly from adjacency matrices. For symmetric matrices spectral methods are popular. IsoMap is a well known approach that learns an embedding through spectral decomposition (Belkin and Niyogi, 2001). Singular-Value Decomposition (SVD) is a technique to learn embeddings that generalises to non-square, non-symmetric matrices (Golub and Van Loan, 1996). General matrix and tensor factorisation techniques form a large sub-field of ML concerned with learning embeddings under various constraints. There are many applications of factorisation methods including e-commerce recommender systems operating on customer-product matrices (Koren et al., 2009; Kolda and Bader, 2008).

We are particularly interested in embeddings where the objective function minimises the prediction error from a function defined by a neural network instead of the error reconstructing a matrix. Matrix and neural network based embeddings have been shown to be related (Levy and Goldberg, 2014), but the formulation remains conceptually distinct. Neural networks learn *distributed representations* of entities where the representation is a pattern of activity distributed across computing units instead of localised to the value stored in a single unit (Hinton et al., 1990). In neural embedding models, the vector space corresponds to the activations of a subset of the network's units and the values are learned through backpropagation (Linnainmaa, 1970).

4.3 Neural Embeddings

Even within the relatively narrow field of neural embeddings, there remains some disagreement about the definition of an *embedding*. Some researchers, particularly in the computer vision community, regard an embedding as the final set of weights before the output layer of a deep network (Jia et al., 2014). However, here and for the remainder of this thesis, we refer to the first matrix of weights in a shallow neural network as a neural embedding. Shallow neural embeddings models have been successfully applied to a wide range of problems including word analogies (Mikolov et al., 2013a; Mnih and Kavukcuoglu, 2013), machine translation (Sutskever et al., 2014), document comparison (Kusner et al., 2015), missing edge prediction (Grover and Leskovec, 2016), vertex attribution (Perozzi and Skiena, 2014), product recommendations (Grbovic et al., 2015; Baeza-Yates and Saez-Trumper, 2015), customer value prediction (Kooti et al., 2017; Chamberlain et al., 2017a) and item categorisation (Barkan and Koenigstein, 2016). In all cases, the embeddings are learned without

labels (unsupervised) from a sequence of entities.

4.3.1 Embeddings for Natural Language Processing

Neural embeddings first came to prominence as a successful tool to represent words in Natural Language Processing (NLP). In the NLP community, it has been known since at least the 1960s that words frequently co-occurring together share meaning (Rubenstein and Goodenough, 1965). This phenomenon is called *the distributional hypothesis* of language. It can be summarised by the phrase "You shall know a word by the company it keeps" and is more precisely defined by Rubenstein and Goodenough (1965) as "the proportion of words common to the contexts of word A and to the contexts of word B is a function of the degree to which A and B are similar in meaning" where a context defines the locality of a word, which is usually taken to be a sentence or a sliding window containing a constant number of words.

Early work on representations of words based on the distributional hypothesis relied on large, sparse count matrices with one column for each element of the vocabulary (Miller and Charles, 1991). Count-based methods were improved by applying various heuristics and then producing embeddings either through SVD or other forms of matrix factorisation (Bullinaria and Levy, 2012). The first model to replace counts with predictions based on distributed representations of words was by Bengio et al. (2003). They used a neural network to implicitly learn embeddings by optimizing model performance at predicting the next word in a sequence. Two related models inspired by Restricted Boltzmann Machines were suggested by Mnih and Hinton (2007). In this paper, they also proposed an important log-bilinear model that only computed hidden activities once per prediction and removed the non-linearities from the hidden layer. The idea of learning distributed representations of words was later generalised to many linguistic tasks by Collobert and Weston (2008).

Initially it was unclear whether count-based or predictive models were superior, with predictive models suffering in particular from very long training times[1]. A large fraction of the computation time in predictive language models is dedicated to explicitly normalising the probability of the next word over all of the words in the vocabulary. Two techniques have been successfully developed that alleviate this problem: hierarchical decompositions and negative sampling. A hierarchical decomposition of the conditional word probabilities first appeared in Morin and Bengio (2005). The words

[1]The model of Collobert and Weston (2008) had to be trained for two months on Wikipedia data.

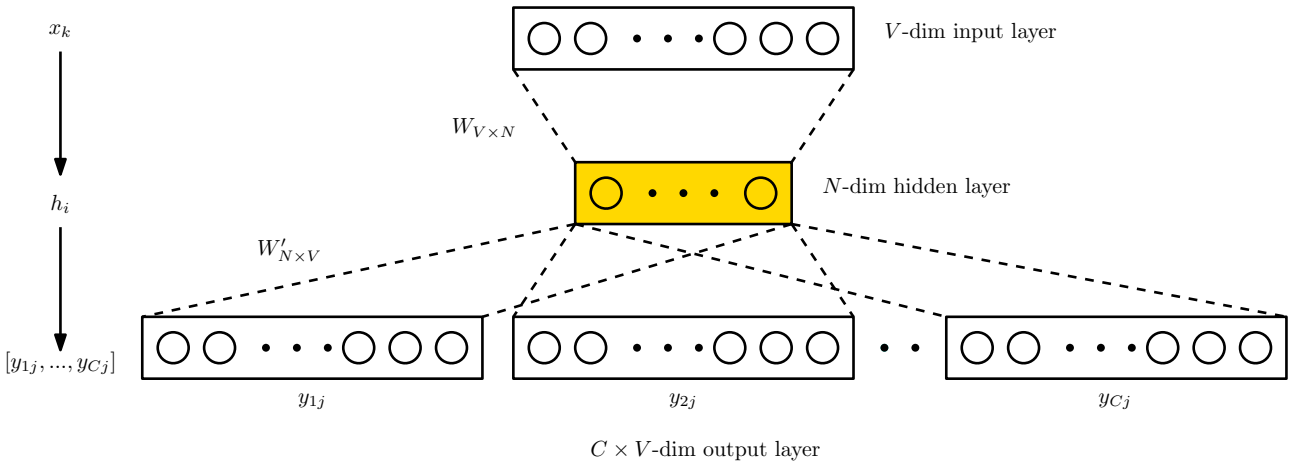


Figure 4.5: The Skipgram architecture has three layers: An input projection layer, a linear hidden layer and a softmax output layer. The objective is to predict a multi-word context from a single input word.

in the vocabulary are represented as the leaves of a binary tree and are indexed by bit-strings that encode the path to each leaf from the root. The model represents the probability of the next word given a set of context words as a sequence of binary decisions conditioned on the context vector. The approach exponentially decreases the amount of normalisation required, improving runtimes by two orders of magnitude. However, the predictive performance is substantially worse and a binary word tree must be constructed semi-manually. A hierarchical approach was later applied by Mnih and Hinton (2009) to the log-bilinear model developed by the same authors (Mnih and Hinton, 2007). In this work, they introduce an automatic algorithm for learning binary word trees that allows words to appear as multiple different leaves and in doing so achieve state-of-the-art performance. Huffman binary trees are applied to simpler log-bilinear models in Mikolov et al. (2013b) that adapt the concept of the word context. Instead of predicting the next word conditioned on a fixed length sequence of previous words, they predict either (1) a central word from the surrounding words in the CBOW model or (2) the surrounding words from a central word in the Skipgram model.

Negative sampling is based on noise-contrastive estimation (Gutmann, 2012) and was introduced to natural language processing by Mnih and Teh (2012). It was later found that negative sampling improves runtimes over the hierarchical softmax without inhibiting predictive performance for state-of-the-art models such as Skipgram and CBOW (Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013a).

4.3.2 The Skipgram Model

The Skipgram model (Mikolov et al., 2013a) is shown in Figure 4.5. It is a shallow neural network with three layers: (1) An input projection layer that maps from one-hot to a distributed representation, (2) a linear hidden layer, (3) an output softmax layer. Previous NLP models predicted a target word from a multi-word context (Bengio et al., 2003). This required an aggregation of the context representations in the hidden layer, forcing the updates of every context word to be identical. Skipgram is able to treat each (input, context) word pair as a separate observation and so can effectively learn from more data.

Skipgram is trained on a sequence of words that is decomposed into (input word, context words)-pairs. The model uses two separate vector representations, one for the input words (W in Figure 4.5) and another for the context words (W' in Figure 4.5), with the input representation comprising the learned embedding. The word pairs are generated by taking a sequence of words and running a sliding *context window* of size C over them. The choice of C is configurable and $C = 10$ is commonly used. As an example the word sequence "chance favours the prepared mind" with $C = 3$ would generate the following training data: $\{$(chance, favours), (chance, the), (favours, chance), ... $\}$. Words are initially randomly allocated to vectors within the two vector spaces. Then, for each training pair, the vector representations of the observed input and context words are pushed towards each other and away from all other words (see Figure 4.6). Figure 4.5 shows that the Skipgram architecture contains three layers. The first is the input / projection layer. It maps from a one-hot representation of a word $\mathbf{x_i}$ to a D-dimensional continuous vector v_i where D is problem specific and $D \ll |V|$. The weights between the input and hidden layers are $W \in \mathbb{R}^{|V| \times D}$ and $v_i = W_{:,i}$. The second layer is the hidden layer. It is a linear layer with an activation that is just the vector representation of the input word v_i. The final layer is an output layer of size $C \times |V|$. The weights are shared for each context word. The output is a sigmoid of the weighted sum of the activations of the hidden layer.

4.3.3 Update Equations

The forward pass of the Skipgram model is defined by the following equations. The hidden layer activation is just the vector representation of the input word

$$\mathbf{h} = W\mathbf{x}_I = \mathbf{v}_{w_I} \tag{4.10}$$

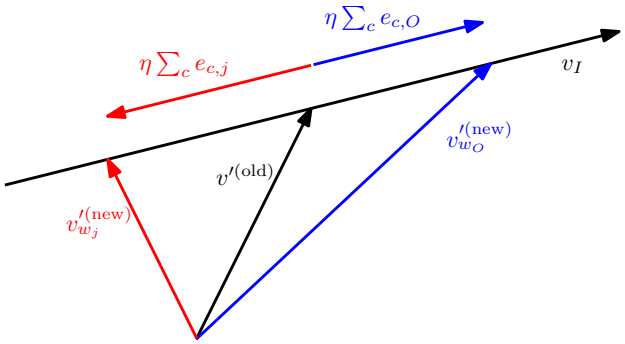


Figure 4.6: Geometric interpretation of the update equations in the Skipgram model. The vector representation of the output vertex $v'^{(\text{new})}_{w_O}$ is moved closer (blue) to the vector representation of the input vertex v_I, while all other vectors $v'^{(\text{new})}_{w_j}$ move further away (red). The magnitude of the change is proportional to the prediction error.

where w_I is the index of the observed input word. Output weights are shared across each context word and so the inputs to the final layer are

$$u_{c,j} = u_j = (W'\mathbf{h})_j = \mathbf{h}^T \mathbf{v}'_{w_j}, \tag{4.11}$$

where W' is the output matrix shown in Figure 4.5. The output is a softmax function

$$y_{c,j} = y_j = p(w_j = w_{c,O}|w_I) = \frac{e^{u_j}}{\sum_i e^{u_i}} \tag{4.12}$$

where $w_{c,O}$ and $u_{c,j}$ are the index of the observed word and the input of the j^{th} unit in the the c^{th} context panel respectively. The loss function is the negative log likelihood of all of the words in the context given the input word:

$$L = -\log p(w_{1,O}, w_{2,O}, ..., w_{C,O}|w_I) \tag{4.13}$$

$$= -\log \prod_{c=1}^{C} \frac{e^{u_{c,O}}}{\sum_i e^{u_{c,i}}} \tag{4.14}$$

$$= -\sum_{c=1}^{C} u_{c,O} + \sum_{c=1}^{C} \log \sum_{i=1}^{V} e^{u_{c,i}}. \tag{4.15}$$

It is important to realise that even though $u_{c,i}$ are repeated we do not drop the c index because when applying the backprop algorithm we calculate derivatives over individual edges of the computational graph, which differs from the normal procedure in calculus. Taking the derivative of the loss w.r.t. $u_{c,j}$

$$\frac{\partial L}{\partial u_{c,j}} = y_{c,j} - \mathbb{1}(w_{c,j} = w_{c,O}) = y_{c,j} - t_{c,j} := e_{c,j} \tag{4.16}$$

where $\mathbb{1}$ is the indicator function such that $t_{c,j}$ is one if the j^{th} word is the output word and zero otherwise and we define $e_j = y_{c,j} - t_{c,j}$ as the prediction error from the j^{th} unit of the c^{th} panel. The error is negative only when $w_{c,j} = w_{c,O}$. Next we calculate the derivative w.r.t. to the elements of W'

$$\frac{\partial L}{\partial W'_{i,j}} = \sum_c \frac{\partial L}{\partial u_{c,j}} \frac{\partial u_{c,j}}{\partial W'_{i,j}} = h_i \sum_c e_{c,j} \tag{4.17}$$

leading to gradient descent update equations of the form

$$W'^{\text{new}}_{i,j} = W'^{\text{old}}_{i,j} - \eta h_i \sum_c e_{c,j}. \tag{4.18}$$

Updates are normally made using stochastic gradient descent, in which case they are given by

$$\mathbf{v}'^{\text{new}}_{w_j} = \mathbf{v}'^{\text{old}}_{w_j} - \eta \mathbf{h} \sum_c e_{c,j} \tag{4.19}$$

where η is the learning rate. There are two important corollaries of Equation (4.19). Firstly as η and $\mathbf{h}$ are shared across context words, updates for each (input, context) pair can be performed independently. Secondly, for every training example, every element of W' must be updated. As W' is a $|V| \times D$ matrix, this update dominates the runtime.

4.3.4 Softmax Optimizations

The coupling of every output unit in the softmax function requires $|V| \times D$ parameters to be updated for each training example. This is impractical when working with large datasets. Two innovations, the hierarchical softmax and negative sampling solve this problem by reducing the time complexity from $O(|V|)$ to $O(\log |V|)$ and $O(k)$ respectively where k is the number of negative samples.

The Hierarchical Softmax is an elegant solution that reduces the runtime associated with evaluating a softmax from $O(|V|)$ to $O(\log |V|)$ using a binary tree with $|V|$ leaves. Each tree vertex encodes a probability density function over its children (Mnih and Hinton, 2009). Probabilities are assigned to words through a random walk down the tree. The specific structure of the tree has a large impact on training time and accuracy (Mnih and Hinton, 2009). A common choice in NLP is the Huffman tree (Mikolov et al., 2013a).

Negative Sampling is a form of Noise Contrastive Estimation (NCE) that provides a simple alternative to the hierarchical softmax (Gutmann, 2012). NCE is an estimation technique that is based on the assumption that a good model should be able to separate signal from noise using only logistic regression. It was first applied to language by Mnih and Teh (2012).

To perform negative sampling a *negative dataset* must be generated. This is often sampled from a problem-specific noise distribution. The negative log likelihood for logistic regression is given by

$$L = -\log \prod_{\{i:y_i=1\}} p(y_i = 1|\mathbf{x}_i, w) \prod_{\{i:y_i=0\}} p(y_i = 0|\mathbf{x}_i, w) \tag{4.20}$$

$$= -\sum_i (y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i) + (1 - y_i) \log \sigma(\mathbf{w}^T\mathbf{x}_i)) \tag{4.21}$$

where y, $\mathbf{x}$ and $\mathbf{w}$ are the target, input and parameters respectively. In negative sampling a loss function over the observed and negative datasets is defined as

$$L = -\log \prod_{\{i:y_i=1\}} p(y_i = 1|\mathbf{x}_i, w) \prod_{\{i:y_i=0\}} p(y_i = 0|\mathbf{x}_i, w) \tag{4.22}$$

where $y_i = 1$ indicates that an example is in the training set and $y_i = 0$ indicates that it is from the negative set.

4.3.5 Skipgram with Negative Sampling

Due to its simplicity and superior time complexity, we favour negative sampling over the hierarchical softmax in practical applications of the Skipgram model. As we only care about producing good embeddings, instead of making well calibrated predictions, the loss function does not need to produce a well specified predictive distribution. Following Equation (4.19), to simplify the exposition we consider just pairs of (input, output) words instead of summing over contexts. The Skipgram with Negative Sampling (SGNS) loss function is given by

$$L = -\sum_{(w_I,w_O)\in\mathfrak{D}} \log \sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - \sum_{(w_j,w_I)\in\mathfrak{D}'} \log \sigma(-\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I}) \tag{4.23}$$

The negative data set is assumed to be k-times larger than the training set. The negative dataset is sampled online from a noise distribution rather than batch-generating it a-priori. There is no strong

theoretical justification for the choice of noise distribution and negative words are usually drawn from the unigram distribution of the words in the training set raised to the power of $\frac{3}{4}$ (Mikolov et al., 2013a). In the online algorithm, for each input word in the observed data, k negative samples $\{W_{neg}\} \sim P_n(w_j)$ are drawn that are not the output word. NCE does not demand that the true output word is excluded from the noise sample, but in practice this is faster. The loss function for a single input-output word pair is given by

$$L = -\log \sigma(\mathbf{v}'^{T}_{w_O} \mathbf{v}_{w_I}) - \sum_{w_j \in W_{neg}} \log \sigma(-\mathbf{v}'^{T}_{w_j} \mathbf{v}_{w_I}) \tag{4.24}$$

$$= -\log \sigma(u_O) - \sum_{w_j \in W_{neg}} \log \sigma(-u_j) \tag{4.25}$$

$$= -\log \sigma(u_O) - \sum_{i}^{k} \mathbb{E}_{w_j \sim P_n} \log \sigma(-u_j). \tag{4.26}$$

The number of negative samples is typically in the range 2–20 and for large datasets just 2–5 negative words per sample are required. The training procedure increases the argument of the first sigmoid in Equation (4.26) and decreases the arguments of all of the negative data sigmoids under the sum. Taking the derivative of the loss w.r.t. u_j gives

$$\frac{\partial L}{\partial u_j} = \begin{cases} \sigma(u_j) - 1, & \text{if } w_j = w_O \\ \sigma(u_j), & \text{if } w_j \in W_{neg} \\ 0, & \text{otherwise} \end{cases} \tag{4.27}$$

and using this to evaluate the derivative of the loss w.r.t. the output weights gives

$$\frac{\partial L}{\partial (\mathbf{v}'_{w_j})_k} = \frac{\partial L}{\partial u_j} \frac{\partial u_j}{\partial (\mathbf{v}'_{w_j})_k} = \frac{\partial L}{\partial u_j} h_k \tag{4.28}$$

$$\frac{\partial L}{\partial \mathbf{v}'_{w_j}} = \begin{cases} (\sigma(u_j) - t_j)\mathbf{h}, & \text{if } w_j \in w_O \cup W_{neg} \\ 0, & \text{otherwise.} \end{cases} \tag{4.29}$$

$$\mathbf{v}'^{new}_{w_j} = \begin{cases} \mathbf{v}'^{old}_{w_j} - \eta(\sigma(\mathbf{v}'^{T}_{w_j}\mathbf{h}) - t_j)\mathbf{h}, & \text{if } w_j \in w_O \cup W_{neg} \\ \mathbf{v}'^{old}_{w_j}, & \text{otherwise} \end{cases} \tag{4.30}$$

where $t_j = 1$ if and only if $w_j = w_O$ and zero otherwise. This update is applied independently for each (input, context) pair and there are approximately ten updates per training example with $O(1)$ time complexity in the size of the vocabulary. In English $|V| \approx 100,000$ and so roughly 10,000 times

less calculations are required when using negative sampling compared to the full softmax evaluated in Equation (4.19). To update the input weights we recall that all of the output logits depend on $\mathbf{h}$

$$\frac{\partial L}{\partial h_k} = \sum_{j:w_j \in w_O \cup W_{neg}} \frac{\partial L}{\partial u_j}\frac{\partial u_j}{\partial h_k} = \sum_{j:w_j \in w_O \cup W_{neg}} \frac{\partial L}{\partial u_j}(\mathbf{v}'^{T}_{w_j})_k \tag{4.31}$$

$$\frac{\partial L}{\partial \mathbf{h}} = \sum_{j:w_j \in w_O \cup W_{neg}} (\sigma(u_j) - t_j)\mathbf{v}'_{w_j} \tag{4.32}$$

$$\mathbf{v}^{new}_{w_I} = \mathbf{v}^{old}_{w_I} - \eta \sum_{j:w_j \in w_O \cup W_{neg}} (\sigma(\mathbf{v}'^{T}_{w_j}\mathbf{h}) - t_j)\mathbf{v}'_{w_j} \tag{4.33}$$

4.3.6 Related Language Embeddings

There are several well-known models that are closely related to Skipgram. Glove is a model that performs global log-bilinear regression instead of operating on local context windows (Pennington et al., 2014). It attempts to combine the best elements from dense matrix factorisation and neural models. Unlike Skipgram, which treats context windows independently, Glove is trained on global co-occurrence statistics and so it uses more information from the underlying data. However, Glove lacks the property that it can be directly trained on sequences of tokens and has not been as broadly applied outside of NLP. *Paragraph Vectors* extend the idea of word embeddings to learn vector representations for entire documents (or paragraphs) (Le and Mikolov, 2014). The authors introduce the concept of a paragraph vector, which is appended to the word vectors within a context and used to predict a target word. In this way, they simultaneously learn word and paragraph embeddings. The model has the appealing property that different length pieces of text can be represented by fixed length vectors. In Skipgram, the input vectors W (see Figure 4.5) are taken to be the word embeddings and the output vectors W' are usually discarded. Mitra et al. (2016) observe that vectors with a high cosine similarity in a single vector space (W or W') are *typically* similar, sharing a type or function, while vectors with high cosine similarity across vector spaces share similar topics. They show that, for the task of web search, it is beneficial to include both properties through a linear combinations of the form $aW + bW'$.

4.4 Neural Embeddings of Graphs

SkipGram can be extended to graph-structured data using random walks to create sequences of vertices. Embedded representations of vertices can then be learned using SGNS by treating vertex

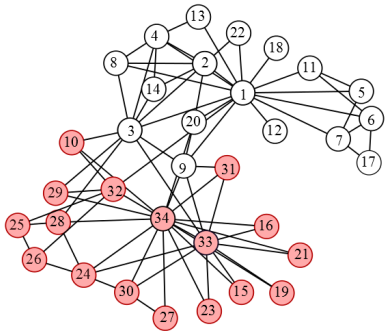


Figure 4.7: The network diagram for Zachary's famous karate club. It is split into two factions centred around vertices 1 (the karate instructor) and 34 (the club president)

IDs exactly analogously to words in the NLP formulation. This was originally proposed as DeepWalk (Perozzi and Skiena, 2014) and extensions varying the nature of the random walks have been explored in LINE (Tang et al., 2015) and Node2vec (Grover and Leskovec, 2016). The main justification for this idea is that digital traces of complex networks are really a form of noisy measurement of a true underlying network. Random walks have been shown extensively to mitigate for false edges and infer the presence of missing ones e.g. by Page et al. (1999).

To illustrate the process of learning neural embeddings of graph structured data we use the karate club network of Zachary (1977) shown in Figure 4.7. It contains 34 vertices and is split into two factions centred on the karate instructor (vertex 1) and the club president (vertex 34). This network has unweighted edges and so random walks can be generated by simply sampling the next vertex to visit based on the current vertex uniformly from $\{1,2,\ldots,$ vertex out-degree$\}$. Concretely, for a random walk starting at vertex 27 we would sample $x \sim U(\{1,2\})$. If $x = 1$ we move to the lowest indexed vertex (30 in this case) and append that vertex to the random walk. We then repeat the process at vertex 30 with $x \sim U(\{1,2,3,4\})$. For large graphs, an intermediate dataset is generated that contains multiple random walks originating at each vertex. Common choices of walk parameters are to use ten walks starting from each vertex, each of length 80 (Perozzi and Skiena, 2014; Grover and Leskovec, 2016; Tang et al., 2015).

To efficiently sample vertices for the random walks on weighted graphs, alias sampling is used (Li et al., 2014b). Sampling from a k-dimensional discrete distribution is naively $O(k)$, but it can be achieved in constant time using alias sampling. The trick is to convert a discrete distribution such as the three state distribution shown in the top line of Figure 4.8 into a uniform mixture of binary variables (bold lines in the bottom part of Figure 4.8). With this representation, sampling is equivalent to rolling a k-sided dice to choose one of the uniform area bold outlined rectangles and then tossing a

coin to select one of the two states within each rectangle. For instance, sampling $x \in \{1, 2, 3\}$ from $p = [0.2, 0.4, 0.6]$ is equivalent to sampling an index $i \sim U(3)$ and then sampling a binary variable from $Bern(y_i)$. This is done in such a way that outcomes with $p < \frac{1}{K}$ are completely contained in one slot and outcomes with $p > \frac{1}{K}$ are spread over multiple slots.

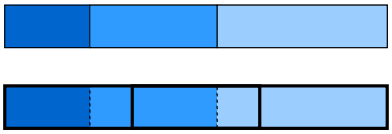


Figure 4.8: Illustration of alias sampling. Sampling from a discrete distribution (top) is decomposed into a sample from a uniform (bottom, bold lines) followed by a Bernoulli (bottom, dashed lines)

4.5 Neural Embeddings of Items, Products and Users

The Skipgram model and all of the methods described in Section 4.3.6 only require a sequence of data that obeys the distributional hypothesis and embedded representations can be learned efficiently (Hinton, 1986). This realisation has led researchers to experiment with embeddings of many types of object.

The Skipgram model has also been applied to a broad array of commercial problems. Barkan and Koenigstein (2016) used SGNS for item-level embeddings in item-based collaborative filtering, which they called item2vec. As a context the authors use a basket of items that were purchased together[2]. Grbovic et al. (2015) mined receipts from email data and used SGNS to generate a set of product embeddings, which the authors called prod2vec. For each customer a sequence of products was built (with arbitrary ordering for those bought together) and then a context window was run over it. The goal was to predict products that are co-purchased by a single customer within a given number of purchases. In the same paper, the authors also proposed a hierarchical extension called bagged-prod2vec that groups products appearing together in a single email. Chamberlain et al. (2017a) used customer product interactions to learn embeddings of customers. They showed how customer embeddings could be warm started and used for long-term forecasting. Baeza-Yates and Saez-Trumper (2015) used a variant of SGNS to predict the next application a mobile phone user would open. The key idea was to consider sequences of application usage within mobile sessions. The data had associated time stamps, which the authors used to modify the original model. Instead

[2]Items are synonymous with products, but the term 'item' is used in the recommender systems literature.

of including every pairwise set of applications within the context, the selection probability was controlled by a Gaussian sampling scheme based on the inter-open duration.

4.6 Summary

In this chapter we have introduced and justified the need for compact representations of graphs. Two types of fundamentally different representations were described: (1) minhash signatures and (2) neural embeddings. Both representations are of a fixed length specified by a free parameter and efficiently encode a notion of distance. However, there are many differences between them. Minhash signatures are arrays of integers that do not belong to a vector space. They encode a notion of the Jaccard distance between sets through Equation (4.2). Minhashing can be applied to graphs by treating vertices as sets of their neighbours and we adopt this approach for search in Chapter 5. Neural embeddings are real valued vectors that encode the Euclidean distance between entities that exist in sequences through the inner product. Neural embeddings can be applied to graphs by taking short random walks over the graph to generate sequences. In Chapters 6,7 and 8 we use neural embeddings in different forms as features to improve machine learning systems.

Chapter 5

Real-Time Community Detection in Full Social Networks on a Laptop

In this chapter we develop a method to rapidly discover communities in large social networks. We are able to do this efficiently by embedding vertices into a metric Jaccard space using the minhashing technique described in Section 4.1. The embedded representation of the graph allows communities that are close in Jaccard distance to a set of *query* vertices to be rapidly discovered using an approximate nearest neighbours technique. We work with massive graphs containing many million vertices and acquiring, analysing and validating results on these datasets are involved subjects that are covered in detail. This chapter is closely related to the published work of Chamberlain et al. (2018).

5.1 Introduction

We are concerned with discovering communities in social networks under three major constraints. Firstly, the raw graphs are large, secondly, the algorithm should give results in real-time and thirdly, it should run on a laptop. These constraints are the requirements of a software system developed at Starcount Insights, which allowed analysts to partially replicate the function of commercial or political focus groups. Specifically we are interested in discovering communities in full social networks, such as Facebook and Twitter. We define real-time to be less than the average human reaction time, which is 0.25s (Hewett et al., 1992).

On social media, people connect with other people (e.g., by being a "friend" or "following" somebody)

forming a *social graph*. Discovering communities in the social graph has a large number of practical applications, which include

- Security, where analysts explore a network looking for groups of potential adversaries;
- Social sciences, where queries can establish the important relationships between individuals of interest;
- E-commerce, where queries reveal related products or users;
- Marketing, where companies seek to optimize advertising channels or celebrity endorsement portfolios.

These applications do not disrupt user experience in the way that sponsored links or feed advertising do, offering an alternative means for social media providers to continue to offer free services.

A concrete example of a commercial use of community detection in Twitter is to provide strategic support to a company that wants to trade in a new geographic region. To enter a new geography a company must understand the competitors, customers and marketing channels in that region. Our system allows them to extract this valuable information by providing the Twitter handles for their existing products, key people, brands and endorsers. In real-time, they can receive the accounts closely related to their company filtered by that market. The output is automatically structured into groups (communities) such as media titles, sportspeople and other related companies. Analysts examine the results, and our system allows them to interactively explore the network by changing the input accounts. We show a high-level illustration for an alcoholic drinks brand in Figure 5.1.

We consider community detection with three practical restrictions:

1. Results need to be returned in real-time,
2. Computations can run on a single laptop,
3. The system can handle large graphs.

A system that satisfies these constraints is vital to analysts who wish to interactively explore and analyse community structures in large graphs. Community detection algorithms can be broadly divided into two forms: global methods, which partition the entire graph, and local methods, which

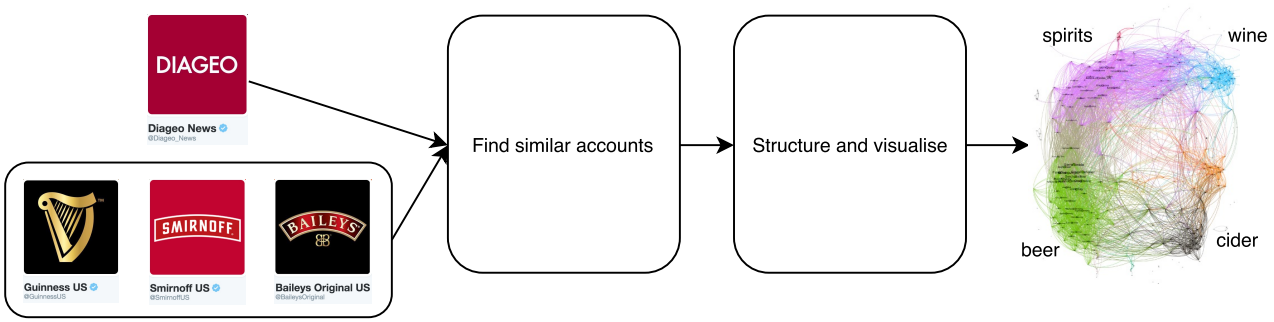


Figure 5.1: Example application of a real-time community detection system. Diageo want to explore the market (competitors, customers, associations etc.) around their brand. They feed in information about themselves ("seeds"). In this example the seeds are the company itself (Diageo) and some of their major brands (Smirnoff, Baileys and Guinness). Our systems finds accounts that are similar/related to the seeds and then structures the similar accounts into communities.

look for vertices that are related to an input vertex and only operate on a small part of the graph. To achieve our objective of a large-scale, real-time community detection system that runs on a laptop, we combine global and local approaches: First, we use local community detection to identify an interesting region of a graph, and then we apply global community detection to help understand that region. In the experimental section we describe several concrete applications, one of which is understanding the local community around and the structure of the US Republican party. In this case an analyst could query the Twitter Follower graph with the accounts 'Donald Trump', 'Marco Rubio', 'Ted Cruz', 'Ben Carson' and 'Jeb Bush'[1]. The system finds the local community around these query vertices, which in this case happens to be the right wing of US politics. The returned results indicate the sub-communities that exist within the US right wing. On seeing these, an analyst may decide that they are specifically interested in one subgroup. In this case one such subgroup are gun / rifle lobbyists and could run a follow up query using 'the national rifles association', 'gun owners of America' and 'the national association of gun rights', which are three accounts surfaced by the original query.

5.2 Background, Key Challenges and Contribution

A social media network is a structure that comprises a graph and a collection of metadata describing the vertices V and / or edges E of the graph. A *community* has no formal mathematical definition, but is generally agreed to be is a collection of vertices $C \subset V$ that share many more edges than would be expected from a random subset of vertices (Gleich and Seshadhri, 2012). In the context of

[1]This relates to data from December 2015

the Twitter Follower graph, which we refer to simply as the *Twitter graph*, a vertex V is a Twitter account, and an (undirected) edge E between V_i, V_j exists if V_i Follows V_j or V_j Follows V_i. Here, and in the remainder of the chapter, we capitalize Follows when describing the Twitter specific meaning of this word. A community might be the set of Twitter accounts belonging to machine learning researchers. In addition to the Twitter graph, the Twitter *network* also includes metadata associated with the accounts (e.g., name, description) and edges (e.g., creation time).

In this chapter, we develop a robust real-time algorithm for community detection for which we focus exclusively on the properties of the graph, i.e. no metadata of the Twitter/Facebook network is required. In particular, we propose that robust associations between social network accounts can be reached by considering the similarity of their neighbourhood graphs (Liben-Nowell and Kleinberg, 2007). The *neighbourhood graph* of a vertex consists of the set of all vertices that are directly connected to it, irrespective of the edge direction. Neighbourhood graphs of Digital Social Networks (DSNs) can be very large; In Twitter, the largest have almost 100 million edges (as of June 2016).

Our proposition of exploiting the neighbourhood graphs relies on the existence of *homophily* in social networks: The homophily principle states that people with similar traits are more likely to form relationships (McPherson et al., 2001). An important assumption that we make is that the DSNs we study are representative of the underlying social networks. In reality the underlying networks and the DSNs themselves are multiplex (Twitter users retweet and comment in addition to Following). Interactions occur over multiple channels and modelling only a simplex network is known to introduce inaccuracies (Shang, 2015). We choose this approach as communication density in the additional layers of our experimental networks are orders of magnitude more sparse than the layers we study[2]. Accordingly, we feel that the assumption that social media accounts with similar neighbourhood graphs are likely to have similar attributes is valid for these datasets.

Our real-time system exhibits the following properties:

1. It produces high quality communities from noisy data. Quality is defined using the community axioms given in Section 5.5.1 (Yang and Leskovec, 2013).

2. It is robust to failure and does not require engineering support.

3. It is parsimonious with the time of its users. Specifically we mean returning results in real-time and having sub-linear time complexity with the size of the network (Discussed in detail

[2]e.g. the median number of retweets from a Twitter user in our dataset is zero, comments are even more sparse

in Section 5.4.3).

The first constraint leads us to use the neighbourhood graph as the unit of comparison between vertices. The neighbourhood graph is generated by the actions of large numbers of independent users in contrast to features like text content or group memberships, which are usually controlled by a single user. The second requirement leads us to search for a solution that does not require large computing resources and can run on a laptop. The third property prescribes a real-time system.

Currently, no tool exists that provides real-time analysis of large graphs on a single commodity machine. Existing methods to analyse local community structure in large graphs either rely on distributed computing facilities or incur excessive runtimes making them impractical for exploratory and interactive work (Clauset, 2005; Bahmani et al., 2011). We introduce a novel real-time analysis tool for detecting communities in large graphs using only a laptop. As a working example we use a dataset of 700 million Twitter users. However, our approach is more generally applicable as it relies only on the general graph structure, not on Twitter-specific data. We provide additional results using Facebook and email data to demonstrate the general applicability. For real-time community detection, we need to solve two core challenges:

1. The graph must fit into the memory of a laptop.

2. Many neighbourhood graphs containing up to 100 million vertices must be compared in milliseconds.

To address the first challenge, we embed the neighbourhood graph of each vertex in Jaccard space, representing them as fixed length minhash signatures (See Section 4.1). This representation is vastly smaller than the size of the graph. There is a trade-off between representation size and approximation accuracy so that the representation can always be configured to fit the graph into memory. Minhash signatures allow for an efficient comparison of neighbourhood graphs by means of the Jaccard similarity. To solve the second challenge and achieve real-time querying we use the elements of the minhash signatures as the basis to build a Locality Sensitive Hashing (LSH) data structure. LSH facilitates querying of similar accounts in constant time. The combination of minhashing and LSH allows analysts to enter an account or a set of accounts and in milliseconds receive the set of most related accounts. From this set we use the minhash signatures to rapidly construct a weighted graph

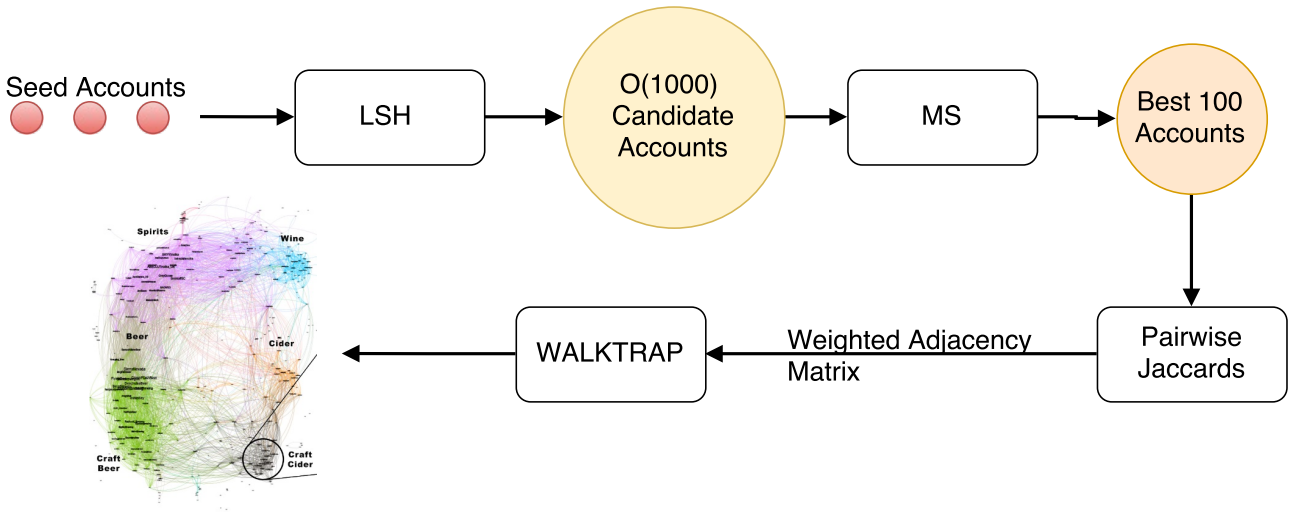


Figure 5.2: The full process diagram. A set of seeds is queried using LSH and Minhash Similarity. The weighted adjacency matrix for the top 100 results is estimated using minhash signatures. The WALKTRAP community detection algorithm is applied to the weighted adjacency matrix, and the results are visualized.

and apply the WALKTRAP community detection algorithm before visualizing the results (Pons and Latapy, 2005). The end-to-end schematic is depicted in Figure 5.2.

Our novel system is practical and combines proven techniques in an innovative way: (1) Minhashing and LSH are applied to the neighbourhood graph (Gibson et al., 2005; Becchetti et al., 2010) for representation and search respectively. (2) Minhashing is normally only used to compare very similar sets. We show that minhashing is effective for community detection when applied to a broad range of neighbourhood graph similarities. (3) We develop an agglomerative clustering algorithm and an original update procedure for minhash signatures in this setting. The novel combination of these techniques allows our system to perform robust real-time community detection on a laptop using graphs that exceed 100 million vertices. The contributions of this chapter are:

1. We establish that robust associations between social media users can be determined by means of the Jaccard similarity of their neighbourhood graphs.

2. We show that the approximations implicit in minhashing and LSH minimally degrade performance and allow querying of large graphs in real-time.

3. System design and evaluation: We have designed and evaluated an end-to-end system for extracting data from social media providers, compressing the data into a form where it can be efficiently queried in real-time.

4. We demonstrate how queries can be applied to a range of problems in graph analysis, e.g., understanding the structure of industries, allegiances within political parties and the public image of a brand.

Our code base is publicly available at our github repository[3], which includes instructions for replicating our experiments.

5.2.1 Related Work

Existing approaches to large-scale, efficient, community detection have three flavors: More efficient community detection algorithms, innovative ways to perform processing on large graphs and data structures for graph compression and search. Table 5.1 shows related approaches to this problem and which of our three constraints they satisfy. The table highlights that there is currently only one way

Table 5.1: Comparison of related work. SCM stands for runs on a Single Commodity Machine.

Method	**Real-time**	**Large graphs**	**SCM**
Modularity optimization (Newman, 2004b)	✗	✗	✓
WALKTRAP (Pons and Latapy, 2005)	✗	✗	✓
INFOMAP (Rosvall and Bergstrom, 2008)	✗	✗	✓
Louvain method (Blondel et al., 2008)	✗	✓	✓
BigClam (Yang and Leskovec, 2013)	✗	✓	✓
Graphlab (Low et al., 2014)	✗	✓	✗
Pregel (Malewicz et al., 2010)	✗	✓	✗
Surfer (Chen et al., 2010)	✗	✓	✗
Graphci (Kyrola et al., 2012)	✗	✓	✓
Twitter WTF (Gupta et al., 2013)	✓	✓	✗
LEMON (Li et al., 2015)	✗	✓	✓
Our Method	✓	✓	✓

to do real-time community detection on large graphs (Twitter WTF Gupta et al. (2013)). However, this method requires a large computing infrastructure and does not run on a laptop. Algorithms that do run on a laptop are not real-time capable and may not even perform community detection on large graphs.

[3]https://github.com/melifluos/LSH-community-detection

Community Detection Algorithms

Community detection methods have been developed in areas as diverse as neuronal firing (Bullmore and Sporns, 2009), electron spin alignment (Reichardt and Bornholdt, 2006a) and social models (Yang and Leskovec, 2013). Fortunato (2010) and Newman (2003) both provide excellent and detailed overviews of the diverse community detection literature. Approaches can be broadly categorized into local and global methods.

Global methods assign every vertex to a community, usually by partitioning the vertices. Many highly innovative schemes have been developed to do this. Modularity optimization (Newman, 2004b) is one of the best known. Modularity is a metric used to evaluate the quality of a graph partition. Communities are determined by selecting the partition that maximizes the modularity. An alternative to modularity was developed by Pons and Latapy (2005) who innovatively applied random walks on the graph to define communities as regions in which walkers become trapped (WALKTRAP). In Rosvall and Bergstrom (2008), random walks are combined with efficient coding theory to produce INFOMAP, a technique that provides a new perspective on community detection: Communities are defined as the structural sub-units that facilitate the most efficient encoding of information flow through a network. All three methods are well optimized for their motivating networks, but do not scale to modern DSNs.

The availability of data from the web, DSNs and services like Wikipedia has focused research attention on algorithms that scale. An early success was the Louvain method that allowed modularity optimization to be used to perform community detection on large graphs (they report 100 million vertices and 1 billion edges). However, the method was not intended to be real-time, and the reported 152 minute runtime on a biopteron 2.2k with 24GB of memory is too slow to achieve real-time performance, even allowing for nearly a decade of hardware advances (Blondel et al., 2008). Another noteworthy technique applied to large graphs is BigClam (Yang and Leskovec, 2013). BigClam is a multiple membership model, meaning that each vertex can be assigned to more than one community. This differs from the Louvain method, which assigns each vertex to a single community. As vertices can belong to more than one community, BigClam can be said to detect overlapping communities. However, in common with the Louvain method, BigClam is not a real-time algorithm that could facilitate interactive exploration of social networks.

In contrast to global community detection methods, local methods do not assign every vertex to

a community. Instead they find vertices that are in the same community as a set of input vertices (seeds). For this reason they are normally faster than global methods. Local community detection methods were originally developed as crawling strategies to cope with the rapidly expanding web graph (Flake et al., 2000). Following the huge impact of the PageRank algorithm (Page et al., 1999), many local random walk algorithms have been developed. Kloumann and Kleinberg (2014) conducted a comprehensive assessment of local community detection algorithms on large graphs. In their study Personal PageRank (PPR) (Haveliwala, 2002) was the clear winner. PPR is able to measure the similarity to a set of vertices instead of the global importance/influence of each vertex by applying a slight modification to PageRank. PageRank can be regarded as a sequence of two step processes that are iterated until convergence: A random walk on the graph followed by (with small probability) a random teleport to any vertex. PPR modifies PageRank in two ways: Only a small number of steps are run (often four), and any random walker selected to teleport must return to one of the seed vertices. Recent extensions have shown that finding the local community around a vertex can be improved by seeding (using as the teleport set) PPR with the neighbourhood graph of that vertex (Gleich and Seshadhri, 2012) and that PPR can be used to initialise *local* spectral methods with good results (Li et al., 2015).

Random walk methods are usually implemented using power iteration; a series of matrix multiplications requiring the full adjacency matrix to be read into memory. The adjacency matrix of large graphs will not fit in memory. Therefore, distributed computing resources are used (e.g., Hadoop). While distributed systems are continually improving, they are not always available to analysts, require skilled operators and typically have an overhead of several minutes per query.

A major challenge when applying both local and global community detection algorithms to real-world social media networks is performance verification. Testing algorithms on a held-out labelled test set is complicated by the lack of any agreed definition of a community. Much early work makes use of small hand-labelled communities and treats the original researchers' decisions as gold standards (Sampson, 1969; Zachary, 1977; Lusseau, 2003). Irrespective of the validity of this process, a single (or small number) of manual labellers can not produce ground-truth for large DSNs. Yang and Leskovec (2015) proposed a solution to the verification problem in community detection. They observe that in practice, community detection algorithms detect communities based on the structure of interconnections. However, results are verified by discovering common attributes or functions of vertices within a community. Yang and Leskovec (2015) identified 230 real-world networks in which

they define ground-truth communities based on vertex attributes. The specific attributes that they use are varied, and some examples include publication venues for academic co-authorship networks, chat group membership within social networks and product categories in co-purchasing networks.

Graph Processing Systems

A complimentary approach to efficient community detection on large graphs is to develop more efficient and robust systems. This is an area of active research within the systems community. General-purpose tools for distributed computation on large scale graphs include Graphlab, Pregel and Surfer (Chen et al., 2010; Malewicz et al., 2010; Low et al., 2014). Purpose-built distributed graph processing systems offer major advances over the widely used MapReduce framework (Pace, 2012). This is particularly true for iterative computations, which are common in graph processing and include random walk algorithms. However, distributed graph processing still presents major design, usability and latency challenges. Typically, the runtimes of algorithms are dominated by communication between machines over the network. Much of the complexity comes from partitioning the graph to minimise network traffic. The general solution to the graph partitioning problem, placing roughly equal numbers of nodes on each machine while minimising the number of inter-machine edges, is NP-hard and remains unsolved. These concerns have led us and other researchers to buck the overarching trend for increased parallelization on ever larger computing clusters and search for single-machine graph processing solutions. One such solution is Graphci, a single-machine system that offers a powerful and efficient alternative to processing on large graphs (Kyrola et al., 2012). The key idea is to store the graph on disk and optimize input/output (I/O) routines for graph analysis operations. Graphci achieves substantial speed-ups compared to conventional systems, but the repeated disk I/O makes real-time operation impossible. Twitter also use a single-machine recommendation system that serves "Who To Follow (WTF)" recommendations across their entire user base (Gupta et al., 2013). WTF provides real-time recommendations using random walk methods similar to PPR. This is achieved by loading the entire Twitter graph into memory. Following their design specification of five bytes per edge, $5 \times 30 \times 10^9 = 150$ GB of RAM would be required to load the current graph. This is an order of magnitude more than available on a laptop, which is our target platform.

Graph Representations and Compression

An alternative to using large servers, computing clusters, or disk storage, is to use compact representations that compress the whole graph to fit into the memory of a single machine. Graph compression techniques were originally motivated by the desire for single machine processing on the web graph. The use of minhash representations (see Section 4.1) and neighbourhood graphs for compression originates from research on storing the differences between graphs instead of raw graphs. Adler and Mitzenmacher (2001) searched for web pages with similar neighbourhood graphs and encoded only the differences between edge lists. The seminal work by Boldi and Vigna (2004) ordered web pages lexicographically endowing them with a measure of locality. Similar compression techniques were adapted to social networks by Chierichetti et al. (2009). They replaced the lexical ordering with an ordering based on a single minhash value of the out-edges, but found social networks to be less compressible than the web (14 bits versus three bits per edge). While the aforementioned techniques achieve remarkable compression levels, they pay the price of slower access to the data (Gupta et al., 2013).

Locality Sensitive Hashing (LSH) is a technique introduced by Indyk and Motwani (1998) for rapidly finding approximate near neighbours in high dimensional space. In the original paper, a parameter ρ governs the quality of LSH algorithms. A small value of ρ implies that near neighbours are likely to be hashed to the same bucket while distant points are likely to be hashed to different buckets. As ρ appears in the exponent of the query time and the space complexity, a lower value of ρ leads to a better algorithm. There is a great deal of work studying the limits on ρ. Of particular interest is Motwani et al. (2005) who use a Fourier analytic argument to provide a tighter lower bound on ρ, which was later bettered by O'Donnell et al. (2009) who exploited properties of the noise stability of boolean functions. Later research uses the structure of the data, through data-dependent hash functions (Andoni et al., 2014) to get even tighter bounds. However, when hash functions are data dependent, only static data structures can be addressed.

5.3 Data

In this section we introduce the data used throughout the chapter and explain how it was obtained. There are three datasets: (1) A Twitter Follower graph, (2) a Facebook (FB) page likes graph and (3) an email interaction graph. The first two datasets were crawled over a period of two years (2014-2016),

while the email dataset is publicly available (Leskovec et al., 2007).

5.3.1 Crawling Social Networks

The FB and Twitter data were obtained by crawling the social networks and downloading content through their public Application Programming Interfaces (APIs). Most providers of social networks offer public APIs to their networks. By doing so they allow 3^{rd} party developers to build applications using their data. This stimulates innovation and embeds their network more deeply into the everyday lives of its users and, for large networks, the global socio-economic and political sphere. Delivering data at massive scale can incur significant costs. To manage these, all networks limit the rate at which data can be downloaded through their APIs. Rate limiting varies between networks, but most work on a per user basis. A user access token is granted whenever a user signs up to an app using Facebook / Twitter etc. This work makes use of several client facing applications (Starcount Playlist, Starcount Vibe and Chatsnacks) to provide user access tokens.

To optimize data throughput while remaining within the DSN rate limits we developed an asynchronous distributed network crawler using Python's Twisted library (Wysocki and Zabierowski, 2011). The crawler consists of a server responsible for token and work management and a distributed set of clients making http requests to DSNs (see Figure 5.3). The server contains a credential man-

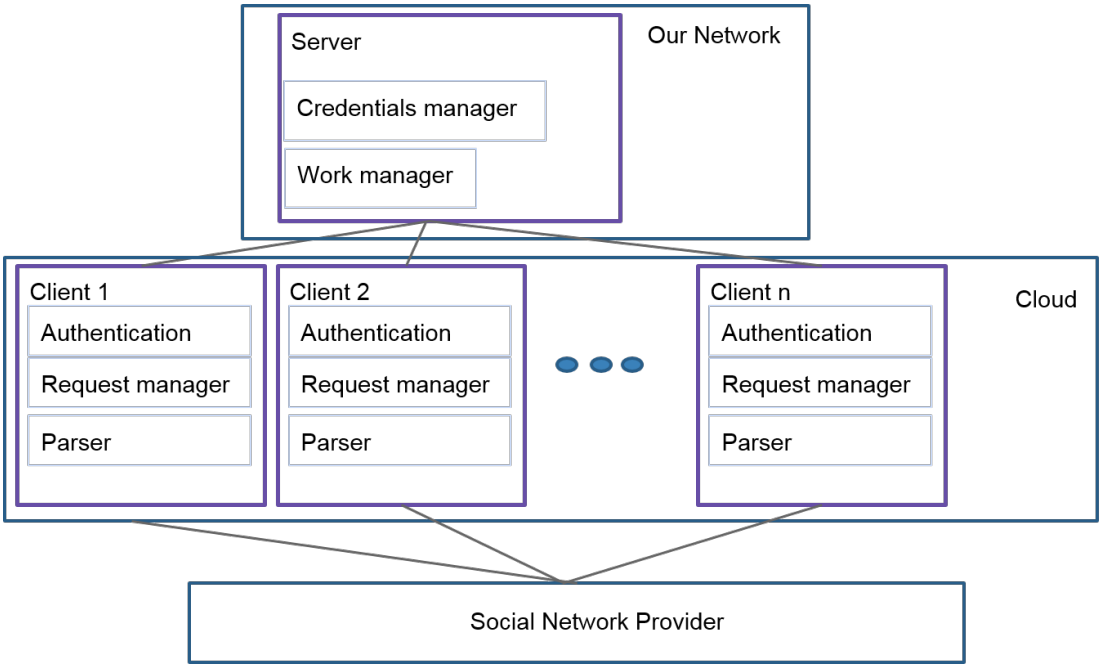


Figure 5.3: Distributed asynchronous system for data acquisition from digital social networks.

ager that holds access tokens in a queue and monitors the number of calls to each API. Once a token

has been exhausted it is put to the back of the queue and locked until its refresh time. The server communicates with the clients over TCP, responding to requests for work and access tokens with account ids and unlocked access tokens or pause responses respectively.

The clients make asynchronous requests to the DSNs, handling response codes, parsing and storing data. A conventional program will *block* while waiting for an http response. When the principal function of a program is to download data, blocking time dominates the runtime. One solution is to run the program using multiple threads. However, for this application, threads carry an unnecessary overhead and induce inefficiencies as data is naively moved between caches by the operating system. The asynchronous programming paradigm offers a superior alternative to explicit multi-threading. Asynchronous programming makes use of an event loop that constantly *listens* for new jobs and does not block while waiting for http responses.

We implemented the system using an 80 MB shared fibre optic connection. However, our activity caused network blackouts in our building and so we designed a distributed system that could be partially deployed in the cloud. The final system is depicted in Figure 5.3. The access tokens and account IDs to query (work) live on a server on our local network. Clients are deployed to Amazon's elastic cloud from where all interactions with DSN servers occurs. We configured the clients to establish persistent connections to the API endpoints. Every time a connection is opened, a handshake must occur. For secure systems (communicating over https), the handshake is particularly onerous, requiring the exchange of security certificates. By establishing persistent connections, we require only one handshake per connection.

5.3.2 Twitter Data

Twitter is the most widely used Digital Social Network (DSN) for academic research, and the data is relatively easy to obtain. At the time of writing the Twitter Follower graph consists of roughly one billion vertices (Twitter accounts) and 30 billion edges (Follows).

To collect data we use the Twitter REST API to crawl the network. Every time a new account is crawled we check the number of Followers in the account metadata and if it is greater than 10,000, we download the full Follower list. While 10,000 is an arbitrary number, accounts with more than 10,000 Followers tend to have public profiles (Wikipedia pages or websites), which are required to verify any results.

Our data set is a snapshot of the Twitter graph from December 2015. We found 675,000 accounts with over 10,000 Followers. They are Followed by a total of 1.5×10^{10} accounts, of which 7×10^8 are unique. We learn minhash representations of the 675,000 largest accounts using the Following patterns of all 7×10^8 accounts in the dataset. Any queries or results returned in the experimentation section are restricted to the 675,000 hashed accounts.

5.3.3 Facebook Data

Unlike Twitter, Facebook is mostly a private network. It is not possible to crawl the Facebook friend graph in the same way as the Twitter Follower graph. However, a relatively small number of Facebook users have public *pages* rather than private *profiles*. Data describing users interactions with FB pages is available to crawl. To collect data from FB we matched the Twitter accounts with more than 10,000 Followers to FB page accounts using a combination of automatic account name matching and manual verification. FB page likes are not available retrospectively, but can be collected through a real-time stream. Having identified the subset of accounts on Facebook corresponding to the large Twitter accounts, we used the Facebook API to collect the interaction streams of each page over a period of two years from early 2014. The resulting Facebook Pages engagement graph contains 450 million vertices (FB accounts) and 700 million edges (Page likes/comments).

5.3.4 Email Data

Due to privacy concerns, neither the Twitter nor the FB datasets can be made publicly available in their raw forms. For reproducibility we provide additional results on a public email network dataset (Leskovec et al., 2007). The network is a directed network of email communication from a large European research organization. Each vertex is an employee and they are uniquely labelled by one of 42 departments. There are a total of 1,005 email accounts and 25,571 edges. An edge (V_i, V_j) exists if the i^{th} email account emails the i^{th} account.[4]

[4]Further details and the data are available at `http://snap.stanford.edu/data/email-Eu-core.html`

5.4 Method

In the following, we detail our approach to real-time community detection in large social networks with the restriction that it runs on a single laptop. Our method consists of two main stages: In stage one, we take a set of seed accounts and expand this set to a larger group containing the most related accounts to the seeds. This stage is depicted by the box labelled "Find similar accounts" in Figure 5.1. Stage one uses a fast nearest-neighbour search algorithm. In stage two, we embed the results of stage one into a *complete weighted graph* where each vertex is connected to every other vertex. The edge weights are given by the Jaccard similarity of the two accounts they connect. This form of graph is known as an *intersection graph* in the mathematics literature, where it is a well studied object (Karoński et al., 1999; Shang, 2010, 2012). We apply a global community detection algorithm to the intersection graph and visualize the results. Stage two is depicted by the box labelled "Structure and visualize" in Figure 5.1.

We use the following notation: The i^{th} user account (or interchangeably, vertex of the network) is denoted by A_i, and $N(A_i)$ gives the set of all accounts directly connected to A_i (the neighbours of A_i). The set of accounts for which we want to discover communities in the network are provided by a user into the system and are called "seeds". They are denoted by $S = \{A_1, A_2, \ldots, A_m\}$ and the community returned by stage one is denoted by $C = \{A_1, A_2, ..., A_n\}$.

5.4.1 Stage 1: Seed Expansion

The first stage of the process takes a user-defined set of seed accounts as input, orders all other accounts by similarity to the seeds and returns an expanded set of accounts similar to the seed account(s). For this purpose, we require three ingredients:

1. A similarity metric between accounts
2. An efficient system for finding similar accounts
3. A stopping criterion to determine the number of accounts to return

In the following, we detail these three ingredients of our system, which will allow for real-time community detection in large social networks on a standard laptop.

Similarity Metric

We treat two accounts as similar if they have similar connections. The set of connections of an account A is the account's neighbourhood graph $N(A)$. The neighbourhood graph is an attractive feature as it is not controlled by a single individual, but by the (approximately) independent actions of large numbers of individuals. The edge generation process in Digital Social Networks (DSNs) is noisy, producing graphs with many extraneous and missing edges. As an illustrative example, the pop stars Eminem and Rihanna have collaborated on four records and a stadium tour ("Love the Way You Lie" (2010), "The Monster" (2013), "Numb" (2012), and "Love the Way You Lie (Part II)" (2010), the Monster Tour (2014)). Despite this clear association, at the time of writing, Eminem is not one of Rihanna's 40 million Twitter followers. However, Rihanna and Eminem have a Jaccard similarity of 18%, making Rihanna Eminem's 6^{th} strongest connection. Using the neighbourhood graph as the unit of comparison between accounts mitigates against noise associated with the unpredictable actions of individuals. The similarity metric that we use to compare two accounts is the *Jaccard similarity* of their neighbourhood graphs. The Jaccard similarity was introduced in Section 4.1. For the remainder of this chapter we use the Jaccard similarity between two accounts as a shorthand for Jaccard similarity between the sets of their neighbourhood graphs

$$J(A_i, A_j) \triangleq J(N(A_i), N(A_j)) = \frac{|N(A_i) \cap N(A_j)|}{|N(A_i) \cup N(A_j)|}\,. \tag{5.1}$$

The Jaccard similarity is related to the Jaccard distance by

$$D(.) = 1 - J(.). \tag{5.2}$$

The Jaccard has two attractive properties as a metric for comparing neighbourhood graphs. Firstly, it is a normalized measure providing comparable results for sets that differ in size by orders of magnitude. Secondly, as described in Section 4.1, minhashing can be used to provide an unbiased estimator of the Jaccard that is both time and space efficient.

Efficient System to Find Similar Accounts

To efficiently search for accounts that are similar to a set of seeds we represent every account as a minhash signature and use a Locality Sensitive Hashing (LSH) data structure based on the minhash signatures for approximate nearest neighbour search.

Computing the Jaccard Similarities Computing the Jaccard similarities in Equation (5.1) is expensive. Each set of neighbours can have up to 10^8 members, and calculating intersections is super-linear in the total number of members of the two sets being intersected. Multiple large intersection calculations can not be processed in real-time. There are two alternatives: Either the Jaccard similarities are pre-computed for all possible pairs of vertices, or they are estimated. Using pre-computed values for all $n = 675,000$ Twitter accounts with more than 10,000 Followers would require caching $\frac{1}{2}n(n-1) \approx 2.5 \times 10^{11}$ floating point values requiring approximately 1 TB and exceeding the specifications of a laptop. Therefore, we efficiently estimate the Jaccard similarities using minhashing. A detailed description is provided in Section 4.1 and here we include just a brief recap. The estimate $\hat{J}(A_i, A_j)$ of the Jaccard similarity $J(A_i, A_j)$ is attained from two signatures by

$$\hat{J}(A_i, A_j) = \frac{I}{K}, \tag{5.3}$$

where K is the number of hash functions composing a signature and we define

$$I = \sum_{k=1}^{K} \delta(h_k(A_i), h_k(A_j)) \,, \tag{5.4}$$

$$\delta(h_k(A_i), h_k(A_j)) = \begin{cases} 1 & \text{if } h_k(A_i) = h_k(A_j) \\ 0 & \text{if } h_k(A_i) \neq h_k(A_j) \end{cases}, \tag{5.5}$$

where similarly to our Jaccard notation we use $h_k(A_i) = h_k(N(A_i))$ for notational simplicity. The variance is given by

$$\text{var}(\hat{J}) = \frac{J(1-J)}{K} \,, \tag{5.6}$$

where we have dropped the Jaccard arguments for brevity. Equation (5.6) shows that Jaccard coefficients can be approximated to arbitrary precision using minhash signatures with an estimation error whose variance scales as $O(1/K)$.

Memory and space improvements of minhashing We use all 700 million Twitter accounts to compute the minhash signatures. However, the memory requirement of minhash signatures is only Kn integers, where K is the number of hash functions and n is the number of considered Twitter accounts. Therefore, it fits into the RAM of a laptop: For $K = 1,000$ independent hash functions and the $n = 675,000$ largest Twitter accounts, only $\approx 4GB$ are required. In comparison to calculating

Jaccard similarities of the largest 675,000 Twitter accounts with $\approx 4 \times 10^{10}$ neighbours, minhashing reduces expected processing times by a factor of $10,000$, and storage space by a factor of $1,000$. Note that our method allows new accounts to be added quickly by simply calculating a single additional minhash signature without needing to add the pairwise similarity to all other accounts.

Algorithm 1 Minhash signature generation.

Require: $M \leftarrow$ number of Accounts
Require: $K \leftarrow$ size of signature
Require: $N(Account) \leftarrow$ All neighbours
$T \in \mathbb{N}^{M \times K} \leftarrow \infty$ ▷ Initialise signature matrix to ∞
index $\leftarrow 1$
for all Accounts **do**
    $P \leftarrow$ permute(index) $\in \mathbb{N}^{1 \times K}$ ▷ Permute the Account index K times
    for all N(Account) **do**
        $T[i] \leftarrow \min(T[i], P)$ ▷ Compute the element-wise minimum of the signature
    index = index + 1
return T ▷ Return matrix of signatures

Efficient Generation of Minhash Signatures Minhash signatures allow for rapid estimation of the Jaccard similarities. However, care must be taken when implementing minhash generation. Calculation of the signatures is expensive: Algorithm 1 requires $O(NEK)$ computations, where N is the number of neighbours, E is the average out-degree of each neighbour and K is the length of the signature (i.e. the number of independent hash functions for estimating the Jaccard similarity). For our Twitter data these values are $N = 7 \times 10^8$, $E = 10$, $K = 1,000$. A naive Python implementation for generating minhash signatures for the dataset described in Section 5.3.2 required six days to run on a desktop computer with 6 physical (12 logical) Intel i7 5930k @3.5GHz cores and 64GB of RAM. This was prohibitive for nightly updates and so we highly optimized this part of the code base. The code was ported to the Python-to-C bridge project, Cython, which allowed us to add type information and compiler directives to turn off array bounds checking as there is a large amount of array dereferencing. We stored the input matrices in contiguous memory, removing any superfluous code such as logging and inline error checking. The loops were then rewritten to be vectorized by the SIMD processor. The fully optimized Cython version of the minhash implementation runs (in parallel on 6 cores) in approximately one hour.

Locality Sensitive Hashing (LSH) Calculating Jaccard similarities based on minhash signatures instead of the full adjacency matrix provides tremendous benefits in both space and time complexity.

System	Typical runtime (s)	Space requirement (GB)
Naive edge list	8,000	240
Minhash signatures	1	4
LSH with minhash	0.25	5

Table 5.2: Typical runtimes and space requirements for performing local community detection on the Twitter Follower network of 700 million vertices and 20 billion edges and producing 100 vertex output communities.

However, finding the closest accounts to the input seeds in Jaccard space is an onerous task. For a set of 100 seeds and our Twitter data set, nearly 70 million minhash signature comparisons would need to be performed, which dominates the runtime. Locality Sensitive Hashing (LSH) is an efficient system for finding approximate near neighbours (Indyk and Motwani, 1998).

LSH works by partitioning the data space. Any two points that fall inside the same partition are potentially similar. Multiple independent partitions are considered, which are invoked by a set of hash functions. LSH can be elegantly formulated with minhash signatures for near neighbour search in Jaccard space. This is illustrated in Figure 5.4 where each column corresponds to an account and each row corresponds to a minhash function. The minhash signatures are divided into bands containing fixed numbers of hash values. LSH exploits that similar minhash signatures are likely to have identical bands. An LSH table can then be constructed that points from each account to all accounts that have at least one identical minhash band. In Figure 5.4, accounts $A1$ and $A2$ have identical hash values in Band 1 and so will be grouped together. We independently apply LSH to every input seed to find all candidates that are similar to at least one seed.

In our implementation, we use 500 bands, each containing two hashes. For every query account the LSH table is used to lookup similar (high Jaccard coefficient) accounts. Each lookup is $O(1)$ and so the total runtime is $O(S)$ where S is the number of seeds in the query. The results are stored as a set (the candidates set) so that any duplicates resulting from separate queries are removed. Table 5.2 shows typical system runtimes for three scenarios: a naive implementation, using just minhashing and using LSH and minhashing. As most accounts share no neighbours, the LSH step dramatically reduces the number of candidate accounts and the algorithm runtime by a factor of roughly 100. LSH is essential for the real-time capability of our system.

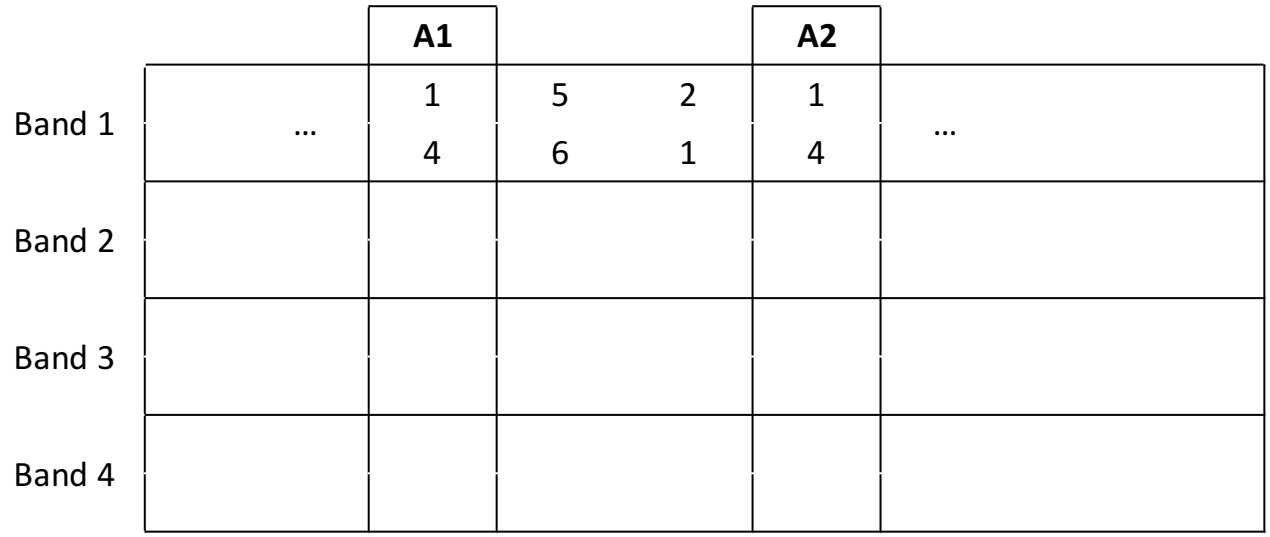


Figure 5.4: Illustration of LSH applied to minhash signatures. Each column represents a signature. The signatures have been banded up, so that each band contains two hashes. Accounts $A1$ and $A2$ will be grouped as similar candidates since they have identical signatures in Band 1.

Sorting Similarities LSH produces a set of candidate accounts that are near to at least one of the input seeds in Jaccard space. In general, we do not want every candidate returned by LSH. Therefore, we select the subset of candidates that are most similar to the seed set defined by the user.

We experimented with two sequential ranking schemes: Minhash Similarity (MS) and Agglomerative Clustering (AC). In both cases, accounts are ranked based by their Jaccard distance to the seed centre $\mathbf{X}$. The j^{th} component of the distance is given by

$$X_j = \frac{1}{n}\sum_{i=1}^{n} D(A_j, S_i)\,, \qquad j = 1, \ldots, M \tag{5.7}$$

where $D = 1 - J$ and M is the number of accounts in the dataset. At each step AC and MS augment the results set C with A^*, the closest account to the seed centre $\mathbf{X}$ such that $A^* \notin C$. However, MS uses a constant value of $\mathbf{X}$ based on the n input seeds while AC updates $\mathbf{X}$ after each step. Figure 5.5 illustrates the ranking process. A set of seeds are shown with their centre ringed. A^* denotes the closest account to the centre.

At each iteration of Algorithm 2, C and $\mathbf{X}$ are updated by first setting $C = S$ and then adding the closest account given by

$$A_t^* = \arg\min_{A_j \notin C_t} \sum_{i=1}^{n} D(A_j, (C_t)_i), \tag{5.8}$$

Algorithm 2 Agglomerative Clustering Algorithm (AC).

Require: Initial seeds S

1: Define candidate members $\bar{A} = LSH(S)$
2: $C_0 = S$
3: **repeat**
4: Compute community centre $\mathbf{X}(\bar{A}, C_t)$, see (5.7)
5: Select next Account: $A^* = \arg\min_{A_i \notin C} \mathbf{X}(A_i, C_t)$
6: Grow community: $C_{t+1} \leftarrow C_t \cup A^*, \quad \bar{A} \leftarrow \bar{A} \backslash A^*$
7: **until** Stopping criterion is met

leading to

$$C_{t+1} = C_t \cup A^*. \tag{5.9}$$

The new centre $\mathbf{X}_{n+1}$ is most efficiently calculated using the recursive online update equation

$$\mathbf{X}_{t+1}(A, C_{t+1}) = \frac{n\mathbf{X}(A, C_t) + D(A^*, C_t)}{n+1}, \tag{5.10}$$

where n is the size of C_t and the scalar operations are broadcast over each element of $\mathbf{X}(A, C_t)$.

Stopping Criterion

Both AC and MS are sequential processes and will return every candidate account unless a stopping criterion is applied. Many stopping criteria have been used to terminate seed expansion processes. The simplest method is to terminate after a fixed number of inclusions. Popular alternatives use local maxima in modularity (Lancichinetti et al., 2009) and conductance (Leskovec et al., 2010). In the context of our application, we choose a different criterion: *coverage*. We refer to the number of unique neighbours of a set of accounts as the coverage of that set.

The reason for choosing coverage as the stopping criterion is driven by one application of our work: To help define an optimal set of influencers to endorse a brand. In this context, we want to answer questions like: "What is the smallest set of athletes that have influence on over half of the users of Twitter?". This is captured by the coverage. Computing the coverage is combinatorial and requires calculating many unions over large sets. However, it can be efficiently approximated using minhash signatures. We exploit two properties of minhash signatures to do this: The unbiased Jaccard estimate through Equation 5.3 and the property that the minhash signature of the union of two sets is the element-wise minimum of their respective minhash signatures. Therefore, the properties of

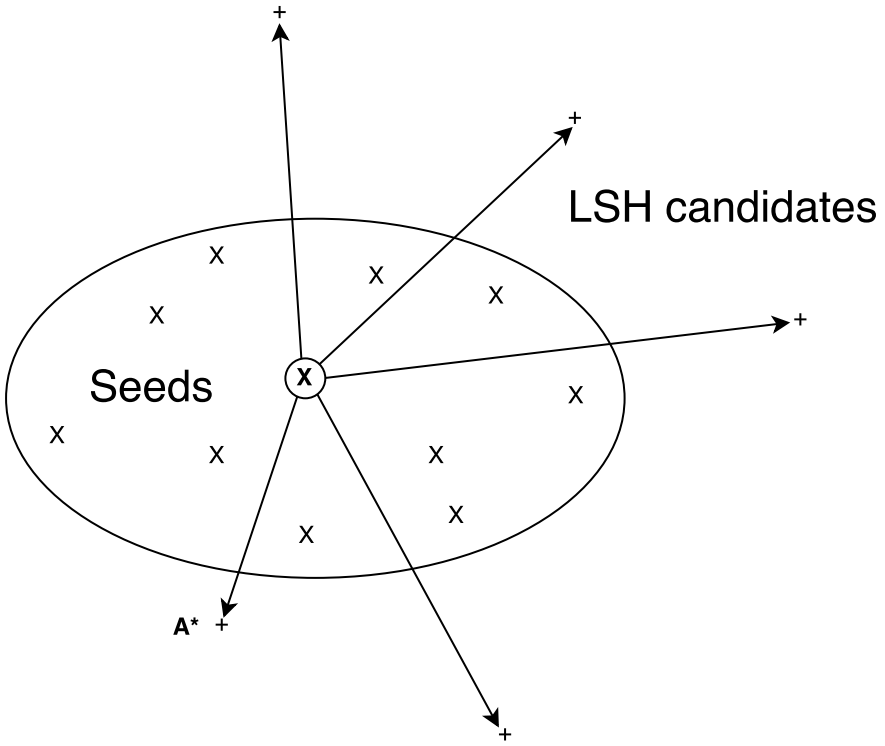


Figure 5.5: Sorting similarities of LSH candidates. The diagram shows a set of seed accounts (X) bounded by an ellipse in Jaccard space. Outside of the ellipse are a set of LSH candidate accounts (+). At each iteration the candidate account (A^*) closest (according to the Jaccard distance) to the centre ($\mathbf{X}$) of the seeds is added to the list of returned values.

minhash signatures allow us to efficiently compute the coverage and use it as a stopping criterion to rank LSH candidates without losing real-time performance. In this section we continue the convention applied to Jaccards of dropping the neighbourhood operator $N(A)$ when defining functions on neighbourhoods of accounts.

Efficient Coverage Computation The coverage y is given by

$$y = \left| \bigcup_{i=1}^{n} N(A_i) \right| , \tag{5.11}$$

which is the number of unique neighbours of the output vertices. Every time a new account A^* is added to the output $|N(C) \cup N(A^*)|$ must be calculated to update the coverage, which is an expensive operation to perform on each addition. Equation (5.12) allows us to rephrase this expensive computation using the Jaccard coefficient (available cheaply via the minhash signatures), which we subsequently use for a real-time iterative algorithm.

Lemma 1 For a community $C = \bigcup_i A_i$ and a new account $A \notin C$, the number of neighbours of the union $A \cup C$ is given as

$$|N(A \cup C)| = \frac{|N(A)| + |N(C)|}{1 + J(A, C)}. \tag{5.12}$$

Proof Following Equation (5.1), the Jaccard coefficient of a new account $A \notin C$ and the community C is

$$J(A, C) = \frac{|A \cap C|}{|A \cup C|}. \tag{5.13}$$

By considering the Venn diagram and utilizing the inclusion-exclusion principle, we obtain

$$|A \cap C| = |A| + |C| - |A \cup C|. \tag{5.14}$$

Substituting Equation (5.14) into the numerator of Equation (5.13) and dropping the Jaccard arguments yields

$$\begin{aligned} |A \cup C| &= \frac{|A| + |C| - |A \cup C|}{J} \\ &= \frac{|A| + |C|}{1 + J}, \end{aligned}$$

which proves Equation (5.12) and Lemma 1.

Lemma 2 Minhash signatures are *composable sketches* and so a community $C = \bigcup_i A_i$ can be represented by a minhash signature

$$H(C) = (h_1(C), h_2(C), \ldots, h_K(C)) \tag{5.15}$$

where $h_k : \mathbb{N}^m \to \mathbb{N}$ and

$$h_k(C) = \min(\{h_k(A_1), h_k(A_2), ..., h_k(A_N)\}), \qquad k = 1, \ldots, K. \tag{5.16}$$

Proof A minhash signature is composed of K independent minhash functions, each of which is a compound function made up of a general mapping and a minimum operation:

$$h_k(A) = \min(g_k(A)), \quad k = 1, \ldots, K, \tag{5.17}$$

where $g_k : \mathbb{N}^m \to \mathbb{N}^m$ and $\min(.) : \mathbb{N}^m \to \mathbb{N}$. Therefore,

$$h_k(\bigcup_i A_i) = \min(g_k(\bigcup_i A_i)) \tag{5.18}$$

$$h_k(C) = \min(\{g_k(A_1), \ldots, g_k(A_K)\}), \tag{5.19}$$

which proves Equation (5.15) and Lemma 2.

At each iteration we identify the next account to add using Equation (5.8) and update the coverage using Equation (5.12). The coverage of the new (augmented) community is

$$|N(C_{t+1})| = \frac{|N(C_t)| + |N(A^*)|}{1 + J(C_t, A^*)}\,. \tag{5.20}$$

The right hand side of Equation (5.20) contains three terms: $|N(C_t)|$ is what we started with, $|N(A^*)|$ is the neighbour count of A^* and $J(C_t, A^*)$ is a Jaccard calculation between the previous community C_t and the new account A^*. The minhash signature of a community is obtained via Equation (5.15). Hence, we can estimate the coverage with negligible additional computational overhead.

The process is stopped after T iterations where T is the smallest integer s.t.

$$|N(C_T)| > n_c \tag{5.21}$$

with n_c the required coverage, which is specified on a task by task basis.

5.4.2 Stage 2: Community Detection and Visualization

Stage 1 expanded the seed accounts used to query the system to include a larger set of similar accounts. Starting from the seeds, this was achieved by (1) using LSH to find a large group of candidate similar accounts (2) filtering down the candidates to the accounts most closely associated to the whole seed set.

In Stage 2, the vertices returned by Stage 1 are used to construct an intersection graph where the edge weights are given by the Jaccard coefficients of the neighbourhood graphs. Figure 5.6 depicts the process of transforming the original unweighted graph into an intersection graph. **A** shows the vertices (red larger circles) that are returned by stage one and their Followers (smaller grey circles). **B** shows a bipartite graph in which each red vertex is connected to its Followers. The specific representation used is a minhash signature that captures the Jaccard similarity between

sets of Followers. The Jaccard similarities obtained through minhash signatures are interpreted as edge weights in **C**, thereby producing a complete intersection graph in which every red vertex is connected to every other by a weighted edge. The edge weights are calculated for all pairwise associations from the minhash signatures through Equation (5.3). This process effectively embeds the original graph in a metric Jaccard space (Broder, 1997). Community detection is run on the weighted graph.

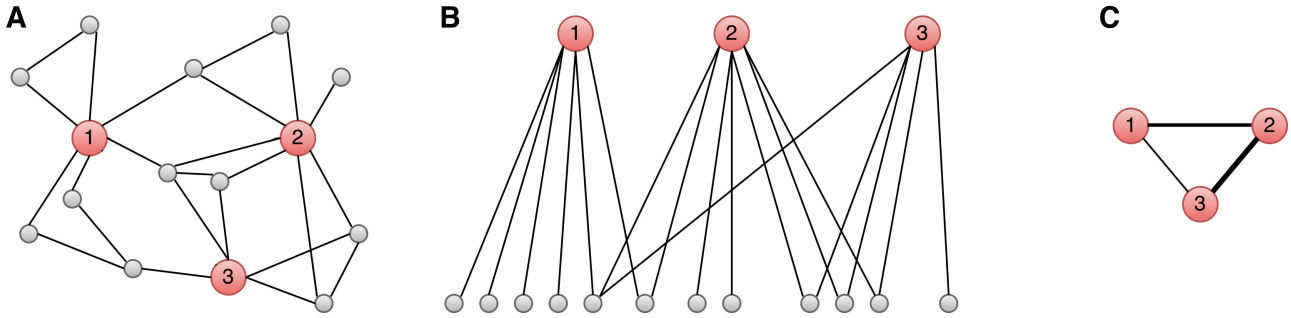


Figure 5.6: Visualizing the intersection graph generation. Interesting vertices are depicted as larger red nodes, and the neighbours as smaller, more numerous grey nodes. **A** shows a complete social network. **B** depicts the overlapping bipartite neighbourhood graphs of the three interesting vertices in **A**. **C** summarizes the network in **A** with an inferred network using the Jaccard similarity measure of the set of neighbouring vertices as edge weights. Vertices connected by high weights are more likely to be in the same community.

The final element of the process is to visualize the community structure and association strengths in the region of the input seeds. We experimented with several global community detection algorithms. These included INFOMAP, Label Propagation, Betweenness Centrality, Leading Eigenvector, Spin-glass and Modularity Maximization (Rosvall and Bergstrom, 2008; Raghavan et al., 2007; Newman, 2006, 2004a; Reichardt and Bornholdt, 2006b). The Jaccard similarity graph is a complete weighted graph whereas most community detection algorithms are designed for binary sparse graphs. As a result, all methods with the exception of label propagation and WALKTRAP were too slow for our use case. Label Propagation had a tendency to select a single giant cluster, thus adding no useful information. Therefore, we chose WALKTRAP for community visualization.

5.4.3 Time Complexity

To analyse the complexity of the end-to-end process we consider a scenario with $|S|$ seed accounts, $|C|$ output accounts, minhash signatures of size K and a total of C_a LSH candidates. Here we first analyse the simplest case of MS with a fixed number of ($|C|$) output values and then discuss how this is modified by applying AC and a dynamic stopping criteria.

The end-to-end process diagram is shown in Figure 5.2. The first step is to run $|S|$ LSH queries with $O(|S|)$ runtime. The results are returned as Python sets and finding the union is $O(\sum_s C_s)$ where C_s is the number of candidates associated with seed s[5]. For each candidate we calculate the average minhash similarity between the candidate and all of the seeds, which is $O(K|S|C_a)$. The values are sorted in $O(C_a \log C_a)$ and the top $|C|$ are retained as the output vertices. Finding all pairwise Jaccard similarities is $O(|C|^2 K)$. The Walktrap runtime reported by Pons and Latapy (2005) for a graph $G(V, E)$ is $O(|E||V|^2)$. We apply Walktrap to the complete Jaccard graph, which has $V = C$ and $|E| = |C|^2/2$ leading to a runtime of $O(|C|^3)$. Putting this all together we have a complexity of

$$T_{MS} = O(|S| + \sum_s C_s + K|S|C_a + C_a \log C_a + |C|^2 K + |C|^3). \tag{5.22}$$

Many of these values are user dependent, typical orders of magnitude for the Twitter data are $|S| = 10$, $|C| = 100$, $C_a = 1000$, $K = 1000$, $C_s = 100$. In which case Equation (5.22) reduces to

$$T_{MS} = O(|C|^2 K + |C|^3). \tag{5.23}$$

For large graphs the only values in Equation 5.22 that change are C_a and C_s. Typically $C_a > C_s \gg C$ and so Equation (5.22) becomes

$$T_{MS} = O(\sum_s C_s + K|S|C_a + C_a \log C_a). \tag{5.24}$$

Therefore the process is linear in the total number of LSH candidates associated with the seeds and slightly super linear in the total number of unique LSH candidates. Applying AC instead of MS introduces additional complexity. In Algorithm 2, there are three stages that are repeated until the stopping criteria is met: (1) Recomputing the centre, (2) finding the nearest account to the centre (3) augmenting the result set. Calculating the distance to the centre is $O(K|S|C_a)$ for the first iteration and then by applying Equation (5.10) is $O(C_a)$ for each subsequent iteration. Augmenting the result set is $O(1)$. This leads to an AC time complexity of

$$T_{AC} = O(|S| + \sum_s C_s + K|S|C_a + C_a \log C_a + |C|^2 K + |C|^3 + |C|C_a), \tag{5.25}$$

which contains an additional $|C|C_a$ term from the MS case. Finally adding a check of the stopping criteria requires evaluating Equation (5.20) at each iteration. This is dominated by the Jaccard calcu-

[5] runtimes are implementation dependent and for python data structures are given in `https://wiki.python.org/moin/TimeComplexity`

lation in the denominator $J(C_t, A^*)$. For MS this introduces an additional $O(|C|K)$ term. However for AC we are calculating $D(C_t, A^*) = 1 - D(C_t, A^*)$ in Equation (5.10) anyway and so there is no additional time complexity.

5.4.4 Space Complexity

Space complexity is dominated by the need to read the LSH table and the minhash table into memory. The minhash table has $O(NK)$ space complexity where N is the number of accounts to be hashed. In the worst case every account hashes to every other account and the LSH table has $O(N^2)$ space complexity for a complete graph. For the sparse complex networks that we are interested in this reduces to $O(N)$ as each vertex only hashes to members of the community that contains it and community sizes do not scale with the size of the network (Leskovec et al., 2008). This leads to a space complexity of $O(NK)$.

5.5 Ground-Truth Communities

Generally, community detection algorithms are based on the structure of the graph (Fortunato and Barthelemy, 2007). To provide a quantitative assessment of our method we require ground-truth labelled communities. However, no ground-truth exists for social networks at the scale we consider. Therefore, in the following, we will provide a methodology for generating ground-truth. The methodology itself must be verified, and we provide an extensive evaluation of the quality of the derived ground-truth based on the *four axioms for good community structures* defined by Yang and Leskovec (2015). These are:

1. Compactness
2. Dense interconnectivity
3. Good separation from the rest of the network
4. Internal homogeneity

However, while communities are detected using structural properties, the performance of community detection algorithms are usually verified by associating each vertex with some functional attributes, e.g., fans of Arsenal football club or Python programmers, and showing that the discovered

communities group attributes together (Yang and Leskovec, 2015). The practice of relating community membership with personal attributes is justified by the *homophily principle* of social networks (McPherson et al., 2001), which states that people with similar traits are more likely to be connected.

We reverse the process of verification by generating ground-truth from personal attributes. To generate attributes, we match Twitter accounts with Wikipedia pages and associate Wikipedia tags with each Twitter account. Wikipedia tags give hierarchical functions like 'football:sportsperson:sport' and 'pop:musician:music'. It is not possible to match every Twitter account, and our matching process discovered 127 tags that occur more than 100 times in the data. Many of them were too vague to be useful such as 'news:media' or 'Product Brand:Brands'. We selected 16 tags that had relatively high frequencies in the data set and evaluated seven metrics for each that are related to the four community quality axioms. The seven metrics are *separability*, *conductance*, *density*, *size*, *cohesiveness*, *clustering* and *conductance ratio*. Separability and conductance measure how well-separated a community is from the rest of the graph. Density and size are largely self-explanatory and measure the community's compactness. Cohesiveness, clustering and conductance ratio measure how internally homogeneous a community is. The mathematical formulation of these metrics and details of how they were calculated is provided in the Section 5.5.1.

The corresponding results are shown in Table 5.3, which is sorted by density. The emboldened rows are visualized in Figures 5.8, 5.10, 5.11 and 5.9. The density is the most important factor to distinguish good from bad communities, varying by two orders of magnitude across the data. This is followed by how well separated (separability) the community is from the rest of the network, which is inversely correlated with conductance by design (see Equations (5.29) and (5.33)). High clustering is also a useful indicator of community goodness for the best communities, but it is less useful for separating communities that consist of many sub-units (e.g., Team Sports) from bad communities (e.g., Food and Drink). Cohesiveness is generally not useful as most communities contain at least one well separated sub-unit.

To establish a clearer view of the density and homogeneity of the ground-truth we visualize the communities using network diagrams and dendrograms. The *network diagrams* are produced in Gephi (Bastian et al., 2009). The layout is generated using the *Force Atlas 2* algorithm (Jacomy et al., 2014). Colours indicate clusters based on Gephi's implementation of modularity optimization. The node (and label) sizes indicate the weighted degree of each node and are scaled to be between 5 and

Table 5.3: Properties of ground-truth communities sorted by edge density. CR stands for Conductance Ratio. High values of clustering, density and separability and low values of CR, conductance and Cohesiveness indicate good communities.

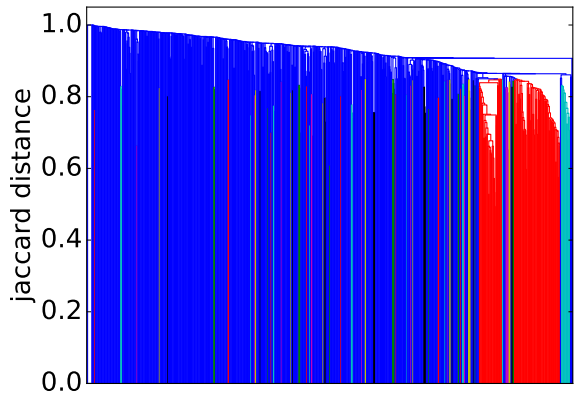
Community	Size	Clustering	Cohesiveness	Conductance	CR	Density	Separability
Mixed Martial Arts	**751**	**6.49E-02**	**4.29E-01**	**5.10E-01**	**1.19**	**3.06E-02**	**4.80E-01**
Adult Actors	352	7.20E-02	1.29E-01	7.70E-01	5.98	2.94E-02	1.50E-01
Cycling	371	6.43E-02	4.51E-01	7.04E-01	1.56	2.50E-02	2.11E-01
Baseball	616	3.64E-02	1.49E-01	7.87E-01	5.29	1.63E-02	1.35E-01
Basketball	**786**	**3.84E-02**	**3.30E-01**	**7.71E-01**	**2.34**	**1.60E-02**	**1.48E-01**
American Football	1295	2.24E-02	3.82E-01	7.40E-01	1.94	9.33E-03	1.75E-01
Athletics	530	3.48E-02	4.13E-01	8.47E-01	2.05	8.21E-03	9.01E-02
Hotel Brand	**836**	**2.20E-02**	**4.53E-01**	**8.37E-01**	**1.85**	**6.16E-03**	**9.71E-02**
Airline	363	2.30E-02	4.41E-01	9.46E-01	2.15	4.35E-03	2.84E-02
Cosmetics	332	3.34E-02	4.87E-01	9.56E-01	1.96	3.55E-03	2.32E-02
Football	4111	3.69E-02	3.95E-01	7.07E-01	1.79	2.93E-03	2.07E-01
Alcohol	**388**	**1.72E-02**	**2.34E-01**	**9.52E-01**	**4.06**	**2.66E-03**	**2.53E-02**
Travel	2038	1.27E-02	4.25E-01	8.29E-01	1.95	2.50E-03	1.03E-01
Model	2096	2.62E-02	4.04E-01	9.01E-01	2.23	1.90E-03	5.50E-02
Electronics	689	1.40E-02	4.38E-01	9.75E-01	2.23	8.78E-04	1.30E-03
Food and Drink	2974	1.76E-02	4.57E-01	9.06E-01	1.98	7.69E-04	5.18E-02

20 pixels. The network diagrams reveal any substructure present within the ground-truth. They contain too much information to easily see the individual accounts. Therefore, we magnify small sub-regions and display Twitter profile images for accounts within them. A weakness of the network diagrams is that different edge weights are hard to perceive.

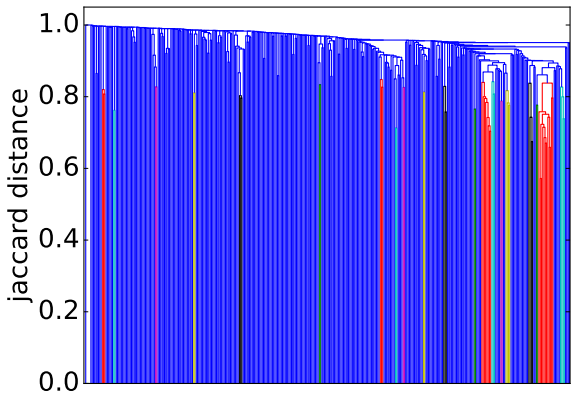
We generated dendrograms (Figure 5.7) to provide a visual representation of the interaction strength within each ground-truth community. The *dendrograms* are agglomerative: All accounts with a Jaccard distance of less than the y-value are fused together into a super-node. Any sub-groups containing more than ten nodes that have no two nodes separated by a Jaccard distance greater than 0.85 have been coloured to indicate sub-communities.

Figure 5.8 shows the Mixed Martial Arts (MMA) community. From Table 5.3 we see that this community is densely connected, strongly clustered and well separated from the rest of the network. The green region in Dendrogram 5.7c is a massive cluster where the distance between any two nodes is less than 0.85. It depicts MMA fighters, mostly fighting in the Ultimate Fighting Championship (UFC). There is a single well-separated sub-community, which is magnified in Figure 5.8 showing Olympic judo fighters. MMA is the *best* community in our study. The Cycling, Adult Actor and Athletics communities are similar in structure to MMA (see Table 5.3).

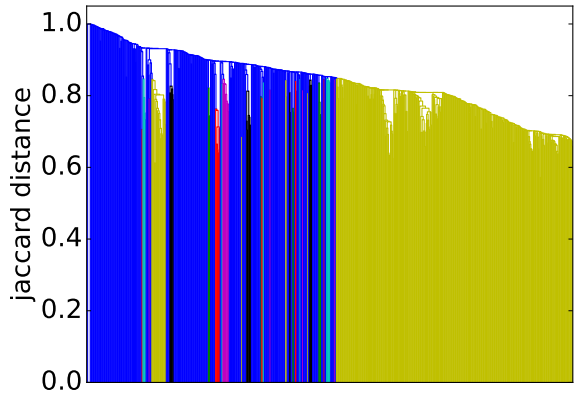
Figure 5.9 shows that the basketball community (largely NBA players) exhibits two large communities (the two NBA conferences). The individual team structure within the divisions is apparent from



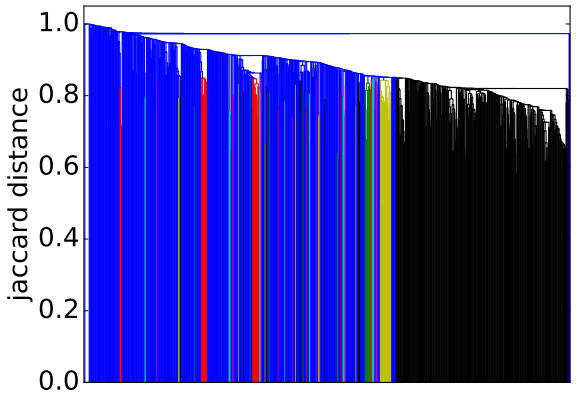
(a) **Industrial groups (here: Hotels).** Small highly connected groups due to sub-brands.



(b) **Industrial groups (here: Alcohol).** Limited interaction.



(c) **Most clearly defined communities (here: Mixed Martial Arts).** Strongly connected; sub-communities mostly due to nationality.



(d) **Team sports (here: Basketball).** Many highly connected sub-groups.

Figure 5.7: Dendrograms showing the strength of interconnection within communities. The vertical axes show the Jaccard distances. Blue areas are not strongly connected. In each non-blue region, no two nodes are separated by a Jaccard distance greater than 0.85. The dendrograms are agglomerative: All accounts with a Jaccard distance less than the y-value are fused together into a super-node. The fusing process is sequential and the x-axis indicates the order of fusing with the first nodes to agglomerate at the right.

the fine banding in Figure 5.7d where many well-connected sub-clusters, each with a distance of less than 0.85 between all pairs of nodes, are visible. We have magnified a small disconnected region of Figure 5.9, which shows players of the Women's National Basketball Association (WNBA). Other team sports (Baseball, football and American football) exhibit similar structural properties.

Figure 5.10 shows that the alcohol industry is split into four major groups representing the different classes of alcoholic drinks (wine, beer, cider and spirits). We have magnified a region of the network that contains mostly English *craft* ciders. Dendrogram 5.7b shows that the alcohol network is mostly poorly connected with only two non-blue regions indicating well connected sub-communities. From Table 5.3 it can be seen that the alcohol network exhibits a low link density and separability, indicat-

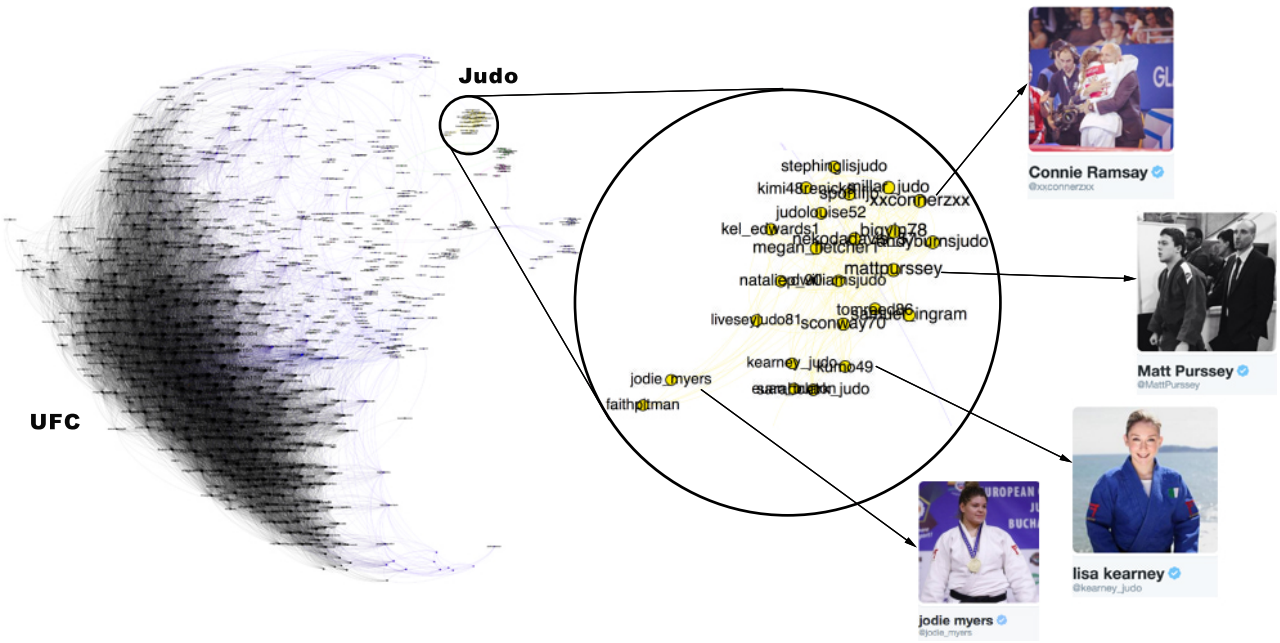


Figure 5.8: Mixed Martial Arts (MMA) community. The MMA community is relatively homogeneous and densely interconnected with high clustering and good separability from the rest of the network. The only disconnected region is the yellow region, which has been magnified to show that it is made up of Olympic judo competitors. This community is well detected by all methods.

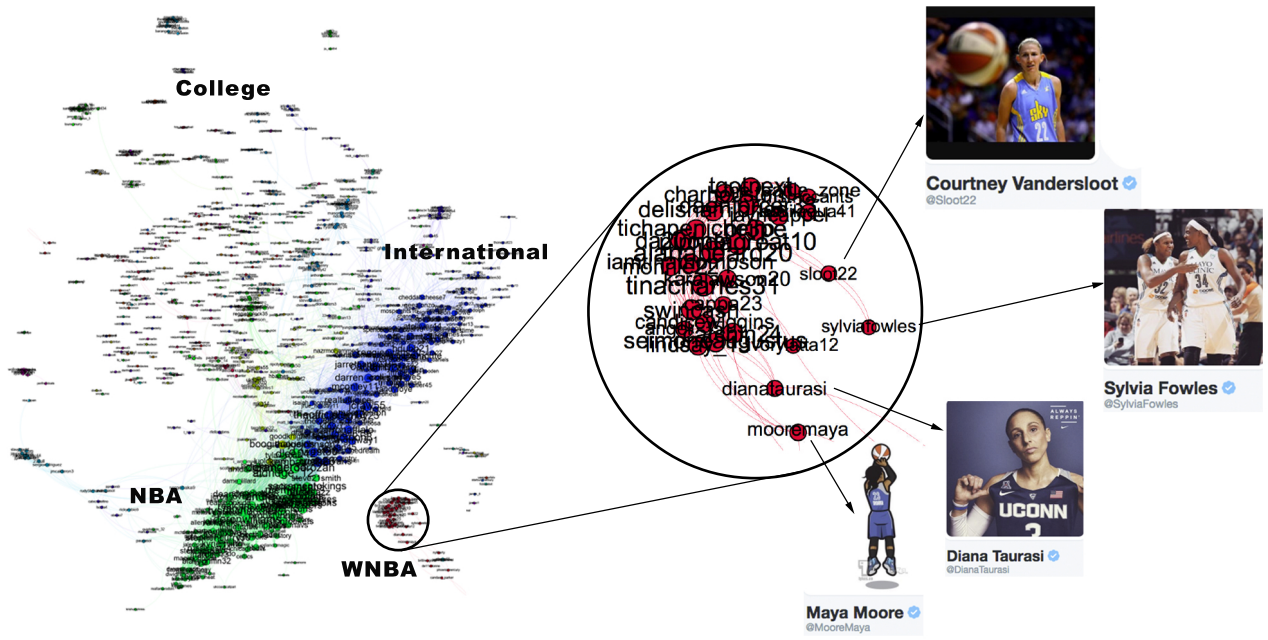


Figure 5.9: Basketball community. The basketball community has attributes similar to the baseball and American football communities: All are densely connected and well separated from the rest of the network. The individual team structure is not apparent in the graph. Instead the two large clusters show teams from the Eastern and Western Conferences. The small peripheral clusters are mostly major college teams. We have magnified an area showing players of the Womens National Basketball Association.

ing that the community lacks distinction from the rest of the network. This is a consistent pattern for other communities drawn from industrial segmentations (Food and Drink, Electronics, Model).

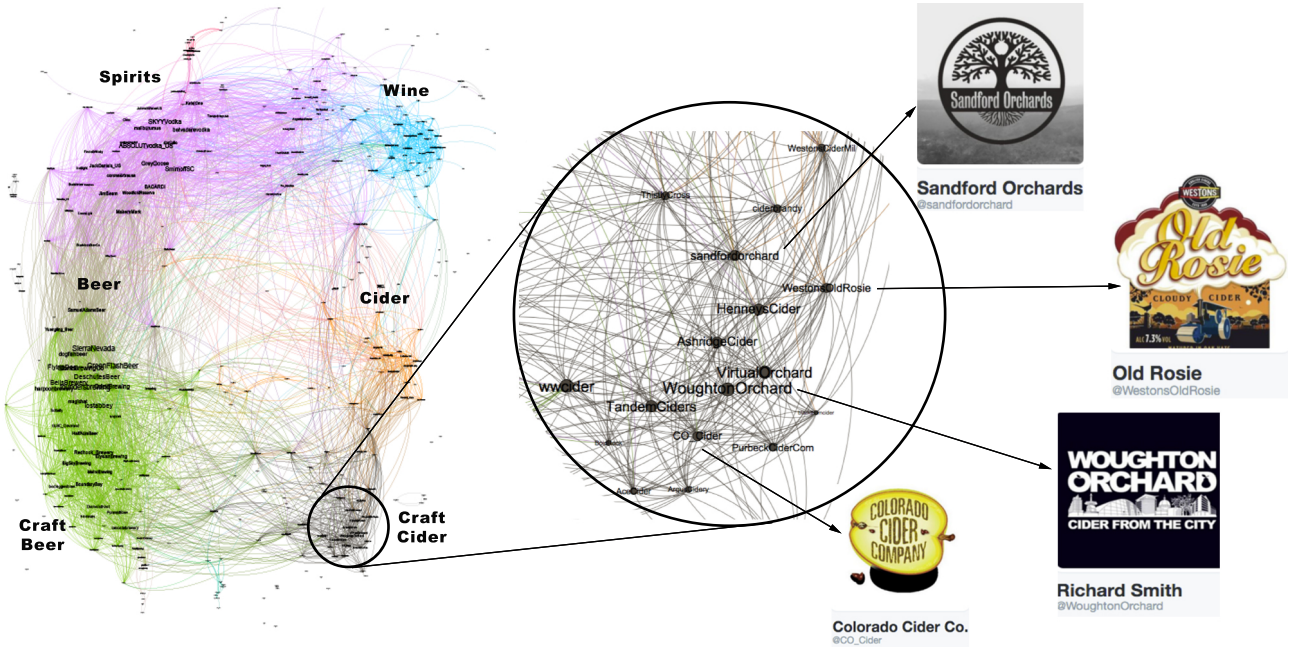


Figure 5.10: Alcohol community. This is a low-density community with poor clustering. It is divided into broad classes of drinks such as beer, spirits and wine. We have magnified an area of the cider sub-community.

Figure 5.11 shows an example of the final group of ground-truth communities: industrial groups with prominent sub-communities. In this case the major sub-communities are the Four Seasons hotel group and hotels located in Las Vegas (magnified). Dendrogram 5.7a shows that while the hotel network is generally poorly connected, there are sizable highly interconnected sub-communities. Table 5.3 shows that the hotel community exhibits low clustering (most accounts are disconnected), high cohesiveness (there are well connected sub-groups) and a low conductance ratio. The travel, airlines and cosmetics communities all share these traits (see Table 5.3).

In summary, we identified four types of ground-truth communities and evaluated their quality based on the four axioms. We found that the community types differ greatly in quality. The group containing mixed martial arts, cycling, athletics and adult actors satisfies the four axioms and form a good set of ground-truth for algorithm evaluation. The group comprising team sports (American football, baseball, basketball and football) satisfy three of the four axioms (they are not homogeneous). The remaining two community types only contain sub-groups that satisfy any of the axioms.

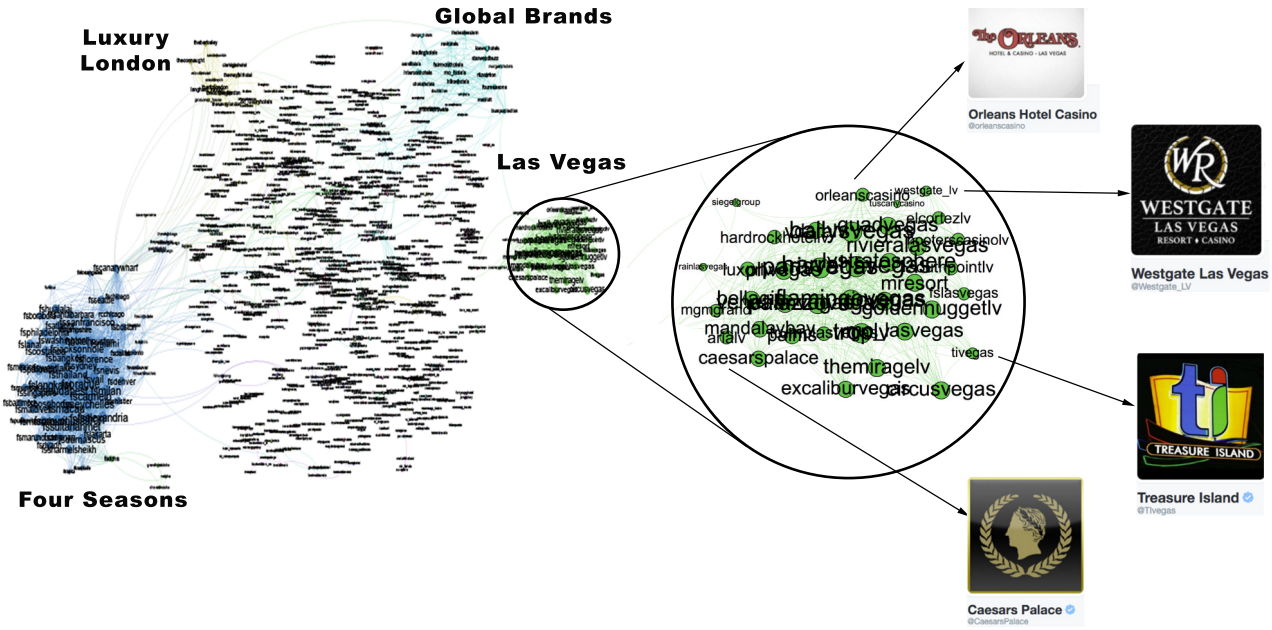


Figure 5.11: Hotels network. The hotels community has low conductance indicating that it is not well separated from the rest of the network. It also has high cohesiveness indicating it contains components that appear to be the true modular units. The two clearly visible subcomponents are the Four Seasons brand in blue to the left and the hotels of Las Vegas, which is magnified.

5.5.1 Community Axioms

Here we provide the technical specification of the community axioms as applied to weighted graphs. This section can be treated as an appendix to Section 5.5 and is not required to understand the remaining sections. It is necessary to define community axioms due to the difficulty in defining ground-truth for communities and the mismatch between structural and functional groups in graphs. Homophily only applies to attributes that improve information flow between individuals. Some attributes have no effect, or are even divisive (for instance right-handed people feel no sense of kinship) and so should not be associated with communities. Additionally, attributes may be at the wrong scale to describe structural sub-units (sports person rather than footballer). An evaluation based on ground-truth that were not communities would have no value. We apply community goodness functions to each prospective 'tag community' to identify to what extent these functional traits generate structurally observable communities.

For each functional group we generate the complete weighted intersection graph by calculating all pairwise Jaccard similarities and evaluate the six metrics in Table 5.3. They are adapted from Yang and Leskovec (2013) to apply to weighted graphs. As we work with a derived graph where each edge weight is the Jaccard similarity of neighbourhoods, the metrics have slightly different interpreta-

tions. Two entities in the derived graph are strongly connected if they have similar neighbourhoods. Since for the large entities the neighbourhood normally has at least an order of magnitude more incoming than outgoing edges, entities are closely related if they have a similar fan/follower base. We define S to be the set of vertices comprising a community and a weighted graph $G(V, E, W)$ where W is a weight matrix. The internal edge weight of S is

$$m_s = \sum_{\{i,j \in S\}} W_{i,j} \,, \tag{5.26}$$

and the weight of edges that cross the boundary of S is

$$c_s = \sum_{\{i \in S, j \notin S\}} W_{i,j} \,. \tag{5.27}$$

The community goodness metrics are then given by:

- **Clustering** exploits the idea that people in communities are likely to introduce their friends to each other. It measures how cliquey a community is. In our paradigm, clustering is high if Followers of a community recommend things for other Followers of the community to like or Follow. If a vertex has k_n neighbours then $\frac{1}{2}k_n(k_n - 1)$ possible connections can exist between the neighbours. The clustering of a node gives the fraction of its neighbours' possible connections that exist. The clustering of a community is the average clustering of each vertex. Clustering is sometimes referred to as the proportion of triadic closures in the network. The weighted clustering of the i^{th} vertex is given by

  $$Cl_i(S) = \frac{W_s^3}{(W_s W_{\max} W_s)_{ii}} \tag{5.28}$$

  where $W_{\max}$ is a matrix where each entry is the maximum weight found within S (Holme et al., 2007).

- **Conductance** is an electrical analogy for how easily information entering the community can leave it. In our context, it is defined as

  $$Con(S) = \frac{c_s}{2m_s + c_s} \,, \tag{5.29}$$

  i.e. it is the ratio of the community's external to total edge weight. A low value means that the the community is well separated from the rest of the network. In our paradigm, conductance is low if the Followers of the community are not interested in other communities.

- **Cohesiveness** measures how easily the community can be split into disconnected components. A good community is not easily broken up. The cohesiveness is given by the minimum conductance of any sub-community. A low value indicates a bad community as there is at least one well-separated sub-community. In our paradigm, low cohesiveness corresponds to members of the community having distinct, non-overlapping Follower groups.

$$Coh(S) = \min_{\{S' \subset S\}} Con(S') \tag{5.30}$$

Iterating through all subsets S' of S is impractical. Thus, we sample S' by randomly selecting ten subsets of starting vertices, running PPR community detection for each and taking a sweep through the PageRank vector to find the minimum conductance cut.

- **Conductance Ratio (CR)** is the ratio of conductance to cohesiveness and defined as

$$CR(S) = \frac{Con(S)}{Coh(S)}\,. \tag{5.31}$$

A large number indicates that the community could be broken up into structural sub-units.

- **Density** is given by the ratio of the community's total internal edge weight to the maximum possible if every edge was present with weight one:

$$Den = \frac{2m_s}{n_s(n_s - 1)} \tag{5.32}$$

A high number indicates a highly interconnected community. In our paradigm, this corresponds to a community with a well-defined Follower base that is interested in most community members.

- **Separability** measures how well the community is separated from the rest of the network. It is the ratio of internal to external edges and so is closely related to conductance:

$$Sep(S) = \frac{m_s}{c_s} \tag{5.33}$$

In our paradigm, a high value indicates that Followers of the community are not interested in much else.

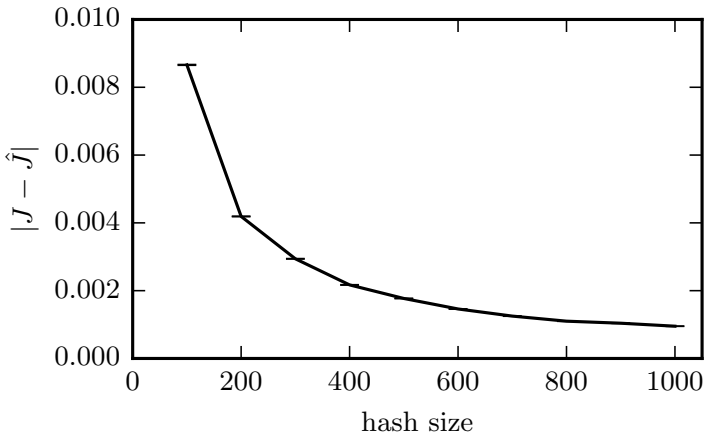


Figure 5.12: Mean absolute error from Jaccard estimation using minhash signatures as a function of the number of the hashes used in the signature. The error bars show twice the standard error using 400,000 data points. They are tight enough to be barely visible in the plot.

5.6 Experimental Evaluation

Our approach to real-time community detection relies on two approximations: minhashing for rapid Jaccard estimation and LSH to provide a fast query mechanism (based on minhashing). We assess the effect of these approximations, and demonstrate the quality of our results in three experiments:

1. We measure the sensitivity of the Jaccard similarity estimates with respect to the number of hash functions used to generate the signatures. This will justify the use of the minhash approximation for computing approximate Jaccard similarities.

2. We compare the runtime and recall of our process on ground-truth communities against the state-of-the-art Personal Page Rank (PPR) algorithm (Haveliwala, 2002) on a laptop.

3. We visualize detected communities and demonstrate that association maps for social networks using minhashing and LSH produce intuitively interpretable maps of the Twitter and Facebook graphs in real-time on a laptop.

5.6.1 Experiment 1: Assessing the Quality of Jaccard Estimates

We empirically evaluate the minhash estimation error using a sample of 400,000 similarities taken from the 250 billion pairwise relationships between the Twitter accounts in our study. We compare estimates using Equation (5.3) to exact Jaccards obtained by exhaustive calculations on the full sets using Equation (5.1). Figure 5.12 shows the mean absolute estimation error (L1 norm) as a function of

Table 5.4: Twitter accounts with the highest Jaccard similarities to @Nike. J and R give the true Jaccard coefficient and Rank, respectively. $\hat{J}$ and $\hat{R}$ give approximations using Equation (5.3) where the superscript determines the number of hashes used. Signatures of length 1,000 largely recover the true Rank.

Twitter handle	J	R	$\hat{J}^{100}$	$\hat{R}^{100}$	$\hat{J}^{1000}$	$\hat{R}^{1000}$
adidas	0.261	1	0.22	2	0.265	1
nikestore	0.246	2	0.25	1	0.255	2
adidasoriginals	0.200	3	0.18	3	0.222	3
Jumpman23	0.172	4	0.13	7	0.166	4
nikesportswear	0.147	5	0.18	4	0.137	5
nikebasketball	0.144	6	0.16	5	0.127	7
PUMA	0.132	7	0.13	6	0.132	6
nikefootball	0.127	8	0.08	17	0.110	9
adidasfootball	0.112	9	0.09	16	0.113	8
footlocker	0.096	10	0.08	17	0.096	11

the number of hashes comprising the minhash signature. Standard error bars are just visible up until 400 hashes. The graph shows an expected error in the Jaccard of just 0.001 at 1,000 hashes. The high degree of accuracy and diminishing improvements at this point led us to select a signature length of $K = 1,000$. This value provides an appropriate balance between accuracy and performance (both runtime and memory scale linearly with K).

Our system uses an iterative procedure that augments the output in similarity order and so the rank correlation is as important as the absolute Jaccard estimation error. To demonstrate that our algorithm is rank preserving we looked at the Nike Twitter account as a seed and computed the Jaccard similarities to other Twitter accounts using a) ground-truth (which is expensive to compute) and b) our proposed estimation using minhash signature of length $K = 1,000$. A top-ten ranked list of Jaccard similarities is given in Table 5.4 for the Nike Twitter account (based on the true Jaccard). Possible matches include sports people, musicians, actors, politicians, educational institutions, media platforms and businesses from all sectors of the economy. Of them, our approach identified four of Nike's biggest competitors, five Nike sub-brands and a major retailer of Nike products as the most associated accounts. This is consistent with our assertion that the Jaccard similarity of neighbourhood sets provides a robust similarity measure between accounts. We found similar trends throughout the data. This observation is consistent with the experience of analysts at Starcount, a London-based social media analytics company, who are using the tool. Table 5.4 also shows how the size of the minhash signature affects the Jaccard estimate and the corresponding rank of similar accounts. We measure the Spearman rank correlation between the true Jaccard similarities (column

R) and those calculated from signatures of length 100 (column $\hat{R}^{100}$) and 1000 (column $\hat{R}^{1000}$) to be 0.89 and 0.97 respectively. The close correspondence of the rank vector using signatures of length 1,000 and the true rank supports our decision to use signatures containing 1,000 hashes.

5.6.2 Experiment 2: Comparison of Community Detection with PPR

In the following, we move from assessing a single component (minhashing) to system-wide experimentation. We evaluate the ability of our algorithm to detect related entities by measuring its performance as a local community detection algorithm seeded with members of ground-truth communities. As a baseline for comparison we use the PPR algorithm. PPR has been identified as the state of the art method for expanding communities from small seed sets by Kloumann and Kleinberg (2014) who conducted a comprehensive assessment of local community detection algorithms on large graphs. In their study, Personal PageRank (PPR) (Haveliwala, 2002) was the clear winner. In addition, they found that the improvement in performance of PPR asymptotes after three-step random walks. Our PPR implementation uses the ground-truth seeds as the *teleport* set and runs for three iterations returning a ranked list of similar accounts.

To produce MS and AC results the seeds are input to an LSH query, which produces a set of candidate near-neighbours. For each candidate the Jaccard similarity is estimated using minhash signatures and the candidates are sorted by either the MS or AC procedures.

In all cases, we sequentially select accounts in similarity order and measure the recall after each selection. The recall is given by

$$\text{recall} = \frac{|C \cap C_{\text{true}}|}{|C_{\text{true}}| - |S|} \tag{5.34}$$

with S the initial seed set, C_{true} as the ground-truth community and C as the set of accounts added to the output. For a community of size $|C|$ we do this for the $|C| - |S|$ most similar accounts so that a perfect system could achieve a recall of one.

Twitter Dataset

In our experimentation with the Twitter dataset we randomly sample 30 seeds from each of the 16 ground-truth communities listed in Table 5.3. It is impossible to provide a fully like-for-like

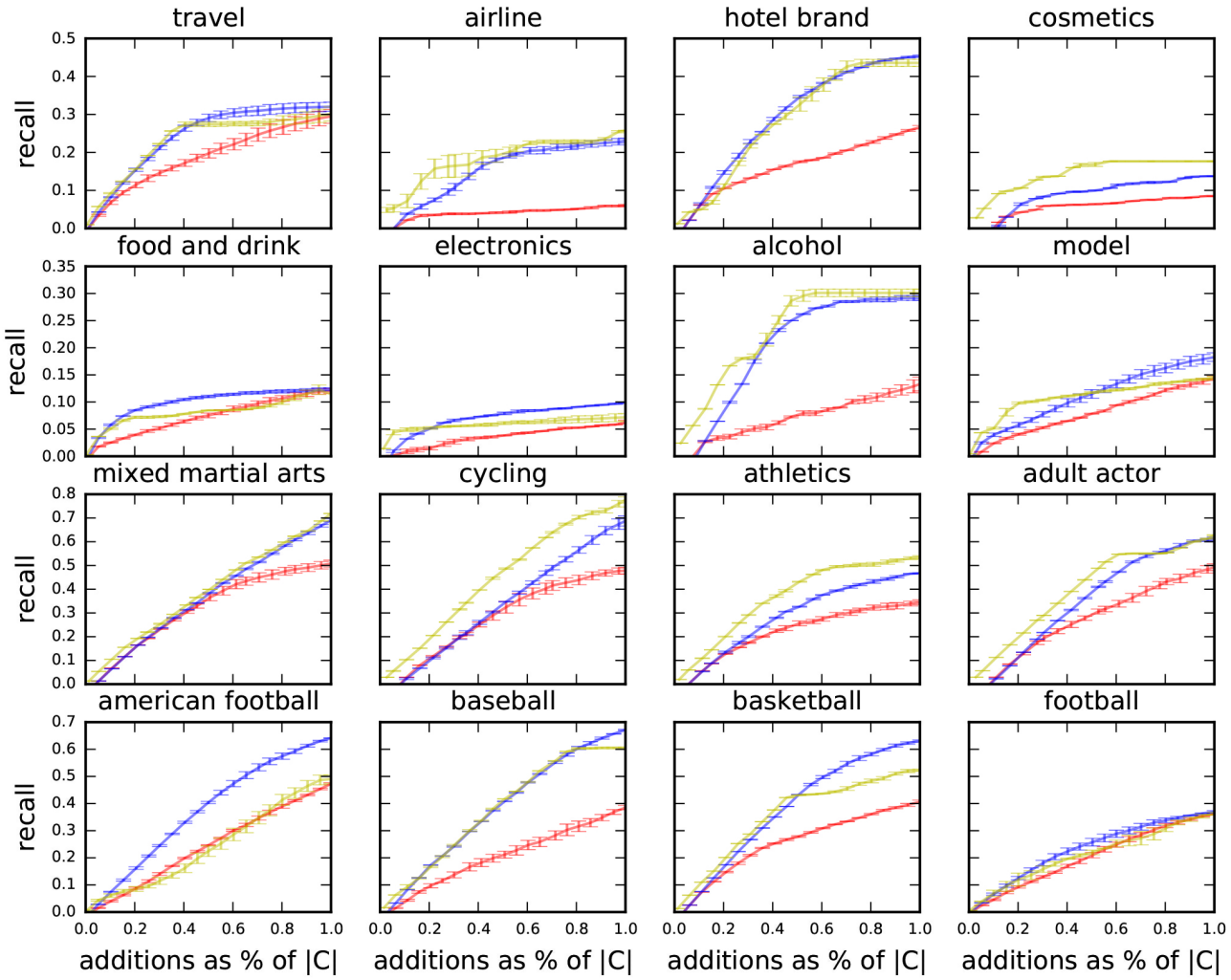


Figure 5.13: Twitter dataset average recall (with standard errors) of Agglomerative Clustering (yellow), Personal PageRank (red) and Minhash Similarity (blue) against the number of additions to the community expressed as a fraction of the size of the ground-truth communities given in Table 5.3. The tight error bars indicate that the methods are robust to the choice of seeds.

comparison with PPR: Running PPR on the full Twitter graph (700 million vertices and 20 billion edges) that we extract features from requires cluster computing and could return results outside of the hashed accounts. The alternative is to restrict PPR to run on the directly observed network of the 675,000 largest Twitter accounts, which could then be run on a single machine. We adopt this latter approach as it is the only option that meets our requirements (single machine and real-time).

We compare MS and AC to PPR operating on the directly observed network of the 675,000 largest accounts. The results of this experiment are shown in Figure 5.13 with the Area Under the Curves (AUC) given in Table 5.5. Bold entries in Table 5.5 indicate the best performing method. In all cases, both MS and AC give superior results to PPR. Figure 5.13 shows the average performance (including standard errors) over five randomly chosen input sets of 30 accounts from C_{true}. The confidence

Table 5.5: Twitter dataset area under the recall curves (Figure 5.13). Bold numeric values indicate the best performing method. Minhash similarity (MS) is the best method in 8 cases, Agglomerative Clustering (AC) in 8 cases and Personalised PageRank (PPR) in none. A perfect community detector would score 0.5.

Tags	PPR	MS	AC
travel	0.186	**0.240**	0.230
airline	0.040	0.151	**0.180**
hotel brand	0.160	**0.294**	0.285
cosmetics	0.055	0.086	**0.143**
food and drink	0.072	**0.099**	0.082
electronics	0.035	**0.069**	0.059
alcohol	0.069	0.199	**0.229**
model	0.078	**0.110**	0.109
mixed martial arts	0.317	0.363	**0.386**
cycling	0.278	0.330	**0.445**
athletics	0.219	0.285	**0.365**
adult actor	0.269	0.347	**0.397**
american football	0.240	**0.371**	0.240
baseball	0.203	**0.379**	0.378
basketball	0.252	**0.380**	0.353
football	0.202	**0.233**	0.212

bounds are tight indicating that the methods are robust to the choice of input seeds. Figure 5.13 is grouped according to the four types of ground-truth communities we discovered, where each type of ground-truth community exhibits different properties. Performance of all methods is affected by the *quality* of the communities. Communities with good values of the metrics given in Table 5.3 tend to have superior recall across all methods. The third row of Figure 5.13 contains the *best* communities as measured by the metrics in Table 5.3. For this group, recalls are as high as 80% (Cycling, using agglomerative clustering). The worst group of communities are the transnational industrial communities in the second row. The lowest recall in row three (Athletics, using PPR) is still higher than the highest recall in the second row of results (Alcohol, using agglomerative clustering). The best performing method for every community in row three of the results is agglomerative clustering. This is because AC is an adaptive method that can incorporate information from early results. The downside of an adaptive method is that pollution from false positives can rapidly degrade performance. This can be seen in the steep decrease in gradient of the AC curves for basketball, baseball and adult actors. The fourth row of the table contains team sports. Team sports also have good metrics in Table 5.3, but differ markedly in structure from the communities in row three. The team sport communities have well-defined multi-modal sub-structures generated by the different teams.

Table 5.6: Email dataset area under the recall curves (Figure 5.14). Bold entries indicate the best performing method. Minhash similarity (MS) is the best method in 10 cases, Agglomerative Clustering (AC) in 5 cases and Personalized PageRank (PPR) in none. A perfect community detector would score 0.5.

Tags	PPR	MS	AC
4	0.162	**0.271**	0.204
14	0.344	0.461	**0.473**
1	0.121	**0.382**	0.372
21	0.206	**0.306**	0.298
15	0.186	**0.396**	0.274
7	0.346	**0.450**	0.447
0	0.127	0.332	**0.384**
10	0.178	**0.345**	0.318
17	0.267	0.496	**0.503**
9	0.057	**0.210**	0.150
19	0.123	0.359	**0.383**
11	0.156	0.354	**0.361**
6	0.007	**0.102**	0.030
23	0.044	**0.103**	0.072
13	0.139	**0.357**	0.306

Both AC and MS are uni-modal procedures that store the centre of a set of data points. For a multi-modal distribution the mean may not be close to the data and so false positives will occur. As AC incorporates false positives into the estimation procedure for all future results MS outperforms AC for all team sport communities. Of the communities in the first and second rows of Figure 5.13 AC is best performing in 50% and MS is best performing in 50%. These communities are all diffuse, but some have a single densely connected region that can be found well by AC.

Email Dataset

The email dataset contains 1,005 vertices (email accounts), each of which is in exactly one of 42 communities (research departments of a large European research institute)[6]. Many of the communities are small and so we only consider the 15 largest (those having more than 25 members). Due to the size of the dataset we use minhash signatures of length 100 instead of 1,000 and only seed the communities with five seeds.

Figure 5.14 shows the average performance (including standard errors) over five randomly chosen

[6]The data and further details are available at `http://snap.stanford.edu/data/email-Eu-core.html`

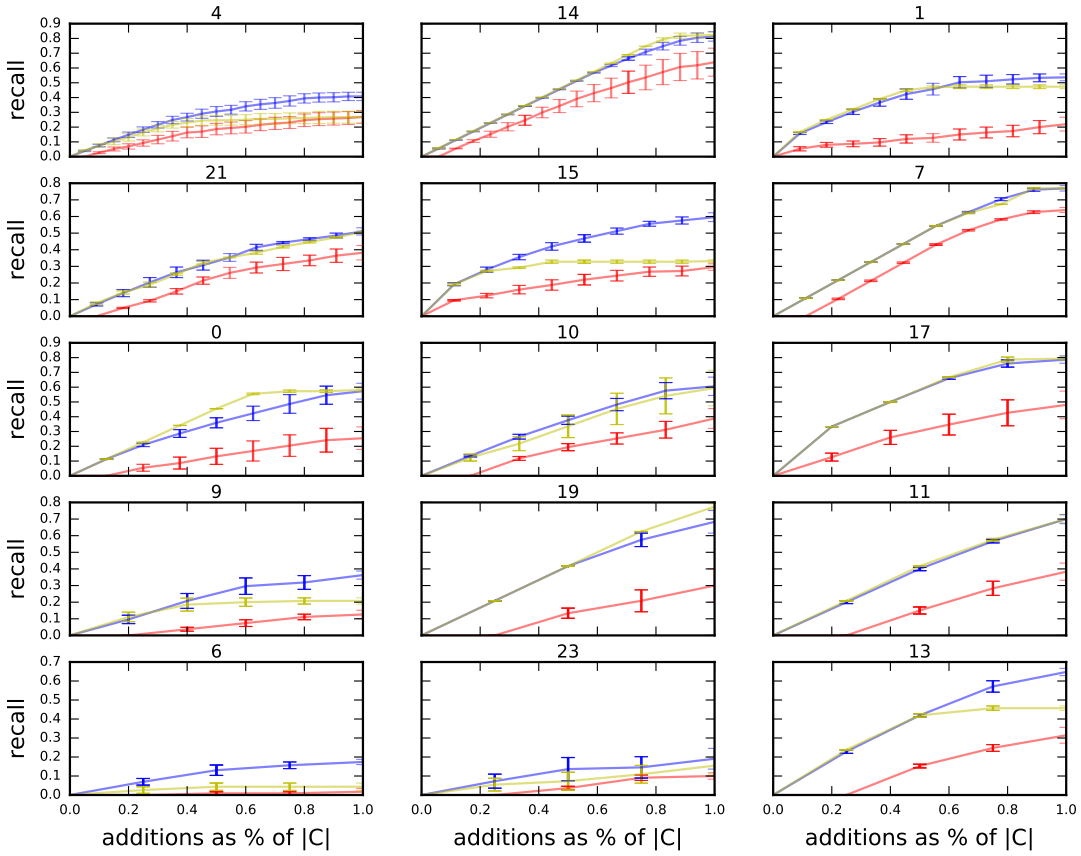


Figure 5.14: Email dataset average recall (with standard errors) of Agglomerative Clustering (yellow), Personal PageRank (red) and Minhash Similarity (blue) against the number of additions to the community expressed as a fraction of the size of the ground-truth communities given in Table 5.3. The tight error bars indicate that the methods are robust to the choice of seeds.

input sets of five email accounts from C_{true}. The confidence bounds are tight indicating that the methods are robust to the choice of input seeds. Table 5.6 gives the area under the curves of Figure 5.14. Again MS and AC outperform PPR in all cases, but this time MS is the best method for the majority of communities (10 cases), while AC is only best in five.

No mapping is provided from community labels to research departments and so for this dataset it is not possible to interpret the results in relation to the semantics of the community label.

Discussion

Much of the difference in performance of these methods derives from their respective ability to explore the graph: PPR is really a global algorithm that has been modified to find local relationships.

Table 5.7: Clustering runtimes averaged over communities.

Method	Mean(s)	Std.Dev.
PPR	12.58	8.83
MS	0.23	0.08
AC	18.6	22.0

After three iterations PPR uses first, second and third-order connections. First-order methods only consider the neighbours of seed nodes. Second-order methods also give weight to the connections of the first-order nodes (neighbours of neighbours) and so on for third-order connections. The ability to explore higher-order connections is the principal reason identified by Kloumann and Kleinberg (2014) for the state-of-the-art performance of PPR. They also note that after two iterations most of the benefit is realized, and that after three iterations there is no more improvement.

Our implementations of MS and AC are effectively second-order methods since they operate on a derived graph where the edge weight between two vertices is calculated from the overlap of the respective neighbourhoods. MS and AC outperform PPR because they are based on many more second-order connections: They run on a compressed version of the full graph instead of a subgraph. PPR is expected to perform better given more computational resources, but the additional complexity, runtime, latency or financial cost required for any scaled up/out solution will violate our system constraints.

Runtime Analysis

Table 5.7 gives the mean and standard deviation of the algorithm runtimes averaged over the 16 communities. MS is the fastest method by two orders of magnitude. Average human reaction times are approximately a quarter of a second. Therefore, MS delivers a real-time user experience (Hewett et al., 1992). As MS is the only method capable of operating in the real-time domain. Since this is a system requirement, we choose the MS procedure for experiment three and in our operational system.

5.6.3 Experiment 3: Real-Time Graph Analysis and Visualization

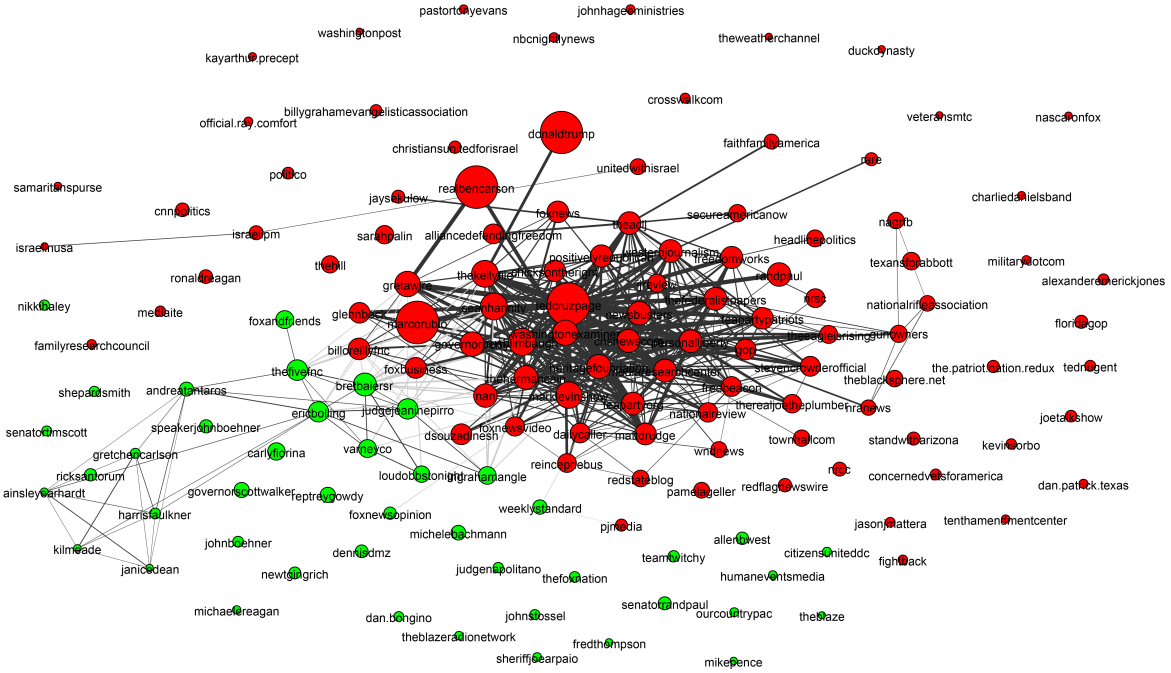
In the following, we provide example applications of our real-time community detection system to graph analysis. Users need only provide a set of seeds, wait for a fraction of a second, and the system

discovers the structure of the graph in the region of the seeds. Users can then interactively explore the discovered communities by providing different seeds based on what previous outputs reveal about the graph structure. Figure 5.15 shows results on the Facebook Page Engagements network while Figure 5.16 and Figure 5.17 use the Twitter Followers graph.

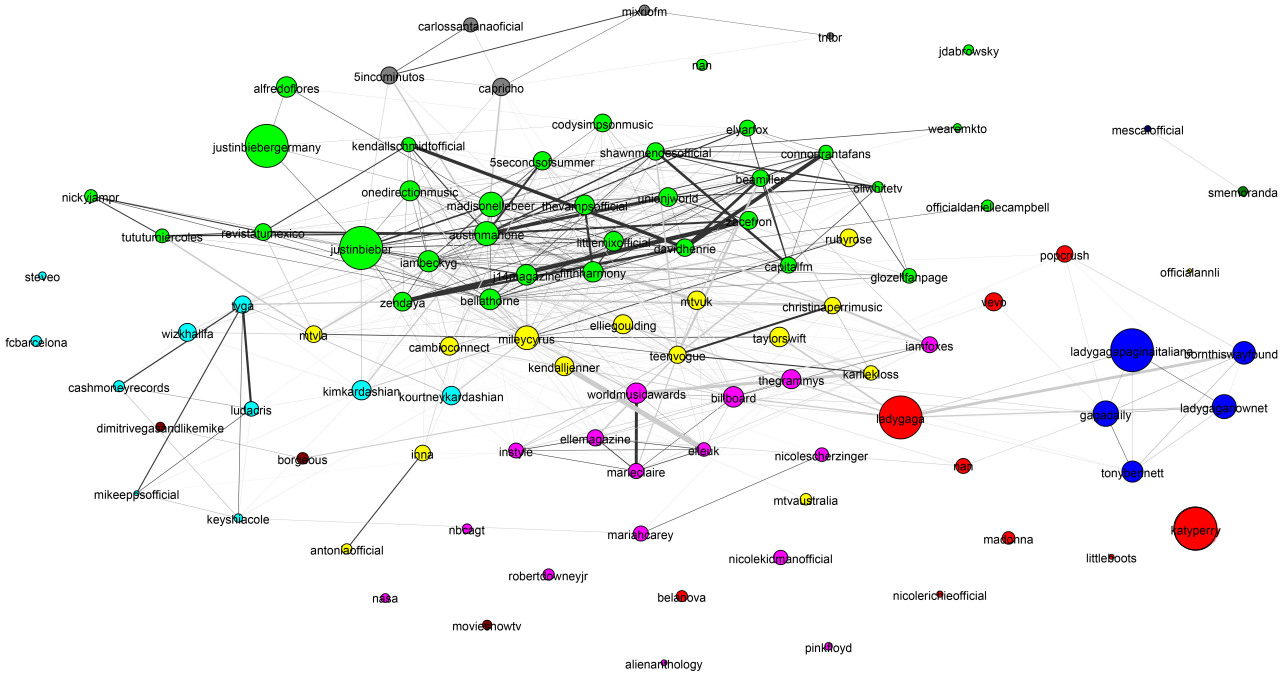
Each diagram is generated by the procedure shown in Figure 5.2: Seeds are passed to the MS process, which returns the 100 most related entities. All pairwise Jaccard estimates are then calculated using the minhash signatures. The resulting weighted adjacency matrix is passed to the WALKTRAP global community detection algorithm. The result is a weighted graph with community affiliations for each vertex. In our visualizations, the vertices are positioned using the Force Atlas 2 algorithm. The thickness of the edges between vertices represents the pairwise Jaccard similarity, which has been thresholded for image clarity. The vertex size represents the weighted degree of the vertex, but is logarithmically scaled to be between 1 and 50 pixels. The vertex colours depict the different communities found by the WALKTRAP community detection algorithm.

In addition to Twitter data, we show some results using the Facebook Pages engagement graph to demonstrate that our work is broadly applicable across DSNs. However, there are some key differences between the Facebook Pages engagement graph and the Twitter Followers graph. As Following is the method used to subscribe to a Twitter feed, Follows tend to represent genuine interest. In contrast, Facebook engagement is often used to signal approval or because a user desires an association. In addition, the Twitter graph documents user interactions between March 2006 and December 2015 (relatively few edges are ever deleted), while the Facebook graph corresponds only to events collected between March 2014, when we began collecting data, and December 2015. As a result, the Twitter dataset contains significantly more data, but with less relevance to current events. Our work uses the vast scale and richness of social media data to provide insights into a broad range of questions. Some illustrative examples are:

- **How would you describe the factions and relationships within the US Republican party?** This is a question with a major temporal component. Therefore, we use the Facebook Pages graph. We feed “Donald Trump”, “Marco Rubio”, “Ted Cruz”, “Ben Carson” and “Jeb Bush” as seeds into the system and wait for $0.25\,\mathrm{s}$ for Figure 5.15a, which shows a densely connected core group of active politicians with Donald Trump at the periphery surrounded by a largely disconnected set of right-wing interest bodies. Note that the last interaction in our



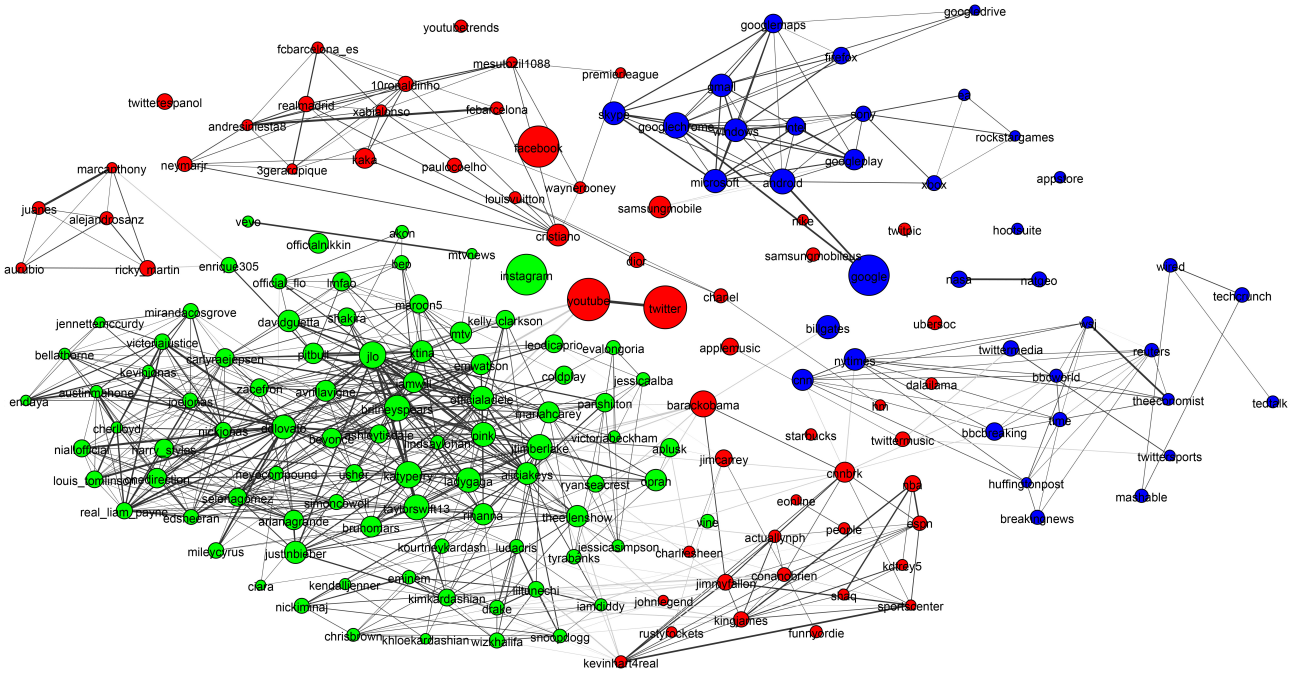
(a) The US republican party in December 2015. Seeds are “Donald Trump”, “Marco Rubio”, “Ted Cruz”, “Ben Carson” and “Jeb Bush”.



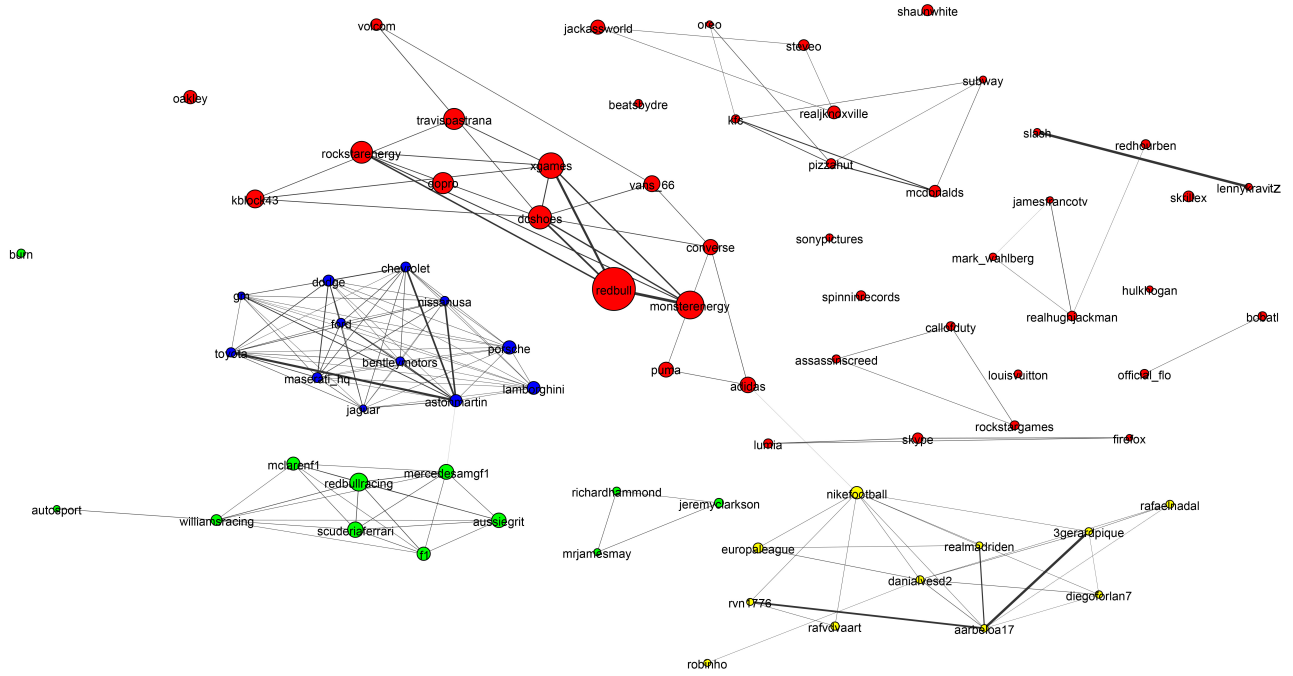
(b) Global Pop Music. Seeds are “Justin Bieber”, “Lady Gaga” and “Katy Perry”.

Figure 5.15: Visualization of the Facebook Pages engagement graph around different sets of seed vertices. The vertex size depicts degree of similarity to the seeds. Edge widths show pairwise similarities. Colors are used to show different communities.

dataset is from December 2015 and Donald Trump’s community structure will have changed significantly since then.



(a) The major social networks. Seeds are Twitter, Facebook, YouTube and Instagram.



(b) The many faces of RedBull.

Figure 5.16: Visualization of the Twitter Follower graph around different sets of seed vertices. Vertex size depicts degree of similarity to the seeds. Edge widths show pairwise similarities. Colours represent different communities.

- **Which factions exist in global pop music?** We feed the seeds “Justin Bieber”, “Lady Gaga” and “Katy Perry” into the system loaded with the Facebook Pages engagement graph and wait for $0.25\,\mathrm{s}$ for Figure 5.15b, which shows that the industry forms communities that group

genders and ethnicities.

- **How are the major social networks used?** We feed the seeds "Twitter", "Facebook", "YouTube" and "Instagram" into the system loaded with the Twitter Followers graph and wait for $0.25\,\mathrm{s}$ for Figure 5.16a, which shows that Google is highly associated with other technology brands. Instagram is closely related to celebrity while YouTube and Facebook are linked to sports and politics.

- **How is the brand RedBull perceived by Twitter users?** We feed the single seed "RedBull" into the Twitter Followers graph and wait for $0.25\,\mathrm{s}$ for Figure 5.16b, which shows that RedBull has strong associations with motor racing, sports drinks, extreme sports, gaming and football.

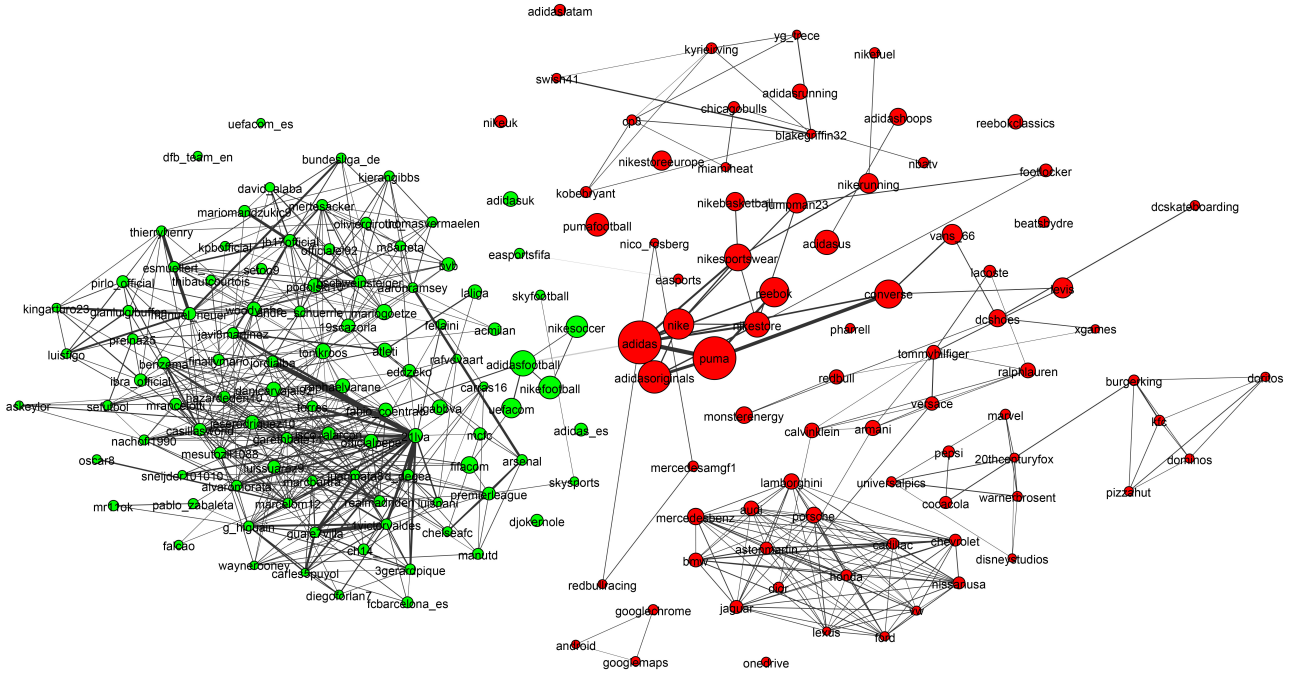
- **How does sports brand marketing differ between the USA and Europe?** We use the Twitter Followers graph. "Adidas" and "Puma" are the seeds for the European brands while "Nike", "Reebok", "UnderArmour" and "Dicks" are used to represent the US sports brands. Figure 5.17a and Figure 5.17b show the importance of football (soccer) to European sports brands, whereas US sports brands are associated with a broad range of sports including hunting, NFL, basketball, baseball and mixed martial arts (MMA).

In all cases, the user selects a set of seeds (possibly of size one) and runs our system, which returns a figure and a table of community memberships in real-time (i.e. within $0.25\,\mathrm{s}$). Analysts can then use the results to supplement the seed list with new entities or use the table of community members from a single WALKTRAP sub-community to explore higher resolution.

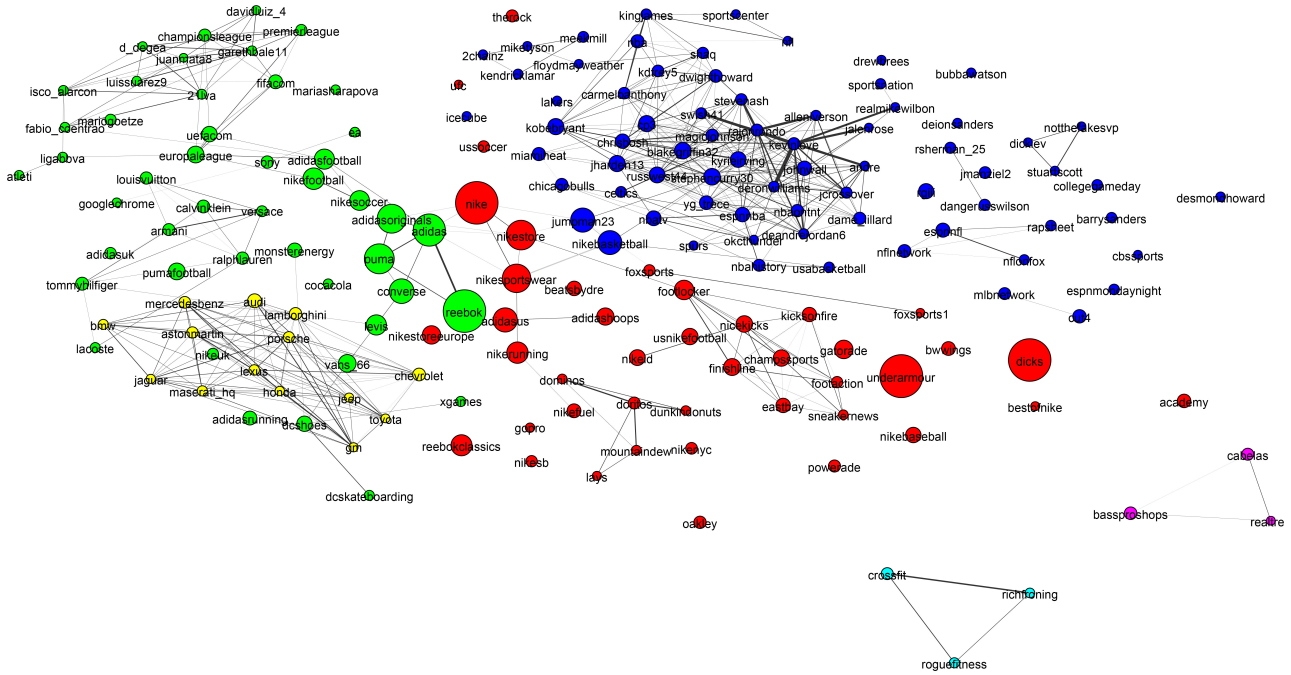
Similar tasks are traditionally conducted with techniques that are expensive and difficult to scale, e.g., telephone polling and focus groups, which often take months to return results. In contrast, our system provides an automatic analysis in the fraction of a second and at minimal cost, which allows for interactive community detection and exploration in large social networks.

5.7 Summary

We have presented a real-time system to automatically detect communities in large social networks. The system is computationally and memory efficient to the extent that it runs on a standard laptop. This work represents a technical advance leading to performance gains that are useful in practice



(a) European sport brands. Seeds are Adidas and Puma.



(b) US sports brands. Seeds are Nike, Reebok, UnderArmour, Dicks.

Figure 5.17: Visualization of sports brands from the Twitter Follower graph around different sets of seed vertices. Vertex size depicts degree of similarity to the seeds. Edge widths show pairwise similarities. Colours represent different communities.

and contains a rigorous evaluation on large social media datasets. The key contributions of this chapter are to demonstrate that (1) using the Jaccard similarity of neighbourhood graphs provides a robust association metric between vertices of noisy and sparsely connected social networks; (2)

working with minhash signatures of the neighbourhood graph dramatically reduces the space and time requirements of the system with acceptable approximation error; (3) applying LSH allows for approximate local community detection on large graphs in real-time with an acceptable approximation error. For interactive and real-time community detection, we have demonstrated that our system finds higher quality communities in less time than the state-of-the-art algorithm operating under the same constraints. Our work has clear applications for interactive knowledge discovery processes that currently rely upon slow and expensive manual procedures, such as focus groups and telephone polling. In general, our system offers the potential for organizations to rapidly acquire knowledge of new territories and supplies an alternative monetization scheme for data owners.

Our approach is good at rapidly exploring the global properties of large graphs at a relatively coarse degree of detail. For our desired application this is acceptable because we are not interested in smaller accounts. However, for use cases where it is necessary to drill down to small accounts with only tens or hundreds of connections, our method would struggle as the approximation quality would degrade and the number of signatures that must be stored would grow exponentially.

A major strength of the system is that it runs on a single machine and so does not require a computing cluster to be maintained or expertise in the associated middleware required to distribute computations. At the time of writing, many small organisations had struggled to extract value from distributed computing and frameworks such as Apache Spark (Zaharia and Chowdhury, 2010) because they lacked the necessary expertise or support to deal with a paradigm that is not yet fully mature. Acquiring support or the internal expertise to manage large distributed systems is expensive. In addition, systems with more components are necessarily more complex and so incur greater downtimes and software iterations are slower.

Our approach is also relatively robust to the presence of *fake Followers*, which are sometimes purchased to inflate the importance of a social media account. Firstly, purchasing fake Followers would only create links to other accounts that had purchased Follows from the same pool of fake accounts. Providing one of our input seeds had not done this, we would not return an account with large numbers of fake Followers. Secondly, the minhash signatures are generated from the aggregate actions of every Follower and so a significant fraction of an account's Followers would need to be fake to distort the algorithm.

We have identified three interesting extensions for future work. Firstly, we treat the input network as binary. In many settings, information is available to weight the edges. This might include message

counts, the time since a connection was established or the type of connection. Efficient methods already exist for working with minhashes of weighted sets (Manasse and Mcsherry, 2010). Therefore, a potentially fruitful progression of this work is to incorporate data with edges that can contain counts, weights and categorizations.

The second extension incorporates developments in the theory of minhashing. b-bit minhashing (Li and König, 2009) and Odd Sketches (Mitzenmacher et al., 2014) provide two promising approaches to extend our system to even larger graphs. Both offer the best cost-benefit trade-off when sets are very similar (Jaccard similarity ≈ 1) or when sets contain most of the elements in the sample space. DSN data typically contains sets that are very small relative to the sample space and with Jaccard similarities $\ll 0.5$. As an example, our Twitter data has a sample space containing 7×10^8 elements with a typical set containing 10^4 elements. The strong theoretical bounds of these algorithms do not hold in these settings. Therefore, a cost-benefit analysis would be required before implementing either in an extension.

Finally, there are many similarity metrics that can be used to compare the vertices of graph structures. We chose the Jaccard similarity because efficient embeddings are available through minhash signatures. However, it is a symmetric function, which limits expressiveness when describing the relationship between two entities. When two accounts have very different size neighbourhoods and the smaller is approximately a subset of the larger, the Jaccard is inconsistent with our intuition that the smaller account is closely related to the larger one, while the converse is not necessarily true. Fan accounts provide a social media example of this phenomenon. An alternate measure relating two sets is their penetration

$$P(A, B) = \frac{|A \cap B|}{|B|} \neq P(B, A),$$

which leads to a directed interaction graph. As the penetration can be defined in terms of the Jaccard

$$P(A, B) = \frac{J(A, B)(|A| + |B|)}{J(A, B)|B|},$$

it can be estimated from minhash signatures in much the same way. Using the penetration would require a method that can exploit directed edges and WALKTRAP (Pons and Latapy, 2005) can be formulated in this way using an asymmetric adjacency matrix to define the random walks.

Chapter 6

Predicting Attributes of Twitter Users

In this Chapter, we develop methods for predicting attributes of Twitter users based on what they Follow[1]. Specifically, we use the Twitter social graph to predict age, occupation and income. After an introductory section, the chapter is broadly split into two parts. In the first part, we develop a Bayesian model that can predict the age of every Twitter user based on what they Follow. The model scales to making predictions for 700m accounts and, unlike previous work based on language, is not restricted by linguistic or geographic boundaries. The work related to age detection is an extension of Chamberlain et al. (2017c). In the second part, we take an existing linguistic model for predicting the income and occupation of Twitter users and show that it can be improved by adding graph embeddings as features. In doing so, we develop the state-of-the-art models for income and occupation prediction on Twitter. The second part of the chapter relates to sections of Aletras and Chamberlain (2017).

6.1 Introduction

The lack of reliable (or usually any) demographic or socio-economic data is a major criticism of the usefulness of Twitter data. Enriching Twitter accounts with this information (e.g., age, occupation and income) is valuable for scientific, industrial and governmental applications such as opinion polling, product evaluations and market research. To learn user attributes, we assume that people with similar traits are likely to have similar Twitter connections. This is an example of the well-known *homophily* principle of social networks (McPherson et al., 2001). People of similar ages are

[1] we use capitalisation to indicate the Twitter specific usage of this word

Figure 6.1: Twitter profile for @williamockam that we created to illustrate our method. The profile contains the name, Twitter handle, number of tweets, number of followers, number of people following and a free-text description field with age information.

likely to be going through comparable age-related life events (e.g. education, child birth, marriage, employment, retirement, wealth changes), and people of similar incomes tend to have related occupations, belong to the same class and may have comparable education, ethnic, or political views. For attribute inference in Twitter, we exploit that most Follows are indicative of a user's interests. Putting things together, we arrive at our central hypothesis that (a) somebody Follows what is interesting to them, (b) their interests are indicative of their attributes. Hence, we propose to infer user attributes based on what they Follow. We created the artificial @williamockam account shown in Figure 6.1 to use as a running example of our method.

6.2 Related Work

There is a large body of excellent research on enhancing social data with learned attributes. A very broad range of attributes have been considered. These include gender (Burger et al., 2011), location (Cheng et al., 2010), ethnicity (Mislove et al., 2011; Pennacchiotti and Popescu, 2011), education (Li et al., 2014a), social class (Filho et al., 2014), occupation (Sloan et al., 2015) and family size (Culotta et al., 2015). In addition to learning attributes Fang et al. (2015) also researched the correlations between various user traits.

A wide array of methods have been applied to infer user attributes. Li et al. (2014a) used a relation extraction approach based on distant supervision to learn profile information of Twitter users such

as spouse, employer and education. Filho et al. (2014) used location information from user 'check-ins' in Foursquare to predict a user's social class. A pattern matching approach to extract age, occupation and social class of Twitter users in the United Kingdom was introduced by Sloan et al. (2015). Sloan (2017) later used a similar approach to analyse the characteristics of the British Twitter population. Culotta et al. (2015) introduced a distant supervised approach to generate labelled data of users mapped to a variety of demographics such as gender, age, income, education, number of children, ethnicity and political preference. The "user name" has also been considered as a source of demographic information. This was first done by Liu and Ruths (2013) to detect gender and later by Oktay et al. (2014) to estimate the age of Twitter users from the first name supplied in the free-text *account name* field (e.g. William in Figure 6.1). In their research, they use US social security data to generate probability distributions of birth years given the name. They show that for some names, age distributions are sharply peaked. A potential issue with this approach is that methods based on the "user name" field rely on knowledge of the user's true first name and their country of birth (Oktay et al., 2014). In practice, this assumption is problematic since Twitter users often do not use their real names, and their country of birth is generally unknown. Images have been used by Fu et al. (2010); Guo et al. (2008) to predict user age with some success. However, computer vision methods are difficult to apply to Twitter data as few accounts have profile images and those that do are often inaccurate or of poor quality.

While many methods have been tried, the majority of the research into attribute learning in social media focusses on information extracted from textual or linguistic features. Schler et al. (2006) popularised this approach by showing that it could be applied successfully to blogs. Huang et al. (2015) proposed a method to classify Sina Weibo[2] users into twelve predefined occupational classes using textual features. Preot et al. (2015) utilised textual features, profile information and user activity features such as number of tweets per day to infer the occupational class of Twitter users. The works of Lv et al. (2016) and Hu et al. (2016) followed similar approaches to (Preot et al., 2015) for predicting user occupation based on linguistic features. Preoţiuc-Pietro et al. (2015) proposed a method to predict user income on Twitter using topical features as well as inferred meta-features such as emotion and affect. Lampos et al. (2016) introduced a supervised model for predicting user socio-economic class (upper, medium, low) from both content based and behavioural features (e.g. number of likes, mentions). Relevant to occupational class prediction, Liu et al. (2016) predict a person's career path using demographic, psycholinguistic and textual features. Work inferring user

[2] A Chinese microblogging site similar to Twitter.

ages from tweets includes (Nguyen et al., 2011; Rao et al., 2010; Zamal et al., 2012). Notably, Nguyen et al. (2013) developed a linguistic model for Dutch tweets that allows them to predict the age category (using logistic regression) of Twitter users who have tweeted more than ten times in Dutch. They performed a lexical analysis of Dutch language tweets and obtained ground truth through a labour intensive manual tagging process. The principal features were unigrams, assuming that older people use more positive language, fewer pronouns and longer sentences. They concluded that age prediction works well for young people, but that above the age of 30, language tends to homogenise. In general, lexical approaches to age prediction suffer from the concept of social age (Nguyen et al., 2013). Social age is determined by life stage (married, children, employment etc.) rather than years since birth, and it has a strong affect on writing style. People often adapt their language to mimic the perceived social norm in a group.

Generally, tweet-based methods struggle to make predictions for Twitter users with low tweet counts. In practice, this is a major problem since we calculated that the median number of tweets for the 700m Twitter users in our data set is only 4 (the *tweets* field shown in Figure 6.1 is available as account metadata for all accounts). Attempts to improve linguistic methods by combining them with network features include Zamal et al. (2012); Pennacchiotti and Popescu (2011), who show that using the graph structure can improve performance at the expense of scalability. Kosinski et al. (2013) showed that attributes of Facebook users can be learned purely from Facebook-Likes. They predicted a broad range of user attributes mined from 58,466 survey correspondents in the US. Their approach of solely using Facebook Likes as features for learning has the benefit of generalising readily to different locales. Culotta et al. (2015) have applied a similar Follower based approach to Twitter to predict demographic attributes, however their approach of using aggregate distributions of website visitors as ground-truth is restricted to predicting the aggregate age of groups of users.

6.3 Probabilistic Age Inference in Twitter

Our work on age inference in Twitter is inspired by the generality of the approaches of Kosinski et al. (2013) and Culotta et al. (2015). However, our setting differs in two ways. Firstly, we use data native to the Twitter ecosystem to generalise from labelled examples to make individual predictions for the entire Twitter population of 700m accounts. Secondly, we do not assume that our sample is an unbiased estimate of the Twitter population and explicitly account for this bias to make good

population predictions. Our age inference method uses ground-truth labels (users who specify their age), which are then generalised to 700m accounts based on shared interests that we derive from Following patterns.

Table 6.1: Ground-truth dataset: Age categories, counts and relative frequencies. Most of the labelled data related to ages under 25. The "features" column gives the average number of accounts followed by the labelled accounts in each age category.

idx	age	count	freq	features
0	under 12	7,753	5.9%	23.7
1	12–13	20,851	15.8%	27.9
2	14–15	30,570	23.1%	30.8
3	16–17	23,982	18.1%	28.7
4	18–24	33,331	25.2%	26.0
5	25–34	9,286	7.0%	23.1
6	35–44	3,046	2.3%	22.6
7	45–54	1,838	1.0%	16.0
8	55–64	962	0.7%	11.4
9	over 65	596	0.5%	11.2

To extract ground-truth labels we crawl the Twitter graph and download user descriptions. A Twitter description is a free-text optional field. Figure 6.1 shows the profile of the fictitious @williamockam account, which we use to illustrate our approach. The description is the text "I am a 22 year old Walt Disney enthusiast and fond of razors". We use the distributed web crawler from Chapter 5, but configured to hit a different Twitter API endpoint. The details of the crawler are provided in Section 5.3. Our crawl downloaded 700m user descriptions. We search the descriptions for REGular EXpression (REGEX) patterns that are indicative of age (e.g., the phrase: "I am a 22 year old" in Figure 6.1) across Twitter's four major languages (English, Spanish, French, Portuguese). The exact REGEX patterns are given in Listing 6.1. An initial run of the REGEX revealed some frequent false positives with terms like 'I feel like I am 80' or 'I am more than 10', which were manually corrected for in the final version.

Listing 6.1: Regex matching run against Twitter descriptions. The code detects age references in English, German, French and Portuguese. Terms including 'feel like', 'think I am' and 'more / less than' were a major source of error in early versions, which led us to write a REGEX that explicitly removes them.

```
awk '{for (i=2; i<=NF; i+=2) {gsub (/,/, "p1p2p3p4p", $i)} print $0 }'
FS="\"" OFS="\"" $x > temp
awk '{print $2, $3, $6}' FS="," OFS="," temp |
sed 's/p1p2p3p4p/\,/g' |
egrep -i "[\'a][mn] [][0-9][0-9][ \,\.\!\;y][ \/yea]|
```

```
[ua]is[ ][0-9][0-9][ \,\.\!\;a][an]|bin[ ][0-9][0-9][ ]|
[hg]o[ ][0-9][0-9][ a][an][on]" >> temp1.csv
sed "s/.*[hghia\']mnso][ ]\([0-9][0-9]\)[ \,\.\!\;ya]
[ \/yean].*/\1/I;s/.*bin[ ]\([0-9][0-9]\)[ ].*/\1/I" temp1.csv |
egrep -v -i "more than [0-9][0-9]|
"think i am [0-9][0-9]|think i'm [0-9][0-9]|
i feel like [0-9][0-9] | depuis [0-9][0-9]|
[a-ln-su-z] an [0-9][0-9]" > temp2.csv
awk '{getline a < "temp2.csv"; print $0","a}' temp1.csv > temp3.csv
```

The first Twitter profile description was written in 2006 and many are out-of-date with potentially incorrect ages. To tackle the stale data problem, we restrict the ground-truth to active accounts. We do not have access to Twitter's web logs and so we defined active accounts to be those that had tweeted or Followed in the last three months. This process discovered 133,000 active users who disclosed their age (i.e. 0.02% of the 700m indexed accounts), which we use as "ground-truth" labels. For each of these we downloaded every account that they Followed. Figure 6.1 shows that @williamockam Follows 73 accounts and we downloaded each of their user IDs.

Applying REGEX matches to free-text fields inevitably leads to some false positives due to unanticipated character combinations when working with large data sets. In addition, many Twitter accounts, while correctly labelled, may not represent the interests of human beings. This can occur when accounts are controlled by machines (bots), accounts are set up to look authentic to distribute spam (spam accounts) or account passwords are hacked in order to sell authentic looking Followers. To reduce the impact of spurious accounts in the training data we note that (1) incorrectly labelled accounts can have a large effect on the model as they are distant in feature space from other members of the class/label (2) incorrectly labelled accounts that have a small effect on the model (e.g. because they only follow one popular feature) do little damage and so can be ignored. To measure the effect of each labelled account on the model we compute the Kullback-Leibler divergence $\mathrm{KL}(P||P_{\setminus i})$ between the full model and a model trained with one data point missing. Here, P is the posterior distribution given the full, labelled dataset and $P_{\setminus i}$ is the posterior using the labelled data set minus the i^{th} data point. This methodology identifies any accounts that have a particularly large impact on our predictive distribution. We flagged any training examples with $\mathrm{KL}(P||P_{\setminus i})$ more than three median absolute deviations from the median score for manual inspection. This process excluded 246 accounts from our training data and examples are shown in Table 6.2. We also randomly sampled 100 data points from across the full ground-truth set and manually verified them by inspecting the

Table 6.2: Spurious data points identified by taking the Median Absolute Deviation of the leave-one-out KL-Divergence.

Handle	Twitter Description	REGEX age	Reason to Exclude
RIAMOpera	Opera at the Royal Irish ... Presenting: Ormindo Jan 11...	11	An Irish Opera
TiaKeough13	My name Tia I'm 13 years old.	13	Hacked account
39yearoldvirgin	I'm 39 years old... if you're a woman, I want to meet you.	39	Probably not 39
50Plushealths	Retired insurance Agent After 40 years of Services.	retired	Using reciprocation software
MrKRudd	Former PM of Australia... Proud granddad of Josie & McLean...	grandparent	Outlier. Former AUS PM

descriptions, tweets and what they Follow.

6.3.1 Age Inference based on Follows

Given a set of 133,000 labelled data points (i.e. Twitter users who reveal their age) we wish to infer the age of the remaining 700m Twitter users. For this purpose, we define a set of features that can be extracted automatically. The features are based on the Following patterns of Twitter users. Once the features are defined, we propose a scalable probabilistic model for age inference. Our age inference exploits the hypothesis that a user's interests are indicative of their age, and that Twitter Follows are a proxy for interests. Therefore, the features of our model are the 103,722 Twitter accounts that are Followed by more than ten labelled accounts, which can be found automatically. Of the 73 accounts Followed by @williamockam, eight had sufficient support to be included in our model. These were: Lord_Voldemort7, WaltDisneyWorld, Applebees, UniStudios, UniversalORL, HorrorNightsORL, HorrorNights and OlanRogers. We formulate the problem as a classification, rather than a regression problem, which is consistent with previous work and standard practice for marketing applications (Nguyen et al., 2013). We use ten age categories with a higher resolution in younger ages where there is more labelled data. For our ground-truth data set, the age categories, number of accounts, relative frequency and average number of features per category are shown in Table 6.1.

Table 6.3 shows the number of labelled accounts Following each feature for @williamockam. The support is the number of *labelled* Followers summed over all age categories, while Followers gives the total number of Followers on Twitter (labelled and unlabelled). A general trend across all features (not only the ones relevant to @williamockam) is that the age distribution is peaked towards "younger" ages as fewer older people reveal their age. This can be seen for the accounts with highest support in our dataset in Table 6.5. To improve the predictive performance of the model in higher age categories we adapted the REGEX in Listing 6.1 to search for grandparents and retirees by adding

Table 6.3: Follower counts for the eight @williamockam features. The support gives their total number of Followers in our labelled data set and Followers is their total number of Followers on Twitter. Fractional counts are from assigning a distribution to grandparents.

Twitter Handle	Support	<12	12–13	14–15	16–17	18–24	25–34	35–44	45–54	55–64	≥65	Followers
Lord_Voldemort7	**273**	5	35	75	55	87	13	0	1	1	1	2.0×10^6
WaltDisneyWorld	**435**	61	100	89	80	65	20	4	7	4	4	2.5×10^6
Applebees	**191**	18	43	38	30	37	9	8	2.33	2.33	3.33	0.57×10^6
UniStudios	**60**	7	7	14	14	13	5	0	0	0	0	0.27×10^6
UniversalORL	**65**	5	13	10	15	14	4	0	1.66	1.66	0.66	0.40×10^6
HorrorNightsORL	**5**	0	0	0	1	3	1	0	0	0	0	0.04×10^6
HorrorNights	**18**	1	3	1	4	6	0	1	0.66	0.66	0.66	0.08×10^6
OlanRogers	**16**	0	2	0	7	7	0	0	0	0	0	0.11×10^6

Table 6.4: Posterior distributions from Equation (6.4) for the eight features Followed by @williamockam. Probabilities are $\times10^{-5}$.

Twitter Handle	Support	<12	12–13	14–15	16–17	18–24	25–34	35–44	45–54	55–64	≥65	Followers
Lord_Voldemort7	**273**	111.7	190.9	258.0	252.3	248.6	145.9	31.9	38.9	77.6	177.5	2.0×10^6
WaltDisneyWorld	**435**	725.0	538.2	441.2	377.6	267.3	233.2	194.2	270.7	254.5	224.4	2.5×10^6
Applebees	**191**	231.8	206.3	176.6	150.3	129.8	137.4	226.7	132.4	139.6	139.2	0.57×10^6
UniStudios	**60**	80.6	56.0	59.3	59.5	49.3	48.1	11.3	2.8	2.3	2.3	0.27×10^6
UniversalORL	**65**	67.4	63.0	56.6	60.5	50.7	42.0	21.1	62.7	86.4	40.6	0.40×10^6
HorrorNightsORL	**5**	0.3	0.7	1.5	4.0	8.3	9.4	2.0	0.3	0.1	0.1	0.04×10^6
HorrorNights	**18**	14.0	13.7	11.3	15.5	16.1	9.4	29.1	29.9	36.8	29.3	0.08×10^6
OlanRogers	**16**	4.3	9.1	10.6	21.9	19.8	5.0	1.6	1.3	1.3	1.3	0.11×10^6

Table 6.5: The accounts with the highest support within the labelled data set.

Twitter Handle	Support	<12	12–13	14–15	16–17	18–24	25–34	35–44	45–54	55–64	≥65	Followers
justinbieber	**20,359**	1517	5179	5737	4202	3073	412	99	67	34	38	8.7×10^7
katyperry	**18,395**	1467	4180	4410	3604	3575	701	158	124	75	102	9.2×10^7
taylorswift13	**15,199**	1207	3417	3674	3045	2919	507	113	117	79	122	8.1×10^7
selenagomez	**14,264**	1270	3578	3691	2847	2339	367	76	43	26	27	4.6×10^7
ArianaGrande	**13,512**	1254	3404	3604	2631	2172	319	50	40	19	20	4.1×10^7
ddlovato	**13,259**	1099	3284	3562	2741	2135	301	53	37	19	28	3.8×10^7
onedirection	**12,834**	979	3472	3778	2767	1622	138	43	20	7	8	3.0×10^7
Harry_Styles	**12,830**	912	3468	3936	2751	1581	120	24	15	9	13	2.9×10^7
NiallOfficial	**12,498**	858	3431	3895	2702	1468	90	24	15	8	8	2.7×10^7
YouTube	**11,688**	926	2496	2687	2193	2287	495	183	154	99	169	6.4×10^7

the terms “grandmother”, “grandfather”, “grandparent”, “retired” in Twitter’s four major languages. This augmented our training data with 176,748 people labelled as retired and 63,895 labelled as grandparents. In our ten-category model, retired people are added to the 65+ category. Grandparents are assigned a uniform distribution across the three oldest age categories, which roughly reflects the age distribution of grandparents in the US (Bureau, 2014)[3], such that we ended up with approximately 374,000 labelled accounts in our ground-truth data.

[3] This value was used as the US is the largest *Twitter country*.

Probabilistic Model for Age Inference

We adopt a Bayesian classification paradigm as this provides a consistent framework to model the many sources of uncertainty (noisy labels, noisy features, survey estimates) encountered in the problem of age inference. Here and for the remainder of the chapter we use calligraphic font to indicate random variables. $\mathcal{Z}$ is a scalar random variable, $\underline{\mathcal{Z}}$ is a vector random variable and $\underline{\underline{\mathcal{Z}}}$ indicates a matrix. Our goal is to predict the age label of an arbitrary Twitter user with feature vector $\mathbf{x}$ given the set of feature vectors X and corresponding ground-truth age labels A. Within a Bayesian framework, we are therefore interested in the posterior predictive distribution

$$P(\mathcal{A}|\underline{\mathcal{X}}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}}) \propto P(\underline{\mathcal{X}}|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}})P(\mathcal{A})\,, \tag{6.1}$$

where $\underline{\mathcal{X}}$ is the feature vector for the user whose age we are predicting, $\{\underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}}\}$ is the labelled ground truth, $P(\mathcal{A})$ is the prior age distribution and $P(\underline{\mathcal{X}}|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}})$ the likelihood. The prior $P(\mathcal{A})$ is based on a survey of American internet users conducted by Duggan and Brenner (2013). They sampled 1,802 over-18-year olds using random cold calling and recorded their demographic information and social media use. 288 of their respondents were Twitter users, yielding a small data set that we use for the prior distributions of over 18s. For under 18s we inferred the corresponding values of the prior using US census data (U.S. Census Bureau, 2010), which leads to the categorical prior

$$P(\mathcal{A}) = \text{Cat}(\pi) = [1, 2, 2, 3, 14, 23, 23, 22, 6, 4] \times 10^{-2}\,. \tag{6.2}$$

The likelihood $P(\underline{\mathcal{X}}|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}})$ is obtained as follows: For scalability we make the assumption that the decision to Follow an account is independent given the age of the user. This yields the factorised likelihood

$$P(\underline{\mathcal{X}}|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}}) = \prod_{i=1}^{M} P(\mathcal{X}_i|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}})^{X_i} \tag{6.3}$$

with $X_i \in \{0, 1\}$ and i indexes the features.[4] We model the likelihood factors $P(\mathcal{X}_i|\mathcal{A}, \underline{\underline{\mathcal{X}}}, \underline{\mathcal{A}})$ as Bernoulli distributions

$$P(\mathcal{X}_i|\mathcal{A} = a) = \text{Ber}(\mu_{ia}), \tag{6.4}$$

[4] We only consider cases where $X_i = 1$ since the Twitter graph is sparse: There are 7×10^8 nodes with 5×10^{10} edges, which implies a density of 1.6×10^{-7}, i.e. the default is to follow nobody. Hence, not following an account does not contain enough information to justify the additional computational cost.

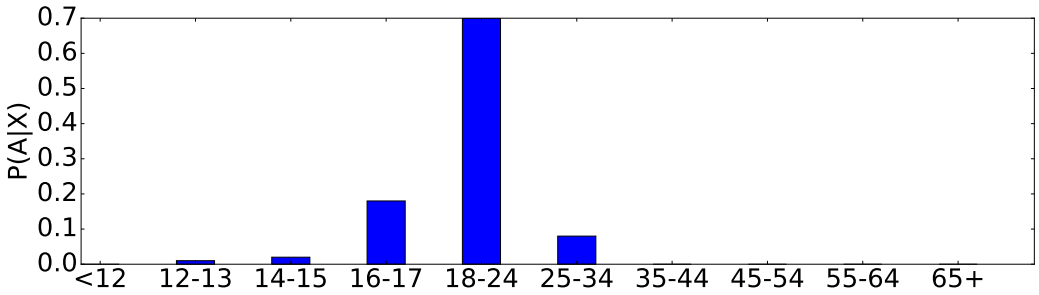


Figure 6.2: Posterior age distribution for @williamockam.

with $i = 1, \ldots, M$, where M is the number of features and there are ten age categories indexed by $a = 1, \ldots, 10$. Since our labelled data is severely biased towards "younger" age categories we cannot simply learn multinomial distributions $P(\mathcal{A}|\mathcal{X}_i)$ for each feature based on the relative frequencies of their Followers (see Table 6.1). To smooth out noisy observations of less popular accounts we use a hierarchical Bayesian model. Inference is simplified by using the Bernoulli's conjugate distribution, the beta distribution

$$\text{Beta}(\mu_{ia}|b_{ia}, c_a) \tag{6.5}$$

on the Bernoulli parameters μ_{ia}. We seek hyper-parameters b_{ia}, c_{ia} of the prior $\text{Beta}(\mu_{ia}|\underline{\underline{\mathcal{X}}}, \underline{\underline{\mathcal{A}}})$, which do not have a large effect when ample data is available, but produce sensible distributions when it is not. This is achieved by setting c_a to be constant across all features (hence dropping the i subscript) and equal to the total number of observations n_a in each age category (the count column in Table 6.1). We then set $b_{ia} = \frac{n_a n_i}{k}$, where $k = 7 \times 10^8$ is the total number of Twitter users and n_i is the number of Followers of feature i (the Followers column of table 6.3 for @williamockam's features). Then, the expected prior probability that a user Follows account i is

$$\mathbb{E}[\mu_{ia}|\mathcal{A} = a] = \frac{b_{ia}}{b_{ia}+c_a} = \frac{n_i}{k+n_i}\,, \tag{6.6}$$

i.e. it is constant across age classes and varies in proportion to the number of Followers across features. The effect of this procedure is to reduce the model confidence for features where data is limited.

Due to conjugacy, the posterior distribution on μ_{ia} is also Beta distributed. Integrating out μ_{ia} we

obtain

$$P(\mathcal{X}_i = 1|\mathcal{A}{=}a, \underline{\underline{\mathcal{X}}}, \underline{\underline{\mathcal{A}}}) = \int_0^1 P(\mathcal{X}_i{=}1|\mu_i, \mathcal{A})P(\mu_i|\underline{\underline{\mathcal{X}}}, \underline{\underline{\mathcal{A}}}, \mathcal{A})d\mu_i \tag{6.7}$$

$$= \int_0^1 \mu_{ia} P(\mu_{ia}|\underline{\underline{\mathcal{X}}}, \underline{\underline{\mathcal{A}}})d\mu_{ia} = \mathbb{E}[\mu_{ia}|\underline{\underline{\mathcal{X}}}, \underline{\underline{\mathcal{A}}}] = \tfrac{n_{ia}+b_{ia}}{n_a+b_{ia}+c_a}\,, \tag{6.8}$$

where n_{ia} is the number of labelled Twitter users in age category a who Follow feature X_i, which are given in Table 6.3 for the @williamockam features and n_a is the number of Twitter users in category a in the ground-truth (See Table 6.1). Performing this calculation yields the likelihoods for the @williamockam features shown in Table 6.4. We are now able to compute the predictive distribution in Equation (6.1) to infer the age of an arbitrary Twitter user. The predictive distribution for @williamockam is shown in Figure 6.2 and is calculated by taking the product of the likelihoods from Table 6.4 with the prior in Equation (6.2) and normalising.

The generative process in our model for the likelihood term in (6.1) is as follows.

1. Draw an age category $a \sim \text{Cat}(\pi)$
2. For each feature i draw $\mu_{ia} \sim \text{Beta}(\mu_{ia}|b_{ia}, c_a)$
3. For each account draw the Follows: $x_i \sim \text{Ber}(\mu_{ia})$

In Table 6.6, we report the five features with the highest posterior age values of $P(\mathcal{A}|\mathcal{X}_i = 1)$ for each age category. The account descriptions are taken from the first line of the relevant Wikipedia page. The youngest Twitter users are characterised by an interest in internet celebrities and computer games players. Music genres are important in differentiating all age groups from 12–45. 25–34 year olds are in part marked by entities that saw greater prominence in the past. This group is also distinguished by an interest in pornographic actors. Age categories 45–54 and 55–64 have the same top five and are differentiated by their interest in religious topics. Users older than 65 are identifiable through an interest in certain sports and politics.

6.4 Age Prediction Experimental Evaluation

We demonstrate the viability of our model for age inference in large social networks by applying it to 700m Twitter accounts. We conduct three experiments: (1) A comparison with the language-

Table 6.6: The features of the model are popular Twitter accounts. This table contains the posterior distributions $p(\mathcal{X}_i = 1|\mathcal{A} = a)$ over age for the five most discriminative (useful) features in each age class.

twitter_handle	description	<12	12–13	14–15	16–17	18–24	25–34	35–44	45–54	55–64	65+
Under 12-year olds											
RosannaPansino	vlogger	**0.40**	0.22	0.15	0.09	0.07	0.02	0.01	0.01	0.01	0.02
AntVenom	minecraft gamer	**0.40**	0.25	0.15	0.09	0.06	0.02	0.01	0.01	0.01	0.01
Bajan_Canadian	internet personality	**0.37**	0.25	0.17	0.10	0.06	0.02	0.00	0.01	0.01	0.01
shaycarl	vlogger	**0.36**	0.20	0.14	0.10	0.07	0.04	0.02	0.02	0.02	0.02
InTheLittleWood	gaming commentator	**0.34**	0.23	0.16	0.11	0.08	0.02	0.01	0.01	0.01	0.02
12–13 year olds											
ivandorschner	child TV presenter	0.18	**0.27**	0.20	0.11	0.09	0.03	0.03	0.02	0.03	0.04
Vikkstar123	youtuber	0.29	**0.26**	0.20	0.14	0.07	0.02	0.01	0.01	0.01	0.02
PeytonList	child actress	0.29	**0.25**	0.20	0.14	0.07	0.02	0.01	0.01	0.01	0.01
G_Hannelius	child actress	0.31	**0.25**	0.18	0.13	0.07	0.02	0.02	0.01	0.01	0.01
Cimorelliband	girlband	0.20	**0.25**	0.23	0.17	0.09	0.02	0.01	0.01	0.01	0.01
14–15 year olds											
therealsavannah	child pop singer	0.10	0.18	**0.27**	0.21	0.12	0.02	0.01	0.03	0.03	0.03
jessicajarrell	child pop singer	0.12	0.21	**0.26**	0.24	0.10	0.02	0.01	0.01	0.01	0.01
TheDylanHolland	child R&B singer	0.12	0.22	**0.26**	0.24	0.11	0.02	0.01	0.01	0.01	0.01
OfficialBirdy	child singer	0.10	0.17	**0.26**	0.24	0.13	0.04	0.01	0.02	0.02	0.02
officialjman	child singer	0.10	0.18	**0.26**	0.28	0.13	0.02	0.01	0.01	0.01	0.01
16–17 year olds											
TannerPatrick	singer	0.05	0.13	0.25	**0.30**	0.18	0.03	0.01	0.01	0.01	0.01
TheWordAlive	metalcore band	0.04	0.11	0.19	**0.29**	0.22	0.09	0.02	0.01	0.01	0.01
MitchLuckerSS	deathcore singer	0.05	0.14	0.23	**0.29**	0.20	0.04	0.01	0.01	0.01	0.02
metrostation	electronic band	0.03	0.07	0.15	**0.29**	0.18	0.10	0.04	0.06	0.06	0.03
BreatheCarolina	electronic band	0.06	0.15	0.22	**0.29**	0.19	0.06	0.01	0.01	0.01	0.01
18–24 year olds											
wecameasromans	metalcore band	0.05	0.13	0.22	0.28	**0.21**	0.06	0.01	0.01	0.01	0.01
Sum41	rock band	0.07	0.11	0.18	0.24	**0.21**	0.09	0.02	0.02	0.03	0.03
hopsin	rapper	0.04	0.09	0.13	0.19	**0.20**	0.09	0.09	0.06	0.05	0.07
Diablo	computer game	0.03	0.06	0.09	0.13	**0.20**	0.17	0.09	0.05	0.06	0.12
paparoach	rock band	0.04	0.09	0.14	0.19	**0.20**	0.12	0.07	0.06	0.06	0.04
25–34 year olds											
icp	hip hop duo	0.02	0.04	0.05	0.09	0.19	**0.37**	0.09	0.04	0.05	0.05
kevinrichardson	boyband member	0.02	0.03	0.05	0.09	0.16	**0.35**	0.12	0.07	0.06	0.04
skulleeroz	boyband member	0.02	0.04	0.06	0.09	0.16	**0.33**	0.12	0.07	0.06	0.05
LeeEvansNews	comedien	0.02	0.03	0.06	0.07	0.17	**0.32**	0.09	0.08	0.09	0.09
miko_lee	adult actress	0.04	0.03	0.03	0.05	0.17	**0.31**	0.08	0.07	0.08	0.14
35–44 year olds											
djspooky	hip hop artist	0.01	0.02	0.03	0.02	0.04	0.15	**0.45**	0.14	0.06	0.08
Mr_Mike_Jones	rapper	0.01	0.01	0.01	0.01	0.03	0.14	**0.44**	0.16	0.09	0.10
HISTORYTV18	history TV channel	0.02	0.03	0.03	0.05	0.09	0.14	**0.36**	0.10	0.06	0.13
TopDawgEnt	record label	0.03	0.07	0.05	0.07	0.11	0.11	**0.36**	0.09	0.03	0.07
DannySwift	boxer	0.02	0.03	0.04	0.07	0.06	0.09	**0.33**	0.12	0.08	0.16
45–54 and 55–64-year olds (identical most-discriminant features)											
JohnBevere	evangelist	0.00	0.00	0.00	0.00	0.01	0.01	0.07	**0.36**	**0.39**	0.15
edstetzer	evangelist	0.00	0.00	0.00	0.00	0.00	0.01	0.07	**0.36**	**0.39**	0.16
ChristineCaine	evangelist	0.00	0.00	0.01	0.00	0.01	0.01	0.07	**0.36**	**0.38**	0.15
womenoffaith	faith group	0.00	0.00	0.00	0.00	0.00	0.02	0.08	**0.36**	**0.38**	0.16
RELEVANT	faith magazine	0.00	0.01	0.00	0.01	0.01	0.01	0.07	**0.35**	**0.38**	0.17
People over 65											
afneil	political journalist	0.00	0.00	0.01	0.01	0.02	0.02	0.04	0.17	0.25	**0.48**
Chris_Boardman	retired cyclist	0.01	0.01	0.01	0.02	0.01	0.01	0.04	0.17	0.25	**0.47**
SkySportsGolf	golf TV channel	0.01	0.02	0.02	0.02	0.03	0.01	0.04	0.16	0.22	**0.46**
IamAustinHealey	retired rugby player	0.04	0.02	0.01	0.01	0.01	0.01	0.04	0.17	0.25	**0.45**
anthonyfjoshua	boxer	0.02	0.03	0.03	0.04	0.09	0.06	0.03	0.08	0.15	**0.45**

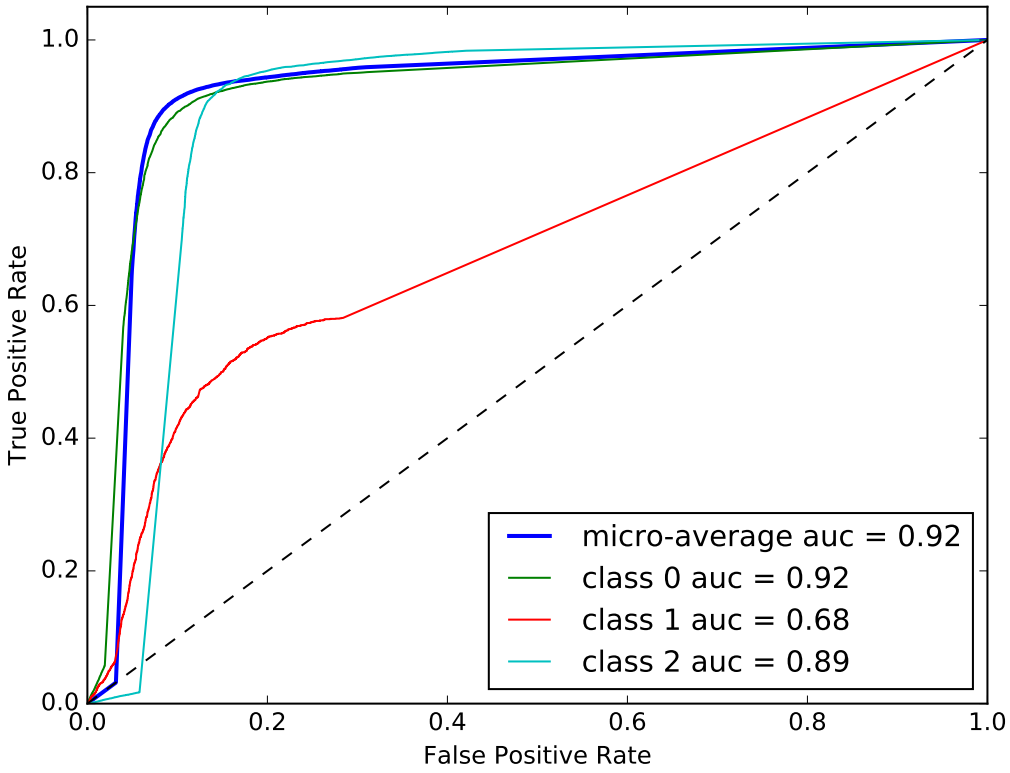


Figure 6.3: Receiver operator characteristics for three class age detection (0 = under 18, 1 = 18–45, 2 = 45+). The dashed line indicates random performance.

based model by Nguyen et al. (2013), which can be considered the state of the art for age inference. (2) A comparison with the survey by Duggan and Brenner (2013). (3) A quality assessment of our age inference on a 10% hold-out set of ground-truth labels and a comparison with the results obtained from inference based solely on the prior derived from census and survey data in (6.2) for age prediction.

6.4.1 Comparison with Dutch Language Model

For comparison with the state-of-the-art linguistic model of Nguyen et al. (2013) that used Dutch language tweets, we consider the performance of our model as a three-class classifier using age bands: under 18, 18–44 and 45+.

Table 6.7 gives the performance of our age inference algorithm on a 10% hold-out test set and the Dutch Language Model (DLM) proposed by Nguyen et al. (2013). Both methods perform equally well with a Micro F1 score of 0.86. The precision and recall show that the DLM approach is efficient, extracting information from only a small training set (support). This is because significant engineering work went into labelling and feature design. In contrast, our feature generation process is

	Our Approach			DLM (Nguyen et al., 2013)		
	<18	**18–44**	**≥45**	**≤18**	**18–44**	**≥45**
Support	7,096	3,676	30,690	1,576	608	310
Precision	0.76	0.39	**0.96**	**0.93**	**0.67**	0.82
Recall	0.68	0.50	**0.95**	**0.98**	**0.75**	0.45
Micro F1	**0.86**			**0.86**		

Table 6.7: Performance for three-class age model.

Table 6.8: Statistics for age prediction on a held-out test set.

		<12	**12–13**	**14–15**	**16–17**	**18–24**	**25–34**	**35–44**	**45–54**	**55–64**	**≥65**
	Test Cases	651	1,731	2,678	2,036	2,670	776	230	5,058	5,145	20,487
Ours	Recall	**0.19**	**0.20**	**0.38**	**0.23**	**0.33**	**0.25**	0.18	**0.32**	**0.41**	**0.30**
	Precision	**0.22**	**0.33**	**0.36**	**0.24**	**0.31**	**0.15**	**0.07**	**0.14**	**0.19**	**0.79**
	Micro F1	**0.31**									
S&C	Recall	0.01	0.02	0.02	0.03	0.14	0.23	**0.23**	0.22	0.06	0.04
	Precision	0.02	0.04	0.06	0.05	0.06	0.02	0.01	0.12	0.12	0.49
	Micro F1	0.07									

automatic and scalable. While we do not achieve the same performance for the lower age categories, for the oldest age category, our approach performs substantially better than the method by Nguyen et al. (2013), suggesting that a hybrid method could perform well. We leave this for future work.

The major advantages of our model over the state-of-the-art approach are twofold: Firstly, we have applied our age inference to 700m Twitter users, as opposed to being limited to a sample of Dutch Twitter users with a relatively high number of Tweets. Secondly, generating our training set is fully automatic and relies only on Twitter data[5], i.e. no manual labelling or verification is required.

Figure 6.3 shows the areas under the receiver-operator characteristics (ROC) curves for our three-class model. The curves are generated by measuring the true positive and false positive rates for each class over a range of classification thresholds. A perfect classifier has an area under the curve (AUC) equal to one, while a completely random classifier follows the dashed line with an $AUC = 0.5$. Performance is excellent for classes under 18 and over 45, but weaker for 18–45 where training data was limited, which we note as an area for improvement in future work.

6.4.2 Comparison with Survey and Census Data

Here we report results on inferring the age of arbitrary Twitter users with a ten age category model. Figure 6.4 shows aggregate classification results for 700m Twitter accounts compared with expected counts based on survey data (S&C) Duggan and Brenner (2013). Our model predicts that over 50%

[5]Nguyen et al. (2013) used additional LinkedIn data for labelling

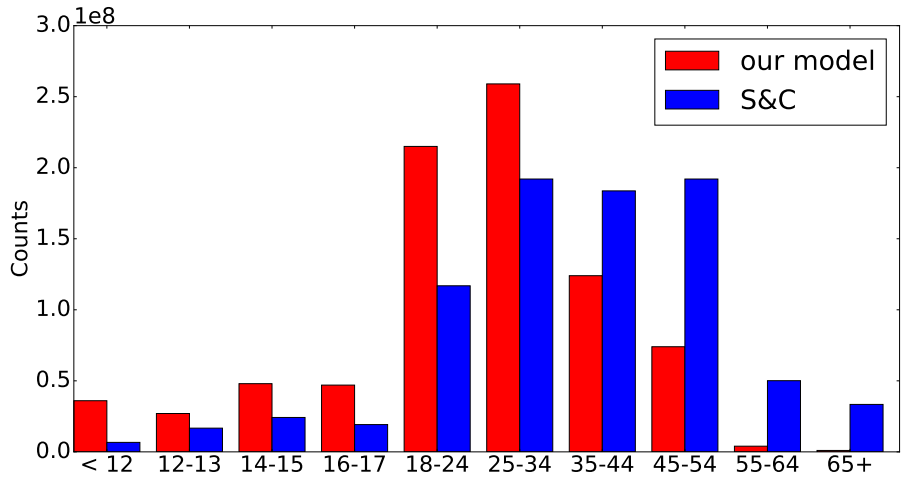


Figure 6.4: Red bars show #accounts that our model allocated to each age class using the mode of the predictive posterior. Blue bars show #accounts that would have been allocated to each age class if ages were drawn from the Survey and Census (S&C) prior.

of Twitter users are between 18 and 35, i.e. the bias of the original training set has been removed due to the Bayesian treatment. It is likely that the S&C data under-represents young people as the increased rates of technology uptake amongst younger people were not take into account when converting the census data.

6.4.3 Quality Assessment

In the following, we assess the quality of our age inference model (10 categories) on a 10% hold-out test data set. Table 6.8 shows the performance statistics for this experiment. The majority of the test cases are in the younger age categories (due to the bias of young people revealing their age) and in older age categories (due to the inclusion of grandparents and retirees). Table 6.8 shows that the precision depends on the size of the data (e.g., predicting 25–44 year categories is hard) whereas the recall is fairly stable across all age categories.[6] Our model significantly outperforms an approach based only on the survey and census data (S&C), which we use as a prior. This highlights the ability of our model to adapt to the data.

Table 6.9: Details of the publicly available age dataset.

attribute	value
$\|V_1\|$	31,389
$\|V_2\|$	50,190
$\|E\|$	1,810,569
avg D_1	57.7
max D_1	2049
std D_1	95.2
avg D_2	36.1
max D_2	4405
std D_2	96.2

(a) Public dataset adjacency matrix statistics. Subscript 1 describes labelled accounts and 2 describes features. V denotes vertices, E edges and D degree.

idx	age range	count
1	10-19	4486
2	20-29	4485
3	30-39	4487
4	40-49	4485
5	50-59	4484
6	60-69	4481
7	70-79	4481

(b) A balanced sample of the dataset made publicly available.

6.4.4 Public Age Dataset

For reproducibility we make an anonymised sample of the data and our code publicly available [7]. The data is in two parts: (1) A sparse bipartite adjacency matrix; (2) a vector of age category labels. This dataset was collected and cleaned according to the methodology described above and then down-sampled to give approximately equal numbers of labels in each of seven classes detailed in Table 6.9b. It includes only accounts that explicitly state an age (ie. no grandparents or retirees). The adjacency matrix is in the format of a standard (sparse) design matrix and includes only features that are Followed by at least ten examples. The high level statistics of this network are described in Table 6.9a.

6.5 Income and Occupation Prediction

Our research on income and occupation takes a different approach to the work on age prediction. The goal of our age detection model was to make massively scalable, language and geography agnostic predictions. For income and occupation learning, we take a public benchmarked dataset and show that we can better the state-of-the-art linguistic models by adding features derived from the structure of the graph.

[6] Without the inclusion of grandparents and retirees in the training set, the predictive performance would rapidly drop off for ages greater than 35.

[7] `https://github.com/melifluos/bayesian-age-detection`

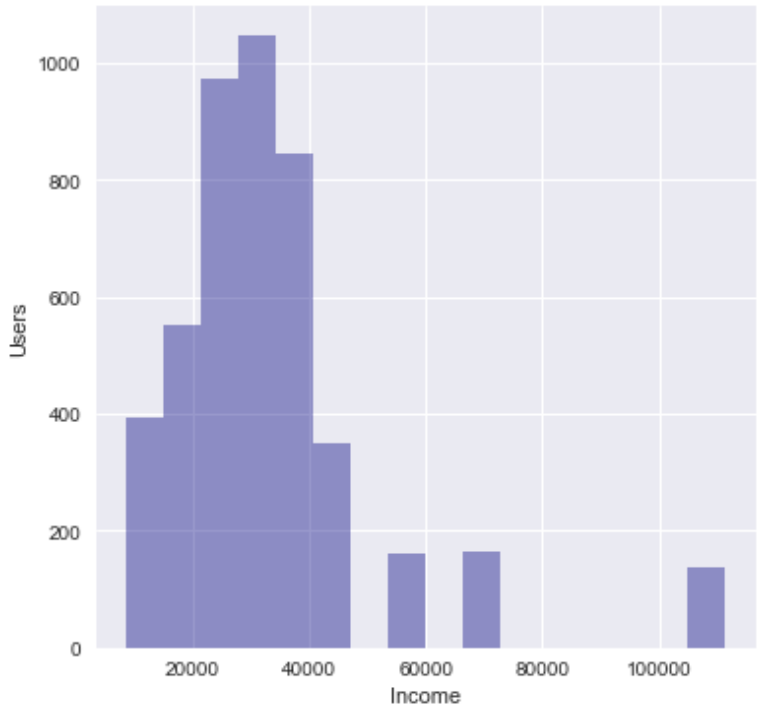


Figure 6.5: Distribution of users and income in pounds sterling (£).

6.5.1 Datasets

The baseline occupational dataset is described in detail by Preot et al. (2015). They use a taxonomy provided by the British Office of National Statistics (ONS) called the Standard Occupational Classification (SOC) (Elias and Birch, 2010) [8]. The SOC is a hierarchical taxonomy of British occupations with four different levels. At the lowest level, specific job titles such as "mechanical engineer" or "legal clerk" are given. The highest level of the hierarchy has just nine job classes and is shown in Table 6.10. Preot et al. (2015) queried Twitter's search API for each of the SOC lowest level job titles, returning a list of 200 accounts that contained the job title in their description fields. Every returned account was manually queried by the authors who removed roughly 50% either because they were spurious or did not tweet enough to learn features for their linguistic model. This process produced $5,191$ occupation labelled accounts.

The income dataset was inferred from the occupation dataset by Preoţiuc-Pietro et al. (2015). To map from occupations to income they use the 2013 ONS Annual Survey of Hours and Earnings (Bird, 2004) and assigned each of the $5,191$ occupation labelled Twitter accounts to the mean annual income of their occupation.

[8] details are available at `https://goo.gl/ExPSoq` accessed 9/12/17

C	Title	count
1	Managers, Directors and Senior Officials	461
2	Professional Occupations	1615
3	Associate Profess. and Technical Occupations	950
4	Administrative and Secretarial Occupations	168
5	Skilled Trades Occupations	782
6	Caring, Leisure and Other Service Occup.	270
7	Sales and Customer Service Occupations	56
8	Process, Plant and Machine Operatives	192
9	Elementary Occupations	131
	Total	4625

Table 6.10: Distribution of users (U) across occupational classes (C).

The graph data to learn embeddings was downloaded again using the same distributed web crawler described in Chapter 5. We configured the crawler to download all friends and followers of the $5,191$ accounts with income and occupation labels. Of the original accounts identified by Preot et al. (2015), some had since been closed or were set to private meaning that their social graph could not be collected. We were able to download the neighbourhood graphs of $4,625$ of the original Twitter users and their distribution over the nine SOC groups are shown in Table 6.10.

6.5.2 Linguistic Features

The baseline models learn Twitter user incomes and occupations from the language of their tweets. A corpus of Twitter language is first extracted from the Twitter Gardenhose stream, which is a 10% representative sample of the entire Twitter stream, from 2 January to 28 February 2011. This corpus yields a vocabulary of $71,555$ Twitter words after infrequent words are removed. The words in the corpus are hard clustered into 200 groups. Twitter users are then represented as the histogram distribution of the words in their last 200 tweets over the clusters.

Preot et al. (2015) experimented with several methods to cluster words and the most successful used word embeddings as described in Section 4.3.1. The embeddings are learned from the baseline corpus of Gardenhose tweets using the Gensim (Rehurek and Sojka, 2010) implementation of SkipGram with Negative Sampling (See Section 4.3.5). They evaluated several different embedding dimensions and found that 50-dimensions gave the best results. Clusters are generated by performing spectral clustering (Verma and Meila, 2003) on a matrix $A_{ij} = \mathbf{v}_i^T \mathbf{v}_j$ where $\mathbf{v}_i$ is the vector embedding of the i^{th} word. The number of topics must be specified a-priori and this was determined empirically to be

200.

6.5.3 Baseline Models

In the experiments of Preot et al. (2015) and Preoţiuc-Pietro et al. (2015) many ML models are used and we repeat the same experimental setup to provide a fair comparison between linguistic models and graph based features. The baseline models are all introduced in Chapter 2 and here we present specific implementation details.

Preot et al. (2015) define the occupational class of a Twitter user as a 9-way classification task and train three distinct classifiers for this task. These are Logistic regression Classifiers (LC) (Zou and Hastie, 2005), Support Vector Machines (SVM) (Joachims, 1998) and Gaussian Process Classifiers (GPC) (Rasmussen and Williams, 2006). All of the classifiers[9] are trained using the one-vs-all approach. The LC model is a standard logistic regression model with an elastic net regularisation (see Section 2.1). An elastic net uses a linear combination of the L1 and L2 penalties to regularise the logistic regression model to reduce model complexity and thus overfitting. Our SVM model (see Section 2.2) uses a Radial Basis Function (RBF) kernel since that allows us to capture nonlinearities within the data. The GPC is a Bayesian non-parametric Gaussian Process for classification (see Section 2.3) using an RBF kernel with Automatic Relevance Determination (ARD) (Neal, 1995) to learn a separate length scales for each feature. We chose to use ARD to give a measure of feature importance, which can be interpreted from the length scales.

Inference in GPs has $O(n^3)$ time complexity and $O(n^2)$ space complexity. As the size of the training data and the dataset is relatively large, we use a Fully Independent Training Conditional (FITC) sparse GPC (Snelson and Ghahramani, 2006) to reduce runtime and RAM usage. In addition to the length scales and the amplitude of Equation (2.36), the sparse GPC is parametrised by the locations of M pseudo-inputs (inducing points), which we learn during training. In our experiments, we use 500 inducing points. GPC inference is not possible analytically and we use Expectation Propagation (Minka, 2001) to approximate the joint posterior as a Gaussian (see Section 2.3).

Inferring the income of a user is defined as a regression task. Given the user feature representations as inputs, we try to predict a real value representing the user's income with the goal of minimising the absolute error between the actual and inferred income. We use variants of the same three

[9]The Gaussian Process models are trained using GPy (`http://github.com/SheffieldML/GPy`). All the other models are trained using Scikit-learn (`http://scikit-learn.org/`).

models, but specialised for regression instead of classification. These are: (1) linear regression (LR), (2) Support Vector machines for Regression (SVR) (Drucker et al., 1997), and (3) Gaussian Process Regression (GPR) (Rasmussen and Williams, 2006). The LR model is a linear regression model, which also uses an Elastic Net regularization term. The SVR model is a nonlinear SVM model modified for regression (see Section 2.2) that also uses an RBF kernel. The last model is a Sparse GPR model, again using an RBF with ARD kernel and 500 inducing points.

6.5.4 Graph Embeddings

To incorporate network information into the language model we use graph embeddings based on unbiased random walks using SkipGram with Negative Sampling (SGNS). This approach is introduced in Section 4.4. In addition to unbiased random walks, biased random walks have also been considered that encourage the walker to either return to the previous vertex or move further away from it (Grover and Leskovec, 2016). We experimented with biased random walks, but chose not to use them as there were no statistically significant benefits and the walks are more expensive to compute as they require a scan over second order network connections. Embeddings were learned from random walks of length 80, with ten walks originating at each labelled vertex. Any vertices occurring less than ten times in the random walk dataset were pruned and vertices with a frequency greater than 1/1000 were downsampled to 1/1000. SGNS was trained using a context window of width ten and five negative samples per positive sample. The same embeddings were used for both tasks (income and occupation).

6.6 Income Experimental Evaluation

Tables 6.11a and 6.11b show the results obtained by the proposed predictive models using the topics baselines (*Topics*), graph embeddings (*Graph*) and their combination (*Graph+Topics*) as feature representations of the Twitter users in the dataset. Note that models using *Topics* as features are the baseline methods. The best performing model using graph based features (*Graph*) achieves an accuracy of 50.44% in the occupational classification task. In income prediction, the MAE is $9,048$ and Pearson's correlation is 0.63. This implies that graph embeddings carry meaningful information about user's socioeconomic attributes.

Table 6.11: A Comparison of the baseline results (Topics) against results using graph embeddings (Graph) and linguistic features combined with graph embeddings (Graph + Topics).

Occupational Class	
Method	**Accuracy (%)**
Majority Class	35.00
Preoţiuc-Pietro et al. (2015)	
LC-Topics	46.57
SVM-Topics	49.47
GPC-Topics	49.64
Ours	
LC-Graph	46.24
SVM-Graph	50.14
GPC-Graph	50.44
LC-Graph+Topics	48.84
SVM-Graph+Topics	52.00†
GPC-Graph+Topics	51.46†

(a) Accuracy of models in predicting user occupational class. † denotes statistical significant different (t-test, $p < 0.01$) method to *GPC-Topics*.

Income		
Method	**MAE (£)**	ρ
Preot et al. (2015)		
LR-Topics	10573	.50
SVR-Topics	9528	.59
GPR-Topics	9883	.60
Ours		
LR-Graph	10811	.50
SVR-Graph	9048‡	.62
GPR-Graph	9532	.63
LR-Graph+Topics	10326	.54
SVR-Graph+Topics	9072‡	.64
GPR-Graph+Topics	9488	.64

(b) Mean Absolute Error and Pearson's correlation coefficient between actual and predicted income. ‡ denotes statistical significant different (t-test, $p < 0.001$) method to *SVR-Topics*. Models were trained to minimise MAE.

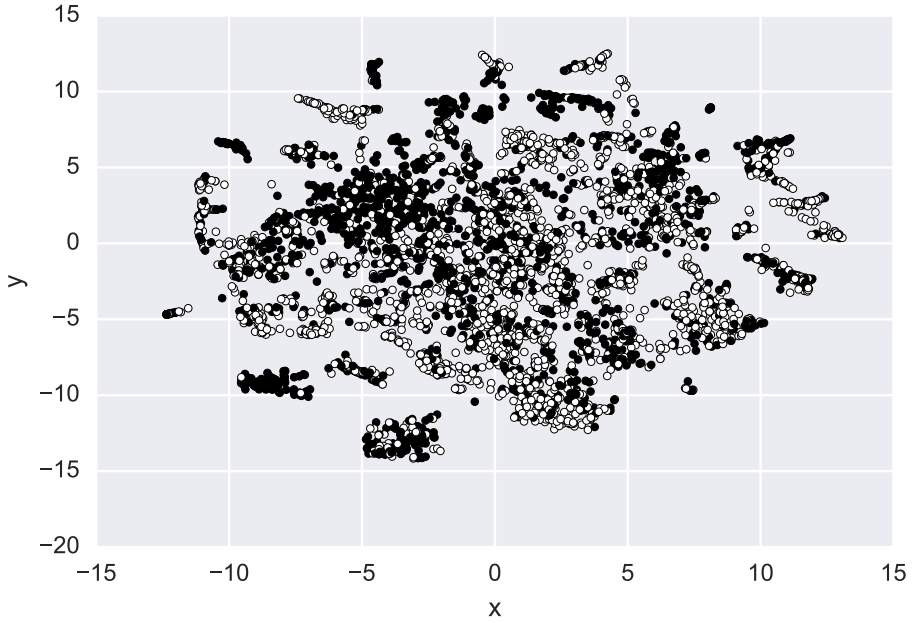


Figure 6.6: A 2-d TSNE plot of the best performing user embedding (32D). Black is users with above median income, white is below.

The graph embedding features perform consistently better than the textual features (*Topics*) for the majority of the predictive models on both tasks except for the LR and LC models. This confirms our first hypothesis that information from the network structure of a user is indicative of socioeconomic attributes. Figure 6.6 shows a 2-d t-SNE (Van Der Maaten and Hinton, 2008) plot of the best performing user embedding. t-SNE stands for t-distributed Stochastic Neighbour Embedding, which is a form of embedding that is well-suited to visualising high dimensional data. It works by constructing two probability distributions that specify the probability that any two points are neighbours, one in the high and one in the low dimensional space. The embedding is then learned by minimising the KL-divergence between the two distributions as a function of the location of the points in the low dimensional space. Figure 6.6 plots accounts with above (black circles) and below (white circles) median predictive incomes. The figure shows that the distributions of low and high income users are markedly different, though still heavily overlapping once projected into two dimensions.

The combination of user graph embeddings and topics (*Graph+Topics*) outperform both feature sets when these are used individually. More specifically, our *GPC-Graph+Topics* model significantly outperforms (t-test, $p < 0.01$) the previous state-of-the-art method, *GPC-Topics* introduced in Preot et al. (2015) on occupational classification. Moreover, our *SVR-Graph+Topics* model significantly outperforms ($p < 0.001$) the best baseline method, i.e. *SVM+Topics*. This confirms that our second hypothesis; network structure and linguistic information are complementary.

Nonlinear models (i.e. SVM, SVR, GPC, GPR) achieve better results in inferring user socioeconomic attributes than linear (LR) models. While in the occupational classification task our best performing model is GPC, the best model in income inference is the SVR instead of GPR. This implies (nonlinear) model selection should not be neglected in these tasks.

An analysis of the errors in the occupational classification task shows that most misclassifications come from adjacent classes. For example, users in classes 1, 3 and 4 are mistakenly classified as class 2. This happens because adjacent classes contain related occupations. However, we notice less dispersion of errors caused by other classes misclassified as class 2 when we use graph embeddings and the combination of graph embeddings and topics. This might be explained by the homophily of users' networks captured by graph embeddings.

In summary, the proposed user graph embeddings can learn more effective user representations for occupation classification and income regression by exploiting the structure of the social network compared to methods that only use linguistic features. This demonstrates that models based on

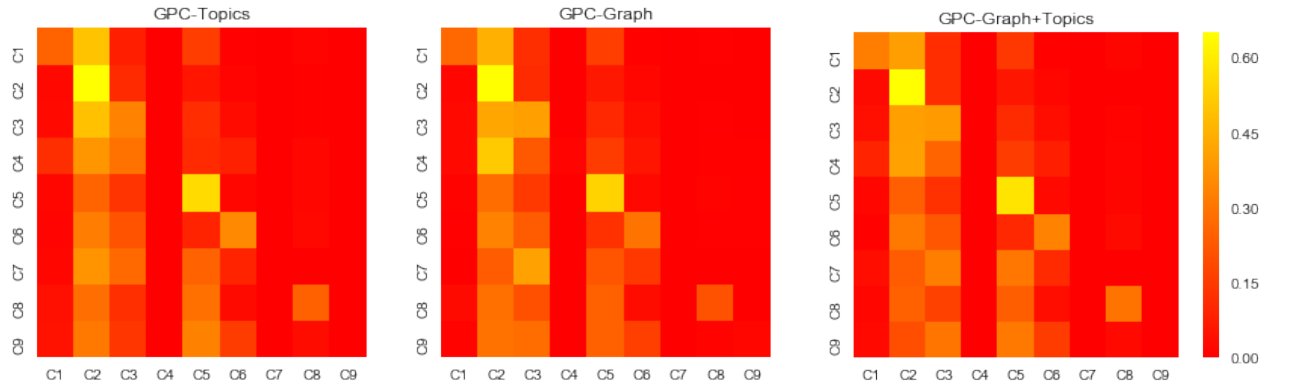


Figure 6.7: Confusion matrices of the classification results (from left to right) of the GPC-Topics (Preoţiuc-Pietro et al. Preoţiuc-Pietro et al. (2015)), GPC-Graph (Our work) and GPC-Topics+Graphs (Our work) respectively. Rows and columns represent the true and inferred occupational class of the users respectively.

network information can be useful for accurately inferring complex socioeconomic characteristics of Twitter consumers, i.e. users that do not post but read content of other users. The combination of network and linguistic features leads to further improvements.

We also investigate how the embedding size affects the performance of the different models in the two tasks. We train graph embeddings with dimensionality $d \in \{16, 32, 64, 128\}$, as described in Section 6.5.4. We then train models for occupational classification (LC, SVM, GPC) and income regression (LR, SVR, GPR) with the learned embeddings.

Figure 6.8 shows the accuracy and MAE obtained. We note that performance in both tasks first increases and then slightly decreases for the nonlinear models. That is because the embedding size is too small to effectively capture user's network information. When dimensionality is larger, then nonlinear models become too complex and may eventually start to overfit to the training data. On the other hand, the performance of the less complex linear models steadily, but slowly increases with larger embeddings.

6.7 Summary

We have presented two successful methods for learning attributes of vertices in graphs when a small proportion of the users are labelled. The first method is a massively scalable hierarchical Bayesian model for predicting user ages under uncertainty. The model exploits generic properties of Twitter users, e.g., what they follow, which is indicative of their interests and, therefore, their age. Our

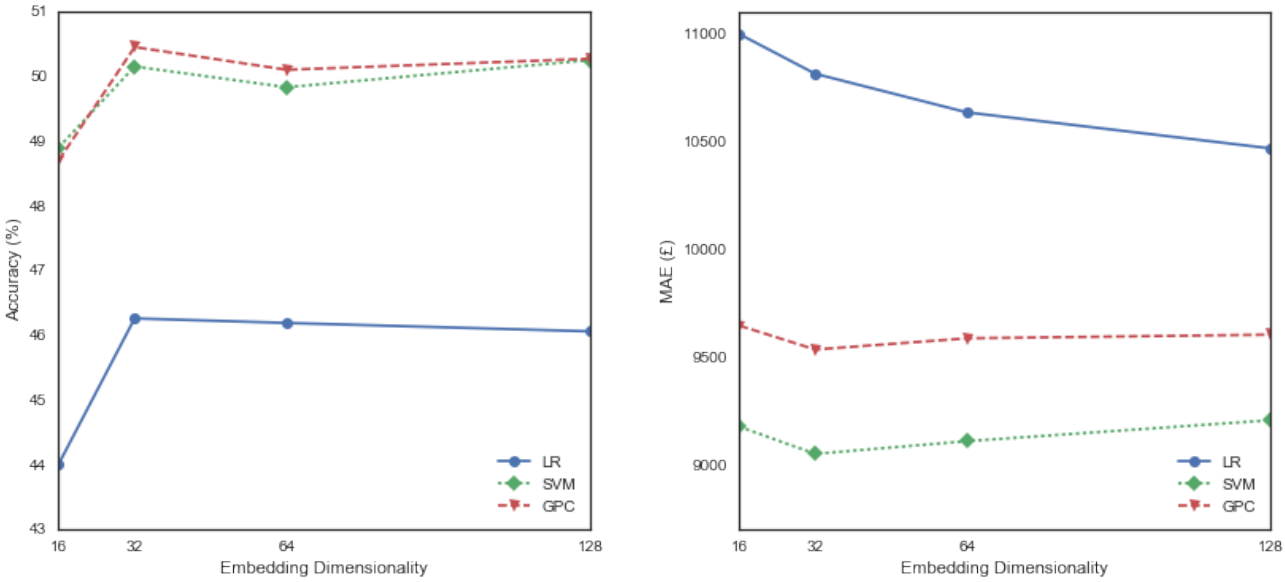


Figure 6.8: The plot on the left shows the effects of the embedding dimensionality on the predictive accuracy of the models using *Graph* features. The plot on the right shows the embedding dimensionality effect on the income regression MAE using *Graph* features to train models.

model performs as well as the current state-of-the-art for inferring the age of Twitter users without being limited to specific linguistic or engineered features. We have successfully applied our model to infer the age of 700 million accounts, demonstrating the scalability of our approach. The method can be applied to any attributes that can be extracted from user profile descriptions. Our model was used in a full-scale industrial system for social media marketing at Starcount Insights and we included extensive details of how the data was produced and processed.

In the second part of the chapter, we showed how the state-of-the-art linguistic model for inferring the income and occupations of Twitter users could be improved upon by incorporating features from the users' social network. This method used neural graph embeddings to efficiently represent users as fixed length vectors based upon their social network. The graph embeddings outperformed the extensively engineered linguistic features and performance was even better when linguistic and graph features were combined. Both sets of experiments require that homophily is measurable in a user's Twitter network. Our results provide clear evidence that this is the case. Specifically Twitter users of similar ages, or with similar incomes and occupations are more likely to be connected to each other.

This chapter has demonstrated the power of learning user attributes from the structure of social networks. Using the social network has many advantages over linguistic features for this task. As many social media users do not post textual content, but still interact in the social graph, graph-

based techniques have much broader applicability. Even when considering only users who produce large quantities of textual content, we showed that graphical information outperforms linguistic features for income and occupation prediction. We hypothesise that the graphical information is *more truthful* than textual content. While both forms of data are publicly available in theory, textual content is far more visible and so it is more likely to project how a user would like to be perceived rather than their true state. This is consistent with the observation that language is adapted based on social constructs (Nguyen et al., 2013). Features derived from the graph are also highly versatile. Both the age and income / occupation models apply, without adaptation, to multiple languages and regions. Graph embeddings are learned completely unsupervised from the graph structure and so can be applied directly to multiple tasks. We demonstrate this by using the same set of embeddings for both the income and occupation prediction tasks.

There are several limitations of our approaches that we would like to address in future work. Firstly, our age detection model makes the assumption that the act of Following accounts is independent given a user's age. This is clearly incorrect as in many cases users Follow highly correlated groups of accounts such as every player from the football team that they support. Failing to incorporate feature dependencies produces predictions that are overconfident and possibly incorrect. The independence assumption was made for engineering and commercial reasons as it was a requirement to run the model every day for every Twitter user at a given cost. It would be interesting to extend the Bayesian model to multiple hierarchical levels that incorporate the community structures present in the graph that were explored in Chapter 5.

Secondly, both datasets use a single snapshot of the graph and do not incorporate temporal information. The study of dynamic graphs has recently attracted the interest of many researchers in the complex networks community Cvetkovski and Crovella (2009); Kim and Leskovec (2013); Zhang et al. (2016) and this appears to be a fruitful direction for future work. It is reasonable to assume that humans are dynamic enough that actions taken, in some cases, many years previously are less indicative of their current state. However, there is no way to account for this in our model. One simple framework to incorporate temporal information would be to generate a weighted graph with edge weights that exponentially decay with time. Random walks over the weighted graph could then be biased to choose more recent edges. A neural graph embedding approach using time-sensitive random walks would generate time-sensitive graph embeddings.

Finally we note that there is a hint of inefficiency about the process of representing vertices of graphs

using neural graph embeddings. Graph embeddings first convert a graph to a set of vertex sequences. Then vectors representing every vertex are randomly initialised and the relative positions of these vectors are re-learned from the sequences. However, there is a great deal of information that is present in the original graph, which could be used to do better than simply randomly initialising vectors in the neural network embeddings layer.

Chapter 7

Customer Lifetime Value Prediction with Embeddings

This chapter describes the Customer LifeTime Value (CLTV) system deployed at ASOS.com in 2017. In parallel with Chapter 6, we first develop a conventional machine learning model and then show how improvements can be made by incorporating graph embeddings. In this case, the embeddings are learned by taking walks over the bipartite customer-product graph that links every ASOS customer to every product they interact with. The key innovation is a method to use embeddings for long-term forecasting that is published in Chamberlain et al. (2017a).

The CLTV system is run every day at ASOS.com and the predictions are used to optimize external marketing, internal customer management processes and to trigger alerts for customers with large changes in CLTV. Machine learning in commercial settings is fundamentally a tool for profit optimization, which introduces additional constraints that are not always present in academic research. Specifically, the cost of training the model and variability/stability of the predictions. It is an increasing trend for commercial ML systems to be run on third party cloud infrastructure (e.g. Amazon EC2, Google Cloud Platform or Microsoft Azure) that is leased per hour. This is the model used at ASOS.com, where for a fixed hardware, the training cost is proportional to runtime. To manage the constraints of cost and prediction stability in a principled way, we develop a generalised loss function for the CLTV model and this is published in Liu et al. (2017b).

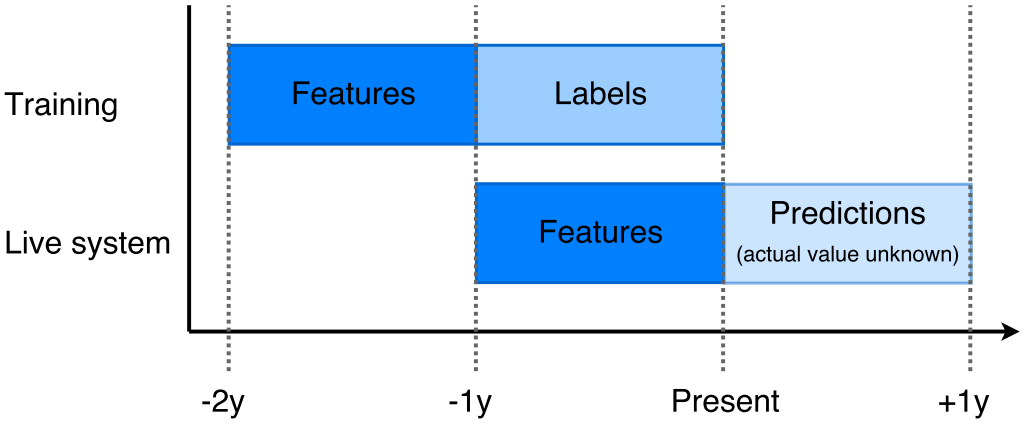


Figure 7.1: Training and prediction timelines for CLTV. The model is retrained every day using customer data from the past two years. Labels are the net customer spend over the previous year. Model parameters are learned in the training period and used to predict CLTV from new features in the live system.

7.1 Introduction

The system addresses two tightly coupled problems: CLTV and churn prediction. We define a customer as churned if they have not placed an order in the last twelve months. We define CLTV as the sales, net of returns, of a customer over a twelve month period. The objective of the system is to improve three key business metrics: (1) the average customer shopping frequency, (2) the average order value, (3) the customer churn rate. The model supports the first two objectives by allowing ASOS to rapidly identify and nurture high-value customers who will go on to have high frequency, high-order value, or both. The third objective is achieved by identifying customers at high risk of churn and modifying the budget for personalised retention activities.

State-of-the-art CLTV systems use large numbers of handcrafted features and ensemble classifiers (typically random forest regressors), which have been shown to perform well in highly stochastic problems of this kind (Vanderveld and Han, 2016; Hardie et al., 2006). However, handcrafted features introduce a human bottleneck, can be difficult to maintain and often fail to utilise the full richness of the data. We show how automatically learned features can be combined with handcrafted features to produce a model that can incorporate domain knowledge and learn rich patterns of customer behaviour from raw data.

The deployed ASOS CLTV system uses the state-of-the-art architecture, which is a Random Forest (RF) regressor with 132 handcrafted features. An introduction to random forests is given in Section 2.4. To train the forest, labels and features are taken from disjoint periods of time as shown

in the top row of Figure 7.1. The labels are the net spend (sales minus returns) of each customer over the past year. The training labels and features are used to learn the parameters of the RF. The second row of Figure 7.1 shows the live system where the RF parameters from the training period are applied to new features generated from the last year's data to predict next year's net customer spend.

Unlike many other ML methods (SVMs, linear regression, decision trees etc.), RF predictions are not deterministic because each tree is trained on a random sample of the training data. We call the variability in predictions due solely to the training procedure the *endogenous variability* and the variability due to changes in the external environment *exogenous variability*. Ideally we would like to measure only exogenous change, which is challenging if the endogenous effects are on a similar or larger scale.

Random forests have several hyperparameters (see Section 2.4.2) that must be carefully selected (Huang and Boutros, 2016). This is usually achieved by running an optimization procedure that searches for hyperparameters that minimise a measure of prediction error. RF hyperparameters (and other machine learning methods) are usually optimized exclusively to minimise error metrics. However, for commercial applications this is not adequate as we must also consider the monetary cost of model training and the stability of the predictions.

It is not possible to find hyperparameter configurations that simultaneously optimize cost, stability and error. For example, increasing the number of trees in a random forest will improve the stability of predictions, reduce the held-out error rate, but increase the cost (due to longer runtimes). We propose a principled approach to this problem using a multi-criteria objective function and Bayesian Optimization (BO). An introduction to BO is given in Section 2.5.

In the remainder of this chapter we provide a detailed explanation of the practical challenges and lessons learned in deploying the ASOS CLTV system and our efforts to improve it using learned features. We experiment with learning representations directly from data using two different approaches: (1) by training a feedforward neural network on the handcrafted features in a supervised setting (see Section 7.4.2), (2) by augmenting the RF feature set with unsupervised customer embeddings learned from the customer product views graph (see Section 7.4.1). The novel customer embeddings are shown to improve CLTV prediction performance significantly compared with our benchmark. Incorporating embeddings into long-term prediction models is challenging because, unlike handcrafted features, they are not easily identifiable. As the embeddings enter the training

objective only through the dot product (See Equation (4.15) and Equation (4.26)) any arbitrary rotation of the axes is possible even for separate training runs on the same dataset. When different datasets are used, as is the case for disjoint training and test periods, the embeddings may be completely unrelated. Figure 7.2 illustrates this point for the special case where the axes are permuted using a four dimensional embedding. Each column in the figure corresponds to a different dimension of the embedding, which is used as a feature for the RF model. In the training period we learn parameters $\{x_1, x_2, x_3, x_4\}$ for each feature. However, as the features are not labelled and their order randomly permutes from training to test time, it is not possible to map the training parameters to the test features.

Andriy's comment that this is much more general has not been addressed

The right hand side of Figure 7.2 shows that the embedding features are associated with the incorrect training parameters because they are randomly permuted with no way of tracking the paths depicted by the arrows in the left panel of the figure. We describe how this problem can be solved within the neural embedding framework using a form of warm start for the test period embeddings in Section 7.4.1.

The main contributions of this chapter are:

1. A detailed description of the large-scale CLTV system deployed at ASOS.com, including a discussion of the architecture, deployment challenges and lessons learned.

2. The development of a framework for jointly optimizing RF hyperparameters over prediction errors, prediction variability and training cost.

3. Analysis of two candidate architectures for hybrid systems that incorporate both handcrafted and learned features

4. Introduction of customer level embeddings and demonstration that these produce a significantly better performance than a benchmark classifier

5. Introduction of neural embeddings for long-term forecasting tasks

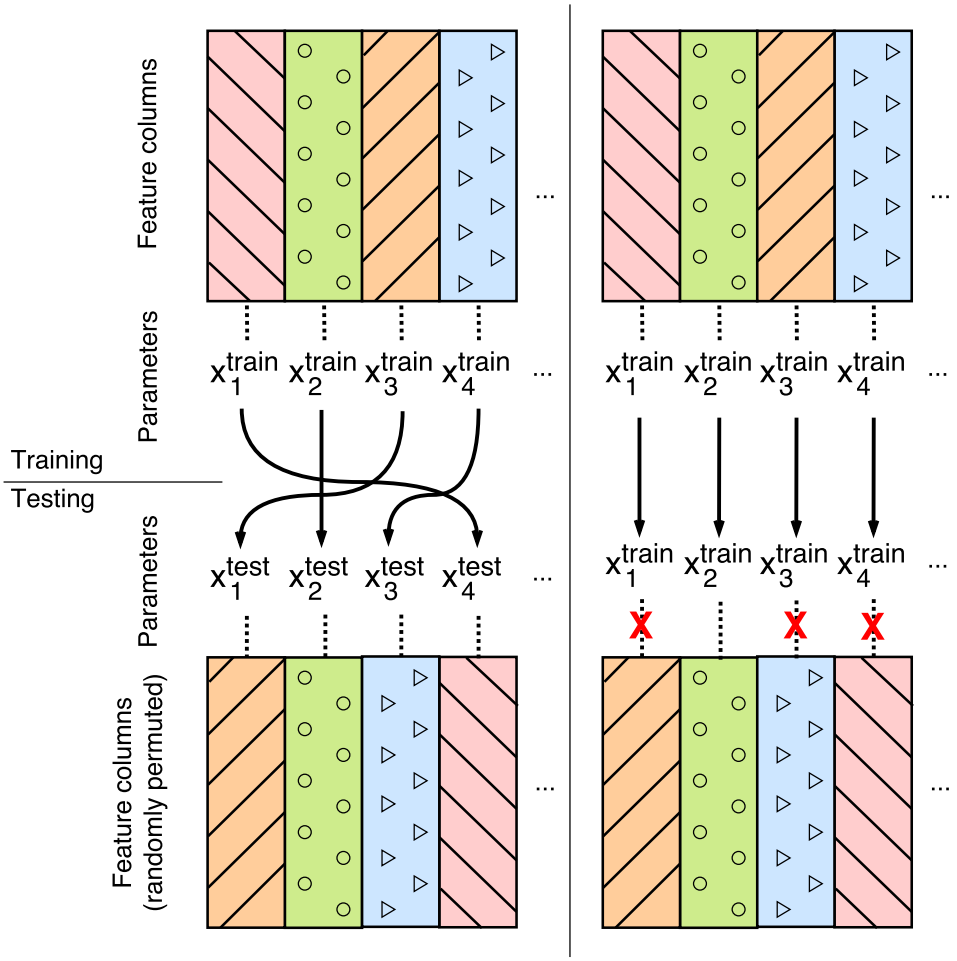


Figure 7.2: Illustration of the challenges of using the components of embedded customer vectors as features for forecasting. Each column represents a component of the vector representation of customers. We have labelled the columns with the parameters that are learned at training time. (Left) The vector components can randomly permute between train and testing time and hence require different learned parameters. (Right) Applying the learned parameters from training time directly to the embeddings in test time will not work as they are no longer attached to the correct component of the embedded vectors.

7.2 Related Work

Statistical models of customer purchasing behaviour have been studied for decades. Early models were hampered by a lack of data and were often restricted to fitting simple parametric statistical models, such as the Negative Binomial Distribution in the NBD model (Goodhart et al., 1984). It was only at the turn of the century, with the advent of large-scale e-commerce platforms that this problem began to attract the attention of machine learning researchers.

7.2.1 Distribution Fitting Approaches

The first statistical models of CLTV were known as Buy 'Til You Die (BTYD) models. BTYD models place separate parametric distributions on the customer lifetime and the purchase frequency and only require customer recency and frequency as inputs. One of the first was the Pareto/NBD model of Schmittlein et al. (1987), which assumes an exponentially distributed active duration and a Poisson distributed purchase frequency for each customer. This yields a Pareto-II customer lifetime and negative-binomial (NBD) distributed purchase frequency. Fader et al. (2005a) replaced the Pareto-II with a Beta-geometric distribution to reduce implementation challenges. Using the Beta-geometric assumes that each customer has a certain probability to become inactive after every purchase. Bemmaor and Glady (2012) proposed the Gamma/Gompertz distribution to model the customer active duration, which is more flexible than the Pareto-II as it can have a non-zero mode and be skewed in both directions.

Recency-Frequency-Monetary Value (RFM) models expand on BTYD by including the additional monetary value feature. Fader et al. (2005b) linked the RFM model, which captures the time of last purchase (recency), number of purchases (frequency) and purchase values (monetary value) of each customer to CLTV estimation. Their model assumes a Pareto/NBD model for recency and frequency with purchase values following an independent gamma/gamma distribution. In the gamma/gamma distribution each value is gamma distributed around the associated customer's mean purchase value, which follows another gamma distribution over all customers. The predicted CLTV is then calculated by multiplying the expected number of purchases from the Pareto/NBD model by the expected purchase value from the gamma/gamma model. The hyperparameters are estimated using maximum likelihood.

While successful for the problems that they were applied to, it is difficult to incorporate the vast ma-

jority of the customer data available to modern e-commerce companies into the RFM/BYTD framework. It is particularly challenging to incorporate automatically learned or highly sparse features. This motivates the need for machine learning approaches to solving this problem.

7.2.2 Machine Learning Methods

The most related work to ours is that of Vanderveld and Han (2016), which is the first work to explicitly include customer engagement features in a CLTV prediction model. They address the challenges of learning the complex, nonlinear CLTV distribution by solving several simpler subproblems. Firstly, a binary classifier is run to identify customers with predicted CLTV greater than zero. Secondly, the customers predicted to shop again are split into five groups based approximately on historical transaction frequency and independent regressors are trained for each group. To the best of our knowledge, deep neural networks have not yet been successfully applied to the CLTV problem. However, there is work on the related churn problem (CLTV equal to zero) by Wangperawong et al. (2016) in the telecommunications domain. In their work, the authors create a two-dimensional array for each customer where the columns are days of the year and the rows describe different kinds of communication (text, call etc.). They used this array to train deep convolutional neural networks. The model also used auto-encoders (Vincent et al., 2008) to learn low-dimensional representations of the customer arrays.

As we are interested in the changes between daily prediction values of our ML system, the predictive stability is an important property. Stability in machine learning has been studied for many years. The notion of instability for bagging models (like random forests) was originally developed by Breiman (1996b,a), and extended explicitly by Elisseeff et al. (2005) to randomised learning algorithms, albeit focusing on generalisation/leave-one-out error (as is common in computational learning theory) rather than the instability of the predictions themselves.

7.3 Customer Lifetime Value Model

The ASOS CLTV model uses a rich set of features to predict the net spend of customers over the next 12 months by training a random forest regressor on historic data. One of the major challenges of predicting CLTV is the unusual distribution of the target variable. A large percentage of customers

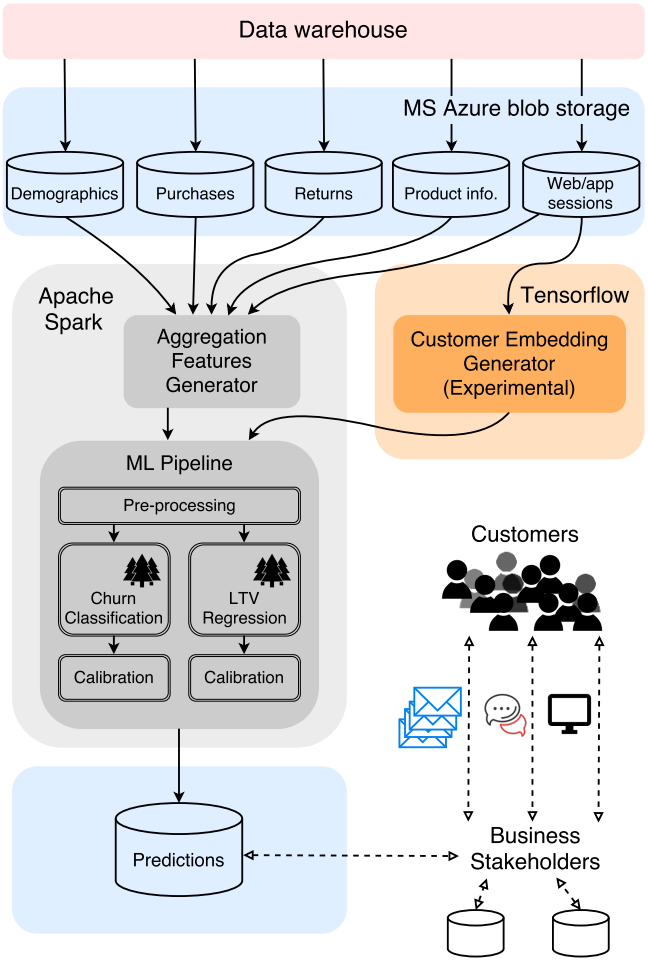


Figure 7.3: High-level overview of the CLTV system. The solid arrows represent the flow of data, and the dashed arrows represents interaction between stakeholders and systems/data. Customer data is collected and pre-processed by our data warehouse and stored on Microsoft Azure Blob storage. The processed data is used to generate handcrafted features in Spark clusters, with web/app sessions additionally used to produce experimental customer embeddings in Tensorflow. The handcrafted features and customer embeddings are then fed through the machine learning pipeline on Spark, which trains calibrated random forests for churn classification and CLTV regression. The resulting prediction are piped to operational systems.

Table 7.1: Feature importances in the baseline RF model by data class.

Data class	Overall Importance
Customer demographics	0.078
Purchases history	0.600
Returns history	0.017
Web/app session logs	0.345

have a CLTV of zero. Of the customers with greater than zero CLTV, the values differ by several orders of magnitude. To manage this problem we explicitly model CLTV percentiles using a random forest regressor. Having predicted percentiles, the outputs are then mapped back to real value ranges for use in downstream tasks.

The remainder of this section describes the features used by the model, the architecture that allows the model to scale to over 15 million customers, and our training and evaluation methodology.

7.3.1 Features

The model incorporates features from the full spectrum of customer information available at ASOS. There are four broad classes of data: (1) customer demographics, (2) purchase history (3) returns history (4) web and app session logs. By far the largest and richest of these classes are the session logs.

We apply random forest feature importance (Friedman et al., 2001a) to rank the 132 handcrafted features. Table 7.1 shows the feature importance breakdown by the broad classes of data and Table 7.2 shows the top individual features. As expected, the number of orders, the number of sessions in the last quarter and the nationality of the customer were important features for CLTV prediction. However, we were surprised by the importance of the standard deviation of the order and session dates, particularly because the maximum spans of these variables were also features. We also did not expect the number of items purchased from the new collection to be one of the most relevant features. This is because newness is a major consideration for high value fashion customers.

The high-level system architecture is shown in Figure 7.3. Raw customer data (see Section 7.3.1) is pre-processed in our data warehouse and stored in Microsoft Azure Blob storage[1]. From Blob storage, data flows through two work streams to generate customer features: A handcrafted feature generator

[1]a Microsoft cloud solution that provides storage compatible with distributed processing using Apache Spark

Table 7.2: Individual feature importances of the top 22 features.

Feature Name	Importance (%)
Number of orders	21
Standard deviation of the order dates	12
Number of session in the last quarter	11
Country	6.4
Number of items from new collection	5.5
Number of items kept	4.9
Net sales	3.9
Days between first and last session	3.9
Number of sessions	3.5
Customer tenure	3.3
Total number of items ordered	2.5
Days since last order	2.1
Days since last session	1.9
Standard deviation of the session dates	1.8
Orders in last quarter	1.6
Age	1.4
Average date of order	0.9
Total ordered value	0.8
Number of products viewed	0.7
Days since first order in last year	0.6
Average session date	0.6
Number of sessions in previous quarter	0.5

on Apache Spark and an experimental customer embedding generator on GPU machines running Tensorflow (Abadi et al., 2015), which uses only the web/app sessions as input. The model is trained in two stages using an Apache Spark ML pipeline. The first stage pre-processes the features and trains random forests for churn classification and CLTV regression on percentiles. The second stage performs calibration and maps percentiles to real values. Finally the predictions are presented to a range of business systems that trigger personalised engagement strategies with ASOS customers.

7.3.2 Training and Evaluation Process

Our live system uses a calibrated RF with features from the past twelve months that is re-trained every day. We use twelve months because there are strong seasonality effects in retail and failing to use an exact number of years would cause daily fluctuations in features that are calculated by aggregating data over the training period.

To train the model we use historic net sales over the last year as a proxy for CLTV labels and learn the random forest parameters using features generated from a disjoint period prior to the label period. This is illustrated in Figure 7.1. Every day we generate:

1. A set of aggregate features and product view-based embeddings for each customer based on their demographics, purchases, returns and web/app sessions between two years and one year ago.

2. Corresponding target labels, including the churn status and net one-year spend, for each customer based on data from the last year.

The feature and label periods are disjoint to prevent information leakage. As the predictive accuracy of our live system could only be evaluated in a year's time, we establish our expectation on the performance of the model by forecasting for points in the past for which we already know the actual values as illustrated in Figure 7.1. We use the Area Under the receiver operating characteristic Curve (AUC) as a performance measure.

7.3.3 Prediction Stability

We are not solely interested in the absolute predictive values because some actions are triggered by changes in predictions. However, we can not simply choose the most stable predictive model as this negatively impacts predictive accuracy and training cost (e.g. a model predicting only a constant value is maximally stable). Here we formalise the notion of RF stability in terms of repeated model runs using the same parameter settings and dataset (ie. all variability is endogenous). We do this for the binary churn prediction problem as it is slightly simpler, but the concept can easily be extended to multiclass classification and regression problems. The expected squared difference between the predictions over two runs is given by

$$\frac{1}{N}\sum_{i=1}^{N}\left[\left(\hat{y}_i^{(j)}-\hat{y}_i^{(k)}\right)^2\right], \tag{7.1}$$

where $\hat{y}_i^{(j)} \in [0,1]$ is the probability from the j^{th} run that the i^{th} data point is of the positive class. We average over $R \gg 1$ runs and define the Mean Squared Prediction Delta (MSPD) as

$$\text{MSPD}(f) \triangleq \frac{2}{R(R-1)} \sum_{j=1}^{R} \sum_{k=1}^{j-1} \left[\frac{1}{N} \sum_{i=1}^{N} \left[\left(\hat{y}_i^{(j)} - \hat{y}_i^{(k)} \right)^2 \right] \right]. \tag{7.2}$$

Rearranging the summations and injecting two expectations of a single datum over model runs $\mathbb{E}_M(\hat{y}_i)$ of opposite signs gives

$$\text{MSPD}(f) = \frac{2}{N} \sum_{i=1}^{N} \frac{1}{R(R-1)} \sum_{j=1}^{R} \sum_{k=1}^{j-1} \left[\left(\hat{y}_i^{(j)} - \mathbb{E}_M(\hat{y}_i) \right) - \left(\hat{y}_i^{(k)} - \mathbb{E}_M(\hat{y}_i) \right) \right]^2. \tag{7.3}$$

By symmetry, the squared terms under the double summation each occur $R-1$ times and so splitting the summation into squared and cross terms leads to

$$\begin{aligned} \text{MSPD}(f) = \frac{2}{N} \sum_{i=1}^{N} \Bigg[& \frac{1}{R(R-1)} \sum_{l=1}^{R} \left[R \left(\hat{y}_i^{(l)} - \mathbb{E}_M(\hat{y}_i) \right)^2 - \left(\hat{y}_i^{(l)} - \mathbb{E}_M(\hat{y}_i) \right)^2 \right] - \\ & \frac{2}{R(R-1)} \sum_{j=1}^{R} \sum_{k=1}^{j-1} \left(\hat{y}_i^{(j)} - \mathbb{E}_M(\hat{y}_i) \right) \left(\hat{y}_i^{(k)} - \mathbb{E}_M(\hat{y}_i) \right) \Bigg]. \end{aligned} \tag{7.4}$$

By considering the grid of pairwise interactions between run j and run k the $\left(\hat{y}_i^{(l)} - \mathbb{E}_M(\hat{y}_i) \right)^2$ term in the inner sum is the main diagonal and the $\left(\hat{y}_i^{(j)} - \mathbb{E}_M(\hat{y}_i) \right) \left(\hat{y}_i^{(k)} - \mathbb{E}_M(\hat{y}_i) \right)$ term contains every off-diagonal element and so they can be combined into a single double sum. The resulting MSPD is

$$\begin{aligned} \text{MSPD}(f) = \frac{2}{N} \sum_{i=1}^{N} \Bigg[& \frac{1}{R-1} \sum_{l=1}^{R} \left(\hat{y}_i^{(l)} - \mathbb{E}_M(\hat{y}_i) \right)^2 \\ & - \frac{1}{R(R-1)} \sum_{j=1}^{R} \sum_{k=1}^{R} \left(\hat{y}_i^{(j)} - \mathbb{E}_M(\hat{y}_i) \right) \left(\hat{y}_i^{(k)} - \mathbb{E}_M(\hat{y}_i) \right) \Bigg]. \end{aligned} \tag{7.5}$$

It is convenient to measure stability on the same scale as the forest predictions and so we use

$$\text{RMSPD}(f) = \sqrt{\text{MSPD}(f)}. \tag{7.6}$$

By considering Equation (7.5) it can be seen that the model instability is closely related to the RF parameter settings. The second term in this equation is a form of covariance of the predictions over multiple training runs. Increasing the training data sample increases the probability that the same input datum will be selected for multiple training runs and thus the covariance of the predictions increases. An increase in covariance leads to a reduction in the RMSPD.

7.3.4 Hyperparameter Optimization

The performance of RF models are strongly dependent on the choice of hyperparameters (see Section 2.4.2). Often hyperparameter settings are chosen that minimise predictive errors. However, in our setting we must find a balance between predictive errors, cost and prediction stability. Our solution to constrained RF hyperparameter optimization is to propose a loss function that is a linear combination of three optimization criteria. The components of our novel RF loss function are: (1) Stability, defined as the RMSPD given in Section 7.3.3. (2) Error, using the Area Under the receiver operating characteristics Curve (AUC). (3) Cost, defined as the training runtime (in seconds). For a fixed hardware specification, runtime is proportional to cost because ASOS.com, in common with many other commercial organisations, lease their ML hardware on a per hour basis.

Loss-function We choose a loss function with free parameters that allows the relative importance of these three considerations to be balanced:

$$L = \beta\,\text{RMSPD} + \gamma\,\text{Runtime} - \alpha\,\text{AUC}, \tag{7.7}$$

where α, β, γ are weight parameters. The weight parameters are specified according to business needs. We recognise the diverse needs across different organisations and thus refrain from specifying what constitutes a "good" weight parameter set. Nonetheless, a way to obtain the weight parameters is to quantify the gain in AUC, the loss in RMSPD, and the time saved in monetary units. For example, if calculations reveal 1% gain in AUC equates to £50 potential business profit, 1% loss in RMSPD equates to £10 loss in business revenue, and a second of computation costs £0.01, then α, β and γ can be set as 5,000, 1,000 and 0.01 respectively.

7.3.5 Parameter Sensitivity

We evaluate the parameter sensitivity of our optimization framework with respect to three of the RF parameters described in Section 2.4.2: The number of trees, the depth of each tree and the fraction of training data available to each tree. This restricts the loss function to the form

$$L = \beta\,\text{RMSPD}(N_t, d, p) + \gamma\,\text{Runtime}(N_t, d, p) - \alpha\,\text{AUC}(N_t, d, p), \tag{7.8}$$

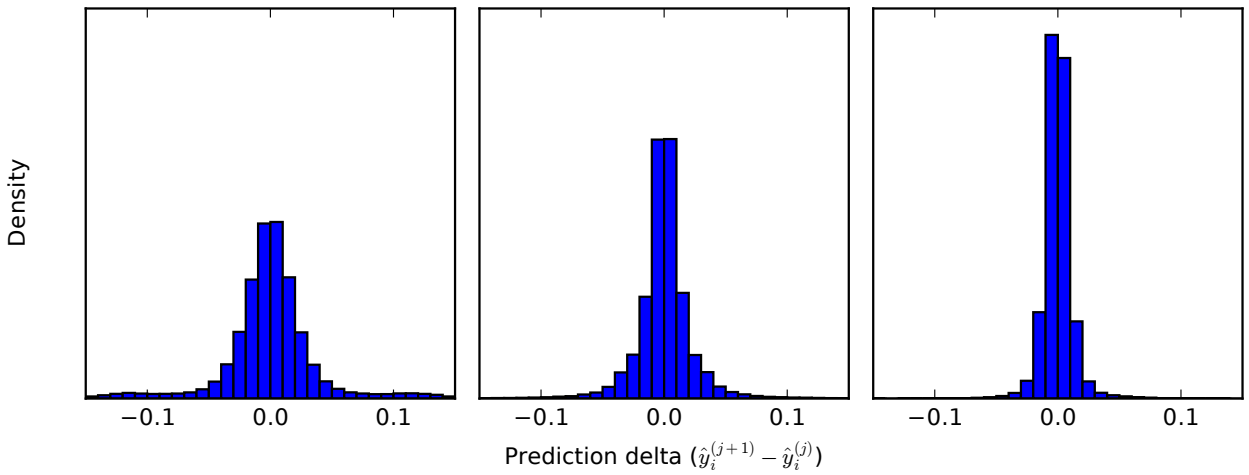


Figure 7.4: The distribution of prediction deltas (difference between two predictions on the same validation datum) for successive RF runs with (from left to right) 8, 32, and 128 trees, repeated ten times. The RMSPD for these three random forests are 0.046, 0.025, and 0.012 respectively. The dataset is split into two halves: the first 25k rows are used for training the random forests, and the latter 25k rows for making predictions. Each run re-trains on all 25k training data, with trees limited to a maximum depth of ten.

where N_t is the number of trees in the trained random forest, d is the maximum depth of the trees, and p is the proportion of data points used in training. For reproducibility, we do this using the public Orange small dataset from the 2009 KDD Cup. The dataset contains 190 numerical and 40 categorical features and binary target labels. Further details are available in Liu et al. (2017).

The generalised prediction performance of a RF increases monotonically with the number of trees, while the runtime grows linearly and the stability of the predictions also improves. The maximum tree depth controls the complexity of each decision tree and the computational cost (runtime) increases exponentially with tree depth. The optimal depth for error reduction depends on the other RF hyperparamaters and the data. Too much depth causes overfitting. Additionally, as the depth increases, the prediction stability will decrease as each model tends towards memorizing the training data. The highest stability will be attained using shallow trees. However, if the forest is too shallow the model will underfit resulting in a high error rate. A reduction in the amount of training data sampled for each tree leads to shorter training times, reducing costs. However, reducing the amount of training data also reduces the generalisability of the model as the estimator sees less training examples, increasing predictive errors. Decreasing the training sample size also decreases the stability of the prediction.

Figure 7.4 visualises the change in the RMSPD with relation to the number of trees in the RF. The

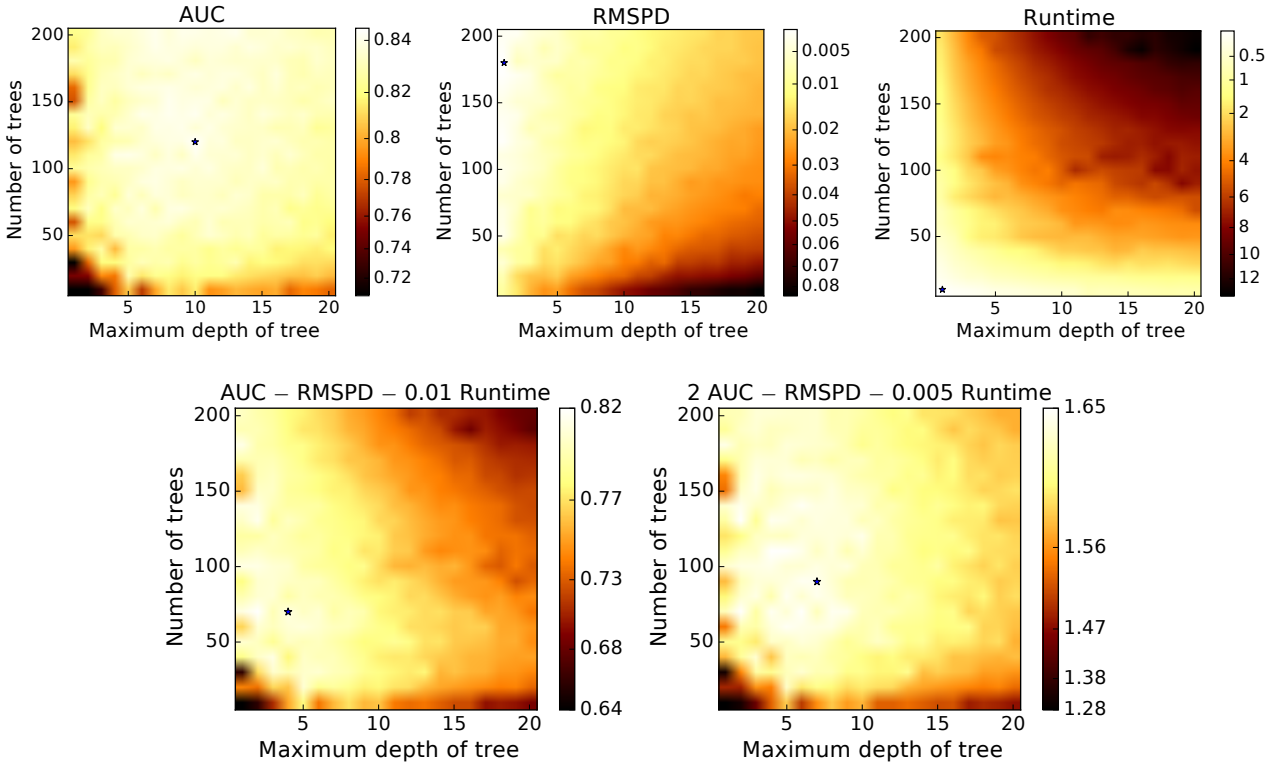


Figure 7.5: The average AUC (top left), RMSPD (top middle), and runtime (top right) attained by random forests with different number of trees and maximum tree depth (training proportion is fixed at 0.5) over five train/test runs. The bottom two plots show the value attained in the specified objective functions by the random forests above. Lighter spots show more preferable parametrizations. The shading is scaled between the minimum and maximum values in each chart. The optimal configuration found under each metric is indicated by a blue star.

plots show distributions of prediction deltas. Increasing the number of trees (going from the left to the right plot) leads to a more concentrated prediction delta distribution, a quality also reflected by a reduction in the RMSPD.

Figure 7.5 shows the AUC, runtime, RMSPD and loss functions averaged over multiple runs of the forest for different settings of number of trees and maximum tree depth. It shows that the AUC plateaus for a wide range of combinations of number of trees and maximum depth. The RMSPD is optimal for large numbers of shallow trees while runtime is optimized by few shallow trees. When we form a linear combination of the three metrics, the optimal solutions are markedly different from those discovered by optimizing any single metric in isolation. We show this for $\alpha = 1, \beta = 1, \gamma = 0.01$ and $\alpha = 2, \beta = 1, \gamma = 0.005$.

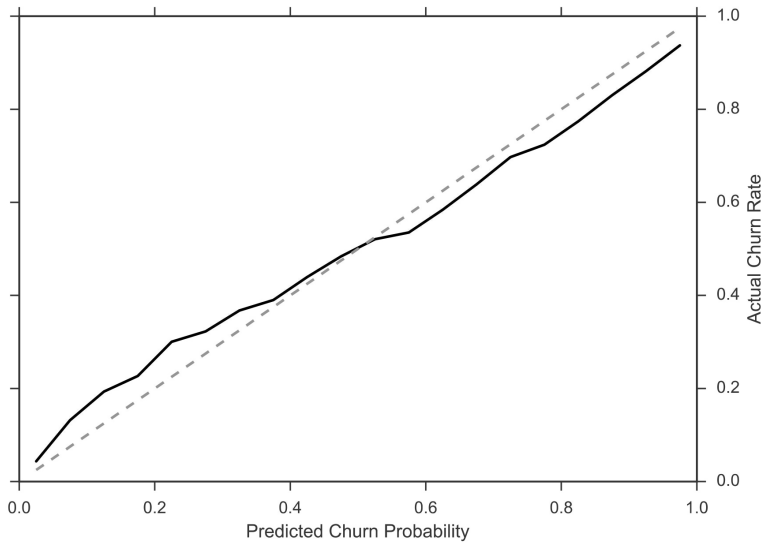


Figure 7.6: Churn prediction density (horizontal axis) and match between predicted probabilities and actual probabilities (black line) versus the optimal calibration (dashed grey line). The predicted probabilities match closely with the actual probabilities. Generated by bucketing predictions into buckets of size 0.01 and evaluating the realised churn rate for each bucket.

7.3.6 Calibration

In this context, calibration refers to our efforts to ensure that the statistics of the model predictions are consistent with the statistics of the data. Model predictions are derived from RF leaf distributions and we perform calibration for both churn and CLTV prediction.

For customer churn prediction, choosing RF classifier parameters that maximise the AUC does not guarantee that the predictive probabilities will be consistent with the realised churn rate (Zadrozny and Elkan, 2001). To generate consistent probabilities, we calibrate by learning a mapping between the estimates and the realised probabilities. This is done by training a logistic regression classifier to predict churn, based only on the probabilities returned by the random forest. The logistic regression output is interpreted as a calibrated probability.

Similarly, to estimate CLTV we have no guarantees that the regression estimates achieved by minimising the Root Mean Squared Error (RMSE) RF loss function will match the realised CLTV distribution. To address this problem, analogously to churn probability calibration, we first forecast the CLTV percentile and then map the predicted percentiles into monetary values. In this case, the mapping is learned using a decision tree. We observe two advantages in performing calibration: (1) the model becomes more robust to the existence of outliers and (2) we obtain predictions, which when aggregated over a set of customers, match the true values more accurately.

7.3.7 Baseline Results

To find the optimal meta-parameters for the RFs we use 10-fold cross validation on a sample of the data. For the churn predictions (see Figure 7.6) we obtain an AUC of 0.798 and calibrated probabilities. Figure 7.6 is generated by bucketing predictions into buckets of size 0.01 and evaluating the realised churn rate for each bucket. For CLTV predictions (see Figure 7.7a) we obtain Spearman rank-order correlation coefficients of 0.56 (for all customers) and 0.46 (excluding customers with a CLTV of 0). We can see in Figure 7.7a that the range and density of the predicted CLTV matches the actual CLTV (excluding those with CLTV of 0). The residuals (prediction - actual) are small and slightly skewed to positive values, peaking at around 0.2 mean CLTVs. This is caused by the difficulty in making CLTV predictions of exactly zero for all churned customers.

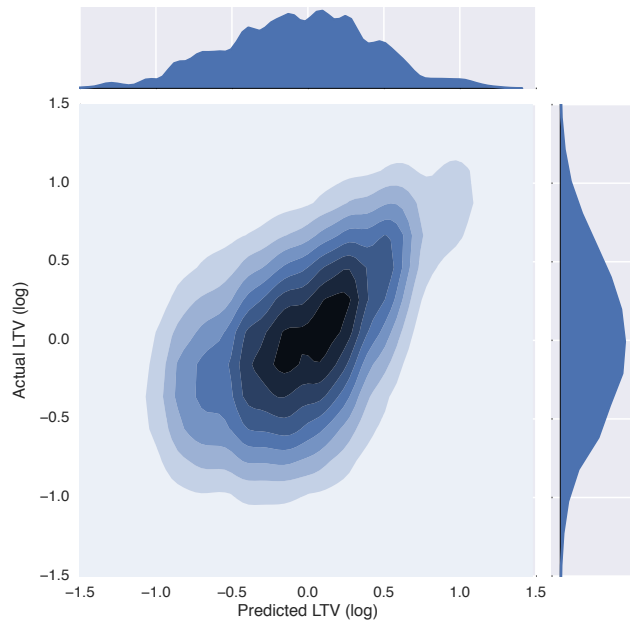
7.4 Improving the CLTV model with Feature Learning

In the remainder of this chapter we describe our ongoing efforts to supplement the handcrafted features in our deployed system with automatic feature learning. Automatic features are directly learned from data to maximise the objective function of a classification or regression task. This technique is frequently used within the realms of deep learning (Jia et al., 2014) and dimensionality reduction (Roweis and Saul, 2000) to overcome some of the limitations of engineered features. Despite being more difficult to interpret, learned features avoid the resource intensive task of constructing features manually from raw data and have been shown to outperform the best handcrafted features in the domains of speech, vision and natural language.

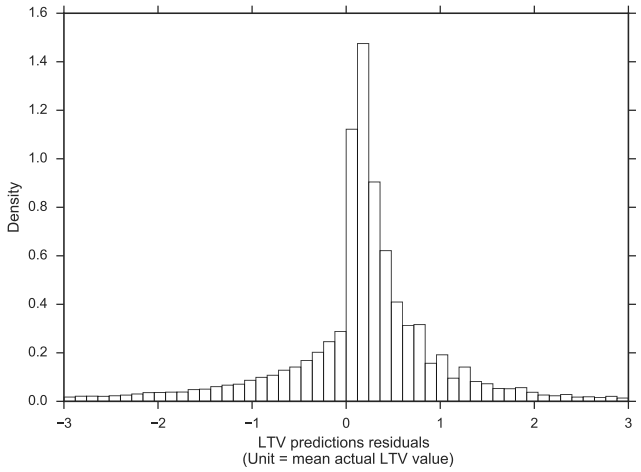
We experiment with two distinct approaches. Firstly, we learn unsupervised neural embeddings from customer-product views graph. Once learned, the embeddings are added to the feature set of the RF model. Secondly, we train a hybrid model that combines logistic regression with a Deep feed-forward Neural Network (DNN). The DNN uses the handcrafted features to learn higher order representations.

7.4.1 Embedding Customers using Browsing Sessions

We learn representations of ASOS customers from the bipartite customer-product views graph (depicted in Figure 7.8a) using Skipgram with Negative Sampling (SGNS), which is described in detail



(a) Predicted CLTV against actual CLTV (excluding customers with an actual CLTV of 0). Units (horizontal and vertical axis). The distribution of the prediction and the actual CLTV are similar in log scale (top and right density plots). The central plot shows the fit between the predictions and the actual values, which have a Spearman rank-order correlation coefficient of 0.46.

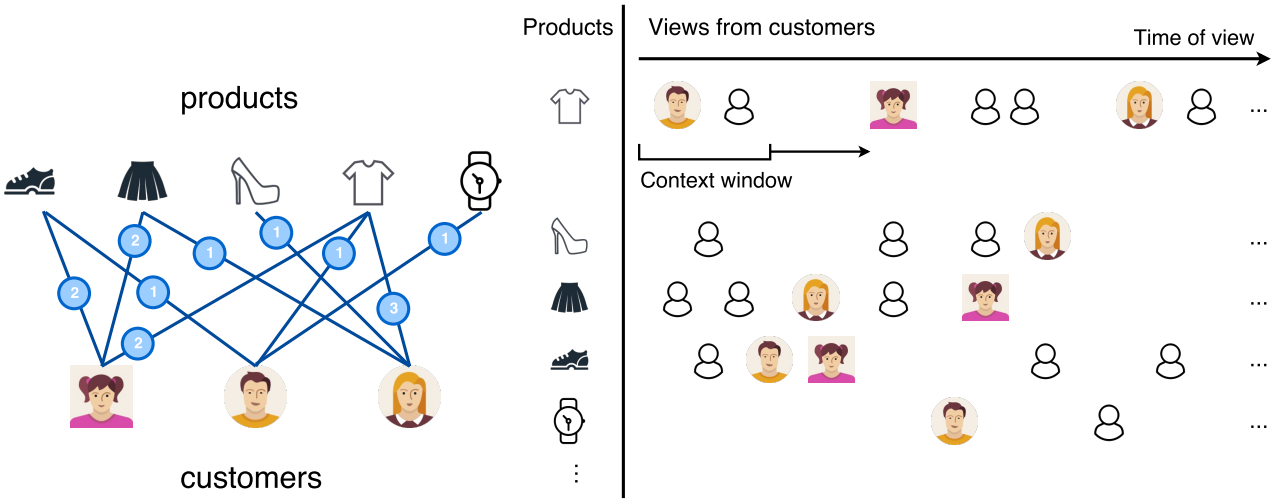


(b) Prediction residuals (prediction - actual). The model tends to slightly overestimate the CLTV as the residuals are slightly shifted to the right.

Figure 7.7: The relationship between predicted and realised CLTV in units of the mean CLTV across all customers. The plots show that the model is well calibrated.

in Section 4.3.5. Previous work has studied embeddings of products based on various forms of customer interactions (Grbovic et al., 2015; Vasile et al., 2016; Barkan and Koenigstein, 2016). Customer embeddings can be produced by aggregating the embeddings of each product that a customer interacts with and this technique is often applied to matrix factorisation embeddings in the recommender systems literature (Covington et al., 2016b; Steck, 2015). However, the aggregation approach fails at the task of producing long-term forecasts when products are relatively short lived as is the case in the fashion industry. For this reason, it is necessary to learn embeddings of customers directly.

To directly learn customer embeddings, the inputs to SGNS are pairs of customers $(C_{\text{in}}, C_{\text{out}})$ and



(a) The bipartite graph of customer product views showing timestamp edge attributes.

(b) Customer sequences derived from the graph.

Figure 7.8: The ASOS customer-product views graph is a sparse bipartite graph that connects every customer who views a product. Sequences are produced by taking the edges from each product and ordering the customers they connect to chronologically. Customer pairs are generated from the sequences using a sliding context window. In this example the context window is of length two and considers only adjacent view events (each pairs in this example) of the same product. Hence, the exact time a product is viewed is ignored. Customers who often appear in the same context window will be close to each other in the embedding representation.

the loss function is:

$$L = -\log \sigma(\mathbf{v}_{\text{out}}^{'T}\mathbf{v}_{\text{in}}) - \sum_{j:C_j \in C_{\text{neg}}} \log \sigma(-\mathbf{v}_j^{'T}\mathbf{v}_{\text{in}}), \tag{7.9}$$

where $\mathbf{v}'_j$ is the output vector representation of customer C_j, v'_{out} is the output vector representation of C_{out}, v_{in} is the input vector representation of C_{in} and $|C|$ is the total number of customers.

To generate customer pairs, a sequence of customer tokens is decomposed using a sliding context window as shown in Figure 7.8b. For every position of the context window, the central customer is used as C_{in} and all other customers in the window are used to form $(C_{\text{in}}, C_{\text{out}})$ pairs. In this way, a window of length three containing (C_1, C_2, C_3) would generate customer pairs (C_2, C_1) and (C_2, C_3). We empirically found that a window of length eleven worked well.

We experiment with two schemes to generate sequences of customer tokens from the graph. (1) Using short random walks over the graph to generate customer sequences. Section 4.4 introduced graph embeddings based on short random walks and these were successfully applied to predict the incomes of Twitter users in Chapter 6. Bipartite graphs contain two distinct entities with edges

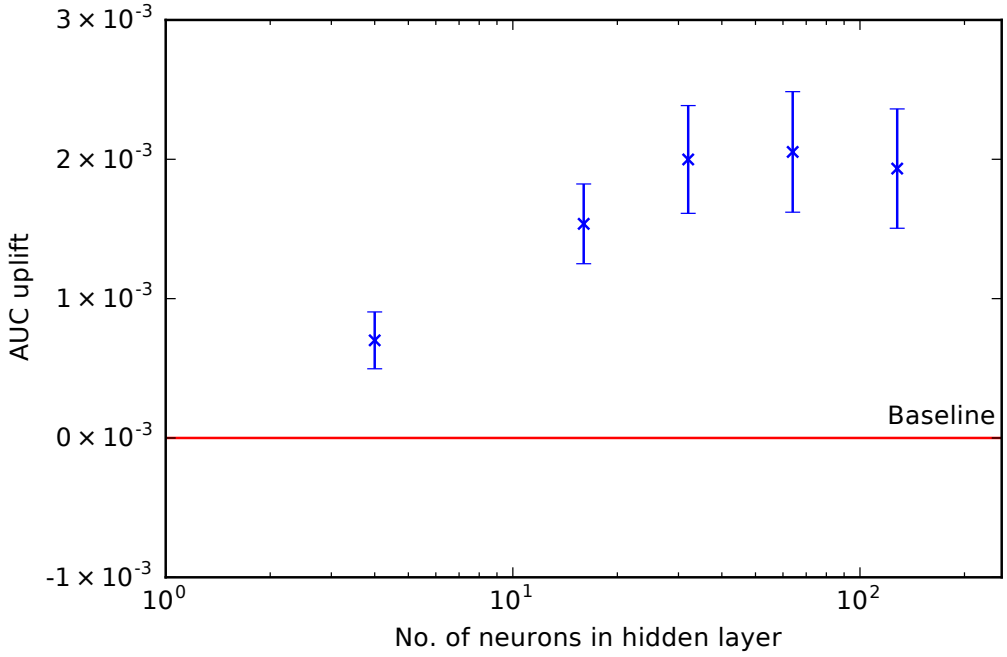


Figure 7.9: Uplift in the area under the receiver operating characteristics curve achieved on random test sets of 20,000 customers with product view-based embeddings against number of neurons in the hidden layer of the SGNS. The error bars represent the 95% confidence interval of the sample mean.

only between entities of different kinds and so to generate sequences of customer tokens it is necessary to take two-step random walks. (2) Using timestamps of product views to generate customer sequences. We produce customer sequences by chronologically ordering the customers associated with the edges of each product in the bipartite customer-product graph. This process is shown in Figure 7.8. Intuitively, the idea is that high-value customers tend to browse products of higher value and less popular products. Of the two approaches only the second produces a significant improvement in the predictive performance of the model. We hypothesise that the timestamps of the edges are important as high value customers often routinely check new products, while lower value customers interact with products that have had their prices reduced. Figure 7.9 shows that we obtained a significant uplift in AUC using embeddings of customers. We experimented with a range of embedding dimensions and found the best performance to be in the range 32-128 dimensions.

Embeddings for the Live System

To use embeddings in the deployed system it is necessary to make a correspondence between the embedding dimensions in the training period and the live system's feature generation period. Figure 7.1 shows that the features for training the CLTV model and features used for the live system

come from disjoint time periods. As the embedding dimensions are unlabelled, randomly initialised and exchangeable in the SGNS loss function (Equation (7.9)), the parameters learned in the training period can not be assumed to match the embeddings used in the live system. The parameter mismatch problem is illustrated in Figure 7.2.

Recall that SGNS begins by randomly initialising all $\mathbf{v}$ and $\mathbf{v}'$. Then, for each customer pair $(C_{\text{in}}, C_{\text{out}})$ with embedded representations $(\mathbf{v}_{\text{in}}, \mathbf{v}'_{\text{out}})$, k negative customer samples C_{neg} are drawn. After the forward pass, $k+1$ output vectors are updated via SGD using backpropagation:

$$\mathbf{v}_j'^{\,\text{new}} = \begin{cases} \mathbf{v}_j'^{\,\text{old}} - \eta \left(\sigma(\mathbf{v}_j'^{\,T} \mathbf{v}_{\text{in}}) - t_j\right) \mathbf{v}_{\text{in}} & \forall j : \; C_j \in C_{\text{out}} \cup C_{\text{neg}} \\ \mathbf{v}_j'^{\,\text{old}} & \text{otherwise} \quad , \end{cases} \tag{7.10}$$

where η is the update rate, σ is the logistic sigmoid and $t_j = 1$ if $C_{\text{in}} = C_{\text{out}}$ and 0 otherwise.

Finally, only one input vector, corresponding to $\mathbf{v}_{\text{in}}$ is updated according to:

$$\mathbf{v}_{\text{in}}^{\text{new}} = \mathbf{v}_{\text{in}}^{\text{old}} - \eta \sum_{j:\, C_j \in C_{\text{out}} \cup C_{\text{neg}}} \left(\sigma(\mathbf{v}_j'^{\,T} \mathbf{v}_{\text{in}}) - t_j\right) \mathbf{v}_{\mathbf{j}}' \quad . \tag{7.11}$$

To solve the parameter mismatch problem, instead of randomly initialising $\mathbf{v}$ and $\mathbf{v}'$ we perform the following initialisation:

- Customers that were present in the training period C_{old}: initialise with training embeddings.
- New customers C_{new}: initialise to uniform random values with absolute values that are small compared to the training embeddings.

In the live system there are four types of $(C_{\text{in}}, C_{\text{out}})$ customer pairs: $(C_{\text{new}}, C_{\text{new}})$, $(C_{\text{new}}, C_{\text{old}})$, $(C_{\text{old}}, C_{\text{new}})$ and $(C_{\text{old}}, C_{\text{old}})$. Equation (7.11) shows that the update to v_{in} is a linear combination of v_{out} and the negative vectors. Therefore a single update of a $(C_{\text{new}}, C_{\text{old}})$ pair is guaranteed to be a linear combination of embedding vectors from the training period. To generate embeddings that are consistent across the two time periods we order the data in each training epoch as (1) $(C_{\text{old}}, C_{\text{old}})$ to update the representation of the old customers, (2) $(C_{\text{new}}, C_{\text{old}})$ to initialise the new customers with linear combinations of embeddings from old customers, (3) $(C_{\text{old}}, C_{\text{new}})$, (4) $(C_{\text{new}}, C_{\text{new}})$. For this scheme to work there must be a large proportion of customers that are present in both the training

and test periods. This is true for customer embeddings, but it is not true for product embeddings as the typical product life is less than six months. This justifies our decision to learn customer embeddings directly instead of using aggregations of product embeddings.

7.4.2 Embeddings of Handcrafted Features

We also investigate to what extent our deployed system could be improved by replacing the RF with a Deep feed-forward Neural Network (DNN). This is motivated by the recent successes of DNNs in vision, speech recognition, and recommendation systems LeCun et al. (2015); Covington et al. (2016b). Our results indicate that while incorporating DNNs in our system may improve the performance, the monetary training cost outweighs the performance benefits.

We limit our experiments with DNNs to the churn problem (predicting a CLTV of zero for the next 12 months). This reduces CLTV regression to a binary classification problem making the predictions and performance metrics easier to interpret.

We experiment with DNNs and hybrid models combining Logistic Regression (LR) and a DNN similar to that of Cheng et al. (2016). The DNNs accept all continuous-valued features and dense embeddings of categorical features, which are described in Section 7.3.1, as inputs. We use Rectified Linear Units (ReLU) activations in the hidden layers and sigmoid activation in the output layer. The hybrid models are LR models incorporating a DNN. The output of the neural network's final hidden layer is used alongside all continuous-valued features and sparse categorical features, as input. This is akin to skip connections in neural networks described by Raiko et al. (2012), with the inputs connected directly to the output instead of the next hidden layer. Training uses the Follow-the-proximally-regularized-leader (FTRL-Proximal) algorithm as described by McMahan et al. (2013) with mini-batch SGD. We use Adagrad for the DNN and L1 regularisation for the LR part.

We evaluate the performance and scalability of the two models with different architectures, and compare them with other machine learning techniques. Neural networks with two, three and four hidden layers, each with different combinations of number of neurons were considered. For each architecture, we train the models using a subset of customer data (the training set), and record (1) the maximum AUC achieved when evaluating on a separate subset of customer data (the test set), (2) the (wall clock) time taken to complete a pre-specified number of training steps. We repeat the training/testing multiple times for each architecture to obtain an estimate on the maximum AUC achieved

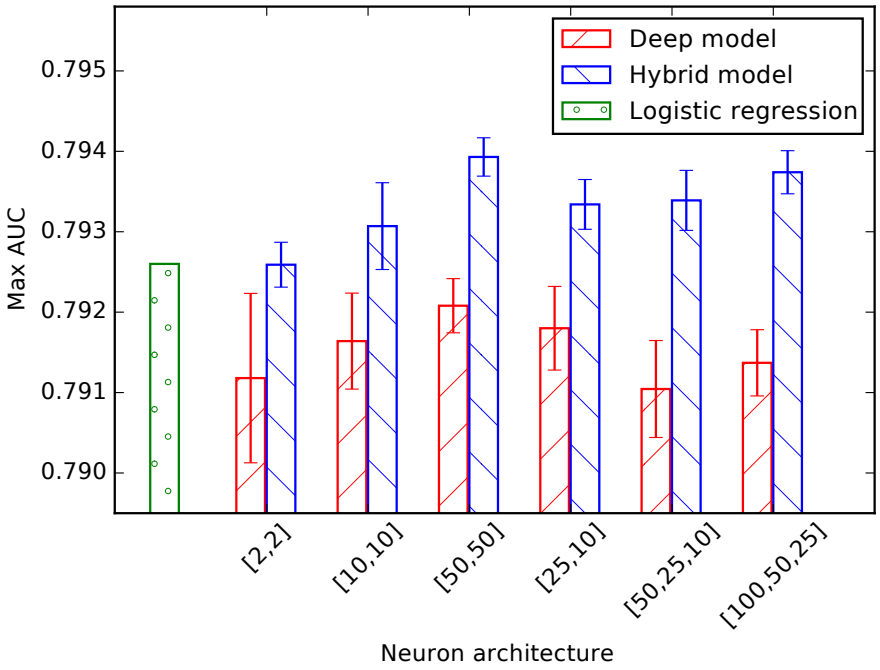


Figure 7.10: Maximum area under the receiver operating characteristics curve achieved on a test set of 50,000 customers in deep feed-forward neural networks and hybrid models with different numbers of hidden layer neurons. The error bars represent the 95% confidence interval of the sample mean. The number of hidden-layer neurons are recorded in the following format: $[x, y]$ denotes a neural network with x and y neurons in the first and second hidden layer respectively, $[x, y, z]$ denotes a neural network with x, y, z neurons in the first, second, and third hidden layers.

and the training time. All training / testing is implemented using the TensorFlow library (Abadi et al., 2015) on a Tesla K80 GPU machine.

Introducing bypass connections in the hybrid models improves the predictive performance compared to a DNN with the same architecture. Figure 7.10 shows a comparison of the maximum AUC achieved by DNNs to the hybrid logistic and DNN models on a test set of 50,000 customers. Our experiments show a statistically significant uplift of at least 1.4×10^{-3} in every configuration of neurons. We believe the uplift is due to the hybrid models' ability to memorize the relationship between a set of customer attributes and their churn status in the LR part. This is complementary to a DNN's ability to generalise based on customers who have similar aggregation features, as described by Cheng et al. (2016).

We estimate the size (in number of neurons) of the hybrid model required to outperform the AUC of the RF model. Figure 7.11 shows that there is a linear relationship between the maximum AUC achieved on the same test set of customers and the number of neurons in each hidden layer in logarithmic scale. We notice a hybrid model with a small number of neurons in each hidden layer

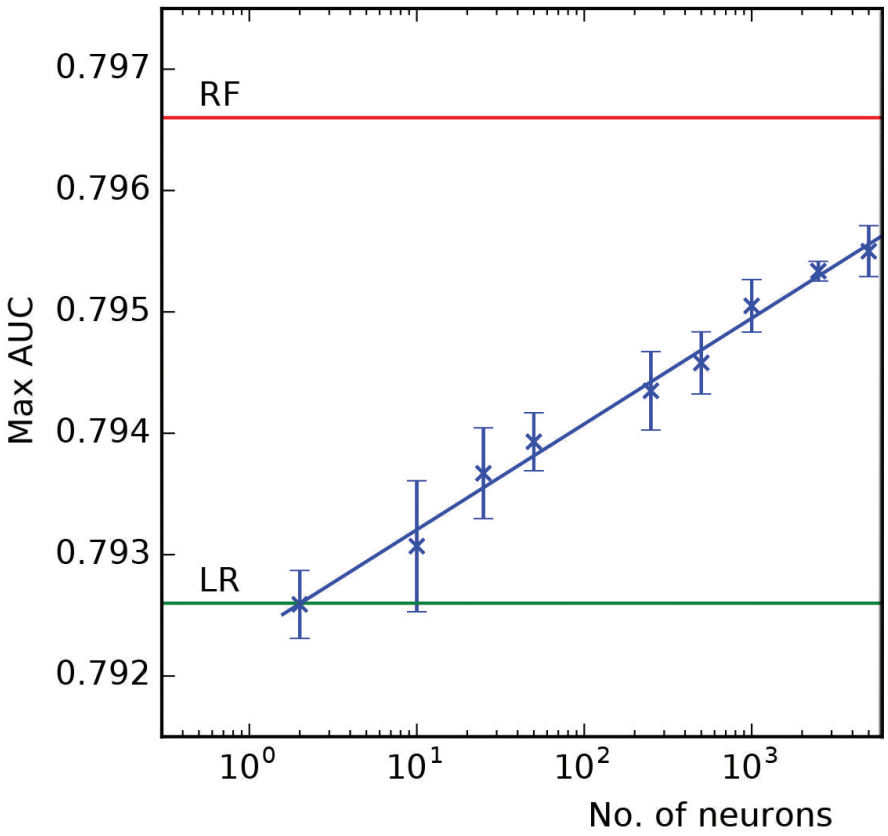


Figure 7.11: Maximum Area Under the receiver operating characteric Curve (AUC) achieved on a test set of 50,000 customers in hybrid models against number of neurons in the hidden layers (in log scale). We only consider hybrid models with two hidden layers, each having the same number of neurons. The error bars represent the 95% confidence interval of the sample mean. The bottom (green) and top (red) horizontal line represent the maximum AUC achieved by a logistic regression model and our random forest model on the same set of customers.

already gives statistically significant improvement in maximum AUC achieved compared with LR, but within the range of our experiments we could not exceed the performance of the RF model.

While the experiments suggest it is possible for our hybrid model, which incorporates a DNN, to outperform the calibrated RF model in churn classification, we believe the monetary cost of training the model outweighs any gains in performance. Figure 7.12 shows the relationship between training cost for the hybrid models and the number of neurons in each hidden layer. The cost of training LR and our calibrated RF model are indicated by horizontal lines. The cost rises linearly with the number of neurons and exponentially with the Max AUC, indicating that a hybrid model that outperformed our calibrated RF model is prohibitively expensive.

We believe the case is similar for CLTV prediction, in which a hybrid model on handcrafted features (whether in numerical values or categorical feature embeddings) can achieve better performance

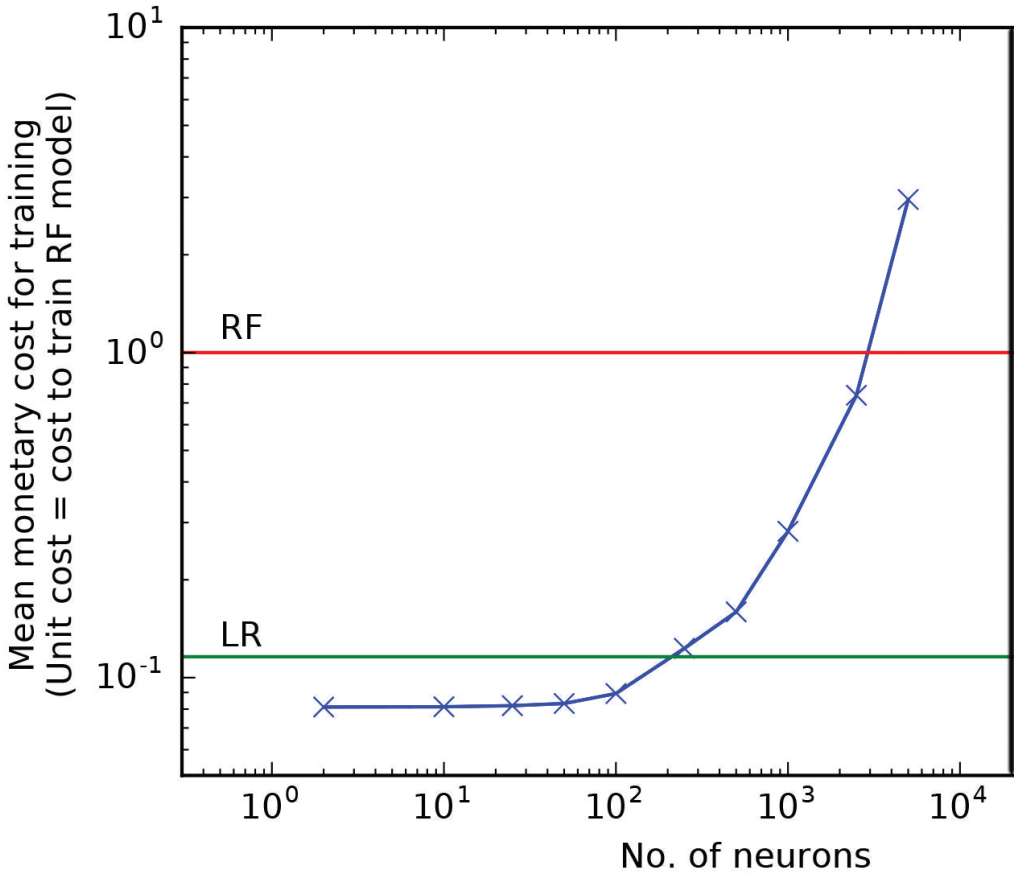


Figure 7.12: Mean monetary cost to train hybrid models on a training set of 100,000 customers against the number of neurons in the hidden layers (both in log scale). The training cost shown is relative to the cost of training our random forest (RF) model. Here we only consider hybrid models with two hidden layers, each having the same number of neurons. The bottom (green) and top (red) horizontal line represents the mean cost to train LR and our RF model on the same set of customers. The cost shown is based on the time required to train each of the models, and the cost of using the computational resources - Spark clusters to train RF models, and GPU VMs to train LR and hybrid models in Microsoft Azure.

than our deployed RF model, though with a much higher cost than is commercially viable. This is supported by our preliminary experiments, which measure the root mean squared error (RMSE) between the hybrid models' predicted percentile and actual percentile of each customer's spend. We observe increasing the number of neurons in hybrid models decreases the RMSE, but are unable to train hybrid models with tens of thousands of neurons due to prohibitive runtimes.

7.5 Summary

We have described the CLTV system deployed at ASOS.com and the main issues we faced while building it. The first half of the chapter introduced the baseline architecture, which achieves state-of-the-art performance for this problem. We highlighted two issues that are often overlooked in the literature: (1) model calibration and (2) generalising optimization to include practical considerations for live commercial ML system, which in this case must take into account prediction stability and training cost. A strength of our approach is that predictions are well calibrated and can be meaningfully aggregated to predict, for instance, total sales in France next year. Unlike previous related work that used a divide and conquer approach (Vanderveld and Han, 2016), we share statistical strength between all samples used for training, making our model more data efficient. The system is also highly practical as it balances the requirements to produce accurate predictions, alert for large changes in predicted values and runs at a cost that ensures net profitability. Given the recent success of representation learning across a wide range of domains, in the second half of the chapter we focussed on our ongoing efforts to further improve the model by learning two additional types of representations: (1) embeddings of customers using session data in an unsupervised setting that augmented the RF features (Section 7.4.1) and (2) training feedforward neural networks on handcrafted features in a supervised setting (Section 7.4.2). We showed that learning an embedding of a rich source of data (products viewed by a customer) in an unsupervised setting can improve the performance over using only handcrafted features. The embedded features alone are powerful. Removing all handcrafted features still provides an AUC of roughly 0.7, compared to 0.8 for the full system, but requiring no specialist knowledge. One of the major advantages of learning customer embeddings is that they are unsupervised and so can be re-used for many tasks, such as a product recommendations, that require customer information. The main alternative approach to the two described ways of learning representations would be to use a deep network to learn end-to-end from raw data sources as opposed to using handcrafted features as inputs. We are starting to explore this approach, which while challenging, we believe might also improve upon the state-of-the-art.

In our analysis of learned features we could not discover a DNN architecture that provided improvements in predictive performance that justified the cost of running the model. There are several limitations to this analysis. Firstly, it is not possible to perform an exhaustive analysis of neural network architectures. We did not experiment with any state-of-the-art recurrent or convolutional networks for instance, nor did we attempt to learn end-to-end from raw features such as the product views

data. Secondly, additional training costs must be compared to the commercial benefits of improving a model, which in practice are difficult to estimate. We estimate the value of ML systems by running random controlled trials such as sending discount coupons to customers above a threshold CLTV given some conditions. However, the trials only shed light on the value of a given model with a given predictive performance and not on the additional value that would be generated by improving the model. Thirdly, the full range of applications of an ML system are not known at the time the system goes into production. Typically the model is designed for a specific use case and, if successful, additional applications emerge. For this reason, the commercial value of a model may be underestimated.

In the future we are interested in treating customers as marked time series of instead of snap shots of data drawn from the last twelve months. Techniques that combine recurrent neural networks with point process models appear to provide a promising direction for research in this area (Du et al., 2015, 2016).

Our baseline model uses 132 handcrafted features. This number is relatively arbitrary and was chosen as the point where no more improvements in predictive performance were generated by including additional features. However, as of September 2017, ASOS had 15 million active customers and there is some evidence that for this amount of training data, improvements could be made using an order of magnitude more features by combining features into what are known as *crossing* or *combinatorial features* (Shan et al., 2016). An example crossing feature form the ASOS dataset is "has purchased shoes AND has purchased jeans".

In Section 7.4.1 we showed that by incorporating embeddings into the RF model we were able to produce significant improvements in predictive performance even when no more improvements were generated by including additional handcrafted features. The SGNS embeddings are vectors that exist in a Euclidean vector space. This choice of geometry is made for convenience and there are many forms of geometry in which customer embeddings could be learned. Having shown that embeddings provide a promosing direction for improving predictive customer models it is interesting to explore whether further improvements could be made by optimizing the underlying geometry of the embeddings. Research in this direction is the subject of the next Chapter 8.

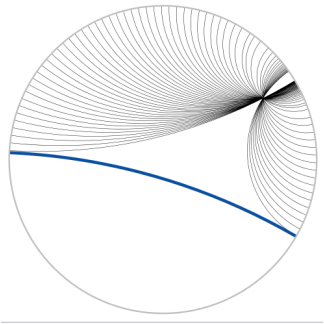
Chapter 8

HyBed: Neural Embeddings of Graphs using Hyperbolic Geometry

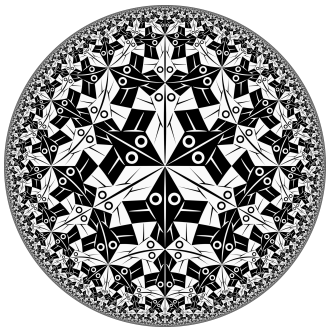
In the previous chapter, we showed how neural embeddings of customers can be learned that improve performance in the important commercial task of predicting future customer value. In this chapter, we develop the theory of neural embeddings in a different direction, by investigating the underlying geometry. We study embeddings of graph-structured data using short random walks as described in Section 4.4. Graph embeddings of this form have been shown to encapsulate vertex similarity and improve performance on tasks including edge prediction and vertex labelling (Perozzi and Skiena, 2014). Embeddings of the form used by Perozzi and Skiena (2014) are learned in high-dimensional Euclidean space. However, recent work has shown that a negatively curved hyperbolic space has many attractive properties for this task. We present HyBed embeddings, a concept that uses these recent insights and learns neural embeddings of graphs that exploit properties of hyperbolic space. We provide experimental evidence that HyBed embeddings significantly outperform Euclidean embeddings on vertex classification tasks for several real-world public datasets. This chapter is an extension of the work presented in Chamberlain et al. (2017b).

8.1 Introduction

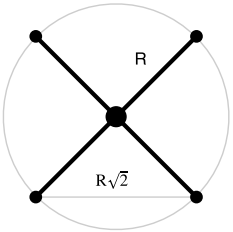
There are two reasons why embedding complex networks in hyperbolic geometry can be expected to perform better than Euclidean geometry. The first is that many complex networks exhibit a hierarchical structure (Newman, 2003). Hyperbolic geometry provides a continuous analogue of tree-like



(a) Parallel hyperbolic lines.



(b) "Circle Limit 1", Escher



(c) A hub and spoke graph.

Figure 8.1: Properties of hyperbolic space. (a) Multiple parallel lines passing through a single point. (b) All tiles are of constant area in hyperbolic space, but shrink to zero area at the boundary of the disk in Euclidean space. (c) Hub and spokes graph. It is impossible to embed this graph in two-dimensional Euclidean space and preserve the properties that (1) all spokes are the same distance from the hub, (2) all spokes are the same distance from each other, and (3) the distance between spokes along the circumference is more than twice the distance to the hub. In hyperbolic space such embeddings exist.

graphs, and even infinite trees [1] have nearly isometric embeddings in hyperbolic space (Gromov, 2007). The second property is that many complex networks have power-law degree distributions, resulting in high-degree hub vertices (Clauset et al., 2007). Figure 8.1c shows a simple hub-and-spoke graph where each spoke is a distance R from the hub and $2R$ from each other. For an embedding in two-dimensional Euclidean space it is impossible to reproduce this geometry for more than two spokes. However, in hyperbolic space, large numbers of spokes that satisfy these geometrical constraints can be embedded because the circumference of a circle expands exponentially rather than polynomially with the radius.

In this chapter, we introduce HyBed graph embeddings that exploit properties of hyperbolic space. We formulate backpropagation for our model in hyperbolic space and show that using an underlying non-Euclidean geometry improves performance in vertex classification tasks across multiple real-world networks.

8.2 Hyperbolic Geometry

Hyperbolic geometry emerged from the difficulties that mathematicians faced trying to prove the parallel postulate from the other axioms of Euclid's geometry. The discovery is jointly credited to

[1] assuming unweighted / unit edges

Nikolai Lobachevsky, Janos Bolyai and Carl Gauss. In Euclidean geometry, given a line L and a point P not on the line, there is exactly one co-planer line that goes through P and does not intersect L. In hyperbolic space however, there are an infinite number of lines parallel to L that pass through P. This is illustrated in Figure 8.1a where every fine line is parallel to the bold, blue line, and all pass through the same point. Hyperbolic space is one of only three types of isotropic space that can be defined entirely by their curvature. The most familiar is flat Euclidean space. Space with uniform positive curvature has an *elliptic geometry* (e.g. the surface of a sphere) and space with uniform negative curvature has a *hyperbolic geometry*, which is analogous to a saddle-like surface. As hyperbolic space is curved, the shortest distance between two points is not a straight line, but a curved *geodesic*. The length of a geodesic is defined through the *metric tensor*. In addition to providing a continuous analogue for tree graphs, many of the properties of complex networks naturally appear when graphs are randomly generated in hyperbolic geometry. Power-law degree distributions, strong clustering and community structure, emerge naturally when points are scattered on the hyperbolic plane and edges connect any two points that are separated by less than a threshold hyperbolic distance (Krioukov et al., 2010).

One important property of hyperbolic space is that it is in some sense *larger* than Euclidean space; the 2D hyperbolic plane cannot be isometrically embedded into Euclidean space of any dimension, unlike elliptic geometry where a 2-sphere can be embedded into 3D Euclidean space etc. The hyperbolic area of a circle or volume of a sphere grows exponentially with its radius, rather than polynomially. For a two-dimensional hyperbolic space of curvature K, the circumference of a circle is given by

$$C = \frac{2\pi}{\sqrt{-K}} \sinh r\sqrt{-K}, \tag{8.1}$$

where r is the radius. For the remainder of the chapter we restrict $K = -1$, in which case a circle of radius 10 has a circumference $C \approx 10000\pi$ instead of 20π. For a radius of 100 this becomes $O(10^{43})$. This property allows low-dimensional hyperbolic spaces to provide effective representations of data in ways that low-dimensional Euclidean spaces cannot. Figure 8.1c shows a hub-and-spoke graph with four spokes embedded in a two-dimensional Euclidean plane so that each spoke sits on the circumference of a circle surrounding the hub. In the distance metric defined by the graph, each spoke is a distance R from the hub and $2R$ from every other spoke. However, in the Euclidean embedding the spokes are a distance of R from the hub, but only $R\sqrt{2}$ from each other. Complex networks often have small numbers of vertices with degrees that are orders of magnitude greater than the median.

These vertices approximate hubs. The distance between spokes tends to the distance along the circumference $s = \frac{2\pi R}{n}$ as the number of spokes n increases, so that the shortest distance between two spokes is via the hub only when $n < \pi$. However, for embeddings in hyperbolic space, the same calculation yields $n < \frac{\sinh R}{R}$, such that an infinite number of spokes can satisfy the property that they are the same distance from a hub, and yet the path that connects them via the hub is shorter than along the arc of the circle. As hyperbolic space can not be isometrically embedded in Euclidean space, there are many different representations that each conserve some geometric properties, but distort others. Here, we use the Poincaré disk model of hyperbolic space.

8.2.1 Poincaré Disk Model

The Poincaré disk models the infinite two-dimensional hyperbolic plane $\mathbb{H}^2$ as the interior of the unit disk. For simplicity we work with the two-dimensional Poincaré disk, but it is easily generalised to the d-dimensional Poincaré ball, where hyperbolic space $\mathbb{H}^d$ is represented by the interior of the Euclidean unit d-ball. Hyperbolic distances grow exponentially towards the edge of the disk. The boundary of the disk $\mathbb{S}^1$ represents infinitely distant points as the infinite hyperbolic plane is squashed inside the finite disk. This property is illustrated in Figure 8.1b where each tile is of constant area in hyperbolic space, but rapidly shrink to zero area in Euclidean space. Although volumes and distances are warped, the Poincaré disk model is *conformal*. The metric tensor for the Poincaré disk is given by

$$ds^2 = \frac{4\left(dx^2 + dy^2\right)}{\left(1 - \left(x^2 + y^2\right)\right)^2} \tag{8.2}$$

Straight lines in hyperbolic space intersect the boundary of the disk orthogonally and appear either as diameters of the disk, or arcs of a circle. Figure 8.1a shows a collection of straight hyperbolic lines in the Poincaré disk. Just as in spherical geometry, shortest paths appear curved on a flat map, hyperbolic geodesics also appear curved in the Poicaré disk. This is because it is quicker to move close to the centre of the disk, where distances are shorter, than nearer the edge. In our proposed approach, we will exploit the circular symmetry of the Poincaré disk. The geometric intuition is that vertices embedded near the middle of the disk can have more 'near' neighbours than they could in Euclidean space, whilst vertices nearer the edge of the disk can still be very far from each other.

8.2.2 Similarities, Angles and Distances

As can be seen from Equation (8.2), The metric tensor in the Poincaré disk is a function only of the radius. Exploiting the angular symmetries of the model using polar coordinates considerably simplifies the mathematical description of our approach and the efficiency of our optimizer. Points in the disk are $x = (r_e, \theta)$, with $r_e \in [0, 1)$ and $\theta \in [0, 2\pi)$. The distance from the origin, r_h is given by

$$r_h = 2 \operatorname{arctanh} r_e \tag{8.3}$$

and the circumference of a circle of hyperbolic radius R is $C = 2\pi \sinh R$. Note that as points approach the edge of the disk, $r_e = 1$, the hyperbolic distance from the origin r_h tends to infinity. In Euclidean neural embeddings, the inner product between vector representations of vertices is used to quantify their similarity. However, unlike Euclidean space, hyperbolic space is not a vector space and there is no global inner product. Instead, given points $\mathbf{x}_1 = (r_1, \theta_1)$ and $\mathbf{x}_2 = (r_2, \theta_2)$ we define a cosine similarity weighted by the hyperbolic distance from the origin as

$$\langle \mathbf{x}_1, \mathbf{x}_2 \rangle_H = \|\mathbf{x}_1\| \|\mathbf{x}_2\| \cos(\theta_1 - \theta_2) \tag{8.4}$$

$$= 4 \operatorname{arctanh} r_1 \operatorname{arctanh} r_2 \cos(\theta_1 - \theta_2). \tag{8.5}$$

It is this function that we will use to quantify the similarity between points in the embedding. We note that using a cosine distance in this way loses some properties of hyperbolic space such as conformality. However, it retains the property that the space expands exponentially with distance from the origin, which allows hierarchical structures with power law degree distributions and small-world phenomena to be represented. Our goal is to learn embeddings that can be trained efficiently and perform well on downstream tasks and trade-offs like this are common in the embedding literature e.g. negative sampling (Mnih and Teh, 2012; Mnih and Kavukcuoglu, 2013).

8.3 Related Work

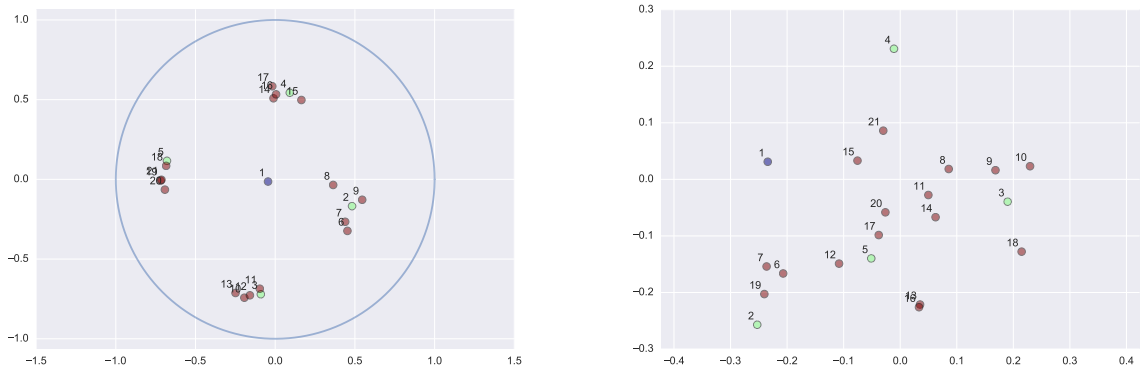
Recent work in the field of complex networks has found that many interesting networks, particularly those with a scale-free structure such as the internet (Shavitt and Tankel, 2008; Boguna et al., 2010) or academic citation networks (Clough et al., 2015; Clough and Evans, 2016) can be well de-

scribed with a hyperbolic geometry. Hyperbolic graph embeddings have been applied successfully to the problem of greedy message routing (Kleinberg, 2007; Cvetkovski and Crovella, 2009) and more recently scalable embeddings of general graphs in low-dimensional hyperbolic space have been addressed by (Bläsius et al., 2016). The work that is most similar to our is by Nickel and Kiela (2017), which appeared independently and concurrently with Chamberlain et al. (2017b) and also uses a shallow neural network to perform the embedding. The key differences are that Nickel and Kiela (2017) try to fit the hyperbolic distance between nodes using Cartesian coordinates in the Poincaré disk, whereas we use a modified cosine distance in a spherical, natural hyperbolic coordinate system. In Euclidean coordinates on the disk, the gradient descent optimizer can move points beyond a radius of one where the notion of distances and gradients breaks down. This problem is resolved by moving points arbitrarily a small Euclidean (but infinite hyperbolic) distance back inside the disk. Our approach, however, is free from this constraint.

A related area of research that explores non-Euclidean geometries is *geometric deep learning* (Bronstein et al., 2017). Geometric deep learning is mostly concerned with how to replicate the success of Convolutional Neural Networks (CNN)s when the underlying data is not arranged on a simple Euclidean grid. Convolutional operations are not well defined on irregular graphs and so signal processing techniques are applied to replicate the function of a Euclidean convolution (Henaff et al., 2015; Defferrard et al., 2016). Recent advances in this field include model simplifications (Kipf and Welling, 2016), an extension to directed graphs (Monti et al., 2018) and applications to recommendation systems (Monti et al., 2017b).

8.4 Neural Embedding in Hyperbolic Space

As in Chapter 7 we use the Skipgram with Negative Sampling (SGNS) model, which is introduced in Section 4.3.5, to generate neural embeddings. We adopt the same notation, but replace words with graph vertices. Therefore the input vertex is denoted w_I and the context / output vertex is w_O. The corresponding vector representations are $\mathbf{v}_{w_I}$ and $\mathbf{v}'_{w_O}$, which are elements of the two vector spaces shown in Figure 4.5, W and W' respectively. Skipgram has a geometric interpretation, shown in Figure 4.6 for vectors in W'. Updates to $\mathbf{v}'_{w_j}$ are performed by simply adding (if w_j is the observed output vertex) or subtracting (otherwise) an error-weighted portion of the input vector. Similar, though slightly more complicated, update rules are given in Equation (4.33) for the vectors in W.



(a) Hyperbolic embeddings

(b) Euclidean embedding

Figure 8.2: Comparison between the embeddings of a complete 4-ary tree with three levels. (a) Hyperbolic embeddings are able to represent the trees branching factor and position the root at the location of the shortest path length. (b) The Euclidean embedding can not approximate the isometries of the tree.

Given this interpretation, it is natural to look for alternatives that improve upon Euclidean geometry.

To learn the embeddings, we replace Skipgram's two Euclidean vector spaces (W and W' in Figure 4.5) with two hyperbolic spaces that are represented by Poincaré disks. We learn embeddings by optimizing an objective function that predicts context vertices from an input vertex, but we replace the Euclidean dot products used in Skipgram with (8.5). A softmax function is used for the conditional predictive distribution

$$p(w_O|w_I) = \frac{\exp(\langle \mathbf{v}'_{w_O}, \mathbf{v}_{w_I} \rangle_H)}{\sum_{i=1}^{V} \exp(\langle \mathbf{v}'_{w_i}, \mathbf{v}_{w_I} \rangle_H)}, \tag{8.6}$$

where $\mathbf{v}_{w_i}$ is the vector representation of the i^{th} vertex, primed indicates context vectors (see Figure 4.5) and $\langle \cdot, \cdot \rangle_H$ is given in (8.5).

8.4.1 Model Learning

We learn the model using backpropagation with Stochastic Gradient Descent (SGD). Optimization is conducted in polar native hyperbolic coordinates where $r \in (0, \infty)$, $\theta \in (0, 2\pi]$. For optimization, this coordinate system has two advantages over the Cartesian Euclidean system used by Nickel and Kiela (2017). Firstly there is no need to constrain the optimizer such that $\|\mathbf{x}\| < 1$. This is important as arbitrarily moving points a small Euclidean distance inside the disk equates to an infinite hyperbolic distance. Secondly, polar coordinates result in update equations that are simple

modifications of the Euclidean updates, which avoids evaluating the metric tensor for each data point. The drawbacks of this coordinate system is that it introduces a singularity at the origin and we need to encode a wrapping of the angular co-ordinate. To address both issues, we initialise all vectors to be in a patch of space that is small relative to its distance from the origin and not near $\theta = 0$. The negative log likelihood using negative sampling is

$$E = -\log \sigma(\langle \mathbf{v}'_{w_O}, \mathbf{v}_{w_I} \rangle_H) - \sum_{w_j \in W_{neg}} \log \sigma(-\langle \mathbf{v}'_{w_j}, \mathbf{v}_{w_I} \rangle_H) \tag{8.7}$$

$$= -\log \sigma(u_O) - \sum_{w_j \in W_{neg}} \mathbb{E}_{w_j \sim P_n}[\log \sigma(-u_j)] \tag{8.8}$$

where $\mathbf{v}_{w_I}$, $\mathbf{v}'_{w_O}$ are the vector representation of the input and context vertices, $u_j = \langle \mathbf{v}'_{w_j}, \mathbf{v}_{w_I} \rangle_H$, W_{neg} is a set of samples drawn from the noise distribution and σ is the sigmoid function. The first term represents the observed data and the second term the negative samples. To draw W_{neg}, we specify the noise distribution P_n to be the unigram distribution of the vertices in the input sequence raised to $3/4$ as in (Mikolov et al., 2013a). The gradient of the negative log-likelihood in (8.8) w.r.t. u_j is given by

$$\frac{\partial E}{\partial u_j} = \epsilon_j = \begin{cases} \sigma(u_j) - 1, & \text{if } w_j = w_O \\ \sigma(u_j), & \text{if } w_j \in W_{neg} \\ 0, & \text{otherwise} \end{cases}. \tag{8.9}$$

The derivatives w.r.t. the components of vectors in W' (in natural polar hyperbolic coordinates) are

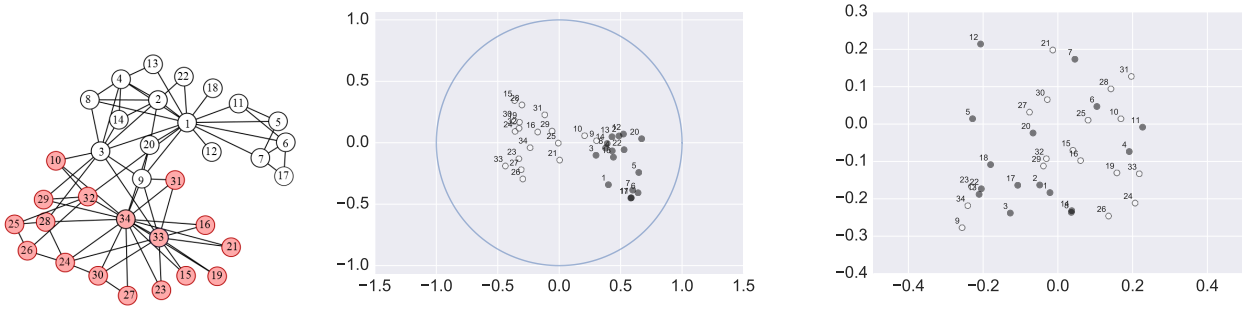
$$\frac{\partial E}{\partial (r'_j)_k} = \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial (r'_j)_k} = \frac{\partial E}{\partial u_j} r_I \cos(\theta_I - \theta'_j) \tag{8.10}$$

$$\frac{\partial E}{\partial (\theta'_j)_k} = \frac{\partial E}{\partial u_j} r'_j r_I \sin(\theta_I - \theta'_j)\,, \tag{8.11}$$

such that the Jacobian is $\nabla_{\mathbf{r}} E = \frac{\partial E}{\partial r} \hat{\mathbf{r}} + \frac{1}{\sinh r} \frac{\partial E}{\partial \theta} \hat{\boldsymbol{\theta}}$. This leads to

$$r'^{new}_j = \begin{cases} r'^{old}_j - \eta \epsilon_j r_I \cos(\theta_I - \theta'_j), & \text{if } w_j \in \chi \\ r'^{old}_j, & \text{otherwise} \end{cases} \tag{8.12}$$

$$\theta'^{new}_j = \begin{cases} \theta'^{old}_j - \eta \epsilon_j \frac{r_I r_j}{\sinh r_j} \sin(\theta_I - \theta'_j), & \text{if } w_j \in \chi \\ \theta'^{old}_j, & \text{otherwise} \end{cases} \tag{8.13}$$



(a) Karate network (b) hyperbolic embeddings (c) Euclidean embeddings.

Figure 8.3: The two factions of the Zachary karate network are linearly separable when embedded in 2D hyperbolic, but not Euclidean space. Both embeddings were run for 5 epochs on the same intermediate random walks.

where $\chi = w_O \cup W_{neg}$, η is the learning rate and ϵ_j is the prediction error defined in (8.9). Calculating the derivatives w.r.t. the input embedding follows the same pattern, and we obtain

$$\frac{\partial E}{\partial r_I} = \sum_{j:w_j \in \chi} \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial r_I} \tag{8.14}$$

$$\frac{\partial E}{\partial r_I} = \sum_{j:w_j \in \chi} \frac{\partial E}{\partial u_j} r'_j \cos(\theta_I - \theta'_j) \tag{8.15}$$

$$\frac{\partial E}{\partial \theta_I} = \sum_{j:w_j \in \chi} \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial \theta_I} \tag{8.16}$$

$$\frac{\partial E}{\partial \theta_I} = \sum_{j:w_j \in \chi} -\frac{\partial E}{\partial u_j} r_I r'_j \sin(\theta_I - \theta'_j)\,. \tag{8.17}$$

The corresponding update equations are

$$r_I^{new} = r_I^{old} - \eta \sum_{j:w_j \in \chi} \epsilon_j r'_j \cos(\theta_I - \theta'_j) \tag{8.18}$$

$$\theta_I^{new} = \theta_I^{old} - \eta \sum_{j:w_j \in \chi} \epsilon_j \frac{r_I r'_j}{\sinh r_I} \sin(\theta_I - \theta'_j)\,. \tag{8.19}$$

Following optimization, the vectors are mapped back to Euclidean coordinates on the Poincaré disk through $\theta_h \to \theta_e$ and $r_h \to \tanh \frac{r_h}{2}$. The asymptotic (in the number of graph vertices) runtime complexity of the update equations (8.12)–(8.13) and (8.18)–(8.19) are the same as Euclidean Skipgram (cf. Equations (4.30) and (4.33)), i.e. the HyBed embedding does not add computational burden.

8.5 Experimental Evaluation

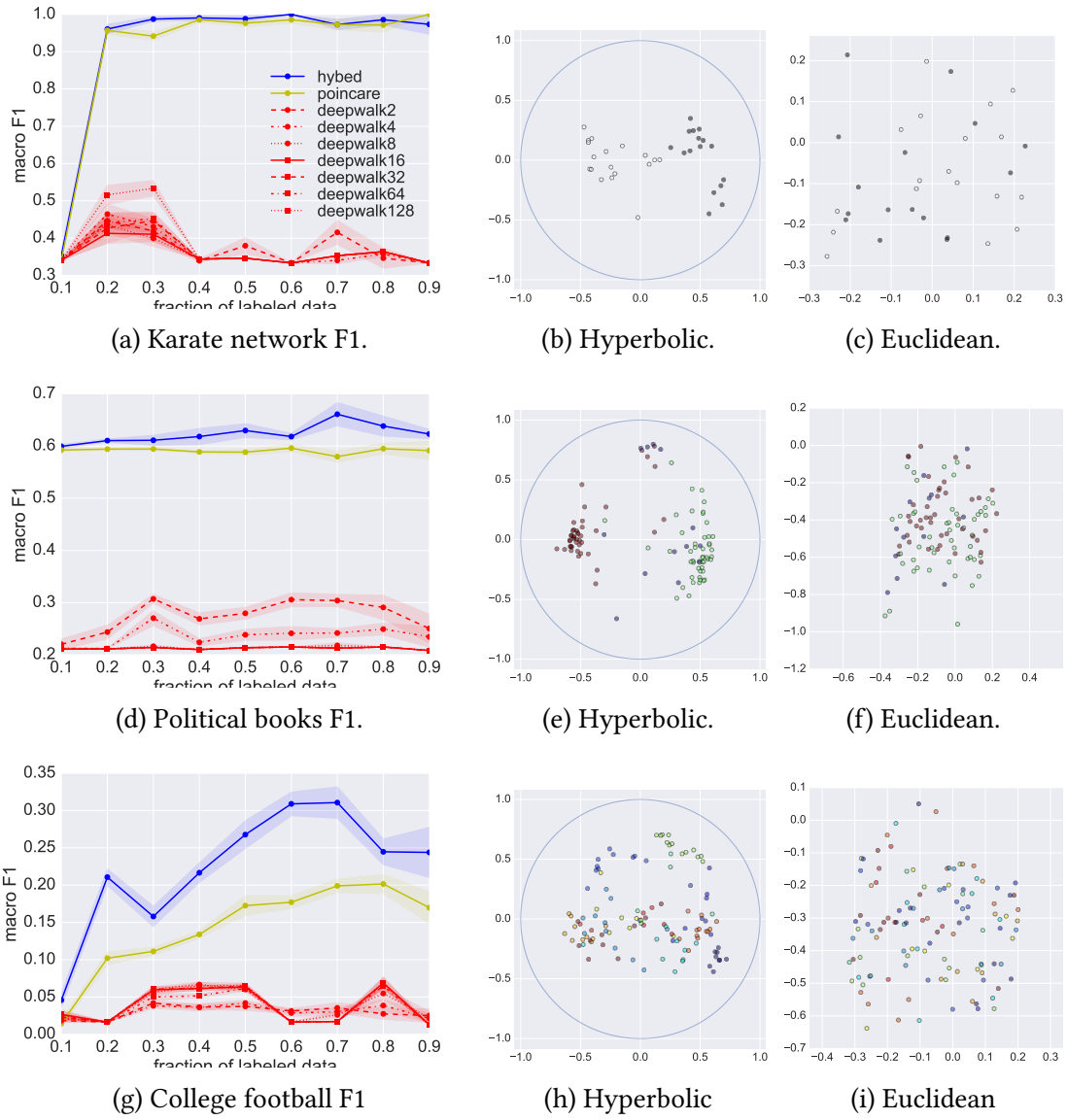
In this section, we assess the quality of HyBed embeddings and compare them to embeddings in Euclidean space. Firstly, we perform a qualitative assessment of the embeddings on a synthetic fully connected tree graph and a small social network. From this analysis it is clear that HyBed embeddings exhibit a number of features that are superior to Euclidean embeddings. Secondly, we run experiments on a number of public benchmark networks, producing both Euclidean and hyperbolic embeddings and contrast the performance of both on a downstream vertex classification task. A TensorFlow implementation and datasets to replicate the experiments are available at our github repository [2].

8.5.1 Qualitative Assessment

To illustrate the usefulness of HyBed embeddings, we compare them with plots of Euclidean embeddings. In all cases, embeddings were generated using five training epochs on an intermediate dataset of ten-step random walks, one originating at each vertex. Figures 8.2 and 8.3 show HyBed embeddings in the 2D Poincaré model of hyperbolic space where the circles of radius 1 is the infinite boundary and Euclidean embeddings in $\mathbb{R}^2$. Figure 8.2 shows embeddings of a complete 4-ary tree with three levels. The vertex numbering is breadth first with one for the root and 2, 3, 4, 5 for the second level etc. The HyBed embedding has the root vertex close to the origin of the disk, which is the position with the shortest average path length. The leaves are all located in close proximity to their parents, and there are clearly four clusters representing the tree's branching factor. The Euclidean embedding is incapable of representing the tree structure with adjacent vertices at large distances (such as 1 and 3) and vertices that are maximally separated in the tree appearing close in the embedding (such as 19 and 7).

Figure 8.3a shows the 34-vertex karate network, which is split into two factions. Figure 8.3b shows the HyBed embedding of this network where the two factions can be clearly separated. In addition, the vertices (5, 6, 7, 11, 17) in Figure 8.3a are the junior instructors, who are forbidden by the instructor (vertex 1) from socialising with other members of the karate club. For this reason they form a community that is only connected through the instructor. This community is clearly visible in Figure 8.3b to the right of the graph. The Euclidean embedding in Figure 8.3c fails to capture these

[2] `https://github.com/melifluos/Hybed`



(a) Karate network F1. (b) Hyperbolic. (c) Euclidean.

(d) Political books F1. (e) Hyperbolic. (f) Euclidean.

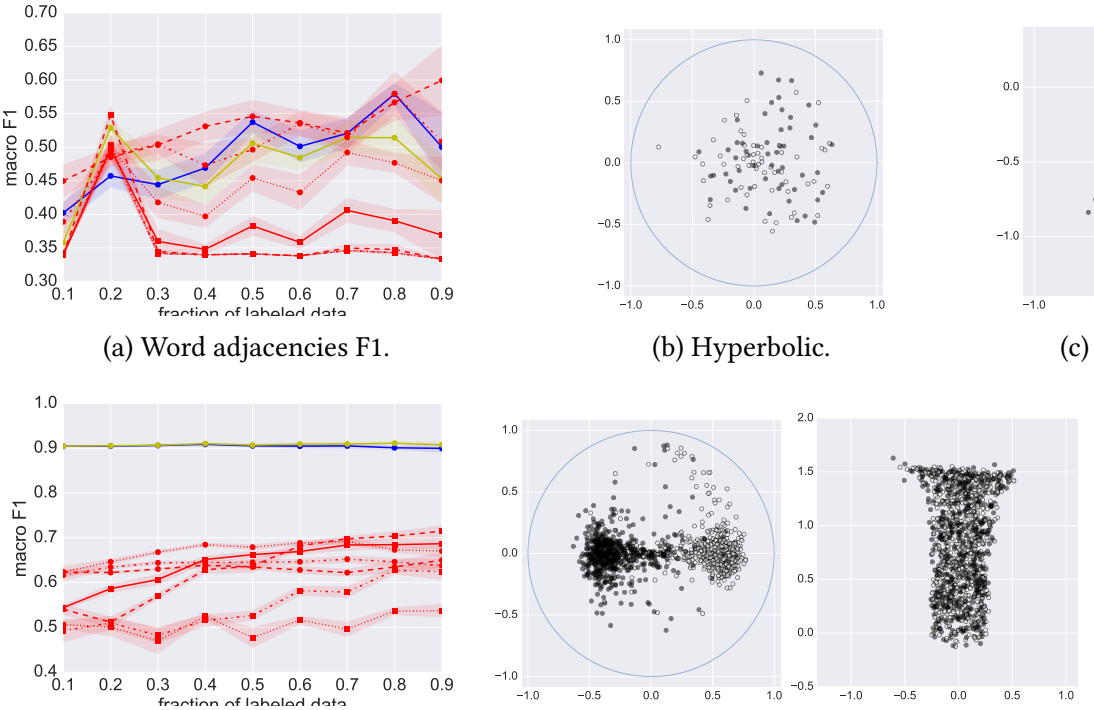
(g) College football F1 (h) Hyperbolic (i) Euclidean

Figure 8.4: Each row contains a line plot of macro F1 score for predicting held-out vertex labels from embedded representations using logistic regression together with the HyBed and 2D Euclidean embeddings. Error bars are standard errors from the mean over ten repetitions.

important features of the underlying graph.

8.5.2 Vertex Attribute Prediction

We quantitatively evaluate the success of HyBed embeddings by using them as features to predict held-out labels of vertices in networks. In our experiments, we compare our embedding to Euclidean embeddings of dimensions 2, 4, 8, 16, 32, 64 and 128 and the hyperbolic embeddings of Nickel and Kiela (2017). To generate the embeddings we first create an intermediate dataset by taking a series of random walks over the networks. For each network we use a set of ten-step random walks with one



(a) Word adjacencies F1. (b) Hyperbolic. (c) Euclidean.

(d) Political blogs F1. (e) Hyperbolic (f) Euclidean

Figure 8.5: Each row contains a line plot of macro F1 score for predicting held-out vertex labels from embedded representations using logistic regression together with the HyBed and 2D Euclidean embeddings. Error bars are standard errors from the mean over ten repetitions.

walk originating at each vertex. We note that this choice makes the learning problem more difficult because it produces an intermediate random walk dataset that is significantly smaller than typically generated.

The embeddings are all trained using the same parameters and intermediate random walk dataset. For Euclidean embeddings we use the gensim (Rehurek and Sojka, 2010) python package, while the HyBed embeddings and our implementation of the embeddings described by Nickel and Kiela (2017) are written in TensorFlow. In both cases, we use five training epochs, a context width of five (giving 10 context vertices per input vertex) and a linearly decaying SGD optimizer with initial learning rate

Table 8.1: Experimental datasets. 'Largest class' gives the fraction of the dataset composed by the largest class and thereby provides the benchmark for random prediction accuracy.

name	—V—	—E—	—y—	largest class	Labels
karate	34	77	2	0.53	Factions
polbooks	105	441	3	0.46	Affiliation
football	115	613	12	0.11	League
adjnoun	112	425	2	0.52	Part of Speech
polblogs	1,224	16,781	2	0.52	Affiliation

of 0.2. We do not prune any vertices.

We report results on five publicly available network datasets for the problem of vertex attribution. 1. Karate: Zachary's karate club contains 34 vertices divided into two factions (Zachary, 1977); 2. Polbooks: A network of books about US politics published around the time of the 2004 presidential election and sold by the online bookseller Amazon.com. Edges between books represent frequent co-purchasing of books by the same buyers; 3. Football: A network of American football games between Division IA colleges during regular season Fall 2000 (Girvan and Newman, 2002); 4. Adjnoun: Adjacency network of common adjectives and nouns in the novel David Copperfield by Charles Dickens (Newman, 2006); 5. Polblogs: A network of hyperlinks between weblogs on US politics, recorded in 2005 (Adamic and Glance, 2005). Statistics for these datasets are recorded in Table 8.1.

The results of our experiments together with the HyBed and 2D Deepwalk embeddings used to derive them are shown in Figures 8.4 and 8.5. The vertex colours of the embedding plots indicate different values of the vertex labels. The legend shown in Figure 8.4a applies to all line graphs. The line graphs show macro F1 scores against the percentage of labelled data used to train a logistic regression classifier with the embeddings as features. Here we follow the method for generating multi-label F1 scores described in (Liu et al., 2006). The error bars show one standard error from the mean over ten repetitions. The blue lines show HyBed embeddings, the yellow lines give the 2D Poincaré embeddings of Nickel and Kiela (2017) while the red lines depict Deepwalk embeddings at various dimensions.

As we use one-vs-all logistic regression with embedding coordinates as features, good embeddings are those that can linearly separate one class from all other classes. Figures 8.4 and 8.5 shows that HyBed embeddings tend to cluster together similar classes so that they are linearly separable from other classes, unlike the Euclidean embeddings.

Table 8.2 gives the average F1 score for each experiment. The hyperbolic methods are greatly superior to Euclidean methods and HyBed is best in three of the five experiments with the highest global average F1 score. The only experiment in which the performance of Euclidean embeddings is comparable with hyperbolic methods is the Word Adjacency dataset and in this case Figures 8.5a shows that the error bars are wide and overlapping.

Table 8.2: Average F1 scores across experiments. HB is HyBed, P is Poincaré and D indicates a Deepwalk embedding. 'All' gives a global average across the five experiments.

experiment	HB	P	D2	D4	D8	D16	D32	D64	D128
karate	**0.91**	0.90	0.37	0.37	0.36	0.36	0.37	0.37	0.38
polbooks	**0.62**	0.59	0.27	0.24	0.21	0.21	0.21	0.21	0.21
football	**0.22**	0.14	0.03	0.03	0.04	0.04	0.04	0.04	0.04
adjnoun	0.49	0.47	**0.53**	0.50	0.44	0.38	0.36	0.36	0.36
polblogs	0.90	**0.91**	0.63	0.64	0.67	0.64	0.63	0.55	0.51
All	**0.63**	0.60	0.37	0.35	0.34	0.33	0.32	0.30	0.30

8.6 Summary

We have introduced HyBed embeddings, which are based on the Skipgram with negative sampling architecture and optimized in native spherical hyperbolic coordinates. Hyperbolic space has the property that power-law degree distributions, strong clustering and hierarchical community structure emerge naturally when random graphs are embedded in it. It is therefore logical to exploit the structure of the hyperbolic space to represent complex networks. We have demonstrated that when applied to the task of classifying vertices of complex networks, hyperbolic embeddings significantly outperform embeddings in Euclidean space. Furthermore, the HyBed approach we develop here performs well when compared to the most comparable method, that of Poincaré embeddings by Nickel and Kiela (2017).

A strength of our approach is its simplicity. Only a small modification of the Euclidean gradients are required to deliver many of the benefits of embeddings in hyperbolic space. For this reason our model has the same time complexity as the Euclidean case, which has been shown to be highly scalable (Mikolov et al., 2013a). However, to achieve simplicity, we give up some of the properties of hyperbolic space. Specifically, our similarity measure loses conformality of the hyperbolic metric and as a consequence closed form curvature tensors, volume elements, the metric tensor, the exponential map and geodesics. For our application, these elements are not necessary, but we lose the full machinery of hyperbolic geometry.

A limitation of our work is that we have only implemented our model in two dimensions. Embeddings in the Poicaré d-ball, instead of the disk, pose no major theoretical challenges. However, writing an efficient software implementation that is parameterised by the embeddings dimensionality has proved challenging and this is ongoing work. All graphs can be embedded in two dimensions, but larger, more complex graphs require higher dimensions to learn embeddings that are effective

in downstream vertex classification tasks (Perozzi and Skiena, 2014; Nickel and Kiela, 2017). It is for this reason that we are only able to present results on graphs of a relatively modest size. We are working to fix this problem and apply HyBed embeddings to random walks over the ASOS bipartite customer product graph to improve applications in recommender systems and the CLTV model described in Chapter 7.

Chapter 9

Conclusion

An ever increasing amount of the world's information is being stored in large graph-structured datasets. The web graph, digital social networks, e-commerce platforms and chat networks (e.g. Whatsapp and Telegram) now contain digital traces of the majority of living humans.[1] At the end of 2017, seven of the world's ten largest companies own or derive significant proportions of their revenue from large graphical information stores.[2] Extracting useful information from large graphs is a problem of great practical and commercial importance.

This research, being funded by an *Industrial Fellowship of the Royal Commission for the Exhibition of 1851*, is applicable to industrial problems within the short to medium term. For this reason we must satisfy many constraints that do not always exist in theoretical work. The first, and most significant, is scalability. In all cases, models must be applicable to graphs containing millions of vertices. Algorithms that scale as $O(n^2)$ or worse, where n is the number of vertices, are not practical in this domain. Absolute runtime is also a critical factor, either because users have an expectation from software (as in Chapter 5) or because models must be re-trained daily (as in Chapter 7). For distributable algorithms, runtimes can be reduced by scaling out the hardware, for instance using the Apache Spark middleware employed in Chapter 7. However, in this case the cost of either buying or leasing hardware must be carefully balanced against any future benefits delivered by improvements in algorithmic performance. Typically the metrics measuring algorithmic performance and business value are not the same and so the cost-benefit analysis is often challenging, and this is an issue that was addressed in Chapter 7.

[1] Data from `https://www.internetworldstats.com/stats.htm` accessed 18/2/18

[2] In Q4 2017 these companies are Apple, Alphabet, Microsoft, Amazon, Facebook, Tencent and Alibaba

Production systems must be supported for the duration of their lifetimes and additional complexity often results in increased downtime or costly maintenance. For this reason, it is often desirable to select simple system designs if the benefits provided by more complex systems are limited. The phenomena of incurring future hidden cost by implementing complex technological systems is often referred to as technical debt (Kruchten et al., 2012), which is known to be particularly acute in machine learning systems (Sculley et al., 2015). The need to support and maintain software places additional constraints on the maturity of any third party libraries or technologies that can be placed into production. Concerns such as this preclude putting pre version 1.0 software into production. This was the case, until recently, with the TensorFlow framework (Abadi et al., 2015) used in Chapters 7 and 8.

9.1 Summary of Thesis Achievements

This thesis has dealt with practical problems of representation and learning in large graphs. With the exception of the final chapter, the research has been applied to industrial systems at two British companies: Starcount Insights, a social media analytics firm and ASOS.com, an e-commerce site specialising in fashion. Here we summarise the achievements of each research chapter.

Chapter 5 described a knowledge discovery system that could replicate the functionality of laborious focus groups. We were able to use data from social media to provide instant answers to questions that are currently addressed by slow and expensive manual surveys. We showed how to explore and structure regions of a graph in real-time using global embeddings of the graph. A full end-to-end system that collected, structured and presented data was built. This system provided the backbone of the capability of Starcount Insights.

In Chapter 6, the infrastructure developed in Chapter 5 to collate large quantities of social media data was re-purposed to collect ground-truth attributes of social media users. We developed methods for predicting age, income and occupations of users based on small quantities of labelled data in a semi-supervised framework. The age prediction model operated at the scale of the full Twitter network and was used to supplement the product developed in Chapter 5 so that users could be filtered by age, or query results could be associated with age distributions. In the second half of the chapter we developed the state-of-the-art model for predicting income and occupation on Twitter by adding graph embeddings to the previously state-of-the-art linguistic model.

Chapter 7 described how we built a production customer profiling system at ASOS that, at the end of 2017, serves over 17 million active customers. While the chapter specifically focusses on the customer lifetime value model, the same basic infrastructure and modelling paradigm has been employed across multiple predictive models. These include predictive measure of customer loyalty, disposable income, churn and engagement. In this chapter we demonstrated successfully how unsupervised embeddings can be used for long-term forecasting models.

In both the knowledge discovery and customer profiling systems described in Chapters 5 and 7 respectively, we used neural embeddings of graphs as an effective way to incorporate complex graphical information into ML architectures. The achievement of Chapter 8 was a reformulation of this framework to exploit hyperbolic geometry, which has beneficial properties for representing the graphical data structures found in social network and user-product interaction graphs.

9.2 Future Work

Here we highlight a number of areas of interesting follow up work and discuss some of the limitations of our methods and how they may be addressed.

In Chapter 7, we showed that embeddings of customers learned from the customer-product interaction graph were an effective feature from which to build predictive models of customer behaviour. To do this, we built an intermediate dataset that, for each product, contained the sequence of customers who viewed it. We reasoned that the sequences contained important information about customer behaviour because customers who frequently appeared close to each other in sequences were likely to have similar properties. Specifically, customers co-located near the start of sequences share a desire for newness, while customers appearing together after a price reduction would have similar attitudes to price or discounting. However, unlike words in passages of text, which are fundamentally a sequence, the sequence of customers is really a converted time-series. Given our hypothesis that customers who view the same products close together in time are similar, including the timestamps of the interactions may improve the quality of the embeddings.

It is generally true that graphs evolve over time and that edges can be associated with timestamps. Incorporating temporal information provides a promising avenue for representation learning. In the Twitter Follower graph used in Chapter 6 for instance, because few edges are deleted, many are old and are no longer representative of the interests of Twitter users. There are two ways that

temporal information can be incorporated into the embeddings framework. (1) Weighting random walks e.g. to favour more recent edges or (2) modifying the context window at the embedding stage. The context window can be adapted by using a fixed time period instead of a fixed number of positions in the sequence. Alternatively, elements can be sampled from the sequence based on a distribution parametrised by the difference between the input and output timestamps (Baeza-Yates and Saez-Trumper, 2015).

While graph embeddings provide a computationally efficient way to learn useful representations of vertices, there are several weaknesses of the approach. The embeddings are learned via a two step process: (1) A series of random walks are taken over a graph. (2) The random walk sequences are used as input data to train a shallow neural network. However, the random walk dataset, which is parametrised by a walk length l and the number of restarts from each vertex n, can be very large. Typical values are $l = 80$ and $n = 10$ (Perozzi and Skiena, 2014; Grover and Leskovec, 2016), meaning that 800 vertex IDs must be stored per vertex in the original graph and the intermediate dataset can exceed the size of the original graph. An edge list representation of a graph $G(V, E)$ requires $2|E|$ space in units of vertex IDs and so whenever $nl|V| > 2|E|$, the intermediate dataset is larger than the input graph. This problem could be addressed by running the end-to-end process as an online algorithm, where steps in the random walk are interleaved with neural network training updates. Thus, eliminating the need to store large intermediate datasets. The second weakness relates to the efficiency of the end-to-end learning process. Neural graph embeddings lose information when they convert the input graph into sequences. The vector representations of each vertex are randomly initialised in a high-dimensional space and aspects of the graph structure are re-learned using a neural network that is trained solely on the sequences. This is motivated by the existence of highly efficient algorithms and software for learning from sequences e.g. (Rehurek and Sojka, 2010), but appears to be unnecessarily destructive. Indeed, some NLP techniques even *construct* a graph from sequences of words to use global relationships (Pennington et al., 2014). One possibility to exploit more of the information present in the original graph, is to use it to initialise the vectors in the neural network embedding layer. In many cases, nodes that should be close to each other e.g. fully connected cliques, are easily identifiable in the input graph. We have started to experiment with initialisation schemes that perform greedy initialisations by sequentially placing neighbours close to each other.

Finally we note one very promising approach for learning and representation of graphs; the emerg-

ing field of *geometric deep learning* (Monti et al., 2017a; Masci et al., 2016; Monti et al., 2017b; Bronstein et al., 2017). Geometric deep learning, of which the graph convolutional neural network (GCNN) is a major part, adapts the successful convolutional neural network framework to the problem of learning from graphs. This is achieved by altering the Euclidean convolutional filters to non-Euclidean geometries (i.e. graphs). Early results indicate that this methodology may be superior for extracting information from graphs than embeddings based on random walks (Kipf and Welling, 2016). However, it should be noted that comparisons such as those by Kipf and Welling (2016) are not truly like for like because the GCNNs are trained with supervision, while graph embeddings are fully unsupervised and so can be applied to multiple tasks without re-training. We did not apply GCNNs in our work because at the time our research was being conducted, the technique was not sufficiently mature to be applicable within a commercial setting to the types of problems that require frequent retraining.

9.3 Applications

In most cases, this research has been applied in industry, and here we expand on some of the uses. The work described in Chapters 5 and 6 is part of a single system used at Starcount Insights to analyse markets and customer groups based on social media data. As a concrete example, we consider a case study where an international beverage company wants to explore the market (competitors, customers, associations etc.) around their brand. They feed in information about themselves in the form of the social media handles of their brands, main employees and endorsers. They are only interested in users who are over 18 and of middle to high income and so they apply filters based on our user attribution models. Our system finds accounts that are similar/related to the seeds and then structures the similar accounts into communities. The communities can be profiled using any attributes that can be learned by the method described in Chapter 6 i.e. any attribute with reasonable support in the profile description data. The output is then used to identify new marketing channels or customer segments.

The applications of the work in Chapters 5 and 6 are not limited to the activities of Starcount Insights and extend into many additional domains. In the field of national security it is common that hostile groups interact through public social media networks. The ability to understand and rapidly identify communities related to groups of known adversaries is an important task.[3] For large e-commerce

[3]while no reference is available, the author was previously employed in an organisation working on this problem.

sites, the customer-product interaction graph can be too large to manually inspect. In these cases, we may wish to discover communities related to groups of individuals or products. Communities in the customer-product graph indicate product clusters or customer segments. Customer segments are important in identifying new groups and providing personalised services to meet their needs. Product clusters are used to identify new product offering and understand the level of sales cannibalisation from adding new product lines.

Large scale social media platforms are usually provided free of charge because revenue is generated by selling targeted advertising. However, treating users in this way can damage the reputations of platforms or the user experience (Allcott and Gentzkow, 2017; Hoadley et al., 2010; Story and Stone, 2007). An important element of our work is that it uses the information in the network, without damaging the products and services that the networks provide. Thus it offers an alternative revenue stream for social media providers.

In Chapter 7, we described the CLTV prediction system deployed at ASOS. The underlying model and system architecture is applied to learn a larger number of customer attributes that allow ASOS to build rich profiles of customers. The profile information is then fed into downstream models (e.g. the recommender systems) or used directly to personalise a customer's interactions on the website. One of the most important uses is in marketing and we conclude by providing more detail in this area. ASOS engages in many marketing channels, but not every channel has the capacity to associate marketing spend with individual customers. In *programmatic marketing* this association can be made using a cookie ID. Programmatic involves real-time bidding for advertising space to be shown to a customer based on a profile associated with their cookie ID. At ASOS, the customer CLTV model is used to modify the bidding strategy for advertising, with increased spend being allocated to customers who are predicted to be of higher value. The churn model is closely related to the CLTV model. It estimates the likelihood of a customer making a purchase again in the next twelve months. We use the churn model to monitor the churn risk of our customer base and take mitigating actions when we believe a customer is likely to churn. One strategy involves sending a discount coupon to customers who's churn risk rises beyond a threshold level, providing certain additional conditions are met. The CLTV model is also useful for finding new customers through a process called *lookalike marketing*. Lookalike marketing is a form of data mining where an external dataset is mined for correlations with a desirable attribute which, in this case, is high predicted lifetime value. External datasets may include credit reference data, Google search terms or Facebook user profiles. Lookalike

marketing allows marketing spend to be redistributed to target certain channels, geographic regions or Google keywords where high value customers are likely to be found.

Finally, our research into hyperbolic embeddings in Chapter 8 is an iteration on the embedding model described in Chapter 7, and so is associated with the same set of applications. However, the applications are much broader as the model has the capacity to improve vertex attribute learning across any graph-structured data.

Bibliography

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. *TensorFlow: large-scale machine learning on heterogeneous distributed systems.* Technical Report (2015), 19 pages.

Lada A Adamic and Natalie Glance. 2005. The political blogosphere and the 2004 U.S. election. *Proceedings of the Third International Workshop on Link Discovery* (2005), 36–43.

Micah Adler and Michael Mitzenmacher. 2001. Towards compressing web graphs. *Proceedings of the Data Compression Conference* (2001), 203–212.

Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* (2002), 74(1) 47.

Nikolaos Aletras and Benjamin P Chamberlain. 2017. Predicting socioeconomic attributes with network and language information. (2017),

Hunt Allcott and Matthew Gentzkow. 2017. Social media and fake news in the 2016 election. *Journal of Economic Perspectives* 31, 2 (2017), 211–236.

Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), 1018–1028.

Ricardo Baeza-Yates and Diego Saez-Trumper. 2015. Wisdom of the crowd or wisdom of a few? *Proceedings of the 26th ACM Conference on Hypertext & Social Media* (2015), 69–74.

Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. 2011. Fast personalized pagerank on mapreduce. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (2011), 973–984.

Oren Barkan and Noam Koenigstein. 2016. Item2Vec : neural item embedding for collaborative filtering. *The 26th International Workshop on Machine Learning for Signal Processing* (2016),

Marco Baroni, Georgiana Dinu, and German Kruszewski. 2014. Don't count , predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (2014), 238–247.

Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. *Proceedings of the Third International AAAI Conference on Weblogs and Social Media* (2009), 361–362.

Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data* 4.3 (2010), 13.

Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems* (2001), 585–591.

Albert C Bemmaor and Nicolas Glady. 2012. Modeling purchasing behavior with sudden death: a flexible customer lifetime model. *Management Science* 58, 5 (5 2012), 1012–1021.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research* 3 (2003), 1137–1155.

Derek Bird. 2004. Methodology for the 2004 annual survey of hours and earnings. *Labour Market Trends* 112 (2004), 457–464.

Christopher M Bishop. 2016. Pattern recognition and machine learning. *Springer-Verlag New York*

Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. 2016. Efficient embedding of scale-free graphs in the hyperbolic plane. *LIPIcs-Leibniz International Proceedings in Informatics* 57 16 (2016).

Vincent D Blondel, Jean-loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of community hierarchies in large networks. *Journal of Statistical Mechanics: Theory and Experiment* (2008), 10008.

Marian Boguna, Fragkiskos Papadopoulos, and Dmitri Krioukov. 2010. Sustaining the Internet with hyperbolic mapping. *Nature Communications* 1, 62 (2010), 62.

P Boldi and S Vigna. 2004. The webgraph framework I: compression techniques. In *Proceedings of the 13th International Conference on World Wide Web* (2004), 595–602.

Leo Breiman. 1996a. Bagging predictors. *Machine Learning* 24, 421 (1996), 123–140.

Leo Breiman. 1996b. Heuristics of instability in model selection. *The Annals of Statistics* 24, 6 (1996), 2350–2383.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. Classification and regression trees (1984), *CRC Press*

Eric Brochu, Vlad M Cora, and Nando de Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

Andrei Z Broder. 1997. On the resemblance and containment of documents. *IEEE Proceedings of Compression and Complexity of Sequences*. 21–29.

Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-wise independent permutations. *Journal of Computer and System Sciences* 60, 3 (2000), 630–659.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

John A Bullinaria and Joseph P Levy. 2012. Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD. *Behavior Research Methods* 44, 3 (2012), 890–907.

Ed Bullmore and Olaf Sporns. 2009. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience* 10.3 (2009), 186–198.

U.S. Census Bureau. 2014. Grandparent statistics. (2014).

John D Burger, John Henderson, George Kim, and Guido Zarrella. 2011. Discriminating gender on Twitter. *Association for Computational Linguistics* 146 (2011), 1301–1309.

Benjamin P Chamberlain, Angelo Cardoso, C H Bryan Liu, Roberto Pagliari, and Marc P Deisenroth. 2017a. Customer lifetime value prediction using embeddings. *Proceedings of the 23nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), 1753–1762. @inproceedingsLiu2017, archivePrefix = arXiv, arxivId = 1706.09865, author = Liu, C. H. Bryan and Chamberlain, Benjamin Paul and Little, Duncan A. and Cardoso, Angelo, booktitle = Joint European Conference on Machine Learning and Knowledge Discovery in Databases ECML-PKDD, eprint = 1706.09865, file = :Users/ben.chamberlain/Downloads/ecml-2017-generalising.pdf:pdf, keywords = bayesian optimisation,machine learning application,model stability,parameter tuning,random forest, mendeley-groups = thesis, title = Generalising Random Forest Parameter Optimisation to Include Stability and Cost, url = http://arxiv.org/abs/1706.09865, year = 2017

C H Bryan Liu, Benjamin P Chamberlain, Duncan A Little, and Angelo Cardoso. 2017b. Generalising Random Forest Parameter Optimisation to Include Stability and Cost. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases ECML-PKDD* (2017).

Benjamin P Chamberlain, James R Clough, and Marc P Deisenroth. 2017b. HyBed: hyperbolic neural graph embedding. *The 13th International Workshop on Mining and Learning with Graphs* (2017).

Benjamin P Chamberlain, Clive Humby, and Marc P Deisenroth. 2017c. Probabilistic inference of Twitter users' age based on what they follow. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases ECML-PKDD* (2017), 191–203.

Benjamin P Chamberlain, Josh Levy-Kramer, Clive Humby, and Marc P Deisenroth. 2018. Real-time community detection in large social networks on a laptop. *PloS One* 13, 1 (2018), e0188702.

Moses S Charikar and Konstantin Makarychev. 2010. Local global tradeoffs in metric embeddings. *SIAM Journal on Computing* 39, 6 (2010), 2487–2512.

Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*. 380–388.

Rishan Chen, Xuetian Weng, Bingsheng He, and Mao Yang. 2010. Large graph processing in the cloud. In *Proceedings of the ACM SIGMOD International Conference on Management of data* (2010), 1123–1126.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong,

Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. *Proceedings of the First Workshop on Deep Learning for Recommender Systems. ACM* (2016), 7–10.

Zhiyuan Cheng, James Caverlee, and Kyumin Lee. 2010. You are where you tweet : a content-based approach to geo-locating Twitter users. *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (2010), 759–768.

Flavio Chierichetti, Ravi Kumar, and Silvio Lattanzi. 2009. On compressing social networks. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009), 219–228.

Aaron Clauset. 2005. Finding local community structure in networks. *Physical Review E* 72.2 (2005), 026132.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark E J Newman. 2007. Power-law distributions in empirical data. *SIAM review* (2007) 51.4 , 661-703.

James R Clough and Tim S Evans. 2016. What is the dimension of citation space? *Physica A: Statistical Mechanics and its Applications* 448 (2016), 235–247.

James R Clough, Jamie Gollings, Tamar V. Loach, and Tim S Evans. 2015. Transitive reduction of citation networks. *Journal of Complex Networks* 3, 2 (2015), 189–203.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing. *Proceedings of the 25th International Conference on Machine Learning* 20, 1 (2008), 160–167.

Paul Covington, Jay Adams, and Emre Sargin. 2016b. Deep neural networks for YouTube recommendations. *Proceedings of the Tenth ACM Conference on Recommender Systems* (2016), 191–198.

Antonio Criminisi. 2011. Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision* 7, 2-3 (2011), 81–227.

Aron Culotta, Nirmal Kumar Ravi, and Jennifer Cutler. 2015. Predicting the demographics of Twitter users from website traffic data. *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), 72–78.

Andrej Cvetkovski and Mark Crovella. 2009. Hyperbolic embedding and routing for dynamic graphs. *IEEE INFOCOM* (2009), 1647–1655.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems* (2016), 3844–3852.

Pedro Domingos. 2012. A few useful things to know about machine learning. *Communications of the ACM* 55, 10 (2012), 78–87.

Harris Drucker, Chris J C Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. 1997. Support vector regression machines. *Advances in Neural Information Processing Systems* (1997), 155–161.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent marked temporal point processes : embedding event history to vector. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* (2016), 1555–1564.

Nan Du, Yichen Wang, Niao He, and Le Song. 2015. Time-sensitive recommendation from recurrent user activities.. *Advances in Neural Information Processing Systems* 1 (2015), 3492–3500.

Maeve Duggan and Joanna Brenner. 2013. The demographics of social media users. *Consultado en*

Peter Elias and Margaret Birch. 2010. SOC2010: revision of the standard occupational classification. *Economic and Labour Market Review* 4, 7 (2010), 48–55.

Andre Elisseeff, Theodoros Evgeniou, and Massimiliano Pontil. 2005. Stability of randomized learning algorithms. *Journal of Machine Learning Research* 6, 1 (2005), 55–79.

Paul Erdos and Alfred Renyi. 1960. On the evolution of random graphs. *Public Mathethmatics Institute Hungarian Academy of Science* 5, 1 (1960), 17–60.

Peter S Fader, Bruce G S Hardie, and Ka Lok Lee. 2005a. Counting your customers? The easy way: an alternative to the Pareto/NBD model. *Marketing Science* 24, 2 (2005), 275–284.

Peter S Fader, Bruce G S Hardie, and Ka Lok Lee. 2005b. RFM and CLV: using iso-value curves for customer base analysis. *Journal of Marketing Research* XLII (2005), 415–430.

Quan Fang, Jitao Sang, Changsheng Xu, and M. Shamim Hossain. 2015. Relational user attribute inference in social media. *IEEE Transactions on Multimedia* 17, 7 (2015), 1031–1044.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Dinani Amorim, and Dinani Amorim Fernández-Delgado. 2014. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research* 15 (2014), 3133–3181.

Renato Miranda Filho, Guilherme R Borges, Jussara M Almeida, and Gisele L Pappa. 2014. Inferring user social class in online social networks. In *Proceedings of the Eighth Workshop on Social Network Mining and Analysis* (2014), 1–5.

Gary William Flake, Steve Lawrence, and C Lee Giles. 2000. Efficient identification of web communities. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2000), 150–160.

Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3 (2010), 75–174.

Santo Fortunato and Marc Barthelemy. 2007. Resolution limit in community detection. *Proceedings of the National Academy of Sciences* 104.1 (2007), 36–41.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001a. The elements of statistical learning. *New York: Springer Series in Statistics* 1, 1 (2001).

Yun Fu, Guodong Guo, and Thomas S Huang. 2010. Age synthesis and estimation via faces: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2010), 1955–1976.

David Gibson, Andrew Tomkins, and San Jose. 2005. Discovering large dense subgraphs in massive graphs. *Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment*, (2005), 721–732.

Michelle Girvan and Mark E J Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99 (2002), 7821–7826.

David F Gleich and C Seshadhri. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), 597–605.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*. Johns Hopkins University Press, Baltimore. 374–426 pages.

Gerald J Goodhart, Andrew SC Ehrenberg, and Christopher Chatfield. 1984. The Dirichlet: a comprehensive model of buying behaviour.. *Journal of the Royal Statistical Society, Series A* 37, 1 (1984), 621–655.

Trey Grainger, Timothy Potter, and Yonik Seeley. 2014. *Solr in action. Cherry Hill: Manning*

Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in your inbox : product recommendations at scale categories and subject descriptors. *SIGKDD 2015: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), 1809–1818.

Mikhail Gromov. 2007. *Metric Structures for Riemannian and Non-riemannian Spaces. Springer Science and Business Media* (2007).

Aditya Grover and Jure Leskovec. 2016. node2vec : scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 855–864.

Guo Guodong, Yun Fu, Charles R Dyer, and Thomas S Huang. 2008. Image-based human age estimation by manifold learning and locally adjusted robust regression. *IEEE Transactions on Image Processing* (2008). 1178-1188

Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. WTF: the who to follow service at Twitter. *Proceedings of the 22nd International Conference on World Wide Web* (2013), 505–514.

Michael U Gutmann, Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13 (2012), 307–361.

Sunil Gupta, Dominique Hanssens, Bruce Hardie, Nathaniel Lin, V Kumar, Nalini Ravishanker, S Sriram, and William Kahn. 2006. Modeling customer lifetime value. *Journal of Service Research* 9, 2 (2006), 139–155.

Taher H Haveliwala. 2002. Topic-sensitive PageRank. *Proceedings of the 11th International Conference on World Wide Web* (2002), 517–526.

Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015),

James Hensman, Alex Matthews and Zoubin Ghahramani. 2015. Scalable variational Gaussian process classification. *Journal of Machine Learning Research* 38 (2015), 351–360.

Thomas T Hewett, Ronald Baecker, Stuart Card and Tom Carey. 1992. *ACM SIGCHI curricula for human-computer interaction* (1992), ACM Press.

Geoffrey Hinton, James L McClelland, and David E Rumelhart. 1990. Distributed representations. *The philosophy of artificial intelligence* (1990), 248–280.

Geoffrey Hinton 1986. Learning distributed representations of concepts. *Proceedings of the eighth annual conference of the cognitive science society* (1986), 1–12.

Christopher M. Hoadley, Heng Xu, Joey J. Lee, and Mary Beth Rosson. 2010. Privacy as information access and illusory control: The case of the Facebook news feed privacy outcry. *Electronic Commerce Research and Applications* 9, 1 (2010), 50–60.

Petter Holme, Sung Min Park, Beom Jun Kim, and Christofer R. Edling. 2007. Korean university life in a network perspective: dynamics of a large affiliation network. *Physica A: Statistical Mechanics and its Applications* 373 (2007), 821–830.

Tianran Hu, Haoyuan Xiao, Thuy-vy Thi Nguyen, and Jiebo Luo. 2016. What the language you tweet says about your occupation. In *Proceedings of the Tenth International AAAI Conference on Web and Social Media* (2016), 181–190.

Barbara F F Huang and Paul C Boutros. 2016. The parameter sensitivity of random forests. *BMC bioinformatics* 17, 1 (2016), 331.

Yanxiang Huang, Lele Yu, Xiang Wang, and Bin Cui. 2015. A multi-source integration framework for user occupation inference in social media systems. *World Wide Web* 18, 5 (9 2015), 1247–1267.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing* (1998), Vol. 126. 604–613.

Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. 2014. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PLoS ONE* 9, 6 (6 2014), e98679.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: convolutional architecture for fast feature embedding. *Proceedings of the 22nd ACM International Conference on Multimedia* (2014), 675–678

Thorsten Joachims. 1998. Text categorization with support vector machines: learning with many relevant features. *European Conference on Machine Learning. Springer, Berlin, Heidelberg* 1398, LS-8 Report 23 (1998), 137–142.

Donald R Jones. 2001. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization* 21 (2001), 345–383.

Micha l Karoński, Edward R. Scheinerman, and Karen B. Singer-Cohen. 1999. On random intersection graphs: the subgraph problem. *Combinatorics, Probability and Computing* 8, 1999 (1999), 131–159.

Myunghwan Kim and Jure Leskovec. 2013. Nonparametric multi-group membership model for dynamic networks. *Advances in Neural Information Processing Systems* (2013), 1385–1393.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *preprint arXiv:1609.02907* (2016).

Robert D Kleinberg. 2007. Geographic routing using hyperbolic space. *Proceeding of 26th IEEE International Conference on Computer Communications* (2007), 1902–1909.

Isabel M Kloumann and Jon M Kleinberg. 2014. Community membership identification from small seed sets. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1366–1375.

Tamara G. Kolda and Brett W. Bader. 2008. Tensor Decompositions and Applications. *SIAM Review* 51, 3 (2008), 455–500.

Farshad Kooti, Mihajlo Grbovic, Luca Maria Aiello, Eric Bax, and Kristina Lerman. 2017. iPhone's digital marketplace: characterizing the big spenders. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (2017), 13–21.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

Michal Kosinski, David Stillwell, and Thore Graepel. 2013. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences of the United States of America* 110, 15 (4 2013), 5802–5805.

Daniel G Krige. 1952. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa* (1952), 201–215.

Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, and Amin Vahdat. 2010. Hyperbolic geometry of complex networks. *Physical Review E* 82, 3 (2010), 036106.

Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: from metaphor to theory and practice. *IEEE Software* vol. 29, 6 (2012), 18–21.

Harold J Kushner. 1964. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering* 86, 1 (1964), 97.

Matt J Kusner, Yu Sun, Nicholas I Kolkin, and Kilian Q Weinberger. 2015. From word embeddings to document distances. *Proceedings of the 32nd International Conference on Machine Learning* 37 (2015), 957–966.

Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. Graphchi: Large-scale graph computation on just a PC. *Symposium on Operating Systems Design and Implementation* (2012), 31–46.

Vasileios Lampos, Nikolaos Aletras, Jens K Geyti, Bin Zou, and Ingemar J Cox. 2016. Inferring the socioeconomic status of social media users based on behaviour and language. *European Conference of Information Retrieval* (2016), 689–695.

Andrea Lancichinetti, Santo Fortunato, and Janós Kertész. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics* 11.3, 033015 (2009).

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *International Conference on Machine Learning* 32 (2014), 1188–1196.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

Daniel D Lee and H Sebastian Seung. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401, 6755 (1999), 788–91.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007).

Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2008. Statistical properties of community structure in large social and information networks. *Proceedings of the 17th International Conference on World Wide Web* (2008), 695–704.

Jure Leskovec, Kevin J. Lang, and Michael W. Mahoney. 2010. Empirical comparison of algorithms for network community detection. *Proceedings of the 19th International Conference on World Wide Web* (4 2010), 631–640.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in Neural Information Processing Systems* (2014), 2177–2185.

Aaron Q Li, Amr Ahmed, Sujith Ravi, and Alexander J Smola. 2014a. Reducing the sampling complexity of topic models. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014), 891–900.

Kang Li, Jing Gao, Suxin Guo, Nan Du, Xiaoyi Li, and Aidong Zhang. 2014b. LRBM: A restricted Boltzmann machine based approach for representation learning on linked data. *2014 IEEE International Conference on Data Mining* (2014), 300–309.

Li Ping and Christian König. 2009. b-bit minwise hashing. *Proceedings of the 19th International Conference on World Wide Web* (2009), 671–680.

Yixuan Li, Kun He, David Bindel, and John E. Hopcroft. 2015. Uncovering the small community structure in large networks: a local spectral approach. *Proceedings of the 24th International Conference on World Wide Web* (2015), 658–668.

David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology* 58, 7 (5 2007), 1019–1031.

Seppo Linnainmaa. 1970. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki* (1970).

Guimei Liu, Tam T Nguyen, Gang Zhao, Wei Zha, and Jianbo Yang. 2016. Repeat buyer prediction for e-commerce. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM* (2016), 155–164.

Wendy Liu and Derek Ruths. 2013. What's in a Name? Using first names as features for gender inference in Twitter. *Analyzing Microtext: Papers from the 2013 AAAI Spring Symposium* (2013), 10–16.

Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. 2017. SphereFace: Deep Hypersphere Embedding for Face Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*

Yi Liu, Rong Jin, and Liu Yang. 2006. Semi-supervised multi-label learning by constrained non-negative matrix factorization. *Proceedings of the AAAI Conference on Artificial Intelligence* (2006), Vol. 6 421–426.

Yucheng Low, Joseph E Gonzalez, and Aapo Kyrola. 2014. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* (2014).

David Lusseau. 2003. The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London B: Biological Sciences* 270(Suppl2 (2003), S186–S188.

Xia Lv, Peiquan Jin, and Lihua Yue. 2016. User occupation prediction on microblogs. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9932 LNCS. 497–501.

Grzegorz Malewicz, Matthew H Austern, Aart J.C Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the International Conference on Management of data*. 135–146.

Mark Manasse and Frank Mcsherry. 2010. Consistent weighted sampling. Technical Report (2010).

Jonathan Masci, Emanuele Rodolà, Davide Boscaini, Michael M. Bronstein, and Hao Li. 2016. Geometric deep learning. *SIGGRAPH ASIA 2016 Courses on - SA '16*. 1–50.

H Brendan McMahan, Gary Holt, D Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: a view

from the trenches. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM* 1222–1230.

Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: homophily in social networks. *Annual Review of Sociology* 27, 1 (8 2001), 415–444.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems.* 3111–3119.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013),

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes* 6, 1 (1 1991), 1–28.

Tom P Minka. 2001. Expectation propagation for approximate Bayesian inference. *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence.* (2001). 362–369

Alan Mislove, Sune Lehmann, and Yong-Yeol Ahn. 2011. Understanding the demographics of Twitter users. *International Conference on Weblogs and Social Media* (2011), 11.5th: 25

Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. 2016. A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137* (2016).

Michael Mitzenmacher. 2004. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics* (2004), 226–251

Michael Mitzenmacher, Rasmus Pagh, and Ninh Pham. 2014. Efficient estimation for high similarities using odd sketches. *Proceedings of the 23rd International Conference on World Wide Web* (2014), 109–118.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. *Advances in Neural Information Processing Systems* (2013), 2265–2273.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning* (2007), 641–648.

Andriy Mnih and Geoffrey Hinton. 2009. A Scalable hierarchical distributed language model. *Advances in Neural Information Processing Systems* (2008), 1081–1088.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *Proceedings of the 29th International Conference on Machine Learning* (2012), 1751–1758.

Jonas Mockus. 1974. On Bayesian methods for seeking the extremum. *IFIP Technical Conference on Optimization Techniques* (1974), 400–404.

Jonas Mockus, V. Tiesis, and Antanas Zilinskas. 1978. Toward global optimization. *Bayesian Methods for Seeking the Extremum* (1978) 117–128.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. 2017a. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *The IEEE Conference on Computer Vision and Pattern Recognition* (2016), 1 3.

Federico Monti, Michael M. Bronstein, and Xavier Bresson. 2017b. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in Neural Information Processing Systems*, (2017) 3700–3710.

Federico Monti, Karl Otness, and Michael M. Bronstein. 2018. MotifNet: a motif-based graph convolutional network for directed graphs. *arXiv preprint arXiv:1802.01572* (2018).

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics* (2005), 246–252.

Rajeev Motwani, Assaf Naor, and Rina Panigrahy. 2005. Lower bounds on locality sensitive hashing. *SIAM Journal on Discrete Mathematics* (2005), 930–935.

Kevin P Murphy. 2014. Machine learning, a probabilistic perspective. *MIT Press*

Radford M Neal. 1995. Bayesian Learning for neural networks. *PhD Dissertation.*

Mark E J Newman. 2003. The structure and function of complex networks. *SIAM Review* 45.2 (2003), 167–256.

Mark E J Newman. 2004a. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems* 38.2 (2004), 321–330.

Mark E J Newman. 2004b. Fast algorithm for detecting community structure in networks. *Physical Review E* 69(6), 066133 (2004).

Mark E J Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics* (2005), 323–351.

Mark E J Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 74, 3 (2006), 036104.

Dong Nguyen, Rilana Gravel, Dolf Trieschnigg, and Theo Meder. 2013. "How old do you think I am?" A study of language and age in Twitter.. In *International Conference on Weblogs and Social Media* (2013), 439–448.

Dong Nguyen, Noah Smith, and Corolyn P Rosé. 2011. Author age prediction from text using linear regression. *LaTeCH '11 Proceedings of the Fifth ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities* June (2011), 115–123.

Maximilian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. *Advances in Neural Information Processing Systems* (2017), 6341–6350

Ryan O'Donnell, Yi Wu, and Yuan Zhou. 2009. Optimal lower bounds for locality sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory* 6, 1 (2009), 9.

Huseyin Oktay, Aykut Firat, and Zeynep Ertem. 2014. Demographic breakdown of Twitter users: an analysis based on names. *Academy of Science and Engineering* (2014), 1–11.

Matthew Felice Pace. 2012. BSP vs MapReduce. *Procedia Computer Science* 9 (2012), 246–255.

Lawrence Page, Sergei Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the Web. Technical Report.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2012. Scikit-learn: Machine Learning in Python. *Journal of machine learning research* 12 (2011), 2825–2830.

Marco Pennacchiotti and Ana-maria Popescu. 2011. A machine learning approach to Twitter user classification. *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media A* (2011), 281–288.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (2014), 1532–1543.

Bryan Perozzi and Steven Skiena. 2014. DeepWalk : online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM* (2014), 701–710.

James Philbin. 2008. Near duplicate image detection : min-hash and tf-idf weighting. *Proceedings of the British Machine Vision Conference* 810 (2008), 812–815.

Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. *Computer and Information Sciences* (12 2005), 284–293.

Daniel Preot, Vasileios Lampos, and Nikolaos Aletras. 2015. An analysis of the user occupational class through Twitter content. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the Seventh International Joint Conference on Natural Language Processing* (2015), 1754–1764.

Daniel Preoţiuc-Pietro, Svitlana Volkeva, Vasileios Lampos, Yoram Bachrach, and Nikolaos Aletras. 2015. Studying user income through language, behaviour and affect in social media. *PloS One* 10, 9 (2015), 10(9), e0138717.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (9 2007), 036106.

Tapani Raiko, Harri Valpola, and Yann LeCun. 2012. Deep learning made easier by linear transformations in perceptrons. *Artificial Intelligence and Statistics* 22, 1 (2012), 924–932.

Rao C Radhakrishna. 1992. Information and the accuracy attainable in the estimation of statistical parameters. In *Breakthroughs in Statistics, Springer* (1992), 235–247.

Delip Rao, David Yarowsky, Abhishek Shreevats, and Manaswi Gupta. 2010. Classifying latent user attributes in Twitter. *Proceedings of the Second International Workshop on Search and Mining User-Generated Contents* (2010), 37–44.

Carl E Rasmussen and Chistopher KI Williams. 2006. Gaussian processes for machine learning. *MIT Press* (2006).

Radim Rehurek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (2010), 45–50.

Jörg Reichardt and Stefan Bornholdt. 2006a. Statistical mechanics of community detection. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 74 (2006).

Jörg Reichardt and Stefan Bornholdt. 2006b. Statistical mechanics of community detection. *Physics Review E* 74, 1 (2006), 016110.

Martin Rosvall and Carl T Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105.4 (2008), 1118–1123.

Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science, New Series* 290, 5500 (2000), 2323–2326.

Herbert Rubenstein and John B. Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM* 8, 10 (1965), 627–633.

Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM* 18, 11 (1975), 613–620.

Samuel F Sampson. 1969. *Crisis in a cloister.* Ph.D. Dissertation. Cornell University, Ithaca.

Jonathan Schler, Moshe Koppel, Shlomo Argamon, and James Pennebaker. 2006. Effects of age and gender on blogging. *Proceedings of AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs* (2006), 199–205.

David C Schmittlein, Donald G Morrison, and Richard Colombo. 1987. Counting your customers: who are they and what will they do next? *Management Science* 33, 1 (1987), 1–24.

Bernhard. Schölkopf and Alexander J Smola. 2002. *Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press* (2002).

D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems* (2015), 2494–2502.

Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and J C Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 421–434.

Yilun Shang. 2010. Degree distributions in general random intersection graphs. *Electronic Journal of Combinatorics* 17, 1 (2010).

Yilun Shang. 2012. Joint probability generating function for degrees of active/passive random intersection graphs. *Frontiers of Mathematics in China* 7, 1 (2012), 117–124.

Yilun Shang. 2015. Deffuant model of opinion formation in one-dimensional multiplex networks. *Journal of Physics A: Mathematical and Theoretical* 48, 39 (2015), 395101.

Yuval Shavitt and Tomer Tankel. 2008. Hyperbolic embedding of Internet graph for distance estimation and overlay construction. *IEEE/ACM Transactions on Networking* 16, 1 (2008), 25–36.

Luke Sloan. 2017. Who Tweets in the United Kingdom? Profiling the Twitter population using the British social attitudes survey 2015. *Social Media + Society* 3, 1 (3 2017).

Luke Sloan, Jeffrey Morgan, Pete Burnap, and Matthew Williams. 2015. Who tweets? Deriving the demographic characteristics of age, occupation and social class from Twitter user meta-data. *PloS One* (2015).

Edward Snelson and Zoubin Ghahramani. 2006. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems* (2006), 1257–1264

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian optimization of machine learning algorithms.. *Advances in Neural Information Processing Systems* (2012), 2951–2959.

Dan Spielman. 2007. Spectral graph theory and its applications. *The 48th Annual IEEE Symposium on Foundations of Computer Science* (2007), 29–38.

Niranjan Srinivas, Andreas Krause, and Matthias Seeger. 2010. Gaussian process optimization in the bandit setting : no regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning* (2010), 1015–1022.

Harald Steck. 2015. Gaussian ranking by matrix factorization. *Proceedings of the Ninth ACM Conference on Recommender Systems* (2015), 115–122.

Louise Story and Brad Stone. 2007. Facebook Retreats on Online Tracking. *The New York Times* (2007), 3 pages.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems* (2014), 3104–3112.

Ali Tamaddoni, Stanislav Stakhovych, and Michael Ewing. 2016. Comparing churn prediction techniques and assessing their performance: a contingent perspective. *Journal of Service Research* 19, 2 (2016), 123–141.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: large-scale information network embedding. *Proceedings of the 24th International Conference on World Wide Web* (2015), 1067–1077.

Jeffrey Travers and Stanley Milgram. 1967. The Small-World Problem. *Psychology Today* 1, 1 (1967), 61–67.

U.S. Census Bureau. 2010. Profile of general population and housing characteristics: 2010. (2010).

Vin de Silva and Joshua B Tenenbaum. 2003. Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems 15* (2003), 721–728.

Laurens J P Van Der Maaten and Geoffrey Hinton. 2008. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

Ali Vanderveld and Angela Han. 2016. An engagement-based customer lifetime value system for e-commerce. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 293–302. .

Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec - product embeddings using side-information for recommendation. *Proceedings of the Tenth ACM Conference on Recommender Systems* (2016), 225–232.

Deepak Verma and Marina Meila. 2003. *A comparison of spectral clustering algorithms.* Technical Report. 1–18 pages.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning* (2008), 1096–1103.

Artit Wangperawong, Cyrille Brun, and Rujikorn Pavasuthipaisit. 2016. Churn analysis using deep convolutional neural networks and autoencoders. (2016), *arXiv preprint arXiv:1604.05377*

Rafal Wysocki and Wojciech Zabierowski. 2011. Twisted framework on game server example. *Proceedings of the Eleventh International Conference The Experience of Designing and Application of CAD Systems in Microelectronics* (2011), 361–363.

Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. ACM (2013), 587–596.

Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42.1 (2015), 181–213.

Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research* 33, 4 (1977), 452–473.

Bianca Zadrozny and Charles Elkan. 2001. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proceedings of the 18th International Conference on Machine Learning* (2001), 609–616.

Matei Zaharia and Mosharaf Chowdhury. 2010. Spark: cluster computing with working sets. *HotCloud* (2010).

Faiyaz Al Zamal, Wendy Liu, and Derek Ruths. 2012. Homophily and latent attribute inference: inferring latent attributes of Twitter users from neighbors. *Proceedings of the Sixth International AAAI Conference on Weblogs and Social Media Homophily* (2012), 387–390.

Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized PageRank on dynamic graphs. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM* 1 (2016), 1315–1324.

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology),* 67, 2 (2005), 301–320.