NOVA IMS Information Management School

MEGI

Mestrado em Estatística e Gestão de Informação
Master Program in Statistics and Information Management

Mobile App Recommendations Using Deep Learning and Big Data

Author: Luís António Galego Pinto

Advisor: Prof. Roberto Henriques

Dissertation presented as partial requirement for obtaining the Master's degree in Statistics and Information Management.

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

MOBILE APP RECOMMENDATIONS USING DEEP LEARNING AND BIG DATA

by

Luís Pinto

Dissertation presented as partial requirement for obtaining the Master's degree in Statistics and Information Management, with a specialization in Marketing Research and CRM.

Advisor: Prof. Roberto Henriques

ABSTRACT

Recommender systems were first introduced to solve information overload problems in enterprises. Over the last decades, recommender systems have found applications in several major websites related to e-commerce, music and video streaming, travel and movie sites, social media and mobile app stores. Several methods have been proposed over the years to build recommender systems. The most popular approaches are based on collaborative filtering techniques, which leverage the similarities between consumer tastes. But the current state of the art in recommender systems is deep-learning methods, which can leverage not only item consumption data but also content, context, and user attributes. Mobile app stores generate data with Big Data properties from app consumption data, behavioral, geographic, demographic, social network and user-generated content data, which includes reviews, comments and search queries. In this dissertation, we propose a deep-learning architecture for recommender systems in mobile app stores that leverage most of these data sources. We analyze three issues related to the impact of the data sources, the impact of embedding layer pretraining and the efficiency of using Kernel methods to improve app scoring at a Big Data scale. An experiment is conducted on a Portuguese Android app store. Results suggest that models can be improved by combining structured and unstructured data. The results also suggest that embedding layer pretraining is essential to obtain good results. Some evidence is provided showing that Kernel-based methods might not be efficient when deployed in Big Data contexts.

KEYWORDS

Spark; Tensorflow; Social Networks; Machine Learning.

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

3Vs	Refers to the three properties of Big Data: Volume, Velocity and Variety.
ALS	Alternating Least Squares.
ANN	Artificial Neural Network.
ARPU	Average Revenue per User.
ASO	App Store Optimization.
AUC	Area Under the Receiver Operating Curve.
BB	Beta-Binomial.
BG	Beta-Geometric.
CBR	Case-Based Reasoning.
CENE	Content-Enhanced Network Representation Learning.
CF	Collaborative Filtering.
CLV	Customer Lifetime Value.
CRM	Customer Relationship Management.
CTR	Click Through Rate.
DMD	Dirichlet Multinomial Distribution.
DNC	Differentiable Neural Computer.
DNN	Deep Neural Network.
E-Commerce	Electronic Commerce.
FC	Fully Connected.
GP-GPU	General Purpose Graphics Processing Unit.
HDFS	Hadoop Distributed File System.
IS	Information Systems
IT	Information Technology
KLR	Kernel Logistic Regression.
KNN	K-Nearest Neighbors.
KS	Kolmogorov-Smirnov.

LDA	Linear Discriminant Analysis.
LinReg	Linear Regression.
LR	Logistic Regression.
LSTM	Long-Short Term Memory.
M-Commerce	Mobile Commerce.
MLP	Multilayer Perceptron.
MLR	Multinomial Logistic Regression.
MSE	Mean Squared Error.
M-KLR	Multiclass Kernel Logistic Regression.
NBD	Negative Binomial Distribution.
NRMSE	Normalized Root Mean Squared Error.
PLS	Partial Least Squares.
RBF	Radial Basis Function.
RFF	Random Fourier Features.
RFM	Recency Frequency Monetary.
RMSE	Root Mean Squared Error.
RNN	Recurrent Neural Network.
ROC	Receiver Operating Curve.
SEM	Structural Equation Modeling.
SEO	Search Engine Optimization.
SGD	Stochastic Gradient Descent.
SVD	Singular Value Decomposition.
SVM	Support Vector Machine.

1. INTRODUCTION

It is often claimed that the internet changed the retail businesses. For the first-time retailers were not limited to an assortment of popular items and were able to profit from having an endless product variety (Brynjolfsson, Hu, & Smith, 2006). This implies that the aggregated demand for niche products is comparable to the top most popular products – a phenomenon known as the "long tail" effect (Anderson, 2008). Two main factors are usually suggested as the cause of this effect (Goel, Broder, Gabrilovich, & Pang, 2010): one related with the supply side (retailers/producers) and another related with the demand side (consumers).

On the supply side, online retailers can include an incredibly large number of items on their assortment when compared to traditional brick and mortar retailers (Brynjolfsson et al., 2006). This would theoretically be an advantage since demand is heterogeneous, and therefore a larger number of items would allow the retailer to provide more utility to a larger number of customers (Quan & Williams, 2017). Some empirical evidence of this has been found (Goel et al., 2010), suggesting that all consumers have a small portion of niche products in their choices. Also, in the context of digital goods distribution (which can include products such as apps, music, video and written content), advances in Cloud Computing also allowed for an almost linear scaling of data storage infrastructure, which is necessary to attain extremely large assortments with "long-tail" properties (Weinhardt et al., 2009).

On the consumer side, information search costs are hypothesized to be lower for consumers in the online context. By providing search engine and recommender system capabilities, users can more easily access the relevant content from the assortment (Schnabel, Bennett, & Joachims, 2018). In the particular case of mobile app stores, search engines have an increasingly key importance, as evidenced by the rising importance of App Store Optimization (ASO) for publishers in mobile app stores (Wilson, 2018). ASO refers to the tactics employed to improve visibility in app stores similarly to Search Engine Optimization (SEO) (Bilgihan, Kandampully, & Zhang, 2016). This appears to confirm that these systems do play a role in lowering information search costs for consumers. Several studies have provided evidence that recommender systems have a positive impact in content discovery in online environments (Brynjolfsson, Hu, & Simester, 2011; D. M. Fleder & Hosanagar, 2007; Pathak, Garfinkel, Gopal, Venkatesan, & Yin, 2010), which should reinforce the "long tail".

While the impact of the "long tail" effect is usually considered very important for businesses, empirical evidence suggests that this effect doesn't occur in every digital market. In markets such as the online music industry, demand tends to follow a "Superstar" effect (Rosen, 1981), also referred to as the "Winner-takes-all" (Frank Robert & Cook Philip, 1995) or "Blockbuster" (D. Fleder & Hosanagar, 2009) effect. In these markets, the demand for the most popular products largely exceeds the aggregated demand for the least popular ones. The mobile app economy has been found to behave in a similar manner (Zhong & Michahelles, 2013). This "Superstar" effect is consistent with marketing science theory, since it can be explained by two other empirical generalizations that tend to occur across markets (Zhong & Michahelles, 2012, 2013): the natural monopoly effect and the double jeopardy law (A. Ehrenberg & Goodhardt, 2002; A. S. C. Ehrenberg, Goodhardt, & Barwise, 1990; A. S. C. Ehrenberg, Uncles, & Goodhardt, 2004). It has been argued that search engines and recommender systems are especially relevant for these markets (Yin, Cui, Li, Yao, & Chen, 2012; Zhong & Michahelles, 2013), as long as they are able to increase sales diversity.

Some authors argue that different methods of recommendation yield different levels of impact in search cost reduction and thus on sales diversity (D. Fleder & Hosanagar, 2009; D. M. Fleder & Hosanagar, 2007; Zhong & Michahelles, 2013). This suggests that "long tail" effects might be achieved in "superstar" markets by having smarter recommender systems, or by avoiding the usual methods based on collaborative filtering, which tend to reinforce the most popular items (Peltier & Moreau, 2012; Zhong & Michahelles, 2013).

While collaborative filtering was for a long time the main method of building recommender systems, several novel approaches have been proposed in recent years (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013; Koutrika, 2018). Amongst these, deep-learning methods for recommendation systems have seen an exponential rise in published research and results suggest that they are capable of outperforming the traditional approaches (S. Zhang, Yao, & Sun, 2017). In particular, these methods have already been successfully employed in mobile app stores (Cheng et al., 2016). The traditional approaches based on collaborative filtering leverage memorization, but deep-learning has been shown to have a higher generalization capacity (Arpit et al., 2017), especially when regularization is applied, which results in more diverse recommendations (Cheng et al., 2016).

Additionally, we must consider the data sources used to train a recommendation model. The traditional approaches rely solely on either item-item or user-item similarities inferred through matrix factorization methods (Koutrika, 2018). Modern Deep Learning approaches can take advantage of all these features simultaneously. Deep Neural Networks (DNN) can detect non-linear relationships between all these signals (Goodfellow, Bengio, & Courville, 2016, p. 169), which is expected to further contribute to recommendation diversity.

Some early recommender engines relied on user ratings, which have been found not to be the best data source for effective recommender systems (Amatriain, 2013). In contrast, many online retailers, such as mobile app stores, have access to a wide variety of data, which includes user features (demographic, geographic, technographic and behavioral), user-generated content (comments, reviews and search queries), as well as content-based features (app descriptions, app usage and app download co-occurrence which can be used to infer implicit ratings and similarity). Additionally, some mobile app stores such as Google Play - in the past through its Google Plus integrations (Amadeo, 2016), currently through Google Play Games (Google, 2018a) and Google Play Family Library (Google, 2018b), and Aptoide - branded as the social app store, are augmented with online social network features (boyd & Ellison, 2007; Pallis, Zeinalipour-yazti, & Dikaiakos, 2011), which include microblogging, social networking, reviewing, commenting and messaging. Add to this the fact that these app stores run in mobile devices with constant connectivity, which adds real-time streaming characteristics to the data.

Given this context, we can argue that mobile app stores are dealing with data that possesses the 3V properties of Big Data (Sivarajah, Kamal, Irani, & Weerakkody, 2017): Volume, Velocity, and Variety. This suggests that app stores will require Big Data management technologies (which rely on distributed processing) to train, validate, test and deploy recommendation models. Traditional memory-based and matrix factorization approaches are hard to scale in the Big Data context, and we can only obtain approximate numerical solutions – see for instance these Apache Spark implementations of K-Nearest

Neighbors (Maillo, Ramírez, Triguero, & Herrera, 2017) and matrix factorization methods (Bosagh Zadeh et al., 2016). In contrast, DNN models are widely used in data-rich environments to process large amounts of unstructured data (Wedel & Kannan, 2016), and thus are a good fit for mobile app store recommender systems, since they are easier to train and deploy in a distributed manner (Alsheikh, Niyato, Lin, Tan, & Han, 2016).

A limitation of DNN is the fact that, since these do not rely on memorization, their response times are higher than the traditional approaches, which presents a problem in online production environments (Cheng et al., 2016). More complex models are costlier to train and deploy, thus a tradeoff must be made between precision and performance. Few studies have analyzed the cost-benefit of different DNN models, along with the impact of the data sources or the different scoring methods in the context of recommender systems.

A study by Cheng (2016) approached some of these questions while introducing Wide and Deep Learning architectures in mobile app store recommendations. However, the study didn't evaluate the economic impact of such a model. Also, while it did analyze the impact of using rich cross-product features, it didn't analyze the impact of different feature augmentation methods (such as using only structured versus unstructured data) or the impact of not using embedding layer pretraining. Finally, it didn't compare different scoring methods in the final layer (a softmax node was always employed). In this thesis, we intend to explore some of these open questions.

Our main objective is to determine the most efficient deep-learning based recommender system architecture for mobile app stores. To achieve this, we need to satisfy three specific objectives:

1. Assess the impact of using unstructured data versus only using structured data.
2. Assess the impact of embedding layer pre-training in the model performance.
3. Determine the efficiency of using Kernel-based methods in the scoring layer.

The efficiency should be measured using an economic profit metric. This metric should incorporate the estimated financial impact of improvements in customer experience. The improvement should be measured against a baseline, which can be estimated from natural monopoly and double jeopardy effects. Double jeopardy effects are regularly observed in online retail and e-commerce (Huang, 2011) and these appear to extend to m-commerce settings, in particular, the mobile app market, as we've already seen (Zhong & Michahelles, 2013). Since these effects explain mobile app choice behavior and e-commerce in general, we can assume it explains mobile app store choice (at least in the Android platform), with public data confirming that a significant portion of Android users use third-party app stores other than Google Play (App Annie, 2017). Finally, the metric should also incorporate a holistic cost accounting of the model training, validation, testing and deployment at Big Data scale, including cloud computing costs.

In the following sections we will present a literature review covering Recommender Systems, Deep Learning and Big Data. Then we will describe the methodology employed to answer our research questions. Next, we will present the main results, followed by the conclusions, which include a discussion of the results and its implications, the limitations of the study and possible directions for future research.

2. LITERATURE REVIEW

2.1. RECOMMENDER SYSTEMS

Recommender systems attempt to solve the problem of information overload (Aljukhadar, Senecal, & Daoust, 2010). Humans have a limited cognitive ability, and thus, our brains evolved to have selective attention (Sayago, Guijarro, & Blat, 2012). It allows us to select the most relevant events or objects to focus our attention by taking clues from the environment. However, while the *sapiens* brain was well adapted to our ancestral environment, it is less adapted to modern digital environments (Huggett, Hoos, & Rensink, 2007), many times resulting in confusion (Sayago et al., 2012).

With the adoption of information systems (IS) in enterprises, information overload problems became increasingly common, triggering the need for recommender systems. As such, the first recommender system was Tapestry (Goldberg, Nichols, Oki, & Terry, 1992), an email filtering system developed at Xerox to deal with a large number of irrelevant emails received by employees. It was also the first time that the term Collaborative Filtering (CF) was employed, which to this day is still one of the most effective and widely used approaches to recommender system design (Su & Khoshgoftaar, 2009).

Since then many well-known successful applications of recommender systems have appeared in major websites such as Amazon.com, YouTube, Netflix, Spotify, LinkedIn, Facebook, Tripadvisor, Last.fm, and IMDb (Ricci, Rokach, & Shapira, 2015).

Research in this field had its biggest boost due to the Netflix prize which had the first edition at the end of 2006 (Bennett & Lanning, 2007), and again in 2008 and 2009 (Grand Prize). The first and last events appear to have had the biggest impact on scientific production in the field, as evidenced from a bibliographic analysis of recommender system research on a 20-year period[1] (Chart 1).

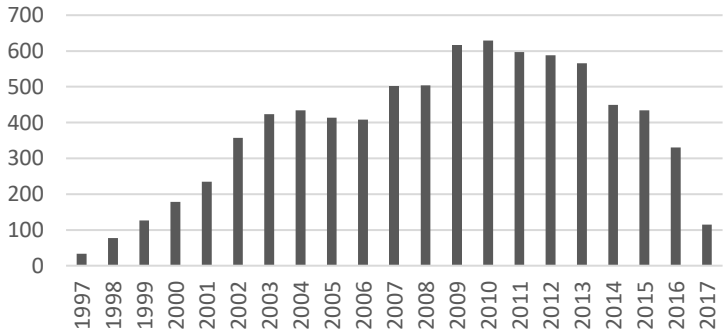


Chart 1 – Number of research papers (with more than 10 citations)
mentioning recommender systems between 1997 and 2017.
(Source: Google Scholar)

2.1.1. Modeling Approaches

Recommender systems can leverage three types of similarities (Katsov, 2018, p. 275): User, Product, and Context. User similarities refer to using the attributes of each customer to infer their intent and

[1] The sample was collected from Google Scholar using Harzin's PoP tool by searching for articles containing the phrase "recommender systems". The analysis only includes papers with more than 10 citations.

preferences. Product similarities refer to using the relationships between users and merchandise which can be expressed in different manners. Many early systems had a focus on explicit ratings, but these relationships can also be inferred through usage data for instance (as a binary variable indicating item consumption, or through counts) (Katsov, 2018, p. 276). CF traditionally leverages either user-based or product-based filtering, or both (hybrid filtering). Finally, context refers to additional signals related to the moment of purchase or the expressed intent (for instance by considering the season of the year). Usually, contextual data is used as a complement to more traditional CF tasks, either by pre-filtering or post-filtering (Katsov, 2018, p. 289).

Another lesser-known type of recommender system includes Knowledge-based filtering (Burke, 2000), which takes advantage of Case-Based Reasoning (CBR) (González-Briones, Rivas, Chamoso, Casado-Vara, & Corchado, 2018). CBR works by asking the user a set of questions that allow the system to produce a recommendation based on previously defined rules. Like contextual recommender systems, CBR features are commonly integrated into recommender systems built with CF methods.

CF methods can further be divided into two types: memory based and model-based (Bobadilla et al., 2013).

Memory-based refers to using instance-based learning, such as K-nearest neighbors (KNN). KNN works by storing a matrix in memory (an instance) that relates users or users and items. For each new user, we take the K nearest objects from the memory instance, using a previously chosen distance or similarity function $f(X_a, X_b)$ that takes the attributes X_a and X_b of each object a and b as input (Aha, Kibler, & Albert, 1991). This distance measure might be Euclidean if the user/item features have metric properties. In that case $f(X_a, X_b)$ can be defined as:

$$f(X_a, X_b) = \sqrt{\sum (X_a - X_b)^2}$$

Jaccard Distance is a common approach if the variables are binary. Under this distance metric we have (Torres, Skaf-Molli, Molli, & Díaz, 2013):

$$f(X_a, X_b) = \frac{|X_a \cup X_b| - |X_a \cap X_b|}{|X_a \cup X_b|}$$

In KNN, the value for K is a hyper parameter. A common rule of thumb is to use the square root of the number of samples (Vrooman et al., 2007). The final recommendations can be based on the top most common items amongst the K-nearest neighbors, or by averaging (if we are working with explicit ratings).

Within model-based approaches, some authors identify two sub-categories (Katsov, 2018, p. 289): regression and latent factor methods.

Within latent factor methods, the most common method is Singular Value Decomposition (SVD). SVD is a matrix factorization technique that essentially performs dimensionality reduction on a user-user or user-item matrix. Mathematically, SVD is nothing more than a generalization of Gaussian elimination, which predated widespread usage of matrices (but is nevertheless consistent with what nowadays is referred to as generalized eigenvalue problem) to non-square and non-real matrixes

(Stewart, 1993). Suppose we have a real matrix A with i rows and j columns, that represent the relationship between users and items, users and users, or users and their attributes.

Our goal is to obtain a decomposition of A, such that:

$$A = U\Sigma V^T$$

$$U = (u_1, u_2, \dots, u_i)$$

$$V = (v_1, v_2, \dots, v_j)$$

Where $\Sigma = diag(\sigma_1, \sigma_2, \dots \sigma_n)$ has nonnegative diagonal elements arranged in descending order of magnitude (the eigenvalues) and U and V are two matrices. The matrix U contains the left eigenvectors, while V contains the right eigenvectors (Gass & Rapcsák, 2004). If A is a user by user matrix, $U = V$. In cases where we have a user by attribute or user by item matrix, U can be interpreted as the eigenvectors of the rows, while V are the eigenvectors of the columns (which can be either items or user attributes). For the final analysis, we should select the relevant matrix of eigenvectors to use as the extracted eigenvectors (which can be either the left or the right depending on the definition of A and the specific problem).

The extracted eigenvectors can be interpreted as latent factors that explain the variance or inertia (Greenacre, 1988) of either the rows or columns of A . The variance or inertia of each factor is nothing more than the eigenvalue (an element of Σ) associated with that eigenvector. These extracted dimensions have metric properties and can then be used to estimate an Euclidean distance between objects before applying KNN. The alternative is to use these as input to another model to score items for a specific user. Several applications of SVD to recommender systems are known (Barragáns-Martínez et al., 2010; Brand, 2003; Paterek, 2007; Sarwar, Karypis, Konstan, & Riedl, 2000).

Other similar approaches based on latent factors include Matrix Factorization using Alternating Least Squares (ALS) and Stochastic Gradient Descent (SGD) learning algorithms (Koren, Bell, & Volinsky, 2009). These approaches are like SVD but tend to generalize better to new cases. This is so because the learning is done using a numerical optimization algorithm which is not only able to better deal with missing values but can also include a regularization term. In these approaches we may have a model as such:

$$\hat{r}_{ij} = v_j^T u_i + \mu_i + b_i + b_j$$

Where $\hat{r}_{ij}$ is the predicted rating of item i for user j, T denotes the matrix transpose, μ_i is the average rating of item i, b_i and b_j are bias terms for the item and user respectively. To perform the optimization, we need to minimize a loss function such as the regularized squared error:

$$R = \begin{bmatrix} r_{11} & \cdots & r_{1j} \\ \vdots & \ddots & \vdots \\ r_{i1} & \cdots & r_{ij} \end{bmatrix}$$

$$\min_{v^*, u^*} \sum (r_{ij} - \mu - b_i - b_j - v_j^T u_i)^2 + \lambda(\| v_j \|^2 + \| u_i \|^2)$$

R is a matrix where each element r_{ij} indicates the rating of item i for user j and $\lambda(\| v_j \|^2 + \| u_i \|^2)$ is a regularization term based on the L2 norm (other types of regularization can be employed such as L1).

Within regression methods, Linear Regression (LinReg) is commonly employed as a baseline model when rating data is available (Mild & Natter, 2002). In the case of recommender systems we need to fit a LinReg model for each item j as such:

$$\hat{r}_{ij} = v_j^T u_i + b_i$$

The standard method is based on ordinary least squares optimization. As such we can obtain estimates for the $\hat{V}$ parameters using the OLS estimator:

$$\hat{V} = (U^T U)^{-1} U^T R$$

A key limitation of such methods is the fact that we can only work with rating data (either implicit or explicit). In cases where we only have binary data (for instance, about if a user either downloaded or not a certain app) we require a different modeling approach. The standard method for binary target variables is Logistic Regression (LR), but in this case, since we're in a multilabel problem, we require Multinomial Logistic Regression (MLR), which can have two forms (Dow & Endersby, 2004): Logit (or Softmax) and Probit. Since logit models are more commonly employed in recommender systems our analysis will focus on these.

Let's suppose we have a set of items $\mathcal{S}$, a set of users $\mathcal{R}$ and a choice matrix L such that:

$$L = \begin{bmatrix} l_{11} & \cdots & l_{1b} \\ \vdots & \ddots & \vdots \\ l_{a1} & \cdots & l_{ab} \end{bmatrix}$$

Where each value of L is 1 if a certain user $a \in \mathcal{R}$ chose item $b \in \mathcal{S}$ and 0 otherwise. Consider now the problem of estimating the probability $P(C_{r,h})$ of a new user $r \in \mathcal{R}$ choosing each item $h \in \mathcal{S}$, and that this probability can be estimated from the vector of user attributes $X_r = \begin{bmatrix} x_{r1} \\ \vdots \\ x_{rk} \end{bmatrix}$. The multinomial logit model then has the following specification (Aurier & Mejía, 2014):

$$\Theta = \begin{bmatrix} \theta_{11} & \cdots & \theta_{1k} \\ \vdots & \ddots & \vdots \\ \theta_{h1} & \cdots & \theta_{hk} \end{bmatrix}$$

$$Y_{r,h} = X_r \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_h \end{bmatrix}^T + \alpha_h$$

$$P(C_{r,h}) = \frac{e^{Y_{r,h}}}{\sum_{s \in \mathcal{S}} e^{Y_{r,s}}}$$

Where $Y_{r,h}$ is a measure of the utility of item h for user r, θ is a matrix of parameters (weights) of each user attribute for each item, and α_h is the constant utility for each item h. We then apply the maximum likelihood estimation method, which is based on cross-entropy loss minimization. The cross-entropy loss function has the following form (Christopher, 2006, p. 209):

$$E(\Theta) = -\sum_{q \in \mathcal{R}} \sum_{w \in \mathcal{S}} l_{q,w} \ln P(C_{q,w})$$

A parameter estimate $\widehat{\Theta}$ can be obtained in an iterative manner using the Newton-Raphson Method. At each iteration, we take the gradient of the error function in respect to one of the parameters θ_p to update its value (Christopher, 2006, p. 210):

$$\nabla_{\theta_p} E(\Theta) = \sum_{q \in \mathcal{R}} [P(C_{q,w}) - l_{q,w}] X_q$$

The final recommendations can be found by simply applying the following recommendation function g to each value $P(C_{q,w})$:

$$g(x) = \begin{cases} 1 \; if \; x > 0.5 \\ 0 \; otherwise \end{cases}$$

If $g(P(C_{q,w})) = 1$ we will recommend item w for user q.

Several empirical applications of MLR to recommender systems are known (D. M. Fleder & Hosanagar, 2007; S.-H. Yang, Long, Smola, Zha, & Zheng, 2011; Zhao, Zhang, Zhang, & Friedman, 2016).

Many other modeling approaches exist for recommender systems. Fischer's Linear Discriminant Analysis (LDA), also known as Multivariate Discriminant Analysis has been employed for recommender systems (K. Kim, 2011). Tree-based machine learning methods such as Decision Trees (Cho, Kim, & Kim, 2002; Gershman & Meisels, 2010), Random Forests (O'Mahony, Cunningham, & Smyth, 2010; H. R. Zhang & Min, 2016) and Gradient Boosted Regression Trees (Ostuni, Di Noia, Mirizzi, & Di Sciascio, 2014) have also been employed.

Bayesian methods were common approaches in the early 2000s (Condli, Madigan, Lewis, & Posse, 1999; Jin & Si, 2004; Miyahara & Pazzani, 2000). Towards the end of the decade some applications of ensemble methods emerged (Jahrer, Töscher, & Legenstein, 2010; Schclar, Tsikinovsky, Rokach, Meisels, & Antwarg, 2009) along with econophysics inspired approaches based on heat and mass diffusion techniques, which have reemerged recently (C. Liu & Zhou, 2010; Lü et al., 2012; Ren, Zhou, & Zhang, 2008; Vidmer, Zeng, Medo, & Zhang, 2015; Y. C. Zhang, Blattner, & Yu, 2007).

Kernel methods have also found applications in recommender systems (Abernethy, Bach, Evgeniou, & Vert, 2008; X. Liu et al., 2016) including the use of Support Vector Machines (SVM) (Fortuna, Fortuna, & Mladenić, 2010; Oku, Nakajima, Miyazaki, & Uemura, 2006; Xia, Dong, & Xing, 2006).

Kernel methods are based on the *kernel trick* also known as *kernel substitution* (Christopher, 2006, p. 292). In these methods we do a mapping of the feature space of new unlabeled inputs x' to the training examples x, using a kernel function k that relies on a basis function φ_q such that:

$$k(x, x') = \varphi(x) T \varphi(x') = \sum_{q \in \mathcal{R}} \varphi_q(x) \varphi_q(x')$$

The basis function can assume several forms. The most commonly employed form is the Radial Basis Function (RBF) (Christopher, 2006, p. 299). Recently scalable approximation methods based on stochastic methods have emerged, such as Random Fourier Features (RFF) (Rahimi & Recht, 2007). Kernel methods can be seen as a form of instance-based methods, where the kernel function acts as a similarity measure between the training set and the new cases (Christopher, 2006, p. 292). These techniques allow us to linearize the feature space so that simpler decision algorithms can be employed.

The application of SVMs to recommender systems involves training an SVM classifier for either each user or each item (Xia et al., 2006). In either case, we are seeking to find a set of items to recommend to each user, by either classifying users for an item or vice-versa. We will assume that a model will be fit for each item, but the inverse specification works in a similar manner.

SVMs rely on the kernel mapping to obtain a transformed feature space where a linear decision boundary can be found based on the support vectors, which are sample cases that lie on the maximum margin hyperplanes in the feature space (Christopher, 2006, p. 330). This maximum margin hyperplane for an item $w \in \mathcal{S}$ can be found by solving (Christopher, 2006, pp. 327–330):

$$\arg\min_{\Theta,b}\left\{\sum_{q\in\mathcal{R}} E_{\infty}([\theta_q k(x, x_q) + b] l_{q,w} - 1) + \lambda||\Theta||^2\right\}$$

Where $E_{\infty}(z)$ is a function that returns 0 if $z \geq 0$ (which happens when the vector corresponding to q is not a support vector), and ∞ otherwise. Alternative multiclass formulations of SVM have been proposed (Hsu & Lin, 2002), which could be used to jointly train a classifier across all items, but no recommender system applications are yet known. Similar methods exist with could potentially be employed to build recommender systems, such as Kernel Logistic Regression (KLR). KLR differs from LR in the definition of $Y_{r,h}$:

$$Y_{r,h} = k(X, X_r)\begin{bmatrix}\theta_1 \\ \vdots \\ \theta_h\end{bmatrix}^T + \alpha_h$$

A multilabel version of KLR can be achieved by performing the same substitution with softmax instead of LR as in Karsmakers, Pelckmans, & Suykens (2007). KLR has been empirically and analytically demonstrated as having the similar performance and behavior of an SVM (Karsmakers et al., 2007), the main difference being the fact that it requires the entire dataset as opposed to only using the support vectors to build a decision margin (Zhu & Hastie, 2005). As such, it is expected that multiclass KLR should behave similarly to multiclass SVM approaches (Karsmakers et al., 2007). Currently, no applications of KLR (or its multiclass version) are known in recommender systems, but it can potentially improve existing LR/Softmax based methods.

Other recommender systems techniques include Artificial Neural Networks (ANN), such as Multilayer Perceptrons (MLP), also known as shallow networks (Goga, Kuyoro, & Goga, 2015). The current state of the art in recommender system design is deep learning methods, which are deep versions of ANN (Cheng et al., 2016; Covington, Adams, & Sargin, 2016; Koutrika, 2018; H. Wang, Wang, & Yeung, 2015; S. Zhang et al., 2017). Both deep and shallow networks can have multiple output nodes that function like either multinomial linear regression or multinomial logistic regression.

2.1.2. Model Evaluation

Model evaluation in the context of recommender systems includes several different metrics which range from multiclass/multilabel versions of traditional machine learning and data mining ones to specific profit-centric indicators (Gilotte, Calauzènes, Nedelec, Abraham, & Dollé, 2018; Ju, Choi, Kim, & Moon, 2017).

We can divide the evaluation metrics into three main groups: Offline, Online and User Studies (Beel & Langer, 2013).

Offline evaluation metrics are the most commonly employed. It assumes that models can be evaluated in terms of prediction/classification accuracy of past item consumption data or ratings. In this context, if an item is recommended to a user who hasn't previously consumed it, we count it as a model failure. It's easy to see that offline metrics are not guaranteed to be realistic, since the fact that an item wasn't previously consumed or rated, doesn't mean it's not relevant to that user.

Online metrics can overcome these issues by testing the recommender system in a production environment. In this context we can apply several specific recommender system metrics, usually derived from marketing evaluation metrics applied in online advertising and e-commerce (Beel & Langer, 2013). The most common metric is the Click-Through Rate (CTR) (Beel & Langer, 2013):

$$CTR = \frac{Clicks}{Impressions}$$

Where impressions are the number of impressions of all recommended items, and clicks are the number of items that were clicked after being recommended. The assumption of this metric is that a clicked item is relevant to the user. While this assumption is not completely realistic, we can argue that its closer to reality than most offline evaluation metrics.

User studies come from the usability and user experience research tradition, which employ survey-based methods. Several standardized psychometric constructs exist to evaluate recommender systems. The ResQues framework proposed by Pu and Chen (2010) includes several constructs that can be measured individually using Factor Analysis or jointly using Structural Equation Modeling (SEM) with either Partial Least Squares (PLS) or Covariance-based estimation (Ayeh, Au, & Law, 2013). These constructs are related to user-perceived qualities, user beliefs, user attitudes and behavioral intentions (Pu & Chen, 2010). Other similar frameworks based on SEM have been proposed such as the Knijnenburg, Willemsen, Gantner, Soncu, & Newell (2012) model which can be used to measure the subjective user experience of a recommender system. These latent psychometric constructs can be used to compare different recommendation models from the point of view of the end-user experience.

We will focus on Offline evaluation since that's the most common approach. Offline evaluation can be thought of as either a classification or a regression (prediction) problem, depending on the type of target variable we have (Herlocker, Konstan, Terveen, & Riedl, 2004). In the former, we are usually working with binary item consumption, while on the latter, we are usually working with explicit or implicit ratings.

Regression-type metrics for recommender system evaluation include the Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Normalized Root Mean Squared Error (NRMSE) (Katsov, 2018, p. 282). In these metrics, we are assuming that the model outputs a prediction $\hat{y}_{r,h}$ for actual rating $y_{r,h}$ by user r for item h, resulting in an error term $e_{r,h}$ such that:

$$e_{r,h} = \hat{y}_{r,h} - y_{r,h}$$

MSE has the following definition:

$$MSE = \frac{1}{|T|} \sum_{(r,h) \in T} {e^2}_{r,h}$$

Where T is a matrix of unseen user ratings for items used for model testing. MSE is not always convenient because its value cannot be easily compared with the original ratings, since it's a squared value (Katsov, 2018, p. 282). RMSE allows us to overcome this issue:

$$RMSE = \sqrt{MSE}$$

RMSE was the target metric for the Netflix prize (Bennett & Lanning, 2007), and is currently the most popular choice for regression type model evaluation (Szabó, Póczos, & Lőrincz, 2012). NRMSE can now be defined as:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}}$$

The advantage of NRMSE is that its value is defined in the range (0,1), which allows us to compare models applied to ratings with different scales.

For classification type models we can employ multiclass/multilabel versions of traditional classification metrics, which can usually be done by macro (across classes) or micro (across cases) averaging (Tsoumakas & Vlahavas, 2007). Consider a binary evaluation measure $M(tp, tn, fp, fn)$ that is calculated based on the number of true positives (tp), true negatives (tn), false positives (fp) and false negatives (fn). Let $tp_\rho, fp_\rho, tn_\rho$ and fn_ρ be the number of true positives, false positives, true negatives and false negatives after binary evaluation for a label ρ. The macro-averaged and micro-averaged versions of M, are calculated as follows, where L is the set of labels (items) (Tsoumakas & Vlahavas, 2007):

$$M_{macro} = \frac{1}{|L|} \sum_{\rho=1}^{|L|} M\left(tp_\rho, fp_\rho, tn_\rho, fn_\rho\right)$$

$$M_{micro} = M\left(\sum_{\rho=1}^{|L|} tp_\rho, \sum_{\rho=1}^{|L|} fp_\rho, \sum_{\rho=1}^{|L|} tn_\rho, \sum_{\rho=1}^{|L|} fn_\rho\right)$$

A popular metric for recommender system evaluation is the F1-Score (Herlocker et al., 2004) which can be defined based on the values of precision and recall:

$$Precision = \frac{tp}{tp + fp} \; Recall = \frac{tp}{tp + fn}$$

$$F1\ Score = 2\ \frac{Precision \times Recall}{Precision + Recall}$$

Where tp is the number of true positives (number of relevant recommendations), fp is the number of false positives (number of irrelevant recommendations) and fn is the number of false negatives (number of would be relevant items that were not recommended). Logically, and for the sake of completeness, we can further define an additional measure tn as the number of true negatives

(number of would be irrelevant items that were not recommended). These evaluation metrics usually assume an offline evaluation setting.

The most common metric for classification type models is the AUC (Verbeke, Dejaeger, Martens, Hur, & Baesens, 2012):

$$AUC = \int_{-\infty}^{+\infty} F_0(s) f_1(s) ds$$

AUC assumes that a classifier produces a score $s = s(x)$ from the features x with a corresponding probability density function of these scores for class k instances $f_k(s)$ and cumulative distribution function $F_k(s)$ with two classes $k = 0,1$. This metric can only be employed to evaluate models that produce probability scores such as softmax regression.

Other related metrics are also used including the Gini coefficient ($Gini = 2 \times AUC - 1$) and the Kolmogorov-Smirnov (KS) statistic which is the maximum distance between a receiver operating curve (ROC) and the diagonal at a specific cut-off value (usually 0.5) (Verbeke et al., 2012).

An empirical study by Forman and Scholz (2009) advises the use of "average AUC" (consistent with the previous definition of a Macro measure) and the "F1-Score computed from false and true positives" (consistent with the previous definition of a Micro measure) to compute multilabel versions of AUC and F1-Score respectively.

Other popular multiclass classification metrics commonly employed to evaluate recommender systems include the Hamming loss and the Jaccard Index. Hamming loss can be defined as the proportion of misses (Luaces, Díez, Barranquero, del Coz, & Bahamonde, 2012):

$$Hamming\ Loss = \frac{FP + FN}{FP + FN + TP + TN}$$

Jaccard Index can be interpreted as a multilabel measure of accuracy based on the already defined Jaccard Distance (Luaces et al., 2012):

$$Jaccard\ Index = \frac{TP}{FP + FN + TP}$$

Additional evaluation metrics include Diversity, Coverage, Serendipity and Novelty (Katsov, 2018, pp. 285–288), which originate from desirable recommender system properties (Ge, Delgado-Battenfeld, & Jannach, 2010; Vargas & Castells, 2011). Most of these might be applied in both offline and online settings. We will review some of the most common measurement approaches.

Diversity is the ability of the recommender system to produce recommendations that are dissimilar (Katsov, 2018, p. 286). To measure diversity, we need to leverage a content-based similarity metric. One approach is to extract features from the item's description/title (using text mining and natural language processing techniques), contents (requiring some form of feature extraction from multimedia/hypermedia content), or some other properties.

In alternative, we can also obtain a similarity metric between items by factorizing a user by items matrix with the methods we've already seen (such as memory-based methods, SVD or even knowledge-based rules). Based on this metric we can define a distance function $dist$ between items:

$$similarity(a, b) \in [0,1]$$

$$dist(a, b) = dist(b, a) = 1 - similarity(a, b)$$

The diversity of the set of recommendations for each user can then be defined as the average distance between all pairs of recommendations.

Coverage refers to the percentage of users that the recommender system can give recommendations to (Katsov, 2018, p. 289). This is important because of the sparse nature of the data for recommender systems, which is also the source of the cold start problem (Schein, Popescul, Ungar, & Pennock, 2002). Another view on coverage is related to catalog coverage which is the percentage of the catalog merchandise that is being recommended (Katsov, 2018, p. 289):

$$Catalog\ Coverage = \frac{1}{|\mathcal{S}|}\left|\bigcup_{u \in \mathcal{R}} Y_u\right|$$

Where Y_u is a recommendation list for user u, $|\mathcal{S}|$ is the cardinality of the set of items $\mathcal{S}$ and $\mathcal{R}$ is the set of all users.

Serendipity is a measure of the extent to which recommendations are attractive and surprising (Katsov, 2018, p. 286). Despite being subjective, heuristic approaches have been suggested based on baseline models that produce trivial recommendations (Ge et al., 2010), which are usually simple recommendation system approaches. From that baseline, we will create a set of expected items. Any item that doesn't belong to this set will be considered unexpected. We can define a usefulness function that returns 1 when a given recommendation is both relevant and unexpected. Serendipity is therefore defined as:

$$Serendipity = \frac{\sum_{item \in \mathcal{S}} Usefulness(item)}{|\mathcal{S}|}$$

Recommendations are considered novel if the user is not aware of the recommended items at the moment the recommendation is provided (Katsov, 2018, p. 285). Many approaches exist to measure novelty: Popularity-based, Distance-based (Vargas & Castells, 2011) and Time-based (Katsov, 2018, p. 285). Popularity-based novelty measurement takes advantage of the long-tail concept. If a relevant recommended item is less popular, we can assume that it might be more novel. Therefore (Vargas & Castells, 2011):

$$Novelty_{Popularity-based}(item, context) = 1 - p(seen|item, context)$$

Distance-based novelty measurement takes advantage of the item features to define a distance measure between items. Then we can leverage the user's past behavior to measure how novel that item might be for a specific user, by considering its past consumption context. Therefore (Vargas & Castells, 2011):

$$Novelty_{Distance-based}(item, user) = \min_{past\ item \in user}[1 - similarity(item, past\ item)]$$

Time-based novelty measurement assumes that the elapsed time between recommendation and action taken on that item by a user indicates its novelty level (Katsov, 2018, p. 285). A larger elapsed time means a higher novelty. Therefore:

$$Novelty_{Time-based}(t_{item}) = \gamma t_{item}$$

Where t_{item} is the elapsed time between the recommendation of the item and the action, and γ is a time weight parameter. The weight parameter can either be set manually based on the retailer's experience, or empirically by combining past behavioral data with a user-based novelty psychometric construct (Pu, Chen, & Hu, 2012). We can therefore use a regression model to estimate the parameter, by assuming that the following relationship is true:

$$Novelty_{User-based}(item) = \gamma t_{item}$$

This novelty measurement approach overlaps with the user studies evaluation approach. In e-commerce settings we can also evaluate the recommender system using profit metrics derived from sales:

$$Profit = \sum_{items} Quantity\ Sold_{item} \times Margin_{item}$$

In environments where we are not selling items (including email, news, multimedia content recommendations, amongst others) this metric cannot be used. In the case of mobile app recommendations, most apps are free to download (94.24% in the Android platform, 88.18% in iOS) (Statista, 2018), which means that this metric can only be used for the small proportion of paid apps. For non-paid items, we need to consider a different approach to profit measurement based on the impact of the recommender system on the Customer Lifetime Value (CLV) (Iwata, Saito, & Yamada, 2008). CLV is commonly used to guide Customer Relationship Management (CRM) processes (Blattberg, Kim, & Neslin, 2008, p. 163).

The revenue associated with each successfully recommended item is the portion of the CLV that can be attributed to that item. Assuming that the revenue associated with each item is constant for all items, then the portion of the CLV attributed to each item can be derived from the item consumption probability (Iwata et al., 2008). In environments such as mobile app stores, users can perform several different actions, but all of these are ultimately related with app consumptions, therefore its plausible to assume that app acquisitions are the major component of the CLV, and that these are heavily influenced by recommender systems.

CLV can be calculated using several methods. Traditionally CLV would be estimated using a Recency, Frequency and Monetary (RFM) model to rank users (Gupta et al., 2006). Modern approaches extend the concept of RFM to obtain a financial estimation of the CLV for each customer.

One approach to obtaining a global average of CLV is to calculate the average of all individual-level CLV values across a database, where the CLV for each customer is given by (Blattberg et al., 2008, p. 108):

$$CLV = \sum_{t=1}^{\infty} \frac{(\widetilde{D}_t - C_t)S_t}{(1+\delta)^{t-1}}$$

Where δ is the discount rate per time unit t, $\widetilde{D}_t$ is the revenue generated by the user on moment t, C_t is the cost of serving user on moment t, S_t is the probability of the customer not churning before

moment t. The RFM model is incorporated into CLV through the S_t ("recency" and "frequency") and $\widetilde{D}_t$ ("monetary").

Several methods to obtain $\widetilde{D}_t$ have been proposed (Blattberg et al., 2008, pp. 130–131): The simplest approach is to assume that $\widetilde{D}_t$ is constant in all periods based on the individual average or the global Average Revenue Per User (ARPU). Trend, causal (based on the user features) and stochastic models are also commonly employed.

To model $S_t = p(alive)$ we need a consumer behavior model. Several have been proposed: Beta Binomial/NBD (BB-NBD) (Jeuland, Bass, & Wright, 1980), NBD-Dirichlet (Goodhardt, Ehrenberg, & Chatfield, 1984), Pareto/NBD (P-NBD) (Schmittlein, Morrison, & Colombo, 1987), Beta-Geometric/Beta-Binomial (BG-BB) (Fader, Hardie, & Berger, 2004), and Beta-Geometric/NBD (BG-NDB) (Fader, Hardie, & Lee, 2005). We will focus our attention on the most recent model by Fader *et al.* (2005), according to which $p(alive)$ is given by (Fader & Hardie, 2008):

$$p(alive|x, t_x, T, r, \alpha, a, b) = \frac{1}{1 + \frac{a}{b+x}\left(\frac{\alpha+T}{\alpha+t_x}\right)^{r+x}}$$

Where x is the number of transactions observed in the time-period $(0, T]$ ("frequency") and $t_x (0 < tx \leq T)$ is the time of the last transaction ("recency"). The model's four parameters r, α, a, b can be estimated using maximum likelihood estimation from the likelihood function:

$$L(r, \alpha, a, b \,|X = x, t_x, T) = \frac{B(a, b+x)}{B(a,b)}\frac{\Gamma(r+x)\alpha^r}{\Gamma(r)(\alpha+T)^{r+x}} + \delta_{x>0}\frac{B(a+1, b+x-1)}{B(a,b)}\frac{\Gamma(r+x)\alpha^r}{\Gamma(r)(\alpha+t_x)^{r+x}}$$

Suppose we have a sample of N customers, where customer i had $X_i = x_i$ transactions in the period $(0, T_i]$, with the last transaction occurring at t_{x_i}. The sample log-likelihood function is:

$$LL(r, \alpha, a, b) = \sum_{i=1}^{N} ln\, L(r, \alpha, a, b \,|X_i = x_i, t_{x_i}, T_i)$$

By maximizing this function using standard optimization methods, we can obtain the parameter estimates. While BG/NBD model and its extensions are the current state of the art, it requires individual-level data to estimate the parameters. The already mentioned NBD-Dirichlet model (Goodhardt et al., 1984) can be an alternative to this model with simpler data requirements.

NBD-Dirichlet results from the combination of two distributions: the Negative Binomial Distribution (NBD) and the Dirichlet Multinomial Distribution (DMD) (Dawes, Meyer-Waarden, & Driesener, 2015). The NBD part describes the category buying behavior of individuals in a market, while the DMD part models the probability of each individual in the market purchasing a specific brand (Goodhardt et al., 1984). The resulting model is therefore given by (Goodhardt et al., 1984):

$$p(r_j|n) = \frac{\binom{n}{r_j}\mathrm{B}(\alpha_j + r_j, S - \alpha_j + n - r_j)}{\mathrm{B}(\alpha_j, S - \alpha_j)}$$

Where $p(r_j|n)$ can be interpreted in the context of online retail as the probability of using r_j times the retailer j amongst n usages of the retailer category, α_j is the usage propensity for retailer j, S is the diversity of usage behavior in the category ($S = \sum_j \alpha_j$) (Bound, 2009). We can assume that $MS_j = \alpha_j/S$ where MS_j is the market share of the retailer j (Wright, Sharp, & Sharp, 2002).

B is the Beta function such that:

$$\mathrm{B}(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$$

Γ is the Gamma function such that:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt, x > 0$$

Accurate estimates for all α_j can be obtained by fitting the DMD for all retailers J in the market using panel data such that (Wrigley & Dunn, 1985):

$$p(r_1, r_2, \dots, r_j|n) = \binom{n}{r_1, r_2, \dots, r_j} \frac{\Gamma(S)}{\Gamma(S+n)} \prod_{j=1}^{J} \left[\frac{\Gamma(\alpha_j + r_j)}{\Gamma(\alpha_j)}\right]$$

Model estimation can be done using the original "mean and zero" method proposed by Goodhardt (1984) or maximum likelihood (Wrigley & Dunn, 1985). Intelligence providers such as App Annie and 42Matters provide app usage intelligence data for the mobile industry, which can be used to estimate these parameters for all platforms. An Excel workbook is available to facilitate the model estimation using maximum likelihood (Rungie, 2003).

An R package is also available (F. Chen, 2016) which can be used to obtain estimates of S by having as input the platform category penetration (category users in the population), platform penetration (users of the specific platform in the population), category usage frequency and the market shares of the different platforms. We can then obtain the platform usage propensity through its market share ($MS_j = \alpha_j/S$). The required input information can be found though several industry publications and market intelligence providers such as Statista and GSMA Mobile Economy.

Under this set of assumptions $p(alive)$ is $p(r_j > 0|n)$, where n is the average usage frequency of the retailer category. NBD-Dirichlet is a realistic assumption, since online consumer behavior in the context of mobile app stores has been shown to follow the Double Jeopardy Law and Natural Monopoly effects described by the NBD-Dirichlet model (Zhong & Michahelles, 2013).

2.1.3. Feature Extraction

As we've seen, model-based approaches to recommender system design can leverage several types of user and context features. We will review some of the most common feature extraction techniques that are relevant for recommender system design in the context of online retailing. Given its social nature, many online retailers generate large amounts of user-generated content (such as comments, reviews, ratings, and likes), keyword search data, social network data. Its interactive and mobile features also generate behavioral data (such as touchstream/clickstream data), geographic coordinate data and technographic (device-related) data.

Behavioral data (touchstream/clickstream) along with technographic data, is usually more structured and easier to quantify. However, in Big Data context also becomes difficult to manage, and may require feature extraction techniques to reduce its dimensionality. Classic autoencoders are widely used for this purpose (Hinton & Salakhutdinov, 2006). Autoencoders are based on the Restricted Boltzmann Machine algorithm, which can be interpreted as a two-layer neural network (a visible and a hidden layer) where each unit takes a binary value $\in [0,1]$. The nodes from the visible layer serve as input to the hidden layer. The energy of a Boltzmann machine with N nodes is defined by (Osogami, 2017):

$$E_\theta(x) = -\sum_{i=1}^{N} b_i x_i - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w_{i,j} x_i x_j$$

Where x is a random configuration of binary states for the nodes, b_i is its bias, $w_{i,j}$ is the weight between a pair of nodes i and j. The parameters are collectively denoted by

$$\theta = (\mathrm{b}_1, \ldots, \mathrm{b}_N, \mathrm{w}_{1,2}, \ldots, \mathrm{w}_{N-1,N})$$

From the energy we can obtain a probability distribution over binary patterns (Osogami, 2017):

$$P_\theta(x) = \frac{exp\ (-E_\theta(x))}{\sum_{\tilde{x}} exp\ (-E\theta(\tilde{x}))}$$

Where the summation with respect to $\tilde{x}$ is over all possible N bit binary values. By optimally setting the values of θ, we can approximate $P_{target}(.)$ with $P_\theta(.)$, meaning that we can reconstruct an input from a compressed representation. Hinton (2006) demonstrated that Autoencoders have superior reconstruction power when compared to Principal Components Analysis (PCA), the most commonly used dimensionality reduction method. Thus, numerical data inputs can effectively be reduced to a smaller number of orthogonal features.

From the types of data previously described, user-generated content and keyword search data are unstructured. Before this data can be operationalized, we need to somehow quantify this data. Autoencoder inspired feature extraction methods have been proposed for this, such as Doc2Vec (Le & Mikolov, 2014). This method, in turn, is based on a previous word embedding method called Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013). Word2Vec consists of a softmax regression model that predicts Wi given the word Wj using a softmax regression model trained on n-grams of a vocabulary V such that (Apache, 2017):

$$p(w_i|w_j) = \frac{exp(u_{w_i}^\top v_{w_j})}{\sum_{l=1}^{V} exp(u_l^\top v_{w_j})}$$

$$u_{w_i} - \text{vector representation of the word } W_i$$

$$v_{w_j} - \text{vector representation of the word } W_j$$

The resulting model allows us to assess which of the V are closest in the semantic space using this cosine distance (the probability of co-occurrence given by the model). If two words are close they can be considered to have identical meanings, which in linguistic terms equals being synonyms. Doc2Vec extends this very same methodology to documents.

Some mobile app stores possess social follower features, resulting in social network data. Social network data is also not structured in the traditional relational database sense, which makes it difficult to represent in SQL tables. Therefore, we can assume that social network data is semi-structured. Its quantification traditionally requires social network analysis techniques, that are usually performed on the entire graph. Popular hand-crafted node-level features include centrality measures (Bloch, Jackson, & Tebaldi, 2016) such as degree (indegree and outdegree), closeness, betweenness, eigenvector, Katz, PageRank, diffusion, and percolation (Piraveenan, Prokopenko, & Hossain, 2013). These social network features have had some applications in recommender systems (Li, Wu, & Lai, 2013). Other more general methods of feature extraction, based on Word2Vec and Random Walk sampling have been proposed such as DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014). DeepWalk, like PageRank, is a very efficient technique for social network feature extraction at Big Data scale, while also having the advantage of providing general features which can be potentially more useful for recommender systems. Random walks are generated for each starting node, resulting in text sentences, whose features can be extracted using Word2Vec. Recently, a more general framework has emerged from DeepWalk, called Content-enhanced Network Representation Learning, or CENE (Sun, Guo, Ding, & Liu, 2016). This new method allows us to extract features from social graphs where each node is associated with textual content (with the possibility of also being extended to other types of multimedia content).

The feature extraction techniques previously mentioned are useful in extracting orthogonal features in the online retail context. Deep learning methods are currently the state of the art in recommender system design, that can leverage all these features to produce recommendations.

2.2. Deep Learning

Deep Learning refers to a specific type of artificial neural network models, capable of performing feature extraction as well as feature processing (LeCun, Bengio, & Hinton, 2015). Thus, deep learning can be defined in opposition to shallow learning, which refers to single data processing layer, a category that includes most traditional machine-learning methods (LeCun et al., 2015), but it's usually employed to refer to traditional ANNs. While shallow models have been theoretically demonstrated as being capable of performing the same tasks as deep models (Cybenko, 1989; Leshno, Lin, Pinkus, & Schocken, 1993), recent research has revealed that to approximate the same functions a very wide shallow network with an exponentially larger number of nodes may be required (Montúfar, Pascanu, Cho, & Bengio, 2014). These results suggest that deep learning networks may be the only feasible option for efficient approximation with finite datasets in complex nonlinear systems (Goodfellow et al., 2016, p. 198).

As we've seen, deep learning originated from shallow learning, which usually means ANNs. These models originated from computational neuroscience, as a method of modeling the behavior of neurons. The first model of its type was the perceptron (Rosenblatt, 1958). This type of model simulates the behavior of the dendrites (neuron inputs), each with different weights, meant to model the effects of long-term potentiation and depression needed for plasticity and learning in the brain (Michmizos, Koutsouraki, Asprodini, & Baloyannis, 2011). The perceptron algorithm then simulates the neuronal response by either outputting -1 or 1 (meant to model neuronal spikes) if a given threshold is reached.

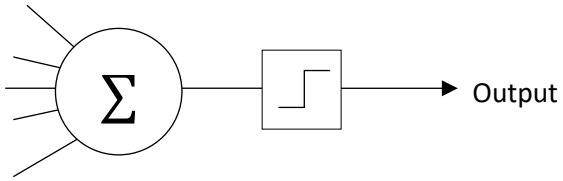


Figure 1 – Perceptron

Later versions of the model were created with more than one layer, called multilayer perceptrons (Hornik, 1991). These networks had an input layer, and output layer and one hidden layer. Each perceptron in the input and hidden layers need to have a continuous output, usually modeled through sigmoid functions.

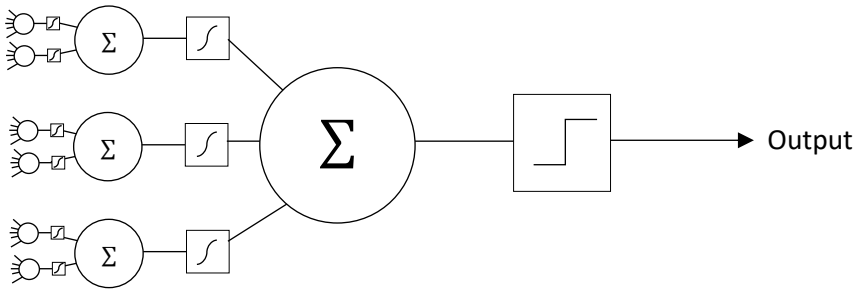


Figure 2 – Shallow Network (Multilayer Perceptron)

Models with a single layer are referred to as shallow networks. Models with multiple hidden layers are deep networks. As we've seen these have different properties, such as being able to more effectively learn more complex relationships with fewer computational resources and data. A seminal work in this field was the application of a deep network to image classification (Krizhevsky, Sutskever, & Hinton, 2012).

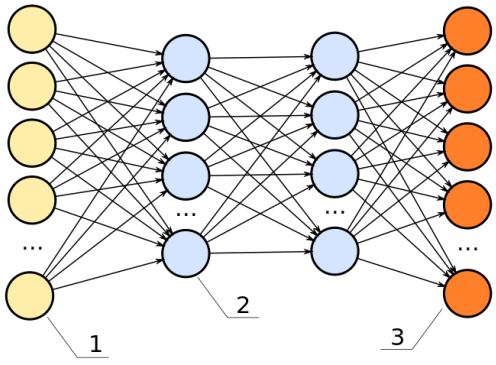


Figure 3 – Deep Network

The quintessential deep learning model is the feed-forward network (FFN). These are capable of learning complex nonlinear relationships from latitudinal raw features using backpropagation and gradient descent and its variants. Extensions have been proposed for autocorrelated data such as convolutional neural networks (CNN) (Rowley, Baluja, & Kanade, 1995).

Deep-learning models have also been combined with other types of machine learning models, including ensemble methods, resulting in models such as the Deep Neural Decision Forests

(Kontschieder, Fiterau, Criminisi, & Bulo, 2015) and Deep SVM (Abdullah, Veltkamp, & Wiering, 2009; Tang, 2013).

More recently, deep learning models have also found use in temporal-sequence modeling. The superiority of deep networks in sequence feature extraction has been demonstrated using methods such as recurrent neural networks (RNN) (Elman, 1990), long short-memory (LSTM) (Hochreiter & Urgen Schmidhuber, 1997) and more recently attention and memory augmented networks (Olah & Carter, 2016).

Memory networks were introduced by Weston, Chopra, & Bordes (2014) as a type of neural network with reading and writing capabilities from a long-term memory cell (Goodfellow et al., 2016). These networks are part of a larger group of attention-based networks (Chorowski, Bahdanau, Serdyuk, Cho, & Bengio, 2015). Following the introduction of memory networks, Graves (2014) introduced the Neural Turing Machine (NTM), inspired by the Turing Machine (Turing, 1937) and von Neumann Architecture (Neumann, 1945). NTM combines a neural network controller (analogous to a CPU) and an addressing scheme for an external memory. The controller may be an RNN or LSTM. In this latter case, we can consider that its longer short-term memory is analogous to classical processor registers (Graves et al., 2014). The current state of the art in memory networks is the Differentiable Neural Computer, or DNC (Graves et al., 2016), which results in a hybrid model that combines neural and computational processing capabilities. Other types of memory networks have also been proposed. A continuous version of the NTM has been proposed based on the algebraic Lie-group theory, which is the Lie-Access Neural Turing Machine, or LANTM (G. Yang, 2016). This architecture introduces a continuous addressing scheme which allows these types of networks to become differentiable end-to-end.

These deep-learning models have been demonstrated as being more efficient than traditional sequence feature extraction techniques such as fast-Fourier transform (Polat & Güneş, 2007) and principal components analysis (PCA) (Gavrilov, Anguelov, Indyk, & Motwani, 2000). Deep learning networks also have limitations. One of the main limitations is the amount of computational power required to train them. Therefore, most deep learning networks are trained in parallel or distributed environments.

Parallel training of deep networks is usually done through General-Purpose Graphics Processing Units (GP-GPUs), given that it significantly reduces training time, making it practical (Campos et al., 2017). Amongst these are CUDA GPUs developed by NVIDIA. CUDA was the first framework for general purpose computing applications of GPUs (Nickolls & Dally, 2010). GPUs are like CPUs in the sense that both are examples of von Neumann Architectures, where there is a separate controller and memory unit. Von Neumann Architectures have long been demonstrated as physical examples of Turing Machines, capable of running any algorithm. The main difference between CPUs and GPUs is that GPUs possess multiple parallel controllers with a shared memory unit. While each core is much simpler than current CPU cores, these are optimized for numerical computations, resulting in increased performance that involves large vector operations (such as deep network training that requires vector convolution). Several software packages and libraries have been developed for deep learning based on these such as Tensorflow (Abadi et al., 2016) and Caffe (Jia et al., 2014).

In the distributed side, Apache Hadoop and Apache Spark are the most widely used ecosystems for distributed computation in clusters, and were originally developed as distributed Big Data platforms. Distributed GPUs are also increasingly being used in the industry to train deep networks (Hazelwood

et al., 2018). These are usually made possible through Hadoop or Spark clusters where each node has both CPU and GPU cores. Two levels of parallelization are made possible through this cluster architecture. This is an example of a distributed parallelization paradigm which can potentially help solve many performance issues in the future. These are however notoriously difficult to program and don't currently have widespread adoption.

2.3. BIG DATA

The term Big Data, as we've seen, usually refers to data that conforms to the 3V's: Volume, Velocity, Variety (Sivarajah et al., 2017). During the last few decades, the amount of data generated by businesses has increased significantly, resulting in the need for specific technology to store and process this data (Oussous, Benjelloun, Ait Lahcen, & Belfkih, 2017). In the context of online platforms, mainly due to its network effects and many-to-many relationships, Big Data becomes the norm, which explains why platforms such as Facebook are storing 600 TB per day (Vagata & Wilfong, 2014). Hadoop is widely used to develop Big Data applications in the industry (M. Chen, Mao, & Liu, 2014). This system has two main components built in Java (Landset, Khoshgoftaar, Richter, & Hasanin, 2015):

1. Hadoop Distributed File System (HDFS) – a file system created to store large amounts of data.
2. MapReduce - distributed data processing engine, based on a programming model with only two functions (Map and Reduce) (M. Chen et al., 2014).

Several other complementary technologies were developed around this, resulting in what is today referred to as the Hadoop Ecosystem. These technologies have extended Hadoop to include SQL-like data querying and data flow tools (Hive and Pig) (Oussous et al., 2017), NoSQL database systems (such as HBase and MongoDB) and a variety of other tools for stream processing (Storm), machine learning (MLib), graph analytics (GraphX), data integration (Sqoop), workflow (Oozie), coordination (Zookeeper) and web interface (Hue) (Landset et al., 2015).

Spark is an alternative to the original MapReduce data processing engine which has been gaining popularity (Twentyman, 2016), which may be explained by its increased ease of use (it currently provides several APIs for Java, Python, and R) and performance, made possible by its in-memory processing features (Shi et al., 2015). Spark can be used to deploy pre-trained machine learning models on Big Data. Spark jobs follow the same basic programming model as MapReduce and are made up of different stages related to the distinct operations that an application executes sequentially (K. Wang & Khan, 2015). Each stage is made up of several tasks that can be executed in parallel across the nodes of the cluster (K. Wang & Khan, 2015).

The Hadoop ecosystem with its filesystem-based data storage also provides a solution to large-scale analysis of unstructured data, which was difficult in SQL environments. These new types of data architectures are centered on the data lake, as opposed to the traditional data warehouse, which was made for structured data environments. Data lakes use different Hadoop ecosystem tools to manage different levels of data structure: from raw files to SQL-like tables (O'Leary, 2014).

3. METHODOLOGY

To answer the research questions of this study, an empirical study was conducted on a Portuguese Android app store, Aptoide. This platform is presented as a "social app store" where some common online social network features exist. Users can create their profiles and follow each other, they have access to a microblogging feature (through the "apps timeline" feature which was still active at the time of the dataset extraction), user-generated comments and app reviews. Additionally, users can download and share apps with their followers. It also has a search engine to find Android apps.

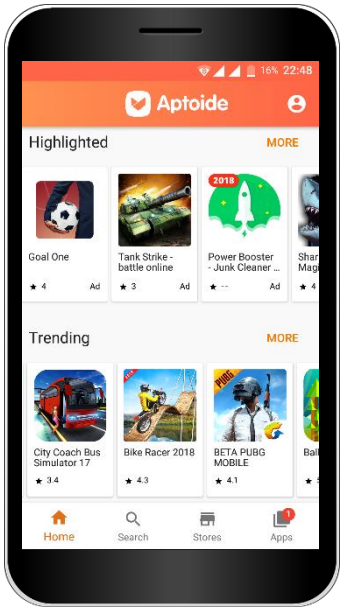


Figure 4 – Recommendations displayed in the Aptoide mobile app home.

Several different models will be compared using a novel profit-centric metric. The different deep learning models compared were defined to answer the three specific research questions. This means that we will do model comparisons across three levels: 1. Feature Embedding, 2. Data Sources and 3. Scoring.

3.1. PROPOSED MODEL

We propose a generic architecture for mobile app recommendations. The basic architecture of our model has three basic layers: feature embedding, feature extraction, and scoring. The proposed model is a fully connected deep network designed to score mobile apps for users using aggregated features at the user level. The raw data can include behavioral, social-network and user-generated content data.

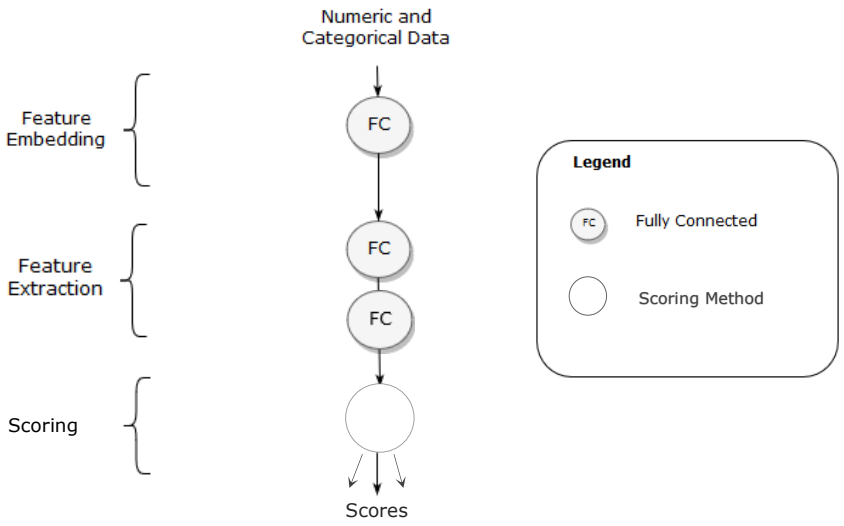


Figure 5 – Deep-Learning Architecture Overview

The three first layers are fully connected components with Relu activation functions. The embedding layer is pre-trained using a combined Word2Vec and Autoencoder procedure from both numerical raw features and one-hot encoded categorical variables. The resulting embeddings are fed into the following layer.

The feature extraction layers take the embedded features for each user in each moment and performs a dimensionality reduction.

These features will be used as input to the final scoring layer. Two variants of the architecture can be constructed using different scoring methods.

The standard approach is to employ Softmax. As a novel approach to recommender system design, we propose an alternative multiclass classification method based on Kernel methods. For this purpose, we will use Tensorflow's implementation of Kernel methods, which is based on Random Fourier Features (RFF) (Rahimi & Recht, 2007). By combining RFF with a softmax output node, we can implement Multiclass Kernel Logistic Regression (M-KLR) (Karsmakers et al., 2007). Tensorflow doesn't currently offer a differentiable SVM or multiclass SVM implementation, but Kernel Logistic Regression (KLR) has been empirically and analytically demonstrated as having the similar performance and behavior as SVM (Karsmakers et al., 2007), the main difference being the fact that it requires the entire dataset as opposed to only using support vectors to build a decision margin (Zhu & Hastie, 2005). As such, it is expected that M-KLR should behave similarly to multiclass SVM approaches.

This proposed architecture can serve as the foundation for the research design described below, where different variants of this architecture will be empirically tested.

3.2. EXPERIMENTAL SETUP

Three specific research questions can be formulated to satisfy the three specific research objectives previously presented:

1. What is the impact of using unstructured data versus only using structured data in the model performance?
2. What is the impact of embedding layer pre-training in the model performance?
3. How efficient are Kernel-based methods in the scoring layer?

The basic architecture to be tested is the recommender system architecture previously described. Different components will be omitted across models to answer each research question. The following table presents a description of the four models:

	Model 1	Three-layer network with fully connected (FC) feature embedding layer (numeric features), feature extraction layer, softmax output and no pretraining.
	Model 2	Four-layer network with one layer of FC feature embedding (numeric and categorical features) and two feature extraction layers, softmax output and no pretraining.
	Model 3	Four-layer network with one FC feature embedding (numeric and categorical features) and two feature extraction layers, softmax output and embedding pretraining.
	Model 4	Four-layer network with one FC feature embedding (numeric and categorical features) and two feature extraction layers, kernel softmax output and embedding pretraining.

The task in our experiment consists of scoring a batch of 10 apps for each user as a multilabel classification problem. We recommend an app if the score is over a threshold (commonly 0.5). If the user has previously acquired that app we consider it a true positive. If not, we consider it a false positive. The same logic is applied to negatives. Our research questions will be answered by the results of each model on this task.

To answer research question 1, we will compare the performance of models 1 and 2. To answer research question 2, we will compare the performance of models 2 and 3. To answer research question 3 we will compare the performance of models 3 and 4. The Python source code for the experiment is available at https://github.com/lgpintomkt/MasterThesis.

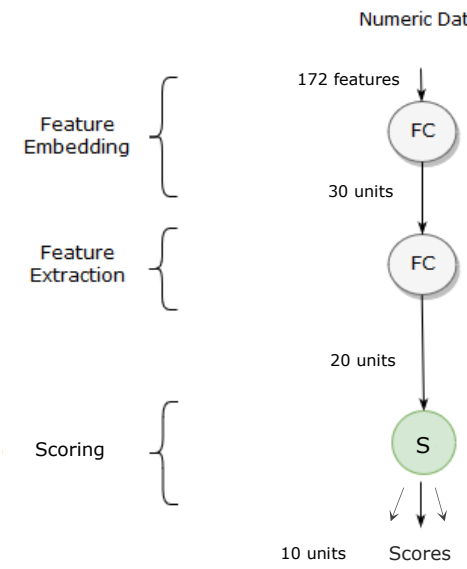


Figure 7 – Architecture of Model 1

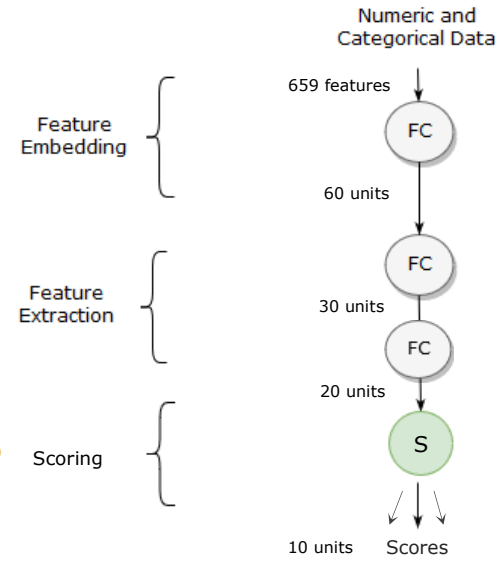
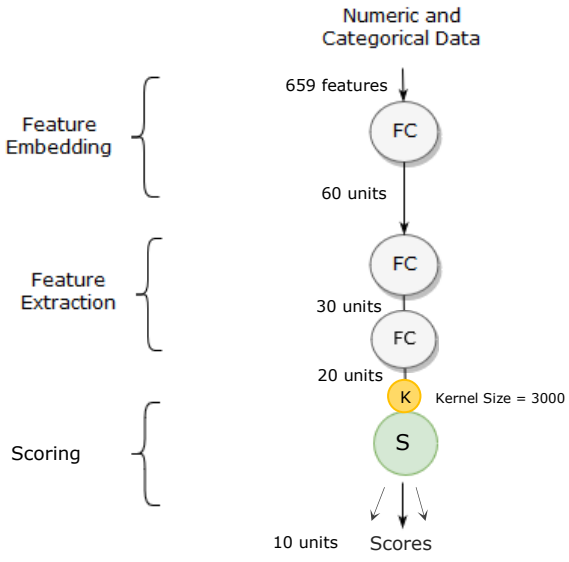


Figure 6 – Architecture of Model 2 and 3



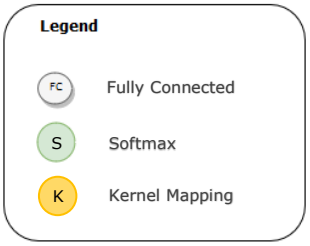


Figure 8 – Architecture of Model 4

3.3. MODEL EVALUATION

Our base model is a classification style model that provides probability scores for items. The different variations will be compared in an offline setting. Therefore, standard recommender system metrics such as Hamming Loss and Jaccard Index along with averaged versions of AUC, Precision, Recall and F1-Score can be employed to compare model performance. In the case of AUC, we will employ Macro averaging, while in the case of F1-Score, Precision and Recall we will use Micro-averaging consistent with the results of Forman and Scholz (2009). An additional Macro average F1-Score metric will also be included for comparison.

We propose an additional novel offline profit-based metric based on the concept of the estimated Average App Download Value ($\hat{V}_{download}$) which is derived from CLV metrics and the NBD-Dirichlet model (full mathematical derivation included in the Appendix A).

We considered that any true positive results in an increment of $+\hat{V}_{download}$ (revenue) while any false positive is counted as decrement of $-\hat{V}_{download}$ (opportunity cost). Additional cost variables were added to compare the efficiency of each models considering the model training, validation, testing and deployment:

$$\hat{V}_{download} = \sum_{t=1}^{h} \frac{nD(\bar{R}_t - \bar{C}_t)[\mathrm{B}(\alpha_j, S-\alpha_j) - \mathrm{B}(\alpha_j, S-\alpha_j+n)]\mathrm{B}(\alpha_j+1, S-\alpha_j+n-1)}{(1+\delta)^{t-1}\mathrm{B}(\alpha_j, S-\alpha_j)^2}$$

$$Profit_{model} = (U\tau_{model} - U\varphi_{model})\hat{V}_{download} - tUCost_{model\ deployment} - Cost_{\substack{training \\ validation \\ testing}}$$

The value for U corresponds to the size of the testing set. The parameters $\alpha, D, \bar{R}_t, \bar{C}_t, \delta$ and h were obtained from internal Aptoide data. The values for $n = 13.52$ and $S = 29.88$ were estimated from recent public industry reports (App Annie, 2018). The values for τ_{model} (true positive rate) and φ_{model} (false positive rate) come from the performance of each model during the testing stage.

The values for $Cost_{model\ deployment}$ and $Cost_{\substack{training \\ validation \\ testing}}$ come from the cloud platforms hourly costs that will be discussed in the following sections.

The deployed model execution time (t) will be estimated using a separate experiment conducted on a Spark environment. In this case, we are interested in the expected execution time for the entire Spark job associated with the model deployment. Remember that Spark jobs are made of multiple stages, and each stage contains several tasks running in parallel in a distributed manner (K. Wang & Khan, 2015). The Tensorflow model to be deployed is encapsulated in an object that will require only a single stage, with several tasks. The total job execution time is therefore given by the following expressions (K. Wang & Khan, 2015):

$$P = \sum_{i=1}^{H} CoreNum_i$$

$$JobTime = JobStartup + StageTime + JobCleanup$$

$$StageTime = StageStartup + max_{v=1}^{P} \sum_{i=1}^{R_v} TaskTime_{v,i} + StageCleanup$$

Where $CoreNum$ is the number of CPU cores of working node i and H is the number of working nodes in the cluster, R_v is the number of sequential tasks executed on CPU core v and P is the total number of CPU cores in the Spark cluster. The number of sequential tasks R_v under these conditions will be the number of user batches (RDD partitions) included in the testing experiment divided by the number of CPU cores. We can separate the total job time in two components as such:

$$Z = JobStartup + StageStartup + StageCleanup + JobCleanup$$

$$V = max_{v=1}^{P} \sum_{i=1}^{R_v} TaskTime_{v,i}$$

$$JobTime = Z + V$$

Since this experiment is to be executed in a single machine cluster we will simulate the typical industrial cluster configurations, by estimating the V and Z components using Ψ sequential trials (the number of trials meant to simulate the number of nodes in the cluster). To simplify the calculations, we will employ a single processing CPU core. We can define two finite sets $\mathcal{V} = \{\mathrm{v}_1, \ldots \mathrm{v}_\Psi\}$ and $\mathcal{Z} = \{\mathrm{z}_1, \ldots \mathrm{z}_\Psi\}$ which represent the Ψ observations of Z and V. Each element of $\mathcal{V}$ represents the sum of all task durations in the trial. From these, we can define two estimators for the actual values. For Z we are looking for the expected value $\mu(Z)$. For V we are looking for the maximum expected value $\mu_*(V)$, which can be estimated using the maximum estimator (ME) (van Hasselt, 2013). Therefore, we have:

$$V \approx \hat{V} \equiv \hat{\mu}_*(V) \equiv \max(\mathcal{V})$$

$$Z \approx \hat{Z} \equiv \hat{\mu}(Z) \equiv \frac{\sum \mathcal{Z}}{|\mathcal{Z}|}$$

The final $\hat{t}$ will then be given by:

$$\hat{t} = \max(\mathcal{V}) + \frac{\sum \mathcal{Z}}{|\mathcal{Z}|}$$

To simulate the typical Big Data setups, we will assume $\Psi = 12$. This cluster size was the same used by K. Wang & Khan (2015) in their experiment, and consistent with the reported industry best practices (Fujitsu, 2017). Even though much larger cluster sizes may occur for specific tasks (Apache Foundation, 2018a), these do not appear to be common. The runtimes were extracted from the Spark Web UI which displays runtimes for tasks, stages, and jobs. We will assume $\xi = 2.958\ USD/hour$ since it's the current pricing of a general purpose m5 12 node cluster on the most popular infrastructure provider (AWS), which includes both the EC2 and EMR costs (Amazon Web Services, 2018).

3.4. DATASET DETAILS

3.4.1. Raw Data

The dataset employed to train and test the different models was extracted from the Aptoide's Big Data lake running on Amazon Web Services (AWS). Table 1 below lists the features available on the dataset.

Table 1 – Features

Geographic[2]	Demographic	Technographic	Behavioral	Social
Latitude Longitude	Language	Android Version Device Model	# Downloads # Searches # Clicks App Downloads Search Queries Clicked Items	Social Network Topology

3.4.2. Feature Engineering

As previously stated, the raw features were subjected to a feature engineering process which will be described in the following section.

The geographic features were stored as latitude/longitude coordinates (in degrees) for each user, however, these cannot be fed directly to a deep learning model. A method similar to Locally Linear Embedding with Geodesic Distance (Varini, Degenhard, & Nattkemper, 2005), based on classical Multidimensional Scaling (MDS) and KNN was employed to extract usable features. The first step was to convert the coordinates in degrees to radians. Once in radian format, a random subsample of 40.000 users was taken from the original sample. From these, a validation and test subsample were also extracted, each being 15% of the overall subsample, with the remaining 70% were used for training.

A distance matrix between all users on the subsample was computed. The employed distance metric was the implementation of the Karney geodesic distance (2013) available on the Python geocoding library GeoPy. Karney's method is currently the state of the art in geodesy, and its widely used to estimate the distance between two latitude/longitude pairs, building upon earlier methods such as Vincenty's formulae (1975). The next step was the application of Singular Value Decomposition (SVD) to the distance matrix, where two eigenvectors were retained. Finally, the embedding was extended to the entire dataset using KNN regression for each eigenvector, where the best K parameter was found to be 5. To assess goodness of fit, the average R^2 of both dimensions was used to evaluate distance reconstruction quality on the test set. The final R^2 value was 0.989, indicating a good feature embedding accuracy. This very high goodness of fit might be explained by having only city-level geographic coordinates, resulting in a perfect coordinate overlap between nearby users.

For the demographic and technographic features, the top 99 most frequent occurrences for each variable were extracted. A 100th feature was included to represent the "other" hypothesis. A one-hot encoded vector was computed for each user.

The behavioral features included clickstream/touchstream variables, search queries, and app download data. From these, app download data were categorical (the user either downloaded or not

[2] The coordinates for each user are the coordinates of the centre of each user's first login city.

a certain app), and the same procedure of the demographic/technographic features was applied. The clickstream data was encoded in a manner consistent with the tidy data approach (Wickham, 2014), where a column encoded the number of clicks/touches of each user, and other columns encoded the percentage of those clicks that were of a certain category/type.

The search query data was subjected to a more complex procedure. The objective was to obtain a categorical one-hot encoding. However, such an encoding should reflect the similarity of mistyped terms or slightly different terms that may refer to the same intent. The approach we followed was derived from standard procedures applied in Natural Language Processing (NLP) based on spell-checking dictionaries (Lehal, 2007). The approach here relied on building a dictionary that would be used to perform one-hot encoding of the search terms. To build this dictionary a sample of the most common 10.000 terms was extracted. A distance matrix was computed for all terms using Damerau-Levenshtein distance. An MDS procedure was then applied to this matrix, where two dimensions were retained. Using these dimensions, a hierarchical clustering procedure was applied, using a height cutoff of 12, resulting in a total of 100 clusters. The final dictionary was built by extracting the most common term from each cluster. The spell-checking procedure was applied to the entire data set by replacing each term with the most similar term according to the Damerau-Levenshtein distance. The search terms employed by the users were then one-hot encoded using the dictionary entries.

The feature extraction for the social network topology data was done using a procedure derived from the DeepWalk method already described. (Perozzi et al., 2014). A random walking was done on this network, with walk length 10, and 0% restarting probability. The resulting random walks allowed us to generate a one-hot encoded vector for each user based on the top 100 most popular profile IDs following a bag-of-words logic. Two centrality measures were also computed and added to the final features: indegree and outdegree. These two features correspond to the number of followers and followed respectively, and capture some additional global network properties that are not so well captured by the random walking procedure (Cui, Wang, Pei, & Zhu, 2017; Dalmia & Gupta, 2018).

The pre-training procedure for the network was the naïve method described in the CENE experiments (Sun et al., 2016) since it was reported to give good results. The same logic of CENE was followed, where we assumed to be observing a directed network where each user is associated not only with other user profiles but also apps, search queries, device models, Android versions, and languages. N-grams were generated for each user from the one-hot encoded variables. The embedding of this network is then achieved using Word2Vec for the categorical variables and an Autoencoder implemented in Tensorflow for the numerical variables.

Finally, every feature was normalized using min-max normalization resulting in values between 0 and 1. The total number of extracted features amounts to 659, of which 172 are numerical, the remaining ones being categorical (one-hot encoded). Users with less than 97% sparsity were kept. The final data is represented as a rank-3 tensor W such that each element $W_{a,b,c}$ corresponds to the value of feature a for user b on day c, over the first 30 days of using the platform. From this tensor, two different matrixes were then extracted (one with just the 172 numerical features and another one with the full 659 feature-set), where the values over the 30 days are summarized for each user. These condensed datasets are going to be the inputs for our models. The pre-processing procedure was executed in a local AMD Quad-Core Processor A4-5000 (1.5 GHz) CPU machine, lasting approximately 8 days.

4. RESULTS

The processed dataset was divided into three parts making up the training, validation and test sets, roughly corresponding to 86%, 2% and 12% of the overall data (74.209 users). The model training, validation, and testing were done on a Nvidia Tesla K80 GPU instance executing a Tensorflow environment using the FloydHub service (https://www.floydhub.com/).

4.1. Model Training and Validation

The training of models 3 and 4 involved a pretraining of the feature embedding architectural component using a Word2Vec (for the unstructured data) and Autoencoder implementation in Tensorflow. Both models were trained over 10 epochs with gradient descent. The resulting embedding parameters where concatenated (cross-loadings were randomly initialized with lower values close to zero) and used as the initialization tensor for the first layer of the final architecture. This pretraining was followed by a global training stage.

The training set was divided into 32 batches of 2.000 users (64.000 overall). The training procedure was done under the classic early stopping method (Prechelt, 2012) dependent on the loss on the entire validation set computed at each epoch (if the validation loss increases on this epoch, the training stops, and the previous weights are retained). The Adam optimizer with the standard parameters (Kingma & Ba, 2014) was used to perform the network training backpropagation.

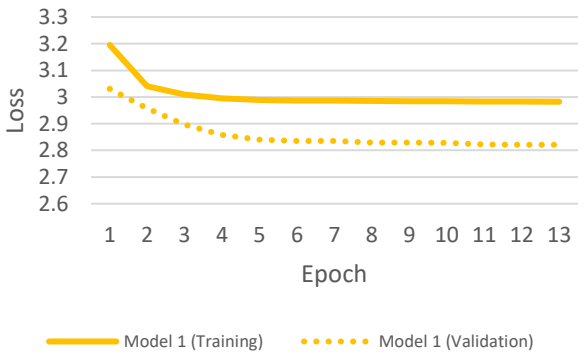


Chart 3 – Training and Validation Loss (Model 1)

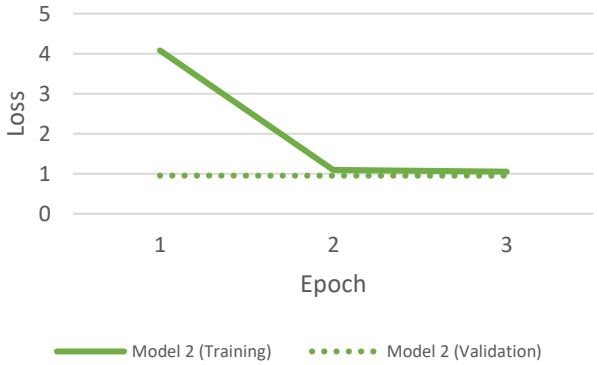


Chart 2 - Training and Validation Loss (Model 2)

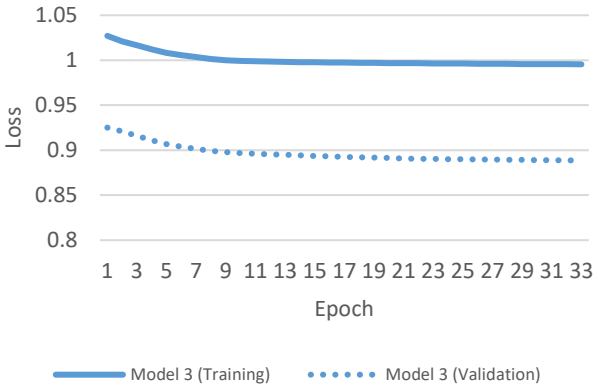


Chart 5 - Training and Validation Loss (Model 3)

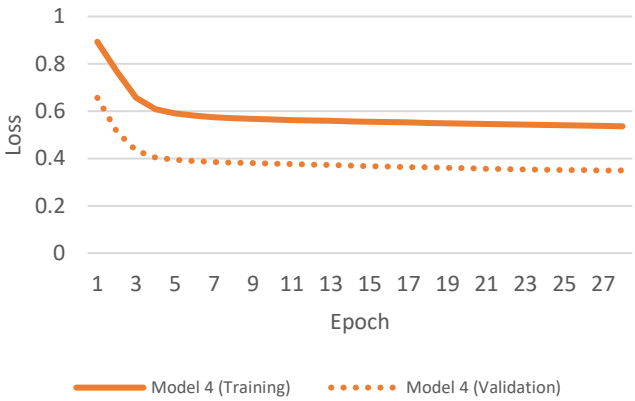


Chart 4 - Training and Validation Loss (Model 4)

4.2. Model Testing

The model testing was done in two parts. The first part was meant to test the model accuracy using standard metrics on the testing set on the Nvidia Tesla K80 GPU instance where the training and validation were done. The average pricing per hour of the infrastructure provider (FloydHub) was used to compute the total costs of training, validation, and testing (1.22 USD/hour).

Table 2 – Computational Statistics of Model Training, Validation and Testing.

	Model 1	Model 2	Model 3	Model 4
Max CPU Utilization	56%	72%	74%	**55%**
Max GPU Utilization	5%	**0%**	6%	27%
Execution Time (minutes:seconds)	03:34	**03:22**	05:35	04:38
Total Cost of Training, Validation and Testing (USD)	$0.071	**$0.067**	$0.111	$0.093

A portion of the testing set (2250 users) was then allocated to a secondary test to obtain the Spark execution time component.

Table 3 – Computational Statistics of Model Deployment

	Model 1	Model 2	Model 3	Model 4
Estimated Serving Latency (ms)	**7.56**	8.45	15.11	15.34
Cost per User	**$ 0.0003104**	$0.0003470	$0.0006209	$0.0006304
Total Cost of Deployment (USD)	**$ 0.70**	$0.78	$1.40	$1.42

The Spark estimated serving latency was measured separately on an AMD Quad-Core Processor A4-5000 (1.5 GHz) CPU machine executing a Spark server on an Ubuntu VM through a Windows 10 host. A single CPU core of this machine was used to execute the experiment. The obtained execution time for each model was used to estimate the Information Technology (IT) infrastructure cost component. A Spark script was executed directly on the PySpark command line tool, which imports the Tensorflow graph from the local disk before applying it to each row of an RDD, according to the method proposed by Databricks (Hunter, 2016).

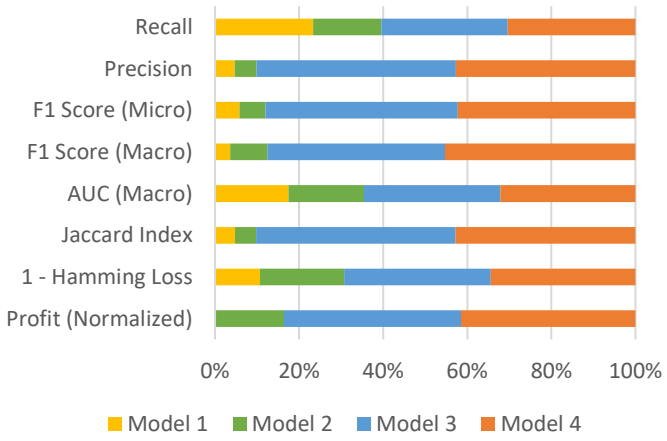


Chart 6 – Experimental Results of Model Testing

The values from both tests were combined to compute the final Profit value for each model.

Hamming Loss, Jaccard Index, AUC (Macro), F1 Score (Macro and Micro), along with Precision and Recall were also compared across models.

In the table below, we present the results of the testing for each model (bold indicates top performer).

Table 4 – Experimental Results of Model Testing (in detail)

	Model 1	Model 2	Model 3	Model 4
Hamming Loss	0.713	0.462	**0.065**	0.076
Jaccard Index	0.031	0.033	**0.314**	0.284
AUC (Macro)	0.523	0.540	**0.973**	0.964
F1 Score (Macro)	0.050	0.123	0.585	**0.626**
F1 Score (Micro)	0.061	0.065	**0.478**	0.442
Precision	0.032	0.035	**0.320**	0.288
Recall	0.731	0.507	0.944	**0.951**
True Positive Rate	2.30%	1.60%	2.98%	**3.00%**
True Negative Rate	26.36%	52.24%	**90.54%**	89.45%
False Positive Rate	70.49%	44.61%	**6.31%**	7.40%
False Negative Rate	0.85%	1.56%	0.18%	**0.16%**
Profit (normalized)	0%	39%	**100%**	98%

The results show that model 1 is the lowest performer in all metrics except Recall (where it significantly outperforms model 2). Except for Recall and Hamming Loss, model 1 and 2 behave similarly, but model 2 beats model 1 overall.

Models 3 and 4 outperform the other two models significantly across all metrics. The difference between the performance of models 3 and 4 is less clear. These two models seem to have a low difference magnitude overall. Still, model 3 outperforms 4 in most metrics except F1 Score (Macro), Recall, True Positive and False Negative Rate. Since Recall depends on the values of the True Positive and False Negative rates we can say that Recall is the main difference between models 3 and 4.

5. CONCLUSION

Our study had the goal of testing a deep-learning architecture for app recommendations in the context of mobile app stores. We pre-processed several features related to user geography, demography, technograhy, past app acquisition, clickstream/touchstream, search queries and social graph. Some of this data were numeric and some were categorical. An unsupervised pretraining procedure was applied to some of the models to generate embeddings for both types of features. Finally, we designed and implemented an experiment that would help us understand our three research questions related with the impact of using unstructured data, the impact of the pretraining procedure, and the efficiency of employing Kernel methods. In the following sections, we will present an interpretation of the results, along with its discussion, implications, limitations of our study and directions for future research.

5.1. DISCUSSION

Our profit metric suggests that the most efficient deep-learning architecture for mobile app stores is model 3, which is a four-layer network with one FC feature embedding and two feature extraction layers, softmax output and embedding pretraining. We will now discuss our results in detail, considering our specific research questions and the previous empirical studies.

1. What is the impact of using unstructured data versus only using structured data in model performance?

Model 1 used 172 raw features, while model 2 used 659. By comparing the performance of model 1 and 2 we conclude that using unstructured data improves the model performance slightly but given the large discrepancy in the number of features, we expected a larger difference magnitude. This improvement can be seen in the accuracy metrics (Hamming Loss and Jaccard Index). By looking at our economic efficiency indicator (Profit), we can see that model 2 is +39% more efficient than model 1.

Additionally, the training for model 2 is significantly faster, taking only 3 epochs to reach convergence. The GPU consumption during the procedure is 0%, which means that this model's fast convergence did not require a GPU instance, which would have resulted in lower costs (and a higher efficiency improvement over model 1).

These results are partially consistent with previous studies. In an experiment by Tan, Xu and Liu (2016), the model exposed to privileged information saw only a modest improvement in performance which is consistent with our results. However, the training time and serving latency increased significantly, while in our case the training time was reduced drastically, and the serving time didn't increase much. We argue that this difference is due to the nature of our unstructured data, which possibly contains information that is highly valuable for recommendations, such as the revealed intent from search queries. This would have made the model training faster, while also increasing accuracy.

2. What is the impact of embedding layer pre-training in the model performance?

Model 2 and model 3 shared the same architecture, but model 2 was not subjected to a layer-pretraining. The results show that the embedding layer pre-training drastically improved performance across the board. These gains in performance are especially notable when it comes to the AUC (Macro)

metric, which went from 0.540 to 0.973. Additionally, our efficiency indicator (Profit) saw an increase of +60%.

These results are partially consistent with the literature. A study by Glorot, Bordes, & Bengio (2011) tested the effect of embedding layer pretraining in Deep Rectifier Networks (networks with Relu activations like ours). While the pretrained models saw a slight improvement, previous results with tanh or softplus activations saw much deeper improvements (Glorot et al., 2011). A simulation study by Erhan, Manzagol, Bengio, Bengio, & Vincent (2009) suggests that when pretraining is not employed the probability of finding poor local minima increases. We believe this is the most probable cause to explain the difference in performance of models 2 and 3.

The Cheng et al. (2016) study in the field of recommender systems for mobile app stores also employed feature embedding procedures for the categorical features, but no benchmark is provided for models without pretraining. The study proposes a novel Wide & Deep architecture that combines both embedded features fed through a deep part, and the raw features fed through the wide part. The best performance is achieved by combining both types of features. The model without the wide part still has a comparable performance, suggesting that most of the predictive accuracy is coming from the pretrained embedded features, which is consistent with our results.

3. How efficient are Kernel-based methods in the scoring layer?

Model 3 and 4 have a nearly identical architecture, with the difference that model 4 adds a kernel mapping layer before the final classifier based on RFF. We can measure the efficiency of the kernel mapping by comparing the Profit metric of the two models. We can see that model 3 has a +2% gain in efficiency when compared with model 4. This is surprising since we expected that the kernel mapping would be able to improve the results of the classifier sufficiently to make it more cost effective.

One possible explanation for this surprising result would be that the task of our experiment was already performing very well, not leaving a lot of room for improvement. However, this doesn't account for the fact that model 4 underperforms in most metrics (even if the difference is small). The only performance improvement appears to be on the Recall metric (and the associated true positive and false negative rates), which suggests that Kernel methods improve Recall, which is not consistent with prior studies with SVM (Maroco et al., 2011; Romero & Koochak, 2015). We could argue however that M-KLR may possess different properties that are not observed in SVM and multiclass SVM. Additionally, we must consider the fact that our implementation is being tested in a Deep Learning framework, as opposed to the Shallow contexts where those empirical studies were conducted. Another key difference is the fact that we used an RBF approximation based on stochastic methods (RFF).

Another observation is that our Kernel model consumed more GPU resources (27%), and took less time to converge than model 3 while maintaining a comparable latency in the Spark environment. The reduced efficiency, when compared with model 3, is explained by the model performance in terms of true and false positive rates. Model 4 is the model that consumes most resources but still performs below the results of Cheng et al. (2016) in terms of serving latency. Keep in mind however that our models are scoring only a tiny batch of 10 apps at a time, while the Cheng model is scoring batches between 50 and 200 apps in parallel.

Given that the sole difference between models 3 and 4 is the existence of a Kernel mapping in the latter, we can conclude that Kernel-based methods might not be sufficiently efficient to build a mobile app recommendation engine at Big Data scale, but more research is necessary to understand why these methods are not effective.

5.2. IMPLICATIONS

Our results have implications for practitioners working in the implementation of recommender systems in mobile app stores. Our results are consistent with previous research (Cheng et al., 2016; Covington et al., 2016) in confirming the overall efficiency of deep learning methods for large-scale recommender systems, which require low serving latency. Additionally, it has been shown that deep learning methods allow us to combine different data sources and different data types (numeric and categorical) which other traditional CF methods are not able to. Our experiments confirm that by augmenting the dataset with categorical features we obtain better models.

In our experiments the preprocessing time took the longest time (8 days), but practitioners should consider investing resources to perform the preprocessing stage in a distributed Big Data environment such as Spark. Currently several parallel implementations of unsupervised learning methods like Word2Vec (Apache Foundation, 2018c), SVD and PCA are already available (Apache Foundation, 2018b). Also, current RDD and Spark Dataframe technology can be queried using SQL or SQL-like methods, meaning that all the necessary data wrangling can be done more efficiently in a distributed environment. This might facilitate the necessary procedures to produce features in the required format. The resulting preprocessed features can then be used to train the deep learning models in the same environment, as it appears to be the current trend in large mobile social platforms like Facebook (Hazelwood et al., 2018).

A key factor for the successful ingestion of unstructured (categorical) data types is the usage of unsupervised pretraining on the embedding layer. In our experiments, the models without pretraining became stuck in poor local minima. The unsupervised pretraining procedure boosted the model efficiency by +60% (as measured by our profit metric), while also raising the AUC from 0.540 to 0.973. This suggests that pretraining is a critical success factor when augmenting the data with categorical features. By employing distributed GPU technology (Moritz, Nishihara, Stoica, & Jordan, 2015), it becomes easier and faster to perform the embedding layer pretraining using Big Data. In fact, this methodology might be easily extended to all layers, resulting in a global layer-wise pretraining procedure, that might be able to further boost these models.

Another key implication is the fact that more sophisticated approaches, such as using app scoring procedures derived from Kernel methods, does not guarantee efficiency at Big Data scale. Our results show that the most sophisticated model not only underperformed in most accuracy metrics (except Recall) but also showed a lower economic efficiency, consuming more resources for deployment. Machine learning practitioners and data scientists seeking to introduce recommender systems in similar environments should consider using simpler methods based on softmax output nodes, which should allow them to fulfill the business requirements at a lower cost.

Also, our experiments show that Tensorflow models can be easily distributed using Spark. While some prior work already exists (Hunter, 2016; H. Kim, Park, Jang, & Yoon, 2016; Moritz et al., 2015), this is

one of the few real life experiments conducted by combining both technologies to deploy deep learning models at Big Data scale.

Finally, this work also has implications for IT/IS project managers tasked with implementing recommender systems in the context of mobile app stores. The concept of Average App Download Value might be helpful for the evaluation of the Earned Value of the modelling tasks within the project, according to a traditional Earned Value Management (EVM) framework (Batselier & Vanhoucke, 2017) widely used in project management (Project Management Institute, 2017, p. 126). From a larger information management perspective, this concept might also be useful to evaluate the impact of the recommender system across the IT, IS, Business Process, Business Benefit and Business Strategy domains (Bytheway, 2014, p. 26), since it incorporates high level marketing effects (consumer behavior and competitive environment) which can then be compared with the overall technological infrastructure costs, like in our profit measure.

5.3. LIMITATIONS

Our study has several limitations. The main limitation has to do with the fact that we've only employed offline evaluation of the models. This has to do with the costs associated with deploying a recommender system, which made the online testing not feasible. The only alternative would have been using a toy setting with a sample of users, but given the time constraints of this thesis, recruiting a large enough sample would have made it difficult. Such methods, however, would have allowed us to measure other properties of the recommender system other than its accuracy and economic efficiency, such as novelty and serendipity.

The second limitation is the task that was used to test the models. We considered a batch of 10 apps that were being scored for all users. In other similar experiments conducted in online settings, much larger batches were employed, with numbers ranging from 50 to 200 per user at a time. In fact, some studies employ extreme multiclass experiments at a large scale (Cheng et al., 2016; Covington et al., 2016), using industrial hardware and software. Three issues appeared when trying to extend the size of the set of classes. First, we didn't have access to industrial grade systems to train, validate, test and deploy models at such an extreme scale. Second, during the pre-processing stage, we ran into memory issues when trying to perform feature engineering procedures. Initially, we had a set of close to 2000 features for each user, with nearly half a million users. Manipulating such a large matrix became impossible given the computational constraints of our hardware. In that context, we could have used a larger set of apps to test the models since we had initially selected the top 500 most popular apps. In the end, we were limited to 100 app acquisition columns per user. Third, we still ran into issues when trying to score more than the 10 most popular apps, given the extreme high sparsity of the dataset we were working with. When we went beyond the top 10 apps we had mostly zeroes, which made it impossible to successfully train the models. In fact, when we attempted to train these models we noticed that they failed to converge over many epochs, and when they did, they return AUC values close to 0.5.

Another limitation is the fact that our model deployment test conducted in Spark was done on a VM as opposed to an actual Spark cluster. While the software environment closely matches the real-world settings, we only employed a single CPU on a cluster with a single node. Therefore, we had to simulate distributed training by running several trials sequentially within the Spark environment. While we tried to replicate as much as possible the real world industrial grade settings, it's possible that our

experiment failed to fully capture the dynamics of a Big Data processing cluster, and thus it might have negatively impacted the quality of our results.

Also, we've tested the model execution time in a batch-processing oriented Spark environment. However, real-life model deployment at the Big Data scale usually implies creating an interface for other systems to use that information. Reference enterprise and software architectures have been proposed for Big Data which suggest the use of Data Loading functionalities to periodically update the serving stores from the data in the raw store (HDFS) (Pääkkönen & Pakkala, 2015). This serving store can be based on traditional SQL datastores, allowing for microservices or more traditional service-oriented architecture (SOA) web services to be deployed to serve the model outputs (Xiao, Wijegunaratne, & Qiang, 2017). These solutions would increase response time, making our results valid only for relative model comparison, since actual serving times would be larger.

A final limitation is the fact that we didn't use any temporal sequence modeling techniques. Our initial plan was to also test three additional variants of our base model based on RNN (Elman, 1990), LSTM (Hochreiter & Urgen Schmidhuber, 1997) and DNC (Graves et al., 2016). These temporal sequence networks would be used to extract features either before the static feature extraction layer or after it. And in fact, our pre-processing procedure was designed to generate a rank-3 tensor W such that each element $W_{a,b,c}$ corresponds to the value of feature a for user b on day c, over the first 30 days of using the platform. However, we once again ran into sparsity related issues, given that most entries of that tensor were zeroes, making it impossible to successfully train these models. What we noticed was that only by using aggregated user level features without the time component, and by using the top 10 apps, we were able to obtain viable models. Current research trends are focused in using session level recommendations (Tan et al., 2016), which leverages temporal sequence networks, and therefore it would have been relevant to test if a general architecture could be efficient for these kind of problems.

5.4. Directions for Future Research

Given the research questions we set out to answer and the limitations of our study we will discuss some possible directions for future research.

First, we can extend our study by testing the same models using different datasets. This will also us to confirm if our results generalize across contexts. For this purpose, we might look to secure additional data sets of other independent Android app stores. In alternative, we might also employ public datasets from other domains, which are widely available (Fortes, 2018). We should test the same models in an online and/or a user study setting, allowing us to gather more information on several other desirable properties of the recommender system such as novelty and serendipity using behavioral and attitudinal metrics. Also, our experiment didn't include any temporal sequence network due to data set limitations, which are the state of the art in session-based recommendations (Tan et al., 2016). Future studies should extend our experiment to temporal sequence contexts using data with less sparsity.

In this thesis, we only considered the effects of the model accuracy and efficiency, but it is known that the user interface plays a dominant role in the success of the recommendations (Zheng, Wang, Zhang, Li, & Yang, 2010). No study to date has addressed the impact of the mobile app store interface in the perceived quality of recommendations. Finally, the model deployment experiment should be extended to a real-world industrial cluster or at least a multi-node cluster in a laboratory setting. This will give

us greater confidence on the quality of the results we've obtained, while also contributing to the nascent academic research in the field of distributed GP-GPU and the efficient deployment of deep-learning models at Big Data scale.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://doi.org/10.1109/TIP.2003.819861.

Abdullah, A., Veltkamp, R. C., & Wiering, M. A. (2009). An ensemble of deep support vector machines for image categorization. *SoCPaR 2009 - Soft Computing and Pattern Recognition*, (1), 301–306. https://doi.org/10.1109/SoCPaR.2009.67

Abernethy, J., Bach, F., Evgeniou, T., & Vert, J.-P. (2008). A new approach to collaborative filtering: operator estimation with spectral regularization. *The Journal of Machine Learning Research*, *10*, 803–826. https://doi.org/10.1145/1577069.1577098

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-Based Learning Algorithms. *Machine Learning*, *6*(1), 37–66. https://doi.org/10.1023/A:1022689900470

Aljukhadar, M., Senecal, S., & Daoust, C. E. (2010). Information overload and usage of recommendations. In *CEUR Workshop Proceedings* (Vol. 612, pp. 26–33).

Alsheikh, M. A., Niyato, D., Lin, S., Tan, H.-P., & Han, Z. (2016). Mobile big data analytics using deep learning and apache spark. *IEEE Network*, *30*(3), 22–29. https://doi.org/10.1109/MNET.2016.7474340

Amadeo, R. (2016). The Google Play Store scraps Google+ integration. Retrieved July 29, 2018, from https://arstechnica.com/gadgets/2016/08/the-google-play-store-scraps-google-integration/

Amatriain, X. (2013). Mining large streams of user data for personalized recommendations. *ACM SIGKDD Explorations Newsletter*, *14*(2), 37. https://doi.org/10.1145/2481244.2481250

Amazon Web Services. (2018). Amazon EMR Pricing. Retrieved June 10, 2018, from https://aws.amazon.com/emr/pricing/?nc1=h_ls

Anderson, C. (2008). *The Long Tail, Revised and Updated Edition: Why the Future of Business is Selling Less of More*. *Word Journal Of The International Linguistic Association*. https://doi.org/30

Apache. (2017). Feature Extraction and Transformation - RDD-based API. Retrieved June 8, 2017, from https://spark.apache.org/docs/2.1.0/mllib-feature-extraction.html#word2vec

Apache Foundation. (2018a). Apache Spark FAQ. Retrieved June 10, 2018, from https://spark.apache.org/faq.html

Apache Foundation. (2018b). Dimensionality Reduction - RDD-based API. Retrieved August 9, 2018, from https://spark.apache.org/docs/latest/mllib-dimensionality-reduction.html#dimensionality-reduction-rdd-based-api

Apache Foundation. (2018c). Feature Extraction and Transformation - RDD-based API. Retrieved August 9, 2018, from https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html

App Annie. (2017). *App Annie App Monetization Report 2016*. Retrieved from http://files.appannie.com.s3.amazonaws.com/reports/1116_App_Monentization_Report_EN.pdf?mkt_tok=eyJpIjoiWXpSaE5UQXlZemcxWm1NMCIsInQiOiJxZE1Dd0J0RnRiOWFUamJaeEc0YkFqZEpPQkNEc3JGZGtjYVNPS1ZseGNlckRTb2FjWTM3UVR3ejRqbnR1bzRtMnFBbkozWmNcL2Z0d29BNUVzcVB2cUtRaHV

App Annie. (2018). *App Annie 2017 Retrospective*. *App Annie*.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., … Lacoste-Julien, S. (2017). A Closer Look at Memorization in Deep Networks. Retrieved from http://arxiv.org/abs/1706.05394

Aurier, P., & Mejía, V. (2014). Multivariate Logit and Probit models for simultaneous purchases: Presentation, uses, appeal and limitations. *Recherche et Applications En Marketing (English Edition)*, *29*(2), 75–94. https://doi.org/10.1177/2051570714535531

Ayeh, J. K., Au, N., & Law, R. (2013). "Do We Believe in TripAdvisor?" Examining Credibility Perceptions and Online Travelers' Attitude toward Using User-Generated Content. *Journal of Travel Research*, *52*(4), 437–452. https://doi.org/10.1177/0047287512475217

Barragáns-Martínez, A. B., Costa-Montenegro, E., Burguillo, J. C., Rey-López, M., Mikic-Fonte, F. A., & Peleteiro, A. (2010). A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition. *Information Sciences*, *180*(22), 4290–4311. https://doi.org/10.1016/j.ins.2010.07.024

Batselier, J., & Vanhoucke, M. (2017). Improving project forecast accuracy by integrating earned value management with exponential smoothing and reference class forecasting. *International Journal of Project Management*, *35*(1), 28–43. https://doi.org/10.1016/j.ijproman.2016.10.003

Beel, J., & Langer, S. (2013). A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems, *xx*(x). https://doi.org/10.1007/978-3-319-24592-8_12

Bennett, J., & Lanning, S. (2007). The Netflix Prize. *KDD Cup and Workshop*, 3–6. https://doi.org/10.1145/1562764.1562769

Bilgihan, A., Kandampully, J., & Zhang, T. (Christina). (2016). Towards a unified customer experience in online shopping environments: Antecedents and outcomes. *International Journal of Quality and Service Sciences*, *8*(1), 102–119. https://doi.org/10.1108/IJQSS-07-2015-0054

Blattberg, R. C., Kim, B.-D., & Neslin, S. A. (2008). *Database Marketing - Analyzing and Managing Customers*. Springer Berlin Heidelberg.

Bloch, F., Jackson, M. O., & Tebaldi, P. (2016). Centrality Measures in Networks. https://doi.org/10.2139/ssrn.2749124

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, *46*, 109–132. https://doi.org/10.1016/j.knosys.2013.03.012

Bosagh Zadeh, R., Meng, X., Ulanov, A., Yavuz, B., Pu, L., Venkataraman, S., … Zaharia, M. (2016). Matrix Computations and Optimization in Apache Spark. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16* (pp. 31–38). New York, New York, USA: ACM Press. https://doi.org/10.1145/2939672.2939675

Bound, J. A. (2009). The S Parameter in the Dirichlet-NBD Model : a Simple Interpretation. *Journal of Empirical Generalisations in Marketing Science*, *12*(3), 1–7. Retrieved from https://www.empgens.com/article/the-s-parameter-in-the-dirichlet-nbd-model-a-simple-interpretation/

boyd, danah m., & Ellison, N. B. (2007). Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, *13*(1), 210–230. https://doi.org/10.1111/j.1083-6101.2007.00393.x

Brand, M. (2003). Fast online SVD revisions for lightweight recommender systems. *Proceedings of the 2003 SIAM International Conference on Data Mining*, *35*(TR-2003-14), 37–46. https://doi.org/10.1137/1.9781611972733.4

Brynjolfsson, E., Hu, Y. (Jeffrey), & Simester, D. (2011). Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales. *Management Science*, *57*(8), 1373–1386. https://doi.org/10.1287/mnsc.1110.1371

Brynjolfsson, E., Hu, Y. J., & Smith, M. D. (2006). From Niches to Riches: Anatomy of the Long Tail. *Sloan Management Review*, *47*(4), 67. https://doi.org/10.2139/ssrn.918142

Burke, R. (2000). Knowledge-Based Recommender Systems. In *Encyclopedia of Library and Information Science: Volume 69 - Supplement 32* (pp. 167–197). https://doi.org/10.1007/978-3-319-29659-3_5

Bytheway, A. (2014). *Investing in Information - The Information Management Body of Knowledge*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-11909-0

Campos, V., Sastre, F., Yagües, M., Bellver, M., Giró-i-Nieto, X., & Torres, J. (2017). Distributed training strategies for a computer vision deep learning algorithm on a distributed GPU cluster. *Procedia Computer Science*, *108*, 315–324. https://doi.org/10.1016/j.procs.2017.05.074

Chen, F. (2016). NBD-Dirichlet Model of Consumer Buying Behavior for Marketing Research. CRAN. Retrieved from https://cran.r-project.org/web/packages/NBDdirichlet/NBDdirichlet.pdf

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, *19*(2), 171–209. https://doi.org/10.1007/s11036-013-0489-0

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., … others. (2016). Wide & deep learning for recommender systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 7–10. https://doi.org/10.1145/2988450.2988454

Cho, Y. H., Kim, J. K., & Kim, S. H. (2002). A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, *23*(3), 329–342. https://doi.org/10.1016/S0957-4174(02)00052-0

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-Based Models for Speech Recognition. *Nips*, 577–585. https://doi.org/10.1016/j.asr.2015.02.035

Christopher, M. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media. Retrieved from https://scholar.google.co.in/scholar?hl=en&as_sdt=0%2C5&q=C.+M.+Bishop%2C+Pattern+Recognition+and+Machine+Learning.+New+York%2C+NY%2C+USA%3A+Springer%2C+2006&btnG=

Condli, M. K., Madigan, D., Lewis, D. D., & Posse, C. (1999). Bayesian Mixed-Effects Models for Recommender Systems. In *ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5B779C69031F4141C0276B05A0092A38?doi=10.1.1.10.6090&rep=rep1&type=pdf

Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16*, 191–198. https://doi.org/10.1145/2959100.2959190

Cui, P., Wang, X., Pei, J., & Zhu, W. (2017). A Survey on Network Embedding. Retrieved from

http://arxiv.org/abs/1711.08752

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314. https://doi.org/10.1007/BF02551274

Dalmia, A., & Gupta, M. (2018). Towards Interpretation of Node Embeddings. *BigNet*, (BigNet), 945–952. https://doi.org/10.1145/3184558.3191523

Dawes, J., Meyer-Waarden, L., & Driesener, C. (2015). Has brand loyalty declined? A longitudinal analysis of repeat purchase behavior in the UK and the USA. *Journal of Business Research*, *68*(2), 425–432. https://doi.org/10.1016/j.jbusres.2014.06.006

Dow, J. K., & Endersby, J. W. (2004). Multinomial probit and multinomial logit: a comparison of choice models for voting research. *Electoral Studies*, *23*(1), 107–122. https://doi.org/10.1016/S0261-3794(03)00040-4

Ehrenberg, A., & Goodhardt, G. (2002). Double Jeopardy Revisited, Again. *Marketing Research*, *14*(1), 40–42. https://doi.org/10.2307/1251818

Ehrenberg, A. S. C., Goodhardt, G. J., & Barwise, T. P. (1990). Double Jeopardy Revisited. *Journal of Marketing*, *54*(3), 82. https://doi.org/10.2307/1251818

Ehrenberg, A. S. C., Uncles, M. D., & Goodhardt, G. J. (2004). Understanding brand performance measures: Using Dirichlet benchmarks. *Journal of Business Research*, *57*(12 SPEC.ISS.), 1307–1325. https://doi.org/10.1016/j.jbusres.2002.11.001

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/10.1016/0364-0213(90)90002-E

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training. In *International Conference on Artificial Intelligence and Statistics* (Vol. 5, pp. 153–160).

Fader, P. S., & Hardie, B. G. S. (2008). *Computing P(alive) Using the BG / NBD Model*. Retrieved from http://brucehardie.com/notes/021/

Fader, P. S., Hardie, B. G. S., & Berger, P. D. (2004). Customer-Base Analysis with Discrete-Time Transaction Data.

Fader, P. S., Hardie, B. G. S., & Lee, K. L. (2005). "Counting Your Customers" the Easy Way: An Alternative to the Pareto/NBD Model. *Marketing Science*, *24*(2), 275–284. https://doi.org/10.1287/mksc.1040.0098

Fleder, D., & Hosanagar, K. (2009). Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity. *Management Science*, *55*(5), 697–712. https://doi.org/10.1287/mnsc.1080.0974

Fleder, D. M., & Hosanagar, K. (2007). Recommender systems and their impact on sales diversity. *Proceedings of the 8th ACM Conference on Electronic Commerce - EC '07*, 192. https://doi.org/10.1145/1250910.1250939

Forman, G., & Scholz, M. (2009). Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement. *HP Labs*, *12*(1), 49–57. https://doi.org/10.1145/1882471.1882479

Fortes, A. (2018). Public Datasets For Recommender Systems. Retrieved August 9, 2018, from https://github.com/caserec/Datasets-for-Recommneder-Systems

Fortuna, B., Fortuna, C., & Mladenić, D. (2010). Real-time news recommender system. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6323 LNAI*(PART 3), 583–586. https://doi.org/10.1007/978-3-642-15939-8_38

Frank Robert, H., & Cook Philip, J. (1995). *The Winner-Take-All-Society*. *Penguin Books*. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+Winner-take-all+society#0%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+winner-take-all+society+Why+the+Few+at+the+Top+Get+So+Much+More+Than+the+Rest+of+Us#0

Fujitsu. (2017). *A Reference Model for High Performance Data Analytics(HPDA) using an HPC infrastructure*. Retrieved from https://sp.ts.fujitsu.com/dmsp/Publications/public/wp-pf4hpc-hpda.pdf

Gass, S. I., & Rapcsák, T. (2004). Singular value decomposition in AHP. *European Journal of Operational Research*, *154*(3), 573–584. https://doi.org/10.1016/S0377-2217(02)00755-5

Gavrilov, M., Anguelov, D., Indyk, P., & Motwani, R. (2000). Mining The Stock Market: Which Measure Is Best? *Acm Sigkdd*, 487–496. https://doi.org/10.1145/347090.347189

Ge, M., Delgado-Battenfeld, C., & Jannach, D. (2010). Beyond accuracy: evaluating recommender systems by coverage and serendipity. *… on Recommender Systems*, 257–260. https://doi.org/10.1145/1864708.1864761

Gershman, A., & Meisels, A. (2010). A Decision Tree Based Recommender System. In G. Eichler, P. Kropf, U. Lechner, P. Meesad, & H. Unger (Eds.), *International Conference on Innovative Internet Community Systems (I 2 CS) – Jubilee Edition 2010* (pp. 170–179). GI. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.232.1772&rep=rep1&type=pdf#page=171

Gilotte, A., Calauzènes, C., Nedelec, T., Abraham, A., & Dollé, S. (2018). Offline A/B testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (pp. 198–206). https://doi.org/10.1145/3159652.3159687

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, *15*, 315–323. https://doi.org/10.1.1.208.6449

Goel, S., Broder, A., Gabrilovich, E., & Pang, B. (2010). Anatomy of the Long Tail : Ordinary People with Extraordinary Tastes. *Business*, *48*(5), 201–210. https://doi.org/10.1145/1718487.1718513

Goga, M., Kuyoro, S., & Goga, N. (2015). A Recommender for Improving the Student Academic Performance. *Procedia - Social and Behavioral Sciences*, *180*(November 2014), 1481–1488. https://doi.org/10.1016/j.sbspro.2015.02.296

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 61–70. https://doi.org/10.1145/138859.138867

González-Briones, A., Rivas, A., Chamoso, P., Casado-Vara, R., & Corchado, J. M. (2018). Case-Based Reasoning and Agent Based Job Offer Recommender System. In Á. Herrero, B. Baruque, J. Sedano, H. Quintián, & E. Corchado (Eds.), *International Joint Conference SOCO'18-CISIS'18-ICEUTE'18* (Vol. 369, pp. 21–33). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-94120-2_3

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from

http://www.deeplearningbook.org

Goodhardt, G. J., Ehrenberg, A. S. C., & Chatfield, C. (1984). The Dirichlet : A Comprehensive Model of Buying Behaviour. *Journal of the Royal Statistical Society*, *147*(5), 621–655. https://doi.org/10.2307/2981696

Google. (2018a). Play games with a Gamer ID. Retrieved July 29, 2018, from https://support.google.com/googleplay/answer/2954594?hl=en

Google. (2018b). Use Google Play Family Library. Retrieved July 29, 2018, from https://support.google.com/googleplay/answer/7007852?hl=en

Graham, C., Bennett, D., Franke, K., Henfrey, C. L., & Nagy-Hamada, M. (2017). Double Jeopardy – 50 years on. Reviving a forgotten tool that still predicts brand loyalty. *Australasian Marketing Journal*, *25*(4), 278–287. https://doi.org/10.1016/j.ausmj.2017.10.009

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines, 1–26. https://doi.org/10.3389/neuro.12.006.2007

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., … Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471–476. https://doi.org/10.1038/nature20101

Greenacre, M. J. (1988). Clustering the rows and columns of a contingency table. *Journal of Classification*, *5*(1), 39–51. https://doi.org/10.1007/BF01901670

Gupta, S., Hanssens, D., Hardie, B., Kahn, W., Kumar, V., Lin, N., … Sriram, S. (2006). Modeling customer lifetime value. *Journal of Service Research*, *9*(2), 139–155. https://doi.org/10.1177/1094670506293810

Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., … Wang, X. (2018). Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. *Proceedings - International Symposium on High-Performance Computer Architecture*, *2018–Febru*, 620–629. https://doi.org/10.1109/HPCA.2018.00059

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, *22*(1), 5–53. https://doi.org/10.1145/963770.963772

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Hochreiter, S., & Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T

Hsu, C. W., & Lin, C. J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425. https://doi.org/10.1109/72.991427

Huang, C.-Y. (2011). Excess Loyalty in Online Retailing. *International Journal of Electronic Commerce*, *16*(2), 115–134. https://doi.org/10.2753/JEC1086-4415160206

Huggett, M., Hoos, H., & Rensink, R. (2007). Cognitive principles for information management: The Principles of Mnemonic Associative Knowledge (P-MAK). *Minds and Machines*, *17*(4), 445–485.

https://doi.org/10.1007/s11023-007-9080-4

Hunter, T. (2016). *Deep Learning with Apache Spark and TensorFlow*. Retrieved from https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html

Iwata, T., Saito, K., & Yamada, T. (2008). Recommendation Method for Improving Customer Lifetime Value. *IEEE Transactions on Knowledge and Data Engineering*, *20*(9), 1254–1263. https://doi.org/10.1109/TKDE.2008.55

Jahrer, M., Töscher, A., & Legenstein, R. (2010). Combining predictions for accurate recommender systems. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '10*, 693. https://doi.org/10.1145/1835804.1835893

Jeuland, A. P., Bass, F. M., & Wright, G. P. (1980). A Multibrand Stochastic Model Compounding Heterogeneous Erlang Timing and Multinomial Choice Processes. *Operations Research*, *28*(2), 255–277. https://doi.org/10.1287/opre.28.2.255

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., … Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. https://doi.org/10.1145/2647868.2654889

Jin, R., & Si, L. (2004). A bayesian approach toward active learning for collaborative filtering. *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 278–285. Retrieved from http://dl.acm.org/citation.cfm?id=1036877

Ju, J. Y., Choi, I. Y., Kim, J. K., & Moon, H. S. (2017). Reinforcement Learning for Profit Maximization of Recommender Systems Reinforcement Learning for Profit Maximization of Recommender Systems Completed Research Paper. In *Proceedings of the P re-ICIS 2017 S IGDSA Symposium*. Seoul, South Korea. Retrieved from https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1003&context=sigdsa2017

Karney, C. F. F. (2013). Algorithms for geodesics. *Journal of Geodesy*, *87*(1), 43–55. https://doi.org/10.1007/s00190-012-0578-z

Karsmakers, P., Pelckmans, K., & Suykens, J. A. K. (2007). Multi-class kernel logistic regression : a fixed-size implementation. In *Advances in Neural Information Processing Systems* (Vol. 20, pp. 1177–1184).

Katsov, I. (2018). *Introduction to Algorithmic Marketing - Artificial Intelligence for Marketing Operations*. Grid Dynamics.

Kim, H., Park, J., Jang, J., & Yoon, S. (2016). DeepSpark: A Spark-Based Distributed Deep Learning Framework for Commodity Clusters, (Nips). Retrieved from http://arxiv.org/abs/1602.08191

Kim, K. (2011). Customer Need Type Classification Model using Data Mining Techniques for Recommender Systems, *5*(8), 279–284.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *ICLR 2015* (pp. 1–15). https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

Knijnenburg, B. P., Willemsen, M. C., Gantner, Z., Soncu, H., & Newell, C. (2012). Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, *22*(4–5), 441–504. https://doi.org/10.1007/s11257-011-9118-4

Kontschieder, P., Fiterau, M., Criminisi, A., & Bulo, S. R. (2015). Deep Neural Decision Forests. In *2015 IEEE International Conference on Computer Vision (ICCV)* (pp. 1467–1475). IEEE. https://doi.org/10.1109/ICCV.2015.172

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, *42*(8), 30–37. https://doi.org/10.1109/MC.2009.263

Koutrika, G. (2018). Modern Recommender Systems: from Computing Matrices to Thinking with Neurons. https://doi.org/10.1145/3183713.3197389

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 1–9. https://doi.org/http://dx.doi.org/10.1016/j.protcy.2014.09.007

Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, *2*(1), 24. https://doi.org/10.1186/s40537-015-0032-1

Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents, *32*. https://doi.org/10.1145/2740908.2742760

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lehal, G. S. (2007). Design and Implementation of Punjabi Spell Checker. *International Journal of Systemics, Cybernetics and Informatics*, 70–75. https://doi.org/10.17485/ijst/2015/v8i27/83917

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867. https://doi.org/10.1016/S0893-6080(05)80131-5

Li, Y. M., Wu, C. Te, & Lai, C. Y. (2013). A social recommender mechanism for e-commerce: Combining similarity, trust, and relationship. *Decision Support Systems*, *55*(3), 740–752. https://doi.org/10.1016/j.dss.2013.02.009

Liu, C., & Zhou, W. (2010). An improved HeatS+ ProbS hybrid recommendation algorithm based on heterogeneous initial resource configurations. *ArXiv Preprint ArXiv:1005.3124*, 1–6. https://doi.org/10.1016/j.physa.2012.06.034

Liu, X., Aggarwal, C., Li, Y.-F., Kong, X., Sun, X., & Sathe, S. (2016). Kernelized Matrix Factorization for Collaborative Filtering. *Proceedings of the 2016 SIAM International Conference on Data Mining*, 378–386. https://doi.org/10.1137/1.9781611974348.43

Lü, L., Medo, M., Yeung, C. H., Zhang, Y.-C., Zhang, Z.-K., & Zhou, T. (2012). Recommender systems. *Physics Reports*, *519*(1), 1–49. https://doi.org/10.1016/j.physrep.2012.02.006

Luaces, O., Díez, J., Barranquero, J., del Coz, J. J., & Bahamonde, A. (2012). Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, *1*(4), 303–313. https://doi.org/10.1007/s13748-012-0030-x

Maillo, J., Ramírez, S., Triguero, I., & Herrera, F. (2017). kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, *117*, 3–15. https://doi.org/10.1016/j.knosys.2016.06.012

Maroco, J., Silva, D., Rodrigues, A., Guerreiro, M., Santana, I., & De Mendonça, A. (2011). Data mining

methods in the prediction of Dementia: A real-data comparison of the accuracy, sensitivity and specificity of linear discriminant analysis, logistic regression, neural networks, support vector machines, classification trees and random forests. *BMC Research Notes*, *4*. https://doi.org/10.1186/1756-0500-4-299

Michmizos, D., Koutsouraki, E., Asprodini, E., & Baloyannis, S. (2011). Synaptic Plasticity: A Unifying Model to Address Some Persisting Questions. *International Journal of Neuroscience*, *121*(6), 289–304. https://doi.org/10.3109/00207454.2011.556283

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Retrieved from http://arxiv.org/abs/1301.3781

Mild, A., & Natter, M. (2002). Collaborative filtering or regression models for Internet recommendation systems? *Journal of Targeting, Measurement and Analysis for Marketing*, *10*(4), 304–313. https://doi.org/10.1057/palgrave.jt.5740055

Miyahara, K., & Pazzani, M. J. (2000). Collaborative Filtering with the Simple Bayesian Classifier. In *PRICAI 2000 Topics in Artificial Intelligence* (Vol. 8046, pp. 679–689). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44533-1_68

Montúfar, G., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the Number of Linear Regions of Deep Neural Networks. https://doi.org/10.1007/978-1-4471-5779-3_4

Moritz, P., Nishihara, R., Stoica, I., & Jordan, M. I. (2015). SparkNet: Training Deep Networks in Spark, 1–12. Retrieved from http://arxiv.org/abs/1511.06051

Neumann, J. von. (1945). *First Draft of a Report on the EDVAC*.

Nickolls, J., & Dally, W. J. (2010). The GPU Computing Era. *IEEE Micro*, *30*(2), 56–69. https://doi.org/10.1109/MM.2010.41

O'Leary, D. E. (2014). Embedding AI and crowdsourcing in the big data lake. *IEEE Intelligent Systems*, *29*(5), 70–73. https://doi.org/10.1109/MIS.2014.82

O'Mahony, M. P., Cunningham, P., & Smyth, B. (2010). An Assessment of Machine Learning Techniques for Review Recommendation. In *Artificial Intelligence and Cognitive Science : 20th Irish Conference* (pp. 241–250). https://doi.org/10.1007/978-3-642-17080-5_26

Oku, K., Nakajima, S., Miyazaki, J., & Uemura, S. (2006). Context-aware SVM for context-dependent information recommendation. *Proceedings - IEEE International Conference on Mobile Data Management*, *2006*, 5–8. https://doi.org/10.1109/MDM.2006.56

Olah, C., & Carter, S. (2016). Attention and Augmented Recurrent Neural Networks. Retrieved August 12, 2018, from https://distill.pub/2016/augmented-rnns/

Osogami, T. (2017). Boltzmann machines and energy-based models. Retrieved from http://arxiv.org/abs/1708.06008

Ostuni, V. C., Di Noia, T., Mirizzi, R., & Di Sciascio, E. (2014). Top-N recommendations from implicit feedback leveraging linked open data. *CEUR Workshop Proceedings*, *1127*, 20–27. https://doi.org/10.1145/2507157.2507172

Oussous, A., Benjelloun, F. Z., Ait Lahcen, A., & Belfkih, S. (2017). Big Data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*. https://doi.org/10.1016/j.jksuci.2017.06.001

Pääkkönen, P., & Pakkala, D. (2015). Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, *2*(4), 166–186. https://doi.org/10.1016/j.bdr.2015.01.001

Pallis, G., Zeinalipour-yazti, D., & Dikaiakos, M. D. (2011). Online Social Networks : Status and Trends. *Data Management*, *331*, 213–234. https://doi.org/10.1007/978-3-642-17551-0_8

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. *KDD Cup and Workshop*, 2–5. https://doi.org/10.1145/1557019.1557072

Pathak, B., Garfinkel, R., Gopal, R. D., Venkatesan, R., & Yin, F. (2010). Empirical Analysis of the Impact of Recommender Systems on Sales. *Journal of Management Information Systems*, *27*(2), 159–188. https://doi.org/10.2753/MIS0742-1222270205

Peltier, S., & Moreau, F. (2012). Internet and the "long tail versus superstar effect" debate: Evidence from the french book market. *Applied Economics Letters*, *19*(8), 711–715. https://doi.org/10.1080/13504851.2011.597714

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14* (pp. 701–710). New York, New York, USA: ACM Press. https://doi.org/10.1145/2623330.2623732

Piraveenan, M., Prokopenko, M., & Hossain, L. (2013). Percolation Centrality: Quantifying Graph-Theoretic Impact of Nodes during Percolation in Networks. *PLoS ONE*, *8*(1). https://doi.org/10.1371/journal.pone.0053095

Polat, K., & Güneş, S. (2007). Classification of epileptiform EEG using a hybrid system based on decision tree classifier and fast Fourier transform. *Applied Mathematics and Computation*, *187*(2), 1017–1026. https://doi.org/10.1016/j.amc.2006.09.022

Prechelt, L. (2012). Early Stopping — But When? (pp. 53–67). https://doi.org/10.1007/978-3-642-35289-8_5

Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK guide)*. Project Management Institute.

Pu, P., & Chen, L. (2010). A user-centric evaluation framework of recommender systems. *CEUR Workshop Proceedings*, *612*, 14–21. https://doi.org/10.1145/2043932.2043962

Pu, P., Chen, L., & Hu, R. (2012). Evaluating recommender systems from the user's perspective: Survey of the state of the art. *User Modeling and User-Adapted Interaction*, *22*(4–5), 317–355. https://doi.org/10.1007/s11257-011-9115-7

Quan, T. W., & Williams, K. R. (2017). Product Variety, Across-Market Demand Heterogeneity, and the Value of Online Retail. *SSRN Electronic Journal*, (2054). https://doi.org/10.2139/ssrn.2993236

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information …*, (1), 1–8. https://doi.org/10.1.1.145.8736

Ren, J., Zhou, T., & Zhang, Y.-C. (2008). Information filtering via self-consistent refinement. *EPL (Europhysics Letters)*, *82*(5), 58007. https://doi.org/10.1209/0295-5075/82/58007

Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. (F. Ricci, L. Rokach, & B. Shapira, Eds.). Boston, MA: Springer US. https://doi.org/10.1007/978-1-4899-7637-6

Romero, F., & Koochak, Z. (2015). *Assessing and Implementing Automated News Classification*.

Retrieved from https://pdfs.semanticscholar.org/41b9/c60ff64349b024a79ba7a2ca7acb68e9a682.pdf

Rosen, S. (1981). The Economics of Superstars. *The American Economic Review*, *71*(5), 845–858. Retrieved from https://www.jstor.org/stable/1803469

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in …. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Rowley, H. A., Baluja, S., & Kanade, T. (1995). Human Face Detection in Visual Scenes, (November).

Rungie, C. (2003). How to Estimate the Parameters of the Dirichlet Model using Likelihood Theory in Excel. *Marketing Bulletin*, *14*, *Techni*(January 2003), 1–9. Retrieved from https://www.researchgate.net/publication/237609308_How_to_Estimate_the_Parameters_of_the_Dirichlet_Model_using_Likelihood_Theory_in_Excel

Sarwar, B. M., Karypis, G., Konstan, J. a, & Riedl, J. T. (2000). Application of Dimensionality Reduction in Recommender System - A Case Study. *Architecture*, *1625*, 264–8. https://doi.org/10.1.1.38.744

Sayago, S., Guijarro, J.-M., & Blat, J. (2012). Selective attention in web forms: an exploratory case study with older people. *Behaviour & Information Technology*, *31*(2), 171–184. https://doi.org/10.1080/01449291003767920

Schclar, A., Tsikinovsky, A., Rokach, L., Meisels, A., & Antwarg, L. (2009). Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems - RecSys '09* (p. 261). New York, New York, USA: ACM Press. https://doi.org/10.1145/1639714.1639763

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '02*, (August), 253. https://doi.org/10.1145/564376.564421

Schmittlein, D. C., Morrison, D. G., & Colombo, R. (1987). Counting Your Customers: Who-Are They and What Will They Do Next? *Management Science*, *33*(1), 1–24. https://doi.org/10.1287/mnsc.33.1.1

Schnabel, T., Bennett, P. N., & Joachims, T. (2018). Improving Recommender Systems Beyond the Algorithm. Retrieved from http://arxiv.org/abs/1802.07578

Shi, J. ., Qiu, Y. ., Minhas, U. F. ., Jiao, L. ., Wang, C. ., Reinwald, B. ., & Özcan, F. . (2015). Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment*, *8*(13), 2110–2121. https://doi.org/10.14778/2831360.2831365

Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V. (2017). Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research*, *70*, 263–286. https://doi.org/10.1016/j.jbusres.2016.08.001

Statista. (2018). *Distribution of free and paid apps in the Apple App Store and Google Play as of 1st quarter 2018*. Retrieved from https://www.statista.com/statistics/263797/number-of-applications-for-mobile-phones/

Stewart, G. W. (1993). On the Early History of the Singular Value Decomposition. *SIAM Review*, *35*(4), 551–566. https://doi.org/10.1137/1035134

Su, X., & Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, *2009*(Section 3), 1–19. https://doi.org/10.1155/2009/421425

Sun, X., Guo, J., Ding, X., & Liu, T. (2016). A General Framework for Content-enhanced Network Representation Learning. Retrieved from http://arxiv.org/abs/1610.02906

Szabó, Z., Póczos, B., & Lőrincz, A. (2012). Collaborative Filtering via Group-Structured Dictionary Learning. In *Latent Variable Analysis and Signal Separation: 10th International Conference, LVA/ICA 2012, Tel Aviv, Israel, March 12-15, 2012. Proceedings* (pp. 247–254). https://doi.org/10.1007/978-3-642-28551-6_31

Tan, Y. K., Xu, X., & Liu, Y. (2016). Improved Recurrent Neural Networks for Session-based Recommendations, 0–5. https://doi.org/10.1145/2988450.2988452

Tang, Y. (2013). Deep Learning using Linear Support Vector Machines. Retrieved from http://arxiv.org/abs/1306.0239

Torres, D., Skaf-Molli, H., Molli, P., & Díaz, A. (2013). BlueFinder. In *Proceedings of the 5th Annual ACM Web Science Conference on - WebSci '13* (pp. 413–422). New York, New York, USA: ACM Press. https://doi.org/10.1145/2464464.2464515

Tsoumakas, G., & Vlahavas, I. (2007). Random k-Labelsets: An Ensemble Method for Multilabel Classification. In *Machine Learning: ECML 2007* (pp. 406–417). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-74958-5_38

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), 230–265. https://doi.org/10.1112/plms/s2-42.1.230

Twentyman, J. (2016). Apache Spark grows in popularity as Hadoop-based data lakes fill up. *Computer Weekly*. Retrieved from http://www.computerweekly.com/feature/Apache-Spark-grows-in-popularity-as-Hadoop-based-data-lakes-fill-up

Vagata, P., & Wilfong, K. (2014). Scaling the Facebook data warehouse to 300 PB. Retrieved from https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

van Hasselt, H. (2013). Estimating the Maximum Expected Value: An Analysis of (Nested) Cross Validation and the Maximum Sample Average, 1–17. Retrieved from http://arxiv.org/abs/1302.7175

Vargas, S., & Castells, P. (2011). Rank and relevance in novelty and diversity metrics for recommender systems. *Proceedings of the Fifth ACM Conference on Recommender Systems - RecSys '11*, 109. https://doi.org/10.1145/2043932.2043955

Varini, C., Degenhard, A., & Nattkemper, T. (2005). ISOLLE: Locally Linear Embedding with Geodesic Distance. In *Knowledge Discovery in Databases: PKDD 2005* (pp. 331–342). Retrieved from http://books.google.com/books?hl=en&lr=&id=FNIjx3PZmosC&oi=fnd&pg=PR2&dq=Lecture+Notes+in+Artificial+Intelligence&ots=M6GcvV1IVt&sig=d9UeAvEE1inT8Dgy5BUDqWELMRg%5Cnhttp://books.google.com/books?hl=en&lr=&id=FNIjx3PZmosC

Verbeke, W., Dejaeger, K., Martens, D., Hur, J., & Baesens, B. (2012). New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. *European Journal of Operational Research*, *218*(1), 211–229. https://doi.org/10.1016/j.ejor.2011.09.031

Vidmer, A., Zeng, A., Medo, M., & Zhang, Y. C. (2015). Prediction in complex systems: The case of the international trade network. *Physica A: Statistical Mechanics and Its Applications*, *436*, 188–199. https://doi.org/10.1016/j.physa.2015.05.057

Vincenty, T. (1975). Direct and Inverse Solutions of Geodesics on the Ellipsoid With Application of Nested Equations. *Survey Review*, *23*(176), 88–93. https://doi.org/10.1179/sre.1975.23.176.88

Vrooman, H. A., Cocosco, C. A., van der Lijn, F., Stokking, R., Ikram, M. A., Vernooij, M. W., … Niessen, W. J. (2007). Multi-spectral brain tissue segmentation using automatically trained k-Nearest-Neighbor classification. *NeuroImage*, *37*(1), 71–81. https://doi.org/10.1016/j.neuroimage.2007.05.018

Wang, H., Wang, N., & Yeung, D.-Y. (2015). Collaborative Deep Learning for Recommender Systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15* (pp. 1235–1244). New York, New York, USA: ACM Press. https://doi.org/10.1145/2783258.2783273

Wang, K., & Khan, M. M. H. (2015). Performance Prediction for Apache Spark Platform. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.246

Wedel, M., & Kannan, P. K. (2016). Marketing Analytics for Data-Rich Environments. *Journal of Marketing*, *80*(6), 97–121. https://doi.org/10.1509/jm.15.0413

Weinhardt, C., Anandasivam, A., Blau, B., Borissov, N., Meinl, T., Michalk, W., & Stößer, J. (2009). Cloud Computing – A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering*, *1*(5), 391–399. https://doi.org/10.1007/s12599-009-0071-2

Weston, J., Chopra, S., & Bordes, A. (2014). Memory Networks, 1–15. Retrieved from http://arxiv.org/abs/1410.3916

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*(10). https://doi.org/10.18637/jss.v059.i10

Wilson, L. (2018). A Complete Guide to App Store Optimization (ASO). *Search Engine Journal*. Retrieved from https://www.searchenginejournal.com/app-store-optimization-how-to-guide/241967/

Wright, M., Sharp, A., & Sharp, B. (2002). Market Statistics for the Dirichlet Model : Using the Juster Scale to Replace Panel Data. *International Journal of Research in Marketing*, *19*(1), 81–90. https://doi.org/10.1016/S0167-8116(02)00049-6

Wrigley, N., & Dunn, R. (1985). Stochastic Panel-Data Models of Urban Shopping Behavior .4. Incorporating Independent Variables into the Nbd and Dirichlet Models. *Environment and Planning A*, *17*(3), 319–331.

Xia, Z., Dong, Y., & Xing, G. (2006). Support vector machines for collaborative filtering. In *Proceedings of the 44th annual southeast regional conference on - ACM-SE 44* (p. 169). New York, New York, USA: ACM Press. https://doi.org/10.1145/1185448.1185487

Xiao, Z., Wijegunaratne, I., & Qiang, X. (2017). Reflections on SOA and Microservices. *Proceedings - 4th International Conference on Enterprise Systems: Advances in Enterprise Systems, ES 2016*, 60–67. https://doi.org/10.1109/ES.2016.14

Yang, G. (2016). Lie Access Neural Turing Machine. Retrieved from http://arxiv.org/abs/1602.08671

Yang, S.-H., Long, B., Smola, A. J., Zha, H., & Zheng, Z. (2011). Collaborative Competitive Filtering: Learning Recommender Using Context of User Choice. *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 295–304. https://doi.org/10.1145/2009916.2009959

Yin, H., Cui, B., Li, J., Yao, J., & Chen, C. (2012). Challenging the long tail recommendation. *Proceedings of the VLDB Endowment*, *5*(9), 896–907. https://doi.org/10.14778/2311906.2311916

Zhang, H. R., & Min, F. (2016). Three-way recommender systems based on random forests. *Knowledge-Based Systems*, *91*, 275–286. https://doi.org/10.1016/j.knosys.2015.06.019

Zhang, S., Yao, L., & Sun, A. (2017). Deep Learning based Recommender System: A Survey and New Perspectives, *1*(1), 1–35. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Zhang, Y. C., Blattner, M., & Yu, Y. K. (2007). Heat conduction process on community networks as a recommendation model. *Physical Review Letters*, *99*(15), 1–5. https://doi.org/10.1103/PhysRevLett.99.154301

Zhao, Q., Zhang, Y., Zhang, Y., & Friedman, D. (2016). *Recommendation based on multiproduct utility maximization PDF Logo* (WZB Discussion Paper SP No. 2016–503).

Zheng, H., Wang, D., Zhang, Q., Li, H., & Yang, T. (2010). Do clicks measure recommendation relevancy? *Proceedings of the Fourth ACM Conference on Recommender Systems - RecSys '10*, 249. https://doi.org/10.1145/1864708.1864759

Zhong, N., & Michahelles, F. (2012). Where should you focus: Long Tail or Superstar? In *SIGGRAPH Asia 2012 Symposium on Apps on - SA '12* (Vol. 3, pp. 1–1). New York, New York, USA: ACM Press. https://doi.org/10.1145/2407696.2407706

Zhong, N., & Michahelles, F. (2013). Google play is not a long tail market: an empirical analysis of app adoption on the Google play app market. *Proceedings of the 28th Annual ACM*, 499–504. https://doi.org/10.1145/2480362.2480460

Zhu, J., & Hastie, T. (2005). Kernel Logistic Regression and the Import Vector Machine. *Journal of Computational and Graphical Statistics*, *14*(1), 185–205. https://doi.org/10.1198/106186005X25619

APPENDIX

APPENDIX A – MATHEMATICAL DERIVATION OF APP DOWNLOAD VALUE

Before we derive an estimator for the App Download Value, we need to establish the notation and the definition of the main mathematical objects.

Definition 1. *Category.*

Let $\mathcal{G}$ denote the set with cardinality $|\mathcal{G}| \in \mathbb{N}$ of all mobile app stores of the same category.

Definition 2. *Market.*

Let $\mathcal{M}$ denote the set with cardinality $|\mathcal{M}| \in \mathbb{N}$ of all users u that have used any mobile app store $j \in \mathcal{G}$ at least once.

Definition 3. *User Base.*

Let $\mathcal{D}_j$ denote the set with cardinality $D \in \mathbb{N}$ of all users that have used the mobile app store $j \in \mathcal{M}$ at least once in the past, which we will refer to as the *user base* of that mobile app store.

Definition 4. *App Downloads.*

Let $s_{ij} \in \mathbb{N}$ denote the number of app downloads of any user $i \in \mathcal{M}$ has done on period t in mobile app store $j \in \mathcal{G}$.

Definition 5. *Mobile App Store Choice Probability.*

Let c_{ij} denote the probability of user i downloading an app from mobile app store $j \in \mathcal{G}$.

Definition 6. *Mean User App Download Frequency.*

Let μ_i denote the mean app download frequency of user i by period.

Definition 7. *Total App Downloads on Mobile App Store until T.*

Let $\tau_{downloads_{j_T}}$ denote the total number of app downloads from mobile app store $j \in \mathcal{G}$ until period T,

$$\tau_{downloads_{j_T}} = \sum_{d \in \mathcal{D}_j} \sum_{t=1}^{T} s_{djt}$$

Definition 8. *Market Share.*

Let MS_k denote the *market share* of the mobile app store $k \in \mathcal{G}$ such that,

$$MS_k = \frac{\sum_{i \in \mathcal{M}} s_{ik}}{\sum_{j \in \mathcal{G}} \sum_{i \in \mathcal{M}} s_{ij}}$$

Definition 9. *User Churn.*

Let p_{ik} be the individual churn probability in mobile app store $k \in \mathcal{G}$ for user $i \in \mathcal{M}$ conditional on the number of app downloads in all mobile app store $j \in \mathcal{G}$ in the period such that,

$$p_{ik} = p(s_{ik} > 0 | \tau_{s_{ij}})$$

Definition 10. *Mobile App Store Churn.*

Let p_j be the churn rate of mobile app store $j \in \mathcal{G}$ such that,

$$p_j = \frac{\sum_{i \in \mathcal{D}} p_{ij}}{D}$$

Definition 11. *Category App Download Frequency.*

Let n denote the app download frequency of mobile app store category $\mathcal{G}$ such that,

$$n = \frac{\sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{G}} s_{ij}}{|\mathcal{M}|} = \frac{\sum_{i \in \mathcal{M}} \tau_{s_{ij}}}{|\mathcal{M}|}$$

Definition 12. *Customer Lifetime Value (CLV) until T.*

Let CLV_{i_T} denote the Customer Lifetime Value of user $i \in \mathcal{D}_j$ until period T such that (Blattberg et al., 2008, pp. 108–109),

$$CLV_{i_T} = \sum_{t=1}^{T} \frac{(1 - p_j)(\tilde{R}_{ijt} - C_{ijt})}{(1 + \delta)^{t-1}}$$

Where $\tilde{R}_{ijt}$ is the revenue generated by user i in period t, C_{it} is the cost of serving the user i on moment t and δ is the discount rate.

Definition 13. *Total Customer Lifetime Value until T.*

Let $\tau_{CLV_{j_T}}$ denote the total Customer Lifetime Value of the mobile app store $j \in \mathcal{M}$ until period T.

$$\tau_{CLV_{j_T}} = \sum_{d \in \mathcal{D}_j} CLV_{d_T}$$

Definition 14. *App Download Probability.*

Let l_j denote the probability of a download occurring on mobile app store $j \in \mathcal{M}$ under a frequentist assumption such that,

$$l_j = \lim_{T \to \infty} \frac{1}{\tau_{downloads_{j_T}}}$$

Definition 15. *App Download Value.*

Let $V_{download}$ denote the value of an app download for mobile app store $j \in \mathcal{M}$, such that,

$$V_{download} = \lim_{T \to \infty} \frac{\tau_{CLV_{j_T}}}{\tau_{downloads_{j_T}}} = \lim_{T \to \infty} (\tau_{CLV_{j_T}}) \times l_j$$

We will now present our main assumptions under the form of axioms that are required to define our statistical estimators. These assumptions are the basic assumptions behind the NBD-Dirichlet model (Goodhardt et al., 1984) and are grounded in the empirical generalizations that have been observed over several decades across markets and geographies (Graham, Bennett, Franke, Henfrey, & Nagy-Hamada, 2017). In particular these patterns have also been observed in the mobile app economy (Zhong & Michahelles, 2013) and amongst online retailers (Huang, 2011). Therefore, we believe these to be a set of realistic assumptions.

Axiom 1. *The probability of a user doing n downloads in a given period follows a Poisson Distribution.*

$$p(s_{ijt} = k) \sim e^{-\mu_i} \frac{days \times {\mu_i}^k}{k!}$$

Axiom 2. *Individual user app download frequencies vary according to a Gamma Distribution.*

$$\tilde{s}_i \sim \frac{\mathrm{e}^{-\mu_i \frac{K}{M}} {\mu_i}^{K-1}}{\Gamma(K) \left(\frac{M}{K}\right)^K}$$

Axiom 3. *User choice of mobile app store over k successive choices follows a Multinomial Distribution.*

$$c_{ij} \sim \frac{k!}{\prod_{t=1}^{T} s_{ijt}} \prod_{g \in \mathcal{G}} \alpha_g$$

Axiom 4. *Individual user mobile app store choice frequencies vary according to a Dirichlet Distribution.*

$$\theta_{ig} \sim \frac{\prod_{g \in \mathcal{G}} (\sum_{t=1}^{T} s_{igt})^{\alpha_g - 1} \Gamma(\sum_{g \in \mathcal{G}} \alpha_g)}{\prod_{g \in \mathcal{G}} \Gamma(\alpha_g)}$$

Axiom 5. *The mobile app store choice probabilities and the average app download frequencies of the different users $m \in \mathcal{M}$ are distributed independently.*

Definition 15. *NBD-Dirichlet Model (Conditional App Downloads Estimator).*

The set of axioms imply that we can obtain the probability $p(r_j|n)$ of a user $i \in \mathcal{M}$ doing $r_j \in \mathbb{N}$ app downloads on period t in mobile app store $j \in \mathcal{G}$ amongst n_i app downloads by employing a compound distribution such that (Goodhardt et al., 1984),

$$\left[\frac{k!}{\prod_{t=1}^{T} s_{ijt}} \prod_{j \in \mathcal{G}} \alpha_j \underset{\alpha_j}{\wedge} \frac{\prod_{j \in \mathcal{G}} (\sum_{t=1}^{T} s_{ijt})^{\alpha_j - 1} \Gamma(\sum_{j \in \mathcal{G}} \alpha_j)}{\prod_{j \in \mathcal{G}} \Gamma(\alpha_j)} \right] \underset{n}{\wedge} \left[e^{-\mu_i} \frac{days \times {\mu_i}^k}{k!} \underset{\mu_i}{\wedge} \frac{\mathrm{e}^{-\mu_i \frac{K}{M}} {\mu_i}^{K-1}}{\Gamma(K) \left(\frac{M}{K}\right)^K} \right]$$

Where α_j is the app download propensity of mobile app store j, M and K are model parameters and $days$ refers to the length (in days) of each time period t.

Therefore,

$$p(r_j|k) \sim \frac{\binom{k}{r_j} \mathrm{B}(\alpha_j + r_j, S - \alpha_j + k - r_j)}{\mathrm{B}(\alpha_j, S - \alpha_j)}$$

Where S is the diversity of usage behavior in the category ($S = \sum_j \alpha_j$) (Bound, 2009). We can assume that $MS_j = \alpha_j / S$ (Wright et al., 2002).

Remark 1. *Mobile App Store Churn Rate Estimator.*

From Definition 15 and the set of axioms it follows that we can obtain an estimator $\hat{p}_j$ for p_j such that,

$$\hat{p}_j = p(r_j = 0|G)$$

Remark 2. *App Download Probability Estimator.*

From Definition 15 and the set of axioms it follows that we can obtain an estimator $\hat{l}_j$ for l_j such that,

$$\hat{l}_j = p(r_j = 1|G)$$

Definition 16. *Average Mobile App Store Customer Lifetime Value Estimator.*

Let $\widehat{CLV}_{j_T}$ be an estimator for CLV_{j_T} such that,

$$\widehat{CLV}_{j_T} = \sum_{t=1}^{T} \frac{(1-\hat{p}_j)(\bar{R}_t - \bar{C}_t)}{(1+\delta)^{t-1}}$$

Definition 17. *Total Customer Lifetime Value Estimator over finite horizon h.*

Let $\hat{\tau}_{CLV_{j_h}}$ be an estimator for $\tau_{CLV_{j_h}}$ such that,

$$\hat{\tau}_{CLV_{j_h}} = \lim_{T \to h} D\widehat{CLV}_{j_T}$$

Definition 17. *App Download Value Estimator.*

Let $\hat{V}_{download}$ be an estimator for $V_{download}$ such that,

$$\hat{V}_{download} = \hat{\tau}_{CLV_{j_h}} \times \hat{l}_j =$$

$$= p\big(r_j = 1 \big| n\big) D \lim_{T \to h} \widehat{CLV}_{j_T} =$$

$$= p\big(r_j = 1 \big| n\big) D \lim_{T \to h} \sum_{t=1}^{T} \frac{(1-\hat{p}_j)(\bar{R}_t - \bar{C}_t)}{(1+\delta)^{t-1}} =$$

$$= \sum_{t=1}^{h} \frac{D(1-\hat{p}_j)(\bar{R}_t - \bar{C}_t)\, p\big(r_j = 1 \big| n\big)}{(1+\delta)^{t-1}}$$

Replacing $\hat{p}_j$ and $p\big(r_j = 1 \big| n\big)$ with their respective expressions we get,

$$\hat{V}_{download} = \sum_{t=1}^{h} \frac{nD(\bar{R}_t - \bar{C}_t)[\mathrm{B}\big(\alpha_j, S-\alpha_j\big) - \mathrm{B}\big(\alpha_j, S-\alpha_j+n\big)]\mathrm{B}\big(\alpha_j+1, S-\alpha_j+n-1\big)}{(1+\delta)^{t-1}\mathrm{B}\big(\alpha_j, S-\alpha_j\big)^2}$$