Insurance recommendation engine using a combined collaborative filtering and neural network approach

Prinavan Pillay (PLLPRI017)

Masters Dissertation

Department of Statistical Sciences
University of Cape Town
Supervisors:
Dr. Sebnem Er.
Mr. Allan Clark

Abstract

A recommendation engine for insurance modelling was designed, implemented and tested using a neural network and collaborative filtering approach. The recommendation engine aims to suggests suitable insurance products for new or existing customers, based on their features or selection history. The collaborative filtering approach used matrix factorization on an existing user base to provide recommendation scores for new products to existing users. The content based method used a neural network architecture which utilized user features to provide a product recommendation for new users. Both methods were deployed using the Tensorflow machine learning framework. The hybrid approach helps solve for cold start problems where users have no interaction history. The accuracy on the collaborative filtering produced 0.13 root mean square error based on implicit feedback rating of 0-1, and an overall Top-3 classification accuracy (ability to predict one of the top 3 choices of a customer) of 83.8%. The neural network system achieved an accuracy of 77.2% on Top-3 classification. The system thus achieved good training performance and given further modifications, could be used in a production environment.

Contents

List of Figures

List of Tables

1. Introduction

The insurance industry has seen a recent surge of development in the data science space, particularly with regards to understanding consumer behaviour and the propensity to purchase new products based on an individual's particular profile and their similarity to other customers (Lumb 2016). These approaches are being used to accelerate digital marketing processes and techniques to advertise the most relevant content to the correct target market. Research into new behavior modelling techniques have thus emerged, which in particular tries to exploit the ability to *up-sell* and *cross-sell* products in the insurer's product catalogue. Product up-sell refers to the ability to convince a customer to purchase a more premium or high-end version of a product that they already own or intend to own, thus spending more money on the same type of service or product offered. Cross-sell refers to the ability to convince a customer to purchase similar items to previously purchased items, thereby increasing their total portfolio of products (Low 2017). Traditionally, cross-sell and up-sell have been sales and marketing tactics which relied on deep customer insight from industry experts. However, given the emergence of scalable computing environments to build machine learning models on massive consumer datasets, automated models have begun to accurately predict up-sell and cross-sell opportunities in the insurance space.

A particular machine learning application which has been widely implemented with propensity modelling is called recommendation engines (Furnas 1995). Recommendation engines have been implemented in a number of production e-commerce, video streaming sites and other massive user-base applications to suggest and predict new user content based on historic item selection. Traditionally, the techniques employed in recommendation engines are usually supervised learning algorithms using neural networks (ibid.), generalized linear models (Jhalani, Kant, and Dwivedi 2016), or collaborative filtering (Rosa 2003) to predict the most relevant unseen content for new and existing users. Unsupervised learning strategies can also be applied, with recent focus on association rule mining, which aims to highlight interesting patterns in purchasing data, and frequent combinations of items that are not directly obvious (Wong and W. Yang 1997).

The proposed research aims to implement a recommendation engine on a local South African insurance database, with the ability to identify the most relevant insurance products for new and existing users. The data consists of a member database with current insurance products, as well as qualitative member data (profession, age, gender, etc.). The research will involve exploring a modelling technique which uses a combination of collaborative filtering, and neural networks. Collaborative filtering will be used for existing users to provide recommendations based on historic selection, and neural networks will be used for new users based on their features (age, gender, occupation, etc.). These hybrid approaches have shown success in massive online platforms including predicting app downloads for the Google Play Store, and Youtube suggestions which involve hundreds of millions of active users (Cheng et al. 2016).

This dissertation firstly presents a literature survey in Chapter 2, which outlines successfully implemented recommendation engines in industry. The proposed methodology and background theory of using collaborative filtering for existing users, and neural networks for new users is then presented in Chapter 3. A data exploration is then conducted in Chapter 4, which provides an overview of the source data set. The model architecture is then presented in detail in Chapter 5, which discusses the detailed implementation of the model. A critical analysis follows in Chapter 6, which analyses the results of testing, and future recommendations to continue research on the system are also proposed. Finally a conclusion and summary is presented in Chapter 7, highlighting key findings of the research.

2. Literature Review

A variety of recommendation engines are implemented commercially on large scale databases, each of which contain a number of advantages and pitfalls. The following literature review discusses a number of different case studies of successful commercial recommendation engines, as well as the underlying algorithms and mathematical approaches used.

2.1 Applications of Recommendation Engines

The following case studies were particularly chosen based on the input data similarity to the insurance dataset structure being tested. Specifically, this involves a set of user features (gender, age, occupation, ethnicity, etc.) and the selection of a product or item.

2.1.1 Movie Recommendations

The use of recommendations for movie viewers is widely researched and implemented both academically and commercially. The first commercial case study involves the Netflix Prize solution for a recommendation engine (Koren 2008). The technique used collaborative filtering to predict user ratings for movies, given no information other than historic ratings from other users and films. The algorithm was designed to cope with high scalability with over $100,480,507$ ratings that $480,189$ viewers provided for $17,770$ movies (ibid.). Matrix factorization was the specific collaborative filtering technique used in the approach. Matrix factorization involves using linear algebra techniques to decompose the sparse user-item matrix into two complete matrices (Chris 2009). When multiplied together, the lower dimensional matrices produce a complete user-item matrix, with a recommendation score for every item and every user. While collaborative filtering techniques have been widely publicized as one of the most common recommendation engine approaches, the major drawback is the requirement to retrain the entire model to provide suggestions to a new user. This is referred to as a cold start problem, since new users have no selection history, and collaborative filtering cannot use any of their features to suggest or predict ratings. (Alan 2014).

2.1.2 Mobile Application Downloads

The next major commercial case study reviewed involves using a recommendation engine for the Google Play Store, to provide recommendations for new application downloads for Android users using wide and deep learning (Cheng et al. 2016). The implementation resulted in a significant increase in app sales after operationalizing the model (ibid.). Wide and deep learning is a new methodology built on a joint training approach which utilizes the benefits of both generalized linear models, as well as neural networks with a joint training (multi-task learning) approach. Joint training refers to the ability of simultaneously optimizing an objective function with multiple models, reinforcing the strengths of each model and mitigating the weaknesses. Generally, with joint training the results of each model's training iteration is backpropogated to the other model during the training process (Ruder 2017). The fundamental advantage of wide and deep learning is to explore the benefits of both memorization and generalization. Memorization relates to the ability to use linear regression to exploit the correlation in datasets, to predict the common sets of occurrences based on historical training data (Cheng et al. 2016). Memorization thus learns particular combinations of products or items that have been selected, and provides recommendations to users based on their previous selections (Arpit et al. 2017), but lacks diversity required to provide new interesting recommendations. Generalization refers to the ability to recommend unique combinations to users that have not necessarily occurred before (Cheng et al. 2016). These transitive properties are usually exploited in non-linear models like neural networks (Krueger et al. 2017). This tends to improve the diversity of recommendations served. However, the transitive properties can tend to recommend irrelevant items and over-generalize in certain cases (ibid.). While potentially quite a powerful technique, this approach is new with few successfully implemented case studies on a wide academic and commercial scale.

2.1.3 Online video streaming and e-commerce

The next case studies reviewed are the recommendation engines employed for e-commerce site Amazon and Youtube video streaming. The underlying algorithms and models used for both these cases are neural networks. Neural networks are usually used for content/model based recommendations, where the metadata surrounding the user-item interactions are also incorporated into the model (Schmidhuber 2015). In the context of Amazon and Youtube, the models utilized various features such as profile data, product information and other characteristics. Neural networks also allow for unstructured data such as product catalogue images or user profile pictures to be incorporated into the training data set. This technique has seen particular success using convolutional neural network architectures (O'Shaughnessy 2009), due to the ability to learn and process low level image features to make predictions with high accuracy. Other models such as recurrent neural networks have been used where sequential input data significantly contributes towards the recommendations provided (Zweig 2015), as in the case of online advertising campaigns on Youtube or Amazon where a sequence of clicks on a website might lead to a product being sold. One of the key advantages of this approach is the ability to perform non-linear transformations of interactions. This allows for complex interactions to be modelled in higher dimensions, or as a combination of different variables, which can express a better relationship with the recommended output. Conventional methods employ linear or algebraic matrix modelling techniques which might fail to model complex interaction patterns and combinations (Charu 2016). One of the major limitations of neural network approaches is the requirement for significant amounts of data to provide high quality recommendations (Cheng et al. 2016).

2.2 Other Case Studies

While the majority of recommendation engines use a variation of techniques discussed above, there are other approaches which have shown success. One of the most notable includes the use of linear regression to recommend academic resources (research papers, books etc.) to students (Thai-Nghe 2010). Firstly, similar to neural networks, generalized linear models use all features as inputs to predict a particular type of output. This differs from collaborative filtering, which only considers the selection history and performs similarity measures to find groups of users based purely on this selection history (Sweeney et al. 2016). Also, due to the simple scalability of linear models, this makes them particularly useful for large scale recommendation engines. Lastly, another advantage of generalized linear models is the interepretability. This refers to the ability to directly understand the relationship between input variables and output variables, which allows one to enhance model performance and understand factors and relationships which contribute towards high quality recommendations. This is often more difficult with other techniques like neural networks, where there is higher order, non-linear transformations of the input data set across multiple hidden layers. Another technique used for predicting purchasing propensity, commonly used in the retail industry, involves using association rule mining in market basket analytics (Wong and W. Yang 1997). Association rules primarily aim to understand the frequency of items that are purchased together. However, since these are primarily targeted towards understanding how items are sold together, and usually do not involve mapping user interactions, these approaches usually do not perform well within the context of user-driven recommendation engines.

2.3 Benefits of Research

As highlighted in the various studies shown above, recommendation engines offer a very relevant application of traditional statistical and machine learning tools to solve real-world customer matching problems. Performing further research in this field can assist in identifying enhancements to the ways current collaborative filtering, neural networks and other methods can be adapted to deal with massive data sets, thereby providing more accurate results and better scalability. This is important since the current surge in data volumes from various academic and industry sources require the current approaches to be refined. Furthermore, as discussed above, different recommendation engine approaches are suited for different input feature sets. Further research can help highlight where ensemble approaches, or joint models need to be implemented for recommendation engines to take advantage of the strengths of many different models to suit a particular feature set.

3. Proposed Methodology

As discussed with the various production recommendation systems in Chapter 2, there are a number of strengths and weaknesses of the various algorithms and approaches. The two key algorithms which are widely implemented are collaborative filtering and neural networks. Collaborative filtering is powerful in recommending content to existing users which have a selection history, but are subject to “cold-start” problems with new users that have no interaction history. Neural networks perform well when taking user and item features into account (age, gender, occupation, etc.) but for users with a wide selection history, this can over-specialize the recommendations where the user is never served new products, but only ones that they already have. Another typical problem experienced with a neural network based approach is keeping users in a filter bubble, where a user’s specific profile (age, gender, location etc.) can limit the variety of suggestions for users (Nguyen et al. 2014).

The proposed approach for an insurance dataset studied in this research will therefore involve a combined strategy of using neural networks for new users to avoid cold start problems, combined with collaborative filtering for existing users who already have one or more products. This chapter involves exploring each of these algorithms in more detail, before testing the performance of each technique in subsequent chapters. The actual implementation and detailed applied architectures of these techniques are discussed in Chapter 5.

3.1 Collaborative Filtering

Collaborative filtering mainly utilizes the selection history of users to generate a sparse user-item (customer-product) matrix mapping existing customers to previously selected products (Charu 2016). This sparse matrix serves as the foundation for collaborative filtering to provide relevant new suggestions of products or content through three different methods, namely user-based collaborative filtering, item-based collaborative filtering or matrix decomposition. The advantage of collaborative filtering is to find similarity without knowledge of the embedding space (Sarwar et al. 2001). This means that hidden features of the products or the users (age, category, preference) do not need to be explicitly stated, but these latent features can be discovered through similarity measures between users or between products/items (Sarwar et al. 2001). For example, in the context of insurance, certain products might be more suited towards older users (retirement annuities) and some products might be suited towards younger users (unit trusts for young professionals). In the absence of an explicit age feature, collaborative filtering can be useful in discovering age as a latent factor based on finding and grouping similar users who typically purchase age-related products. In this way, collaborative filtering is known to be both a supervised and unsupervised learning technique, where the discovery of latent features usually follows an unsupervised process of clustering users, and the ability to perform a prediction or recommendation for a specific product follows a supervised learning process.

3.1.1 User and item based collaborative filtering

In the case of user-based collaborative filtering, new products are recommended to users based on the choices of users most similar to the current user being recommended. After a sparse user-item matrix is obtained for all users, a similarity function is applied to every user to find similar users, and new products are served to users based on the most similar users choosing those products (Bracha 2015). For the current insurance data set, a cosine similarity function can be used, with the following formula:

$$Similarity(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{||\vec{a}||\,||\vec{b}||} = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{3.1}$$

As shown above, vector $\vec{a}$ and $\vec{b}$ represent the list of N ratings that 2 users have given for a set of products or items. The cosine similarity is determined by dividing the dot product of the two vectors, by the product of the vectors lengths. This determines the similarity of the 2 users. Once determining the similarity, products that are chosen by users are recommended for those users who are most similar to them that have not yet purchased those products.

Example 1. The above similarity can be demonstrated with an example from the insurance data set with 3 users and 3 products. In this case, the matrix displays which products a user has purchased or not purchased as a binary representation (1 for purchased, 0 for not purchased). Consider the user-item matrix in Table 3.1.

Customer	Unit Trust	Retirement Annuity	Life Insurance
1	0	1	1
2	1	0	0
3	0	1	0

Table 3.1: User item matrix for insurance customers

Using equation 3.1, the similarity between Customer 1 and Customer 3 can be calculated as follows:

$$\begin{aligned} Similarity(C_1, C_3) &= \frac{(0 \times 0) + (1 \times 1) + (1 \times 0)}{\sqrt{0^2 + 1^2 + 1^2} \times \sqrt{0^2 + 1^2 + 0^2}} \\ &= 0.707 \end{aligned}$$

Similarity ranges from 0 to 1, with 0 indicating highly dissimilar and 1 indicating highly similar. This indicates that customer 1 & 3 are quite similar, and they might have a preference for similar products. Both customers have purchased a Retirement Annuity (RA) product, while customer 1 has also purchased Life Insurance. Based on the similarity, this means we can now cross-sell Life Insurance to customer 3, based on his/her similarity to customer 1. Using the same approach, it can be calculated that the similarity between customer 1 & 2 is 0, and the similarity between customer 2 & 3 is also 0. This means that we would not be cross-selling any products between these two customers.

Item-based collaborative filtering is very similar to user-based collaborative filtering, but differs in that new recommendations are provided based on product similarity, as opposed to user similarity. In the example demonstrated above, this would perform the same calculation between products instead of users (between unit trust, RA and Life Insurance policy). New products are thus recommended to existing users if they are most similar to products the user has already purchased.

Typical similarity functions such as the cosine similarity shown above are efficient for binary interactions (seen/unseen, purchased/not purchased, etc.), while Pearson correlation can also be used for continuous variables (Buda 2010). In the case of larger databases with many users, K-nearest neighbour techniques are used to recommend frequent content only among the K-most similar users (Alan 2014), or other clustering techniques are applied to determine similarity in higher dimensions.

3.1.2 Matrix factorization

Matrix factorization is a subset of collaborative filtering. The outcome of similarity functions is to map the similarity of users and items along an N-dimension embedding space. For example, if we are mapping insurance product similarity in 1 dimension based on age, this would involve determining where each insurance product lies on the scale of “young to old”. These features may not always be obvious in the data set, and may not be as intuitive as age or gender. These hidden dimensions are known as latent features, which need to be experimentally determined. These are many dimensions which define similarity, and collaborative filtering aims to produce a vector for each item or user with a value which shows its preference for a particular dimension.

Example 2. Consider the example of customers and products shown in Table 3.2. In this case, customers provide explicit ratings or preferences for products on a scale of 1-5, with 5 indicating high preference and 1 indicating low preference. If a customer does not have a preference or has not purchased the product, it is indicated by n/a.

Customer	Unit Trust	RA	Life Insurance	Malpractice	Disability
A	3	n/a	1	n/a	1
B	1	n/a	4	1	n/a
C	3	1	n/a	3	1
D	n/a	3	n/a	4	4

Table 3.2: User item matrix for insurance customers with explicit ratings

This ratings matrix can be decomposed into latent features. Let's consider 2 latent features, gender and smoking habits. The customers can be expressed as either smokers/non-smokers, or male/female as shown in Table 3.3.

Customer	Gender (1= Male)	Habit (1=Smoker)
A	1	0
B	0	1
C	1	0
D	1	1

Table 3.3: Latent Features for customers

Similarly, the same latent features need to be applied to the products. In this case, we can assign a hypothetical range of how suitable a product is for a particular hidden feature. For example, on a scale of 0 to 5, let's assume disability insurance scores 3/5 in its suitability for smoking customers. Table 3.4 shows these scores for the products along the hidden dimensions.

Dimension	Unit Trust	RA	Life Insurance	Malpractice	Disability
Gender=Male	3	1	1	3	1
Habit=Smoker	1	2	4	1	3

Table 3.4: Latent Features scores for products

Now to determine if the latent features are correct, we express the user and item feature tables from Tables 3.3 and 3.4 in matrix form. The customer-feature matrix has the following form:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

The product-feature matrix has the following form:

$$\begin{bmatrix} 3 & 1 & 1 & 3 & 1 \\ 1 & 2 & 4 & 1 & 3 \end{bmatrix}$$

Now we multiply the customer features matrix by the product features matrix to get the following user-item matrix:

$$\begin{bmatrix} 3 & 1 & 1 & 3 & 1 \\ 1 & 2 & 4 & 1 & 3 \\ 3 & 1 & 1 & 3 & 1 \\ 4 & 3 & 5 & 4 & 4 \end{bmatrix}$$

The above matrix can once again be expressed in tabular form as shown in Table 3.5. This view shows that the original ratings are preserved, indicating that these latent features are suitably assumed. Furthermore, based on these latent features there are recommendations for ratings that were previously unknown (n/a). In this case, the hidden or latent features were trivial to deduce, but in practice there may be hidden features that are not as intuitive and predictable, and this process would need to be repeated with new latent feature sets, until an appropriate matrix is obtained.

Customer	Unit Trust	RA	Life Insurance	Malpractice	Disability
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

Table 3.5: User item matrix for insurance customers with explicit ratings

The discovery of these hidden features or dimensions are the main tuning parameters for collaborative filtering (Resnick et al. 1994). We can generally tune the algorithm to determine the best amount of latent features, provided that the number is less than half the harmonic mean of the number of users and number of items as shown below (ibid.):

$$k < \frac{U \times V}{2(U + V)} \tag{3.2}$$

where k represents the number of latent features, U represents the number of users and V represents the number of items. Selecting k values in this range can be used to appropriately find the divisors needed to factor the matrix into latent features (Spiliopoulos, Vouros, and Karkaletsis 2007).

Matrix factorization involves using linear algebra techniques to decompose the sparse user-item matrix into two complete matrices (Chris 2009), similar to the user and product feature matrices shown in Example 2 above. This is usually more efficient than performing user-based and item-based collaborative filtering individually, and then combining the results to make predictions as performed in the example. Specifically, matrix factorization involves factorizing the user-item interaction matrix into a user-feature matrix, and item-feature matrix. When multiplied together, the user-feature and item-feature matrices produce a complete user-item matrix, with a predicted recommendation score for every item and every user, as shown below:

$$A'_{ij} = U_i V_j \tag{3.3}$$

where $\mathbf{A}'$ represents the predicted matrix with rows i and columns j obtained from multiplying user features $\mathbf{U}$ and item features $\mathbf{V}$. The aim of the decomposition is to preserve the original user-item mappings while simultaneously obtaining new user-item combinations, as was the case in the example demonstrated. When decomposing the matrix, the original $(i \times j)$ matrix is split into two lower dimensional $(i \times k)$ and $(k \times j)$ matrices. The lower dimensional matrices consist of one matrix that has a row for every user, and one matrix that has a column for every product. The value of k is referred to as the number of latent factors, and constitutes the hyperparameter which needs to be tuned to generate accurate results, since different values of k produce complete matrices with different accuracies. Solving for the optimal $\mathbf{A}'$ involves minimising the loss function expressed below with least squared error:

$$min \sum_{i,j \in obs} (A_{ij} - U_i V_j)^2 \tag{3.4}$$

where $\mathbf{A}$ represents the original sparse matrix. True observations *obs* (actual purchased products) are located at row i and column j of matrix $\mathbf{A}$, and the sum is therefore computed for all non-zero entries in $\mathbf{A}$. The dimensions of $\mathbf{U}$ and $\mathbf{V}$ correspond to the number of latent features k.

This is much more computationally efficient than working with the entire sparse matrix, since the user-feature and item-feature matrices are of much smaller dimensions. Given a particular user ID, we can multiply the user features for that particular user by the product features to find predicted ratings for all products for that user. Vice versa, to find all user ratings for a particular product, we can use the product ID and multiply all item features by user features to find the ratings of all users for that particular product ID.

The matrix factorization approaches can be optimized using various algorithms to produce user and product feature matrices in a computationally efficient way. Techniques usually implemented for the purpose of recommendation engines include the following algorithms:

- Stochastic gradient descent (Gemulla et al. 2011)
- Alternating Least Squares (Takács et al. 2009) - discussed in detail in section 3.1.3
- Singular value decomposition (Koren, Bell, and Volinsky 2009)

All methods can be parallelized, which leads to faster computation times. However, stochastic gradient descent is weaker at handling unobserved interaction pairs (Takács et al. 2009). In the case of a sparse interaction matrix, with many unobserved interactions, it therefore would be more advisable to use alternating least squares in the algorithm approach. Singular value decomposition also performs poorly with unobserved pairs, since it requires real values for every entry of the matrix. This requires us to explicitly set unobserved interactions as null or zero, which can bias the predicted ratings significantly (ibid.).

Typical matrix factorization usually involves predicting numerical ratings for items by users. In the case of insurance product items, the user-item matrix purely provides user-item mappings, without explicit ratings. Explicit ratings are common in the case of movie or ratings datasets, where a score or rank is provided by each user for an item. A modification to the typical matrix factorization will be implemented with implicit feedback, which aims to weight the interaction with a particular interaction bias (Xing 2015). This interaction bias can be provided from any of the other features available in the data. For example, the insurance premium, the number of years that the user has owned the insurance product can be used as an interaction bias or implicit feedback mechanism. If a particular user has owned a product for many years, this might indicate a strong preference for the product.

3.1.3 Weighted Alternating Least Squares Algorithm

As discussed, the method of matrix factorization that will be implemented is the alternating least squares technique. The main aim of the alternating least squares method is to choose latent features which minimize the least square error when making recommendations, as shown in equation 3.4. This method however, only sums the error over the observed instances (actual purchases). A better approach for this is to weight the unobserved interactions to provide a low confidence score for these unobserved interactions, and add these to the least squares error sum as shown below (He et al. 2016):

$$\sum_{i,j \in obs} (A_{ij} - U_i V_j)^2 + w_0 \times \sum_{i,j \notin obs} (0 - U_i V_j)^2 \tag{3.5}$$

where in this case the unobserved interaction pairs are weighted with w_0. The weighted values for these pairs allow for better recommendations, and the optimal weight can be experimentally determined. Recall from section 3.1.2, the number of latent factors k needs to be selected in advance. The goal of the model is to choose the number of latent factors which produce the smallest least squares error.

Example 3. For this example we will revisit the user item matrix in Table 3.2, from Example 1 and 2 in section 3.1.2.

Customer	Unit Trust	RA	Life Insurance	Malpractice	Disability
A	3	n/a	1	n/a	1
B	1	n/a	4	1	n/a
C	3	1	n/a	3	1
D	n/a	3	n/a	4	4

Table 3.6: User item matrix for insurance customers with explicit ratings

Express the table in matrix form A_{ij}:

$$\begin{bmatrix} 3 & n/a & 1 & n/a & 1 \\ 1 & n/a & 4 & 1 & n/a \\ 3 & 1 & n/a & 3 & 1 \\ n/a & 3 & n/a & 4 & 4 \end{bmatrix}$$

Create an initial random customer feature matrix, with 2 latent features U_i:

$$\begin{bmatrix} 0.2 & 0.5 \\ 0.3 & 0.4 \\ 0.7 & 0.8 \\ 0.4 & 0.5 \end{bmatrix}$$

Create an initial random product feature matrix, with 2 latent features V_i:

$$\begin{bmatrix} 1.2 & 3.1 & 0.3 & 2.5 & 0.2 \\ 2.4 & 1.5 & 4.4 & 0.4 & 1.1 \end{bmatrix}$$

Multiply the user-feature and product-feature matrices to get a recommendation matrix U_iV_i:

$$\begin{bmatrix} 1.44 & 1.37 & 2.26 & 0.7 & 0.59 \\ 1.32 & 1.53 & 1.85 & 0.91 & 0.5 \\ 2.76 & 3.37 & 3.73 & 2.07 & 1.02 \\ 1.68 & 1.99 & 2.32 & 1.2 & 0.63 \end{bmatrix}$$

Now calculate the standard mean square error for all cases that there was a true interaction/observation:

$$\sum_{i,j \in obs} (A_{ij} - U_i V_j)^2$$

$$= (1.44-3)^2 + (2.26-1)^2 + (0.59-1)^2 + (1.32-1)^2 + (1.85-4)^2 + (0.91-1)^2$$
$$+(2.76-3)^2 + (3.37-1)^2 + (2,07-3)^2 + (1.02-1)^2 + (1.99-3)^2 + (1.2-4)^2 + (0.63-4)^2$$
$$= 35.67$$

For the cases with no interaction n/a, weight the interaction with w_0 of 0.1 and calculate:

$$w_0 \times \sum_{i,j \notin obs} (0 - U_i V_j)^2$$

$$= 0.1((0-1.37)^2 + (0-0.7)^2 + (0-1.53)^2 + (0-0.5)^2 + (0-3.73)^2 + (0-1.68)^2 + (0-2.32)^2)$$
$$= 0.27$$

Therefore, the total error in the solution is:

$$\sum_{i,j \in obs} (A_{ij} - U_i V_j)^2 + w_0 \times \sum_{i,j \notin obs} (0 - U_i V_j)^2$$
$$= 35.94$$

As shown above, this is a large error since the actual range of the ratings is between 1-5. Hence, the particular choice of latent features might not be a good fit, and a new selection will need to be initiated. To prevent overfitting, we can now also add regularization into the loss function. In summary, this produced the following loss function:

$$\sum_{i,j \in obs} (A_{ij} - U_i V_j)^2 + w_0 \times \sum_{i,j \notin obs} (0 - U_i V_j)^2 + \lambda(\sum_u ||U_i||^2 + \sum_j ||V_j||^2) \qquad (3.6)$$

Optimizing and solving for both the customer features U_i and product features V_j at the same time is not trivial. We rather initialize random customer features and product features. We then keep product features constant, and solve for the user features:

$$u_i = (\sum_{r_{ij \in r_{i*}}} v_j v_j^T + \lambda I_L)^{-1} \sum_{r_{ij \in r_{i*}}} r_{ij} v_j \qquad (3.7)$$

where $r_{ij \in r_{i*}}$ represents all columns of row i in original user-item matrix, v_j represents product features which will be held constant, and regularization with Ridge Regression (L2 Regularization) is conducted with constant λ. This process is repeated for every customer u_i in the batch selected, to solve for the customer features.

Once we have solved for a set of user features, we then alternate by keeping the user features constant, and then solving for the product features. This is the alternating nature of the WALS algorithm. Similarly, to solve for the latent features for every product v_j, we use the following formula:

$$v_j = (\sum_{r_{ij \in r_{*j}}} u_i u_i^T + \lambda I_L)^{-1} \sum_{r_{ij \in r_{*j}}} r_{ij} u_i \tag{3.8}$$

Where $r_{ij \in r_{*j}}$ represents all rows of column j in the original user-item matrix, u_i represents user features that will be kept constant, and regularization with Ridge Regression (L2 Regularization) is conducted with constant λ. This process is repeated for every product V_j in the batch selected. The pseudo-code for this algorithm is shown in Appendix B. This is an iterative process which obtains an optimal **U** and **V** matrix, to produce the best latent features for each customer and each product, such that when they are multiplied together the ordinary least squares error is minimized in the predicted matrix. After a customer feature matrix, and product feature matrix is computed, the accuracy is calculated, and the process is repeated until the error decreases to a desired level.

Example 4. The test data from Example 3 can be further extended to demonstrate solving for customer and product features, using the same user-item matrix of customer and product interactions, as well as the randomly generated user feature matrix, and product feature matrix. Using the initialized feature matrices with 2 latent dimensions, we can optimize and solve for the best latent features as follows:

$$u_i = (\sum_{r_{ij \in r_{i*}}} v_j v_j^T + \lambda I_L)^{-1} \sum_{r_{ij \in r_{i*}}} r_{ij} v_j$$

We can experimentally pick a regularization constant of 2.3. For the first user u_1, the customer features can be calculated by keeping the first row of the random generated product features V_j constant.

$$u_1 = (\begin{bmatrix}1.2\\2.4\end{bmatrix}\begin{bmatrix}1.2 & 2.4\end{bmatrix} + \begin{bmatrix}3.1\\1.5\end{bmatrix}\begin{bmatrix}3.1 & 1.5\end{bmatrix} + \begin{bmatrix}0.3\\4.4\end{bmatrix}\begin{bmatrix}0.3 & 4.4\end{bmatrix}$$

$$+ \begin{bmatrix}2.5\\0.4\end{bmatrix}\begin{bmatrix}2.5 & 0.4\end{bmatrix} + \begin{bmatrix}0.2\\1.1\end{bmatrix}\begin{bmatrix}0.2 & 1.1\end{bmatrix} + 2.3\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix})^{-1}$$

$$\times(3\begin{bmatrix}1.2\\2.4\end{bmatrix} + 1\begin{bmatrix}0.3\\4.4\end{bmatrix} + 1\begin{bmatrix}0.2\\1.1\end{bmatrix})$$

$$= \begin{bmatrix}0.6 & 0.2\end{bmatrix}$$

This returns the optimized user features for u_1. We repeat the process for the remaining user features. We then alternate to solve for the product features using equation 3.6. This produces the following user feature matrix after 1 iteration:

$$\begin{bmatrix} 0.6 & 0.2 \\ 0.3 & 1.4 \\ 1.8 & 0.5 \\ 1.7 & 1.9 \end{bmatrix}$$

The following product feature matrix is obtained after 1 iteration:

$$\begin{bmatrix} 2.2 & 1.5 & 1.9 & 2.4 & 0.8 \\ 1.4 & 1.6 & 4.8 & 0.5 & 2.6 \end{bmatrix}$$

When the user feature and product feature matrices are multiplied, this produces the following prediction matrix:

$$\begin{bmatrix} 1.6 & 1.22 & 2.16 & 1.6 & 1 \\ 2.62 & 2.69 & 7.29 & 1.45 & 3.88 \\ 4.66 & 3.5 & 5.82 & 4.75 & 2.74 \\ 6.4 & 5.59 & 2.35 & 4.2 & 5.64 \end{bmatrix}$$

The error calculated from the above matrix is 28.32, which is lower than the initial 35.94 calculated for Example 1. This indicates that already after 1 iteration of optimizing the customer and product features, the error is decreasing. This process can then be repeated until a desired error is achieved.

3.1.4 Other collaborative filtering techniques

There are other subset techniques of collaborative filtering which purely model item-item interaction. Such techniques usually use association rule mining in market basket analytics to understand purchase propensity (Wong and W. Yang 1997). Association rules primarily aim to understand the frequency with which items are purchased together. For example, if unit trusts and retirement annuities are frequently purchased products, they will be recommended regardless of the user. Due to the less personal nature of item-item interactions, and the abundance of user data available in our dataset, item-item interactions will not be considered in this research, but will be noted for literature purposes.

3.2 Neural Networks

The use of neural networks for both supervised and unsupervised learning has gained popularity in recent years due to the advancements of hardware architectures and scalable computing environments which are optimized for neural network training models (Schmidhuber 2015). In the context of recommendation engines, neural networks are usually used for content/model based recommendations, where the meta-data surrounding the user-item interactions are also incorporated into the model (O'Shaughnessy 2009). For example, with insurance data, this does not simply restrict the data to the user-item matrix as an input source, but also uses user features (age, profession, gender, etc.) as well as product features (category, price, etc.). An advantage of neural networks is that interaction terms do not usually need to be explicitly handled prior to input. Interaction terms usually are seen in regression problems, where a combination of terms serve as a better input to the model than the terms individually. For example, in insurance, premium amount and age might have significant correlation since older individuals tend to pay higher life insurance amounts. Due to the multiple layers of a neural network, these interactions are usually derived implicitly through the modelling process (Schmidhuber 2015). The following discussion provides an overview of the methodology of a neural network architecture primarily suited for a classification problem. The exact architecture and implementation details for the insurance data-set will be provided in Chapter 5, after the data has been described and analysed. However, a sample architecture of a typical feed-forward neural network with 4 input features, one hidden layer and a classification output is shown in Figure 3.1.

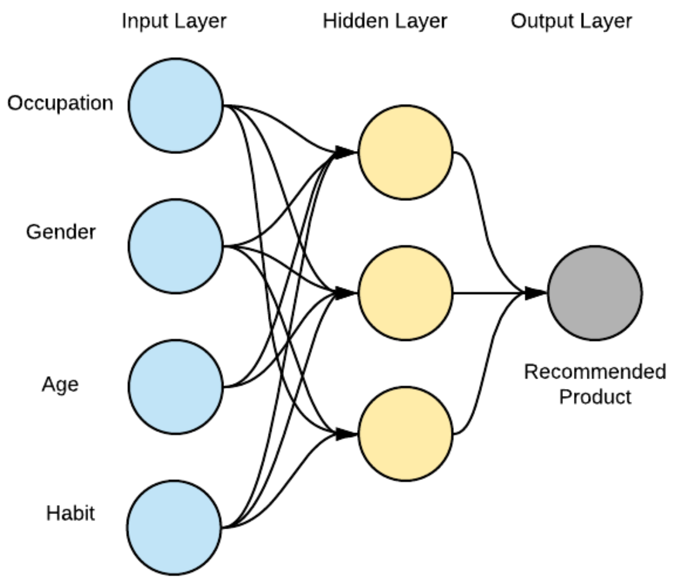


Figure 3.1: Typical Feed-Forward Neural Network

The network of neurons (represented as circles) across the various layers provide a set of transformations which map a set of inputs towards a distinct output (ibid.). In this case it would be to determine the best product to recommend an insurance customer (classification output). The feed forward architecture connects multiple layers of neurons, and an input propagates through the network to return an output value or prediction. To determine the activation a^i_j, which is the j^{th} neuron value in the i^{th} layer, the following formula is used (Nielsen 2015):

$$a^i_j = \sigma(\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j) \tag{3.9}$$

As shown in formula 3.8, the activation of a particular neuron is dependant on the activation of a previous layer, where w^i_{jk} is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer applied to the j^{th} neuron in the i^{th} layer. b^i_j refers to the bias of the j^{th} neuron in the i^{th} layer. The bias for a nueron is analogous to an offset or intercept in linear models, where the bias delays the onset of the activation function (ibid.). Lastly, σ refers to the activation function. Given a set of input data points, the activation function is an important element which determines whether the network of neurons will start firing for the particular input. This checks whether the input data is within the allowable range of data points required for the particular model (Haykin 1994). There are a number of different activation functions used, the most common being sigmoid, hyperbolic tangent and rectified linear unit, depending on the bounds of the input data points. The following sigmoid function will be used in the sample architecture:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3.10}$$

The above concepts can best be illustrated through an example. Consider the basic neural network architecture shown in Figure 3.4.

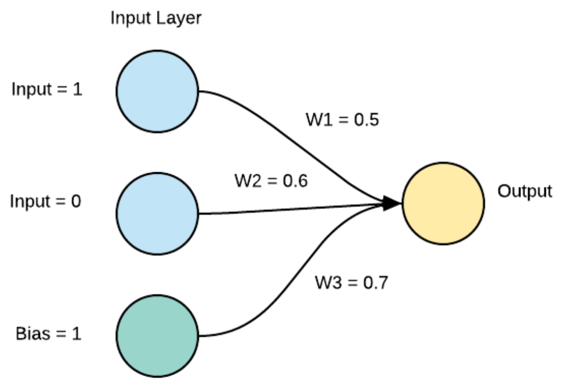


Figure 3.2: Simple neural network example

This configuration has 2 inputs, a bias and a single output. In the context of insurance recommendation, this could represent two binary feature inputs (male/female, and smoker/non-smoker) to recommend one of two products (sickness benefit or unit trust). Weights are randomly initiated, and these will need to be adapted to enhance the accuracy of the neural network. Using the formula 3.9 with activation function 3.10, the output can be calculated as follows:

$$Output = a((I_1 \times W_1) + (I_2 \times W_2) + (B \times W_3))$$
$$Output = a((0 \times 0.5) + (1 \times 0.6) + (1 \times 0.7))$$
$$Output = a(1.3)$$

We now apply the sigmoid activation function a

$$Output = \frac{1}{1 + e^{-1.3}}$$
$$Output = 0.79$$

As shown, the output in this case is 0.79. In the case of a product recommender, an output range of 0-0.5 could correspond to one product, and an output in the range of 0.5-1.0 correspond to a different product. If there is a large discrepancy in the outputs from expected results, the weights can be adjusted to better model the mapping of inputs to outputs. Furthermore, the above system can also represent a single layer in a complex neural network. The inputs in this case can represent the outputs from previous layers, and the outputs represent the inputs to the next layer. However, the calculations and methodology remains the same to derive the final output or recommendation.

In practice, the neural network can range from simple single layer networks which resemble generalized linear models with each neuron representing a different variable, to multi-layered networks which contain a number of non-linearities. For example, simpler neuron architectures are popular in the finance industry, especially around risk behaviour. A case study shows that a single layer neural network can be used to determine whether a credit card customer will default on a payment (Bherde 2015). Multi-layer architectures are common in cases where the predicted output is highly nonlinear, such as speech and natural language prediction (Graves, Mohamed, and G. Hinton 2013). In the case of insurance data modelled in this study, the various features of a customer (age, gender, occupation, etc.) are used as inputs during the training process, and the weights of each neuron in corresponding layers are adapted in order to produce an output that most accurately maps the input to the predicted output (Schmidhuber 2015).

The goal of neural network training is to minimize a cost function for regression or for classification. Cost functions assist in determining the error of the predicted values from true values, using various algorithms. One of the most typical algorithms use gradient descent methods (O'Shaughnessy 2009). With gradient descent, it is important to ensure that the function which measures the difference is differentiable with respect to the outputs. This ensures that a gradient search can be performed to find a local or global minima which reduces the overall error. Once a desired error level is obtained, weights no longer need to be adjusted in the neural network and a converged solution is achieved. For the insurance product predictor that will be designed, the model will provide a classification output. For classification, one of the most widely used cost functions are the cross-entropy cost function for multi-layer neural networks (Nielsen 2015):

$$C = -\sum_{i=1}^{K} y_i \log \widehat{y_i} \tag{3.11}$$

The above equation calculates the total loss over all K outputs, based on the difference between the expected output y and the output generated from the neural network $\widehat{y}$.

Finally, a major pitfall to avoid when designing a neural network is preventing overly complex architectures from overfitting the training data. Generally to prevent overfitting, neural network training involves model fitting with regularization to penalise the model for complexity. In each iteration of training, minimizing the cost function includes a regularization term which penalizes the weights of the neural network to prevent overfitting. This makes the model better at generalisation for new data. One of the more commonly implemented regularization techniques includes Ridge Regression (L2 Regularization), which adds a regularization or "penalty" term to the Cost function as shown below:

$$C = -\sum_{i=1}^{K} y_i \log \widehat{y_i} + (\sum_{j=1}^{n} ||\mathbf{w}^j||^2)\frac{\lambda}{2m} \tag{3.12}$$

The above equation adds a regularization term to the cross-entropy cost function. The regularization term multiplies the sum of all sqaured weights $\mathbf{w}$ across the n neurons with a regularization term λ, scaled by m number of inputs. To better understand the technique of regularization, we will revisit the example discussed earlier. Table 3.7 provides different configurations of the neural network setup in Figure 3.2.

Architecture	Inputs	W_1	W_2	W_3	$\hat{y}$	y
1	$I_1 = 0, I_2 = 1$	0.5	0.6	0.7	0.79	1
2	$I_1 = 0, I_2 = 1$	0.2	0.5	0.8	0.79	1

Table 3.7: Different configurations of weights for simple neural network

As shown, both configurations produce the same expected output. Now, to consider the effects of regularization, we compare the regularized loss function to a standard mean square error calculation. This is calculated using an experimentally chosen λ of 2.3. Using equation 3.6, the cross-entropy loss is calculated for Architecture 1 as follows:

$$C = -(1 \times log(0.75)) + (0.5^2 + 0.6^2 + 0.7^2)(\frac{2.3}{2 \times 2})$$
$$C = 1.274$$

The cross entropy loss can be calculated in a similar way for Architecture 2. The results are indicated in Table 3.8, along with the mean square error for comparison.

Architecture	$\hat{y}$	y	$(y - \hat{y})^2$	λ	Cross Entropy Loss
1	0.79	1	0.044	2.3	1.274
2	0.79	1	0.044	2.3	1.079

Table 3.8: Different configurations of weights for simple neural network

As shown in the example, 2 different architectures that produce the same output and same mean square error can be penalised differently with a regularised loss function. In this case, architecture 1 is penalised more heavily based on the choice of weights. This indicates that model 1 may be more prone to overfitting. In context of the insurance product recommender, this means that architecture 2 may be better at recommending products for users.

The learning method for neural networks depends on the type of problem being solved (supervised, unsupervised, or a combination). In the case of the insurance predictions, this would be considered a supervised learning problem. Neural networks also allow for unstructured data such as product catalogue images or user profile pictures to be incorporated into the training data set. In the case of fashion or retail recommendation engines, this technique has seen particular success with convolutional neural network architectures (O'Shaughnessy 2009), due to the ability to learn and process low level image features to make predictions with high accuracy. Other models such as recurrent neural networks have been used where sequential input data significantly contributes towards the recommendations provided (Zweig 2015), as in the case of online advertising campaigns where a sequence of clicks on a website might lead to a product being sold. There are therefore a number of different architectures possible based on the format of the input data (Covington, J. Adams, and Sargin 2016). For the purpose of this research, a typical feed-forward neural network will be used (Schmidhuber 2015).

One of the key advantages of using a neural network approach for recommendation engines include the ability to perform non-linear transformations of user-item interactions. Conventional methods, including collaborative filtering, employ linear modelling techniques which might fail to model complex interaction patterns and combinations (Charu 2016). Furthermore, neural networks can also be used alongside collaborative filtering strategies, as a form of preproccesing of information prior to typical collaborative filtering techniques being employed (Bracha 2015). Often user-item interactions may not be directly obvious, as in the case of obtaining sentiment from a product review or comment. Such cases require powerful natural language processing and text mining techniques which utilize neural networks to build a candidate user-item matrix which can then be processed using traditional collaborative filtering techniques (L. Zhang, S. Wang, and B. Liu 2018).

3.3 Combined Approach

There are drawbacks to standard collaborative filtering techniques which rely solely on user-item ratings as the source data. Collaborative filtering allows for general recommending ability across broad categories, while other techniques such as neural networks utilize user and item features and allow for deeper modelling of localized relationships. Furthermore, with standard collaborative filtering techniques, there is the requirement to retrain the entire model to provide suggestions to a new user, only once the new user has made some purchases. As previously mentioned, this is referred to as a “cold start” problem, since new users have no product history, and the model cannot use any features or relationships to suggest or predict ratings (Alan 2014). Constantly retraining the model for new users is also computationally unfeasible.

One of the major limitations of neural network approaches is the requirement for significant amount of data to provide high quality recommendations (Cheng et al. 2016). Another drawback of neural networks is the requirement for significant hyper-parameter tuning, as opposed to other techniques. This also tends to give rise to a lack of interpretability due the multiple hidden layers and non-linearities inherent in the models for neural network architectures. This makes it difficult to determine aspects like variable importance in a model. However, there have been large scale implementations of neural network based recommendation systems (Covington, J. Adams, and Sargin 2016), as commercially seen in the case of Youtube and Amazon.

Collaborative filtering and neural networks therefore both have a number of strengths and weaknesses. In summary, new users with no (or limited) interaction history are better suited for recommendations from neural networks. In the context of insurance users, this could represent customers that don’t hold any products and have no purchase history. Because there is limited data on their choices, neural networks can use other features of the customer like age, profession, gender and smoking habit. Neural networks perform well with this metadata about the customer, and can produce high quality recommendations. Users and items with significant interaction data are more suited for recommendations with collaborative filtering. This represents insurance customers that have been members or customers for a long time, and have multiple products in their portfolio. To test this across the models, validation data for the collaborative filtering model will include customers with only an interaction history. For neural networks, validation data will involve only new customers with no interaction history.

4. Data Exploration

This chapter aims to explore the enterprise insurance data-set available for analysis, as well as all relevant statistical features required for quality machine learning model development. Note that the following analysis is conducted after extraction from a production data warehouse. This warehouse is the running database system where the core tables are located. Following the extraction of this data, wrangling operations are required to format data appropriately for statistical analysis.

4.1 Data Schema and Entity Relationship Diagram

An entity relationship diagram of the databases extracted from the warehouse is shown in Figure 4.1.

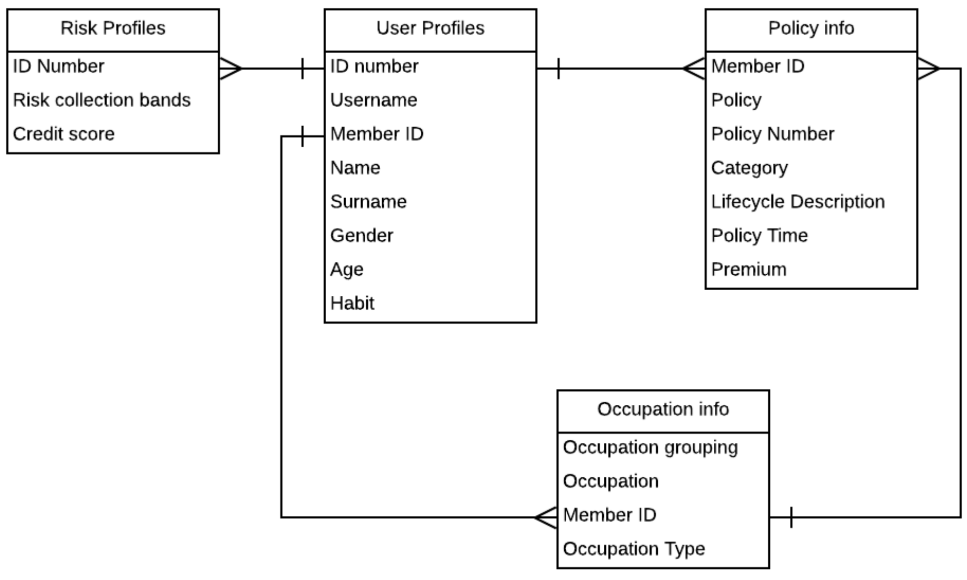


Figure 4.1: Entity Relationship Diagram of current database structure

The following key tables are stored in the data warehouse:

- User profiles - this table consists of core identification data for the various insurance clients. Due to privacy and ethical concerns, this data will not be used directly in the modelling process. The member ID will be retained as a primary key and used as a point of reference across other tables, along with the gender and age of the member, since these will be useful features for content based models.
- Risk profiles and credit score - This table provides key risk bands for members as supplied by the South African credit bureau, along with a credit score. This information is used to calculate members' affordability of certain insurance products, as well as verifying authenticity of applications.
- Policy information - This table consists of the policy details of existing members and various terms relating to the products. Policy time refers to how long the particular policy holder has been active as a member.
- Occupation information - The occupation information table is used to determine the profession and industry details of current members. This is mainly used to determine the fit of an appropriate insurance product for different professionals.

4.2 Preprocessing and Analysis

After data extraction was completed from the data warehouse, the final dataset that was generated for model building is displayed in the schema in Table 4.1.

Field Name	Data Type	Examples
Member ID	Integer	61627384, 263748423, 237483948
Occupation	String	Financial, Medical, Scientific
Premium	Float	140.89 , 690.4, 1223.34
Gender	String	Male, Female
Habit	String	Smoker, Non-Smoker
Age	Integer	63, 67, 31
Time	Integer	12, 8, 15
Product	String	DISA, SPPI Supp A, Sickness Benefit

Table 4.1: Details of insurance variables used for recommendation engine

As shown, the key fields that are extracted are occupation, member age, policy time (years), gender, habit (smoker/non-smoker), premium amount and product type. The machine learning model will thus aim to predict the product type, providing the age, gender, occupation, premium amount, habit, and policy time as inputs. As shown in Figure 4.2, there are a number of different products that can be chosen. The codes for the various products are explained in Appendix C. Typical products include retirement annuities for people looking to invest in a pension account, sickness cover for people who are unable to work and require insurance payouts for days of work missed, disability benefits for insurance cover in the case of permanent injuries, as well as other general purpose insurance products. Some products are specific to certain professions. For example, in the medical or legal industry, certain liability insurance may be required to protect professionals against malpractice or other issues.

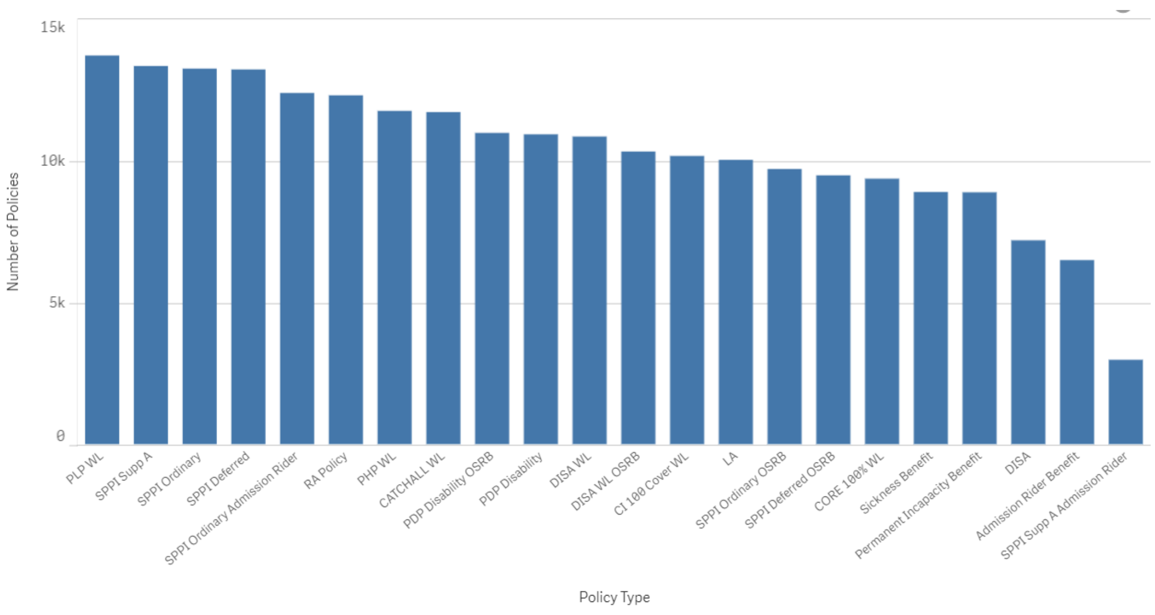


Figure 4.2: All products in data showing number of corresponding members

Age, gender and smoking habit are useful input features for a product recommender. There are key products which target people of specific age groups (young working professionals or people close to retirement). There are also gender specific products (unit trusts for females). In terms of health and well-being, smoking habits might place individuals at risk for certain health-driven insurance products (sickness benefits). Premium amount is an important feature to consider as an input as this can give an indication of the budget that a prospective customer can provide towards the insurance product. Individuals seeking more comprehensive cover will have higher budgets for their monthly instalments. Policy time is an indication of how long the particular customer has been a member with the insurance company, and how long a particular product has been in force on the customer profile. However, policy time can vary based on different products that are chosen. Usually, customers that are happy with their products retain them for many years.

The preprocessing involves transforming the data set into useful inputs which can be efficiently processed by the various modelling strategies discussed in Chapter 3. For efficient model building, this is important since the data can be unbalanced, contain missing values or contain too many anomalies which can reduce the accuracy of the statistical models being generated.

4.2.1 Feature Analysis

The next stage involves analyzing the various features for statistical relevance, and ensuring that the data sets are balanced. This ensures that there is enough training data for the various numerical and categorical features to build high quality models. The first feature that will be analysed is the output product feature, which originally contained 38 different types of products which members can purchase. However, when analysing the number of people with various products, it is evident that certain products have very few members (8-20) while other products have large member databases (over 10 000). Including products with few members increases the complexity of the modelling process, since this increases the sparsity of the user-item matrix for collaborative filtering, as well as the extensive mapping required for categorical variables. This often involves using modelling techniques to deal with high cardinality variables. Products with fewer than 100 members were neglected. This produces the distribution of products as shown in Figure 4.2. The distribution between 22 different products shows between $5000 - 15000$ members each, which is a significant amount of data per product for modelling purposes. Credit scores, ID numbers and Risk bands were ommitted from the dataset due to restrictions by the credit bureau based on POPI requirements (Protection of Private Information act 2013).

The next statistical comparison involves comparing relationships between the various features, to understand if there are particular features which directly or indirectly affect the output feature product. The scatter plot in Figure 4.3 displays the distribution of the various products according to age and product price. In this case, premium refers to the monthly amount that people invest in their product. Higher premiums indicate larger portfolios and investments into insurance products. Age and premium certainly seem to drive product choice. There also appears to be similar products grouped together across the scatter plot. For example, RA and LA products are typically chosen by older people and premiums are very expensive, while Sickness Benefits have lower premiums and are mostly purchased by younger people.

Following this, the next exploration conducted was to understand the relationship between the different genders, premiums and whether smoking affects the insurance products. Figure 4.3 has shown distinct groups where products are spread across a range of premium amounts. The various codes for the products are explained in Appendix C. Figure 4.4 displays how this premium amount is affected by smoking habits for both male and female. In general, smokers tend to have higher premiums (possibly due to health risks and especially among male members), and in general males tend to pay more for insurance than women. This is important to factor in, since the type of product recommended will therefore be influenced by gender and smoking habit.

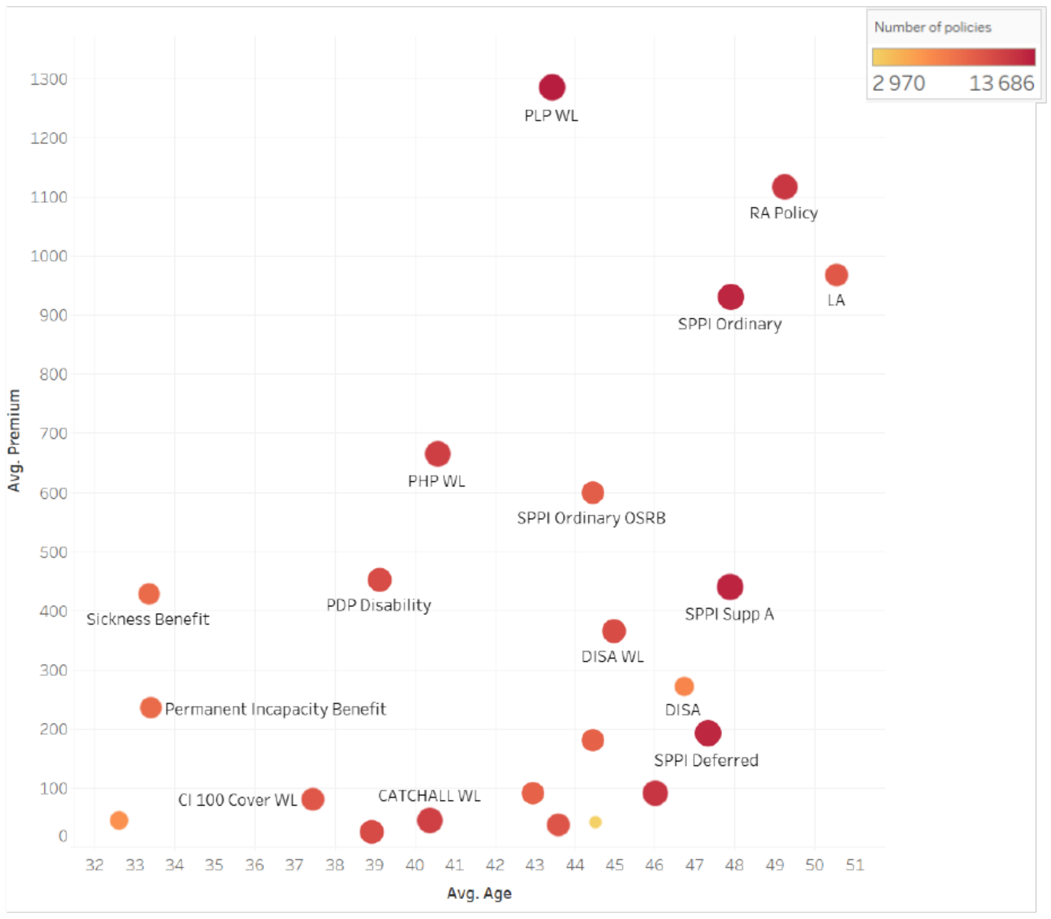


Figure 4.3: Scatter Plot showing Average Age and Average Premium for various products

Lastly, another set of important features is the occupation group and time of policy. We explore the relationship of these features in Figure 4.5, where premiums vary significantly for the occupation groups. Medical and legal professionals for example, pay much higher premiums possibly due to the higher risks in the profession or due to affordability for high end products. Policy time refers to how long individuals have retained a particular policy. As shown in the various groups, policy time does vary between the different profession groups, with legal and medical groups keeping products the longest, and science and academic groups having products or membership status for the least amount of time on average. Other views are shown in Appendix A.

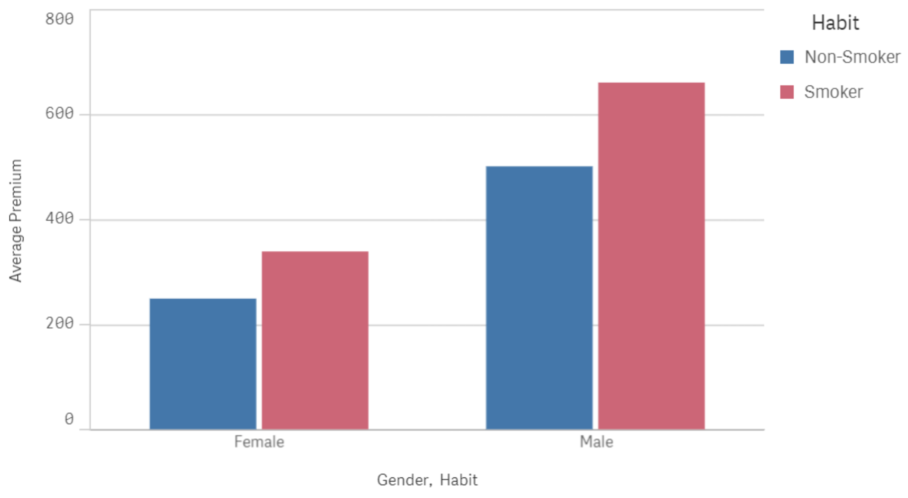


Figure 4.4: Bar graph showing the variation in premium amounts by gender and smoking habits. This indicates generally higher premiums for males and for smokers.

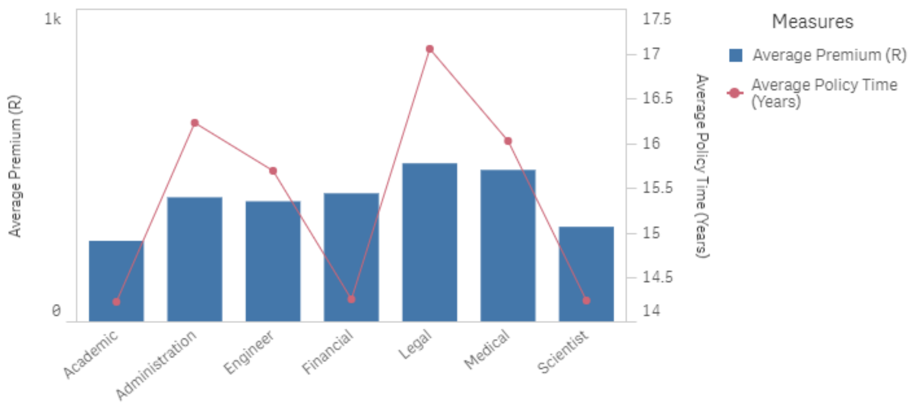


Figure 4.5: Bar graph showing average premium amounts and policy time for different occupation groups

4.2.2 Cluster Analysis

After the initial descriptive analysis provided above, it is evident that the data set exhibits some form of grouping according to its various features. This is clear with certain age groups, genders or professions having a preference for certain types of products. A more statistical way to explore the potential groups in a data set is to perform a cluster analysis. Clustering is an unsupervised machine learning technique, since there is no target variable to predict (Xu and Wunsch 2008). Instead, the analysis aims to determine whether the data can be divided into a set of groups, in which each group contains similar data points (ibid.).

The first step in performing the cluster analysis involves determining a similarity measure between all the data points, using a distance measure or other similarity function. The particular metric that will be used in this case is the Gower dissimilarity function (Xu and Wunsch 2008). The Gower dissimilarity is particularly useful in that it can be used to measure the distance between data points whose features are a mix of both categorical and continuous values, as in the case of the insurance data set. The Gower dissimilarity is essentially a measure between 0-1 of the average dissimilarity between 2 data points (rows). The dissimilarity is averaged across all features of the data points, and is calculated with the following function to determine the dissimilarity between rows i and j in the dataset:

$$d(i,j) = \frac{1}{p}\sum_{f=1}^{p} d_{ij}^{(f)} \tag{4.1}$$

where p represents the total features in a row, and f represents the particular feature being measured. The particular dissimilarity measure $d_{ij}^{(f)}$ is dependant on the type of feature being compared. For continuous variables, the following formula is used:

$$d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{R_f} \tag{4.2}$$

In the above equation, the absolute value of the difference between row i and j for feature f is divided by R_f, which represents the range of the continuous variable. For categorical variables, $d_{ij}^{(f)}$ is 0 if the values have the same value, and 1 otherwise.

Once the pairwise dissimilarities are obtained between all points in the data set, a clustering technique known as partitioning around medoids (PAM) can be applied (also known as k-medoids clustering) (ibid.). This algorithm selects certain points in the data set as cluster centers, and aims to minimize the distance between points labelled as a part of the cluster, and the center of the cluster. This process is iteratively repeated with new cluster centers selected, until an optimal number of clusters are formed (ibid.).

Prior to performing PAM clustering, the number of clusters need to be selected. The optimal number of clusters can be determined by measuring the average cluster silhouette width for various cluster sizes (Aranganayagi and Thangavel 2007). The silhouette width contrasts the average distance of points within the same cluster, with points in other clusters (ibid.). Generally, a higher silhouette width indicates better separability in clusters. Figure 4.6 displays the various average silhouette widths for different cluster sizes after PAM clustering is applied on the data points. As shown, both 5 clusters and 3 clusters generates high silhouette widths. However, when visualised, 5 clusters produced more interpretability and grouping in the data. The optimal number of clusters is therefore chosen as 5.

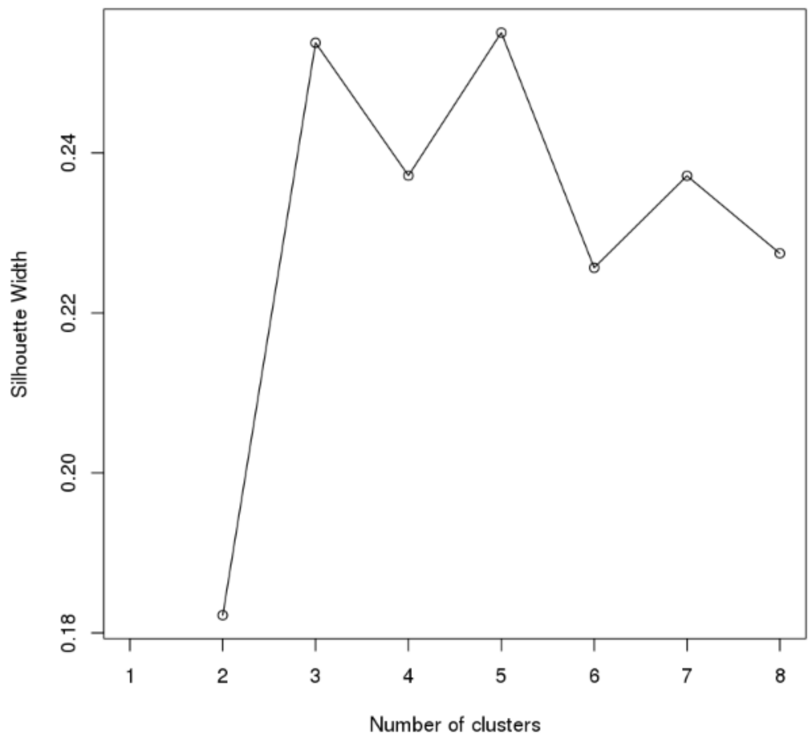


Figure 4.6: Average silhouette widths for different cluster sizes after PAM clustering is performed

After the clustering is performed, a lower dimensional visualization of the clusters can be obtained, which can help in identifying general groups and whether certain points clearly cluster in certain regions. Figure 4.7 displays this visualization.

As shown, there are distinct regions of the xy plane associated with each cluster. Hence, data points can be well grouped and separated according to their particular cluster, with minimal overlapping regions. There are outlier points which do not fit exactly within their cluster region, however this is due to boundary cases in data where feature sets are very similar. A final analysis would be to list highlighted features of each cluster, to determine the particular features that are strongly associated with each cluster. For continuous variables age, policy time and premium, the average values of the group are indicated. For example, the average age of members in cluster 1 is 43 years old. For categorical variables, the most prominent variable is indicated, along with its representation in the group. For example, the most prominent members in Cluster 2 are medical professionals, making up 76.34% of the group. Table 4.2 shows the results of the clustering process, for features that were distinctly similar among the group.

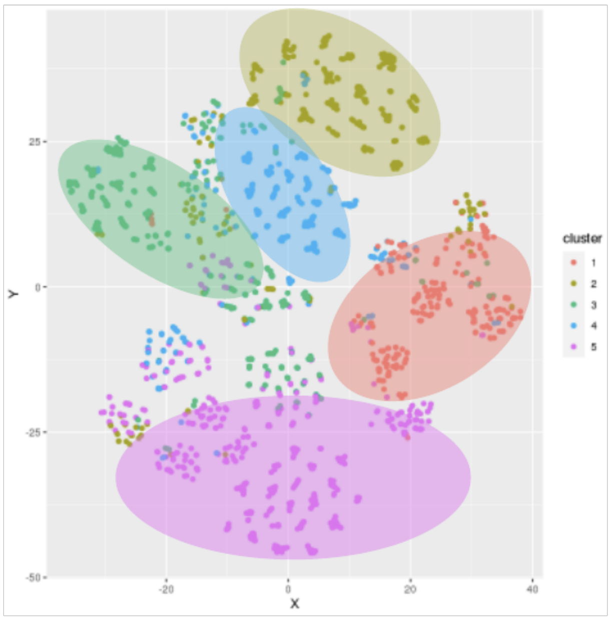


Figure 4.7: Lower dimensional visualization of clustering

Feature	Cluster 1	Cluster 2	Cluster 3
Prominent Occupation	Legal 61.23%	Engineer 76.34%	Legal 71.63%
Average Age	43	41	50
Average Premium	751.46	343.59	727.34
Average Policy Time	16.58	14.35	22.05
Prominent Product	PLP WL 22.23%	SPPI Supp A 26.21%	RA 21.34%
Prominent Habit	Smoker 74.78%	Non-smoker 67.23%	Non-smoker 66.23%
Prominent Gender	Male 58.42%	Male 60.23%	Male 63.34%

Feature	Cluster 4	Cluster 5
Prominent Occupation	Financial 81.23%	Medical 76.34%
Average Age	45	35
Average Premium	483.46	218.59
Average Policy Time	15.58	10.35
Prominent Product	SPPI Ord 32.83%	Sickness Benefit 26.72%
Prominent Habit	Non-smoker 69.28%	Non-smoker 73.91%
Prominent Gender	Male 54.34%	Female 88.23%

Table 4.2: Feature Highlights after PAM Clustering

As shown in table 4.2, the most prominent feature in each group is occupation, with the 5 clusters mainly separating into the Medical, Legal, Engineering and Financial Groups. Premium and policy time are continuous variables which vary among the groups. Finally, the most common product is also specific to each group. The clustering process therefore indicates that there is definitely key features which drive certain types of product uptake. For example, from the above table, it can be seen that female medical professionals in their thirties, who are aiming to pay around R300.00 for premiums, and who have had insurance for over 10 years will be suited for the Sickness Benefit.

4.2.3 Categorical Transformations

The next step of feature engineering is dealing with categorical data. One-hot encoding be used to map the distinct values of each of the categories to new columns, with binary fields indicating the presence or absence of the feature. For example, gender will be mapped into *isMale* and *isFemale* columns which contain 1's and 0's indicating presence or absence of these features. This is usually referred to as the creation of dummy variables (Harris 2011). Since certain features can be derived from each other (as in the case of *isMale*/*isFemale*), some variables are dropped to ensure multi-collinearity does not take place. The variables that are dropped become the reference categories.

4.2.4 Continuous Distributions

In addition to processing the categorical features, the continuous variables (policy time and age) are also analysed to assess whether data normalization, standardization and skewness should be addressed. The plots in Figure 4.8 display the distributions of age, policy time and premium amount.

The major concern to address in the histogram is the monthly premiums, since this data is skewed, with the majority of premiums between 0-1000. A log transform is performed on this variable, to produce Figure 4.9. As shown, the log transformed premium amounts are better distributed across the new range.

Following this transformation, the key age ranges are from 20-80, the transformed premium amounts are between 0 and 5 and the valid policy times are from 0-50 years. Negative values are discarded since these are invalid values for all features. Due to the significantly different ranges, data normalization will be implemented. Since age and policy time have a much larger range than premiums, the large difference in variance would load more on the principal components in this data set. Normalization, as opposed to standardization, will be used since irrelevant outliers in the data were already eliminated in the previous step. This is important since removing outliers ensures that variance caused by these outliers will not be scaled as well, which can reduce the range of valid data. This ultimately produces all values within a range of 0 and 1 for premium, age and policy time using formula 4.3.

$$x_* = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.3}$$

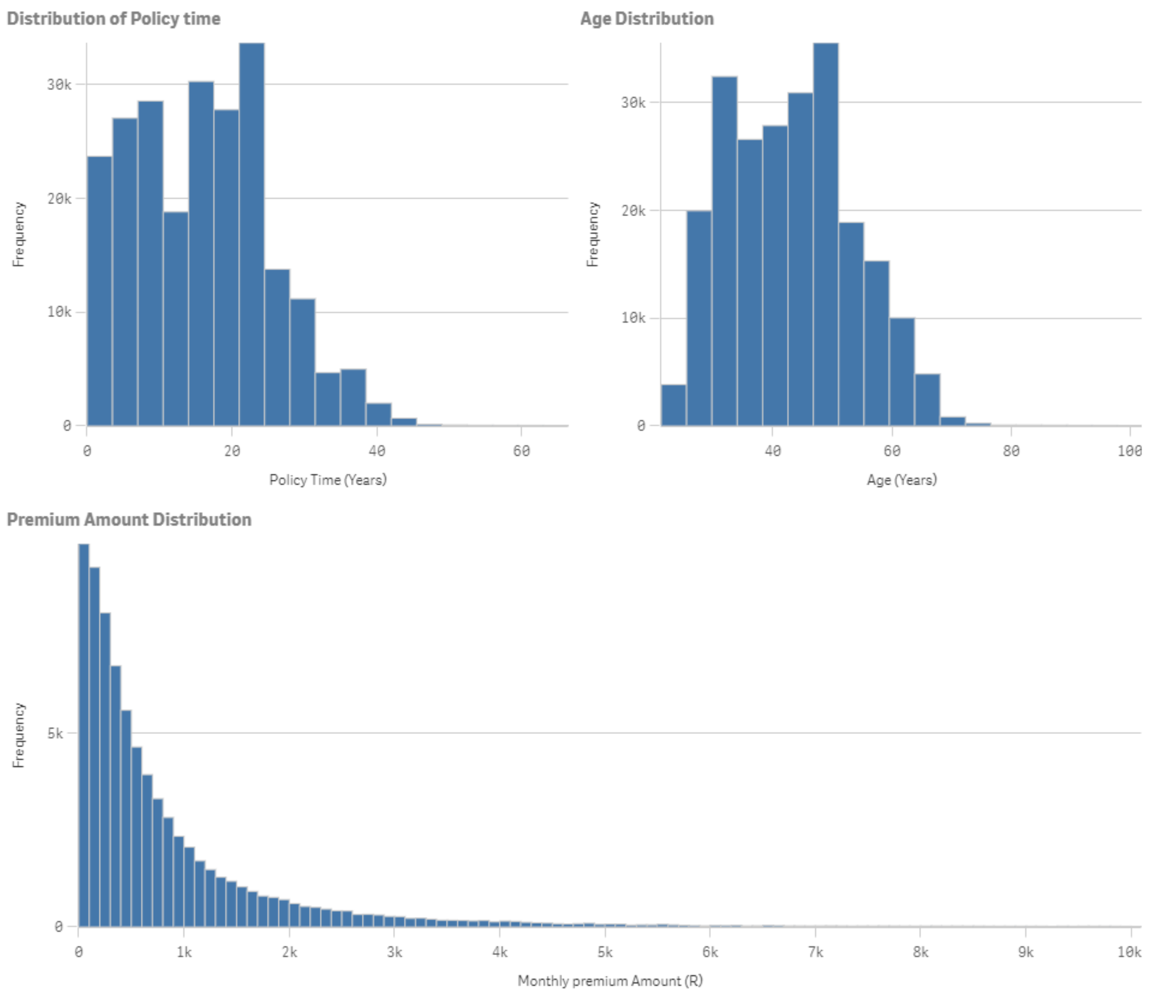


Figure 4.8: Continuous variable distributions

where $x_{max} - x_{min}$ refers to the range of the untransformed data.

4.2.5 Correlation Test

Correlation tests are applied on the data to determine relative importance of features to the target label "product". This provides an indication of predictability of the target column, based on a combination of the input columns. Considering the large amount of categorical columns in the data, we use the Chi Square test to determine whether there is possible association between the outcome variable and the input variables (McHugh 2013). Appendix A provides a detailed implementation of the Chi Square test. As calculated in Appendix A, statistical relevance is proved between the various fields and the target field "product". This indicates that a modelling process could potentially be viable.

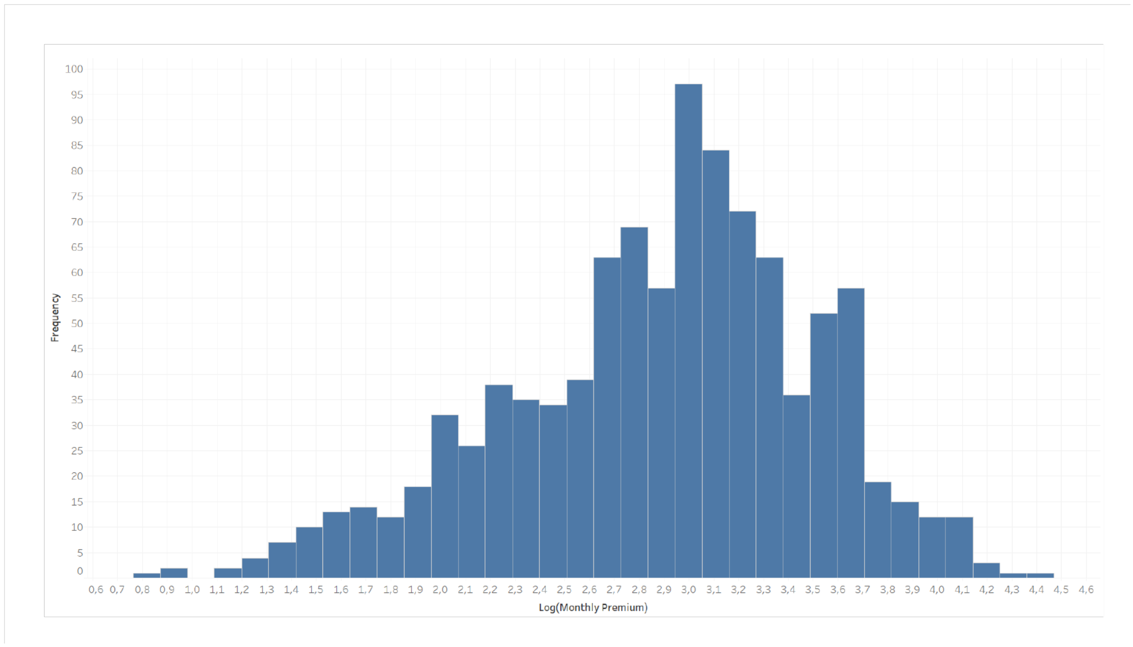
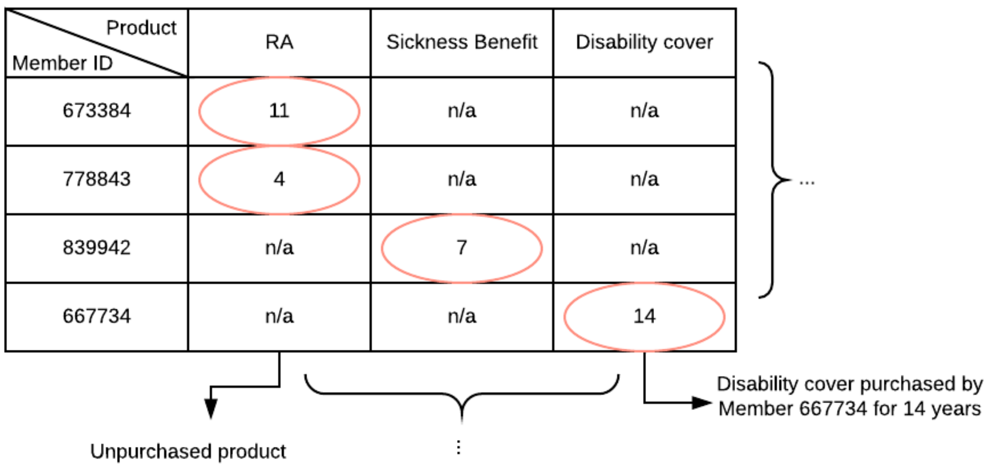


Figure 4.9: Log transform of monthly premiums to address skewed data issues.

4.2.6 User Item Matrix Generation

The final dataset generated can be used for content based recommender models (linear regression and neural networks) but for collaborative filtering based systems, a user item matrix is required. The final step of data prepossessing involves generating a sparse matrix of member numbers and products. The interaction value of the matrix is shown with the policy time (how long the member has had that particular policy). As discussed later, this policy time will be used for implicit feedback purposes. The other qualitative features (gender, age, occupation, etc.) are discarded for the user-item matrix. An example of this matrix is shown in Figure 4.10.



Member ID \ Product	RA	Sickness Benefit	Disability cover
673384	11	n/a	n/a
778843	4	n/a	n/a
839942	n/a	7	n/a
667734	n/a	n/a	14

...
Unpurchased product
⋮
Disability cover purchased by Member 667734 for 14 years

Figure 4.10: Generation of User Item Matrix. Unpurchased items indicated with n/a

5. Model Implementation

The recommendation engine will be built using both a collaborative filtering technique, as well as neural networks. As discussed, the collaborative filtering technique will primarily be used to provide existing users with recommendations, while the neural networks learning system will be used to provide recommendations for new users. This will aim to address the cold start problem discussed previously. Thus, two separate models will be built and tested, and the results of each system will be analysed independently. No joint models (multitask learning) will be implemented at this stage, where the results of each model's training iteration is backpropogated to the other model during the training process. Essentially, the system will use an ensemble model approach, where models are trained independently but combined for classification purposes (Dietterich 2000). Based on new input data presented, the appropriate model will be chosen. The following section provides details and requirements to implement the system specifically on Tensorflow (Abadi et al. 2016), a machine learning library with high level functionality and support for both matrix factorization and neural networks. The general structure of developing a Tensorflow model involves defining the input function to map data into the library, building the machine learning model and then training and performing hyperparameter tuning to optimize the accuracy.

5.1 Collaborative Filtering Model

As discussed previously, the collaborative filtering approach that will be used is matrix factorization, which involves decomposing the sparse user-item matrix into a user-feature and item-feature matrix. The dimensions of these smaller matrices depend on the number of latent or hidden features chosen for the decomposition. Specifically, the matrix factorization will be conducted using Weighted Alternating Least Squares (WALS) methodology discussed in Chapter 3 to obtain the ideal latent features. This approach is proven to be faster since the algorithm can be computed in a distributed, parallel approach (He et al. 2016). Furthermore, this algorithm has techniques for dealing with unobserved interaction pairs, as in the case of a sparse user-item matrix.

5.1.1 Input data into the model

For this component, we are considering how data needs to be supplied to the Tensorflow WALS estimator, which is a high level library used to perform WALS with parallel processing. The following input requirements need to be addressed:

- **Mapping** - creating a user-ID mapping and item-ID mapping for algorithm efficiency to process the variables more easily in Tensorflow. This mapping means user IDs need to start from $(0, 1, 2, ...)$ and items need to start from $(0, 1, 2, ...)$. The first step therefore involves building the member-ID to user-ID mapping, as well as the product to item-ID mapping. The mapping is essential to make business sense of the predictions and recommendations.
- **Implicit Rating** - determining an implicit rating column for the WALS algorithm. This implicit rating column could be an indication of the user preference for a particular product. This is usually best explained through a continuous numerical feature. This provides better indication of a user's preference for a particular product, instead of labelling the interaction as a binary interaction (1 or 0 for purchased/not purchased). In the case of the insurance data, the purchase affinity could be influenced by how long they have kept a certain policy (eg. 20 years with an RA policy). If a user has had a product for a long time, this is a good indication that they are happy with this product, and a similar user might want this product too. Another feature which could provide implicit feedback is premium amount. People who invest more into their particular insurance and investment schemes tend to prefer their service more. However, premiums could also be higher due to risk, hence this might not be as clear an implicit feedback indicator. The policy time is thus used as an implicit indicator, and is scaled for a value between 0-1 (based on longest and shortest policies) to indicate a rating value for the model.
- **Generate Tensorflow records** - this step involves generating the input rows and columns in a format that can be inputted to the Tensorflow WALS package. The full algorithm and details of this parser function is discussed in Appendix B, which describes how the record structures are built in Tensorflow, as well as how each iteration of the algorithm handles batches of rows and columns from the sparse matrix.

5.1.2 Model Parameters for WALS Library

The inputs required for the WALS model library are shown in Table 5.1.

5.1.3 Model Output

Once the optimal latent features are determined, the item and user features are saved in a features dictionary, for remapping into customer and product IDs. The output

Model parameter	**Description**	**Starting Value**
Number of users	Indicates the total number of insurance customers.	75 308
Number of products	Indicates the total number of insurance products available for purchase.	22
Number of latent factors	Indicates the number of hidden features we are predicting (number of dimensions) for the users and items.	5
Training input function	This is the input function created to take in the pre-processed TFrecords for the training dataset.	Custom Parser
Evaluation input function	This is the input function created to take in the pre-processed TFrecords above for the evaluation data set.	Custom Parser
Number of steps	Number of training and evaluation steps	1000
Training size	Number of rows	182 036
Evaluation data size	Number of rows	45 509

Table 5.1: Key input fields for the WALS library

of the model can be in the form of a list of items for users, as well as a list of users for items. In the case of user driven recommendation, this will provide a list of insurance products that a particular user might want to purchase. In the case of item driven recommendations, if a marketing campaign requires who the top users are for a particular product, we can use this method.

We also need to specify the "Top-N" items per user, or users per item for computational efficiency, so we do not store unnecessary predicted ratings. Typically, only a few recommendations are provided to target the user. This will depend on the size of the product dataset. In the case of the insurance products, there are 22 products in total. For this we will use a top 3 measure where only the top 3 recommended products are stored for each user. The full list of recommendations can be provided if required, but 3 were chosen since this represented about 10% of the total product portfolio (22 insurance products). This Top-3 metric is also used as an accuracy measure, where evaluation will determine whether any of the top 3 predicted purchases correspond with actual purchases. For the validation data set, if any of the top 3 products recommended are the ones that the user actually purchased, this will be classified as an accurate prediction.

5.1.4 Hyperparameter Tuning

Hyperparameter tuning is a key element required to optimize the model for higher accuracies. In the context of the collaborative filtering model, the Root Mean Square Error (RMSE) is the key metric used as evaluation for prediction quality. For each item recommended to the user, the error calculates the deviation of the rating for the product from the true value with the following formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(Predicted_i - Actual_i)^2}{n}} \tag{5.1}$$

where RMSE is a numerical error calculated, for each of the n evaluation points (predicted recommendations for all users). In this case the error is calculated only on the top recommendation for each user. The true values in this case is the actual implicit rating that customer would provide a product. Recall that in the case of this dataset, there is no explicit rating, so the policy time was used as an implicit rating. The objective function for hyperparameter tuning is a function of the various model parameters, which needs to be optimized to find the correct combination of hyperparameters.

Bayesian optimization is used as a method of hyperparameter tuning (Swersky, Snoek, and R. P. Adams 2013). Bayesian optimization is powerful in optimizing parameters when the mathematical form of the objective function is difficult to compute (ibid.). Tensorflow's native Bayesian optimization library is applied for hyperparameter tuning. The full details of the algorithm is discussed in Appendix D. After tuning the model, parameters are obtained in their optimal form as shown in Table 5.2.

Model parameter	**Value**
Number of latent factors K	8
L2 Regularization constant	0.236
Weight on unobserved interactions	0.045
Weight on observed interactions	97
Feature Weight Exponent	1
Alternating Least Squares Iterations	40

Table 5.2: Hyperparameter Tuning

Figure 5.1 shows a graphical representation of the hyperparameter tuning process. The left graph shows the accuracy produced for various combinations of hyperparameters. The green line shows the optimal combination of hyperparameters which correspond to Table 5.2, producing the lowest RMSE on the validation data. Due to computation limitations and the size of the dataset, only 10 hyperparameter tuning trials were conducted, each trial consisting of 50 epochs of training. The training process for each trial is shown in the right hand side of Figure 5.1. The trial which generates the highest accuracy after 50 training epochs corresponds with the green line on the left graph of configurations.

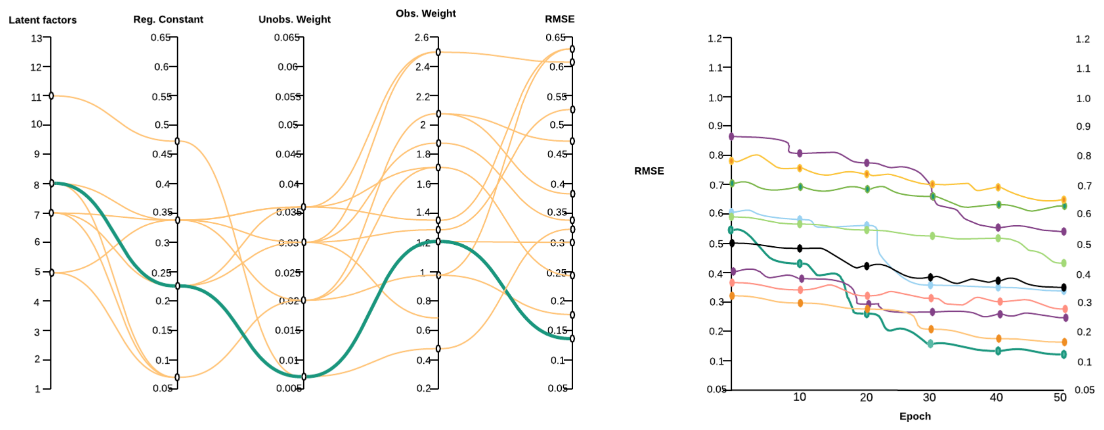


Figure 5.1: Hyperparameter tuning for neural network recommendation engine. Selection of optimal parameters are shown on the left, as well as accuracy for each configuration tested on the right.

5.2 Neural Network Architecture

The above model using WALS matrix factorization is important in making recommendations for existing users with products. However, as discussed, this has a cold start problem where users with no selection history will not be able to receive recommendations. The following model architecture aims to use a neural network approach using various user features to make recommendations.

5.2.1 Content based models

Content based recommendation models involve using features of a user to predict a particular field or target. In the case of the insurance dataset, this involves using age, premium range, gender and smoking habits to predict a product that the user might purchase. This is primarily targeted for new users who have no history of prior purchases, but we know some details about them. The model that will be used is a Tensorflow feed forward neural network using a custom estimator. A custom estimator in Tensorflow is a ground-up neural network design in Tensorflow where all layers, inputs and parameters are customizable (ie. not prebuilt model). Building the architecture of the neural network involves developing the input function to feed data into the model, creating the embeddings for the feature columns, implementing a model function, training and evaluating the dataset.

5.2.2 Build the feature columns for the model

Tensorflow neural networks involve embedding features into columns which can be used by the neural network model. This involves a transformation of numerical and categorical data into embedded features represented as numerical values. The various layers of the neural networks contain weights which undergo multiplication and addition operations. There exists a number of embedding techniques which can help map categorical variables into numerical inputs.

The first embedding will be for the "occupation grouping" column. This column consists of 7 distinct occupation groupings. The embedding in this case will be as a categorical column with vocabulary list. For Tensorflow feature columns, the library cannot handle string features directly, and the data needs to be encoded into integers (Martın Abadi et al. 2015). The vocabulary is the set of categorical labels. The input features for the neural network are more effective as numerical values, so this embedding will create a mapping of the occupation categories to an integer ID ranging from 1-7. This creates a lookup dictionary with a vocabulary (category list) of all possible occupations. Out of vocabulary values are ignored, unless out of vocabulary values are specified. In this case, the embedding assigns these values a new integer ID beyond the expected ranges. This is common in practice, for example when an occupation is passed to the model which has not been dealt with before. For the case of this model, we will only be working with the occupation groupings in the training and test datasets. The same approach is used for the columns gender, and smoking habit. The vocabulary is stored as an in-memory vocabulary, since there is not a significant amount of distinct categorical values for each of the features. In the case of a feature with many variations, this would typically be implemented via a vocabulary file. For the numerical columns policy time, premium amount and age, direct continuous variables are supplied as inputs to the neural network.

The last type of feature which will be used in the model is the concept of crossed column features (interaction features). Often some features make more sense combined rather than having the network train weights for the feature independently. For example, premium amount might be highly dependant on policy time (how long somebody has been a member with the insurance company). Thus, we create a crossed feature (interaction term) between policy time and premium amount and feed this into the neural network.

The various feature embeddings discussed above can be shown with an example. Consider a set of features with raw values as shown in Table 5.3. The various transformations are applied to these raw features to obtain the final feature embeddings as displayed in Table 5.4, which is inputted to the first layer of the neural network.

Variable	**Value**
Occupation	Lawyer
Gender	Male
Habit	Non-smoker
Policy Time	12 years
Premium Amount	R300
Age	65

Table 5.3: Sample feature in raw form

Variable	**Embedding**	**Input value**
Occupation	Categorical Vocabulary	3
Gender	Categorical Vocabulary	1
Habit	Categorical Vocabulary	1
Policy Time	Direct	12
Premium Amount	Direct	300
Age	Direct	65
$\sqrt{Premium} \times PolicyTime$	Crossed	207

Table 5.4: Variable Embedding

5.2.3 Developing the Input Function

The next step involves writing the input function for the neural network, to pass the data into the first layer of the model. This function essentially reads the data from the feature embedding and feeds this into the first layer of the neural network. The input function involves specifying the feature columns as well as the label column. The label column in this case is “product”, which corresponds to the predicted product type for a particular set of input features. Essentially this input function builds an input pipeline to the model which yields batches of features and labels. The batches are obtained according to the rules of the model specified below.

5.2.4 Model Activation and Architecture

The model function involves specifying the details of the neural network architecture that will be used to train the model. The first step involves specifying the number of layers, with the number of neuron units in each layer. Default values are used initially, and will be optimized in the hyperparameter tuning process later. The model function also initializes with the feature embeddings, as well as the number of distinct labels (products) that can be classified. In terms of the activation function, a Rectified Linear Unit (ReLU) is used due to its computationally faster performance (Dahl, Sainath, and G. E. Hinton 2013), as the training does not slow down or stop as the error gradients updates become very small, like in the case of sigmoid activation functions. (ibid.).

The output layer of the neural network uses a function which predicts an “n-class” probability since the output is categorical in nature. This aims to classify an observation into an appropriate class. Specifically the logits from the final layer are transformed into probabilities using the softmax function which outputs a vector which represents the probability distribution for the various intended outcomes. The logit scores are the raw scores from the final layer of the neural network, which are mapped into the probabilities using the following operation.

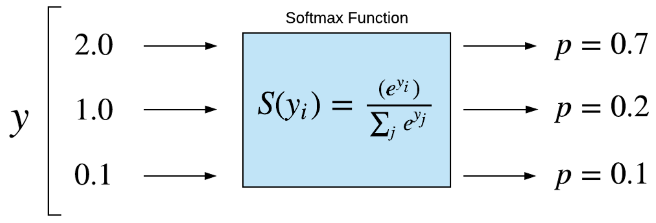


Figure 5.2: Softmax function for converting logits to probabilities

The above figure shows how the final layer outputs y are transformed into probabilities for classification.

Loss functions help in evaluating the performance of the neural network architecture to the particular dataset being trained. When predictions vary significantly from intended values, the losses are larger. The particular loss function in this case is the sparse softmax cross entropy loss function (W. Liu et al. 2016). This expands on the loss function discussed in section 3.2 and is particularly suited for mutually exclusive classification losses, as in the case of a specific product recommender. This function performs particularly well when the output uses an integer representation of the various labels. A summary of the model parameters are shown in Table 5.5.

5.2.5 Model Output

The final model output is similar to the collaborative filtering model. This is in the form of a list of ranked items for users, as well as a list of ranked users for items. In terms of the accuracy measures, classification error is calculated based on the number of incorrectly classified labels. A Top-3 accuracy measure is also implemented, based on whether the top 3 recommended products are within the actual products purchased. Because content based models tend to recommend the same product that a user already has, for user IDs in the evaluation set, all products are removed from the training set, so this does not overfit the data points.

Model parameter	**Description**	**Value**
Number of users	Indicates the total number of insurance customers.	75 308
Number of products	Indicates the total number of insurance products available for purchase.	22
Number of layers	Hidden Layers	3 - initialize
Categorical Embeddings	Categorical Features	Gender, Habit, Occupation
Continous variables	Direct model input	Policy Time, Age, Premium Amount
Activation Function	Neuron activation	ReLu
Number of steps	Number of training and evaluation steps	250
Loss Function	Used to calculate error	Cross Entropy with L2 Regularization
Training size	Number of rows	173 036
Evaluation data size	Number of rows	41 309
Output function	converting logits to probability	Softmax

Table 5.5: Key input fields for neural network

5.2.6 Hyperparameter Tuning

Similarly to collaborative filtering, we employ hyperparameter tuning to optimize the model parameters listed in Table 5.6, to find the best overall accuracy. In this case, we optimize for classification accuracy as well as the Top-3 accuracy. The strategy chosen for hyperparameter tuning is the grid search method, with 10 trials. This refers to the number of times a model is retrained with various combinations of tuning paramaters. To prevent overfitting, L2 regularization is used. When the upper and lower boundary for the various parameters are provided, grid search exhausts the various options and combinations to find the best combination of metrics. For each combination of hyperparameters, a model is trained on the test dataset, and evaluated on a validation dataset. The model with the highest accuracy is chosen as the best set of hyperparameters. Table 5.6 shows the hyperparameters for the Tensorflow neural network after tuning.

Figure 5.3 shows a graphical representation of the hyperparameter tuning process. The left graph shows the accuracy produced for various combinations of hyperparameters. The green line shows the optimal combination of hyperparameters which correspond to Table 5.6, producing the highest classification accuracy on the validation data. Due to computation limitations and the size of the dataset, only 10 hyperparameter tuning trials were conducted, each trial consisting of 75 epochs of training. The training process for each trial is shown in the right hand side of Figure 5.3. The trial which generates the highest accuracy after 75 training epochs corresponds with the green line on the left graph of configurations.

Hyperparameter	**Value**
Hidden layers	[100,80,110]
Dropout	0.0053
Learning rate	0.017
Optimizer	Adam optimization

Table 5.6: Hyperparameter Tuning neural network

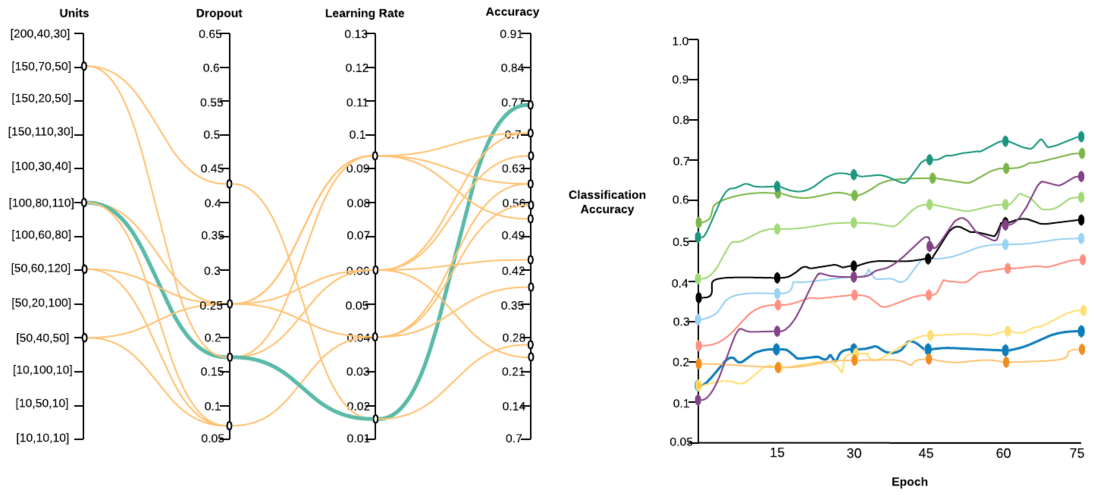


Figure 5.3: Hyperparameter tuning for neural network recommendation engine. Selection of optimal parameters are shown on the left, as well as accuracy for each configuration tested on the right

5.2.7 Other Model Evaluation Metrics

The above discussion explains metrics used to optimize model performance with RMSE and Top-N accuracy. However, there are other metrics which can be used to assess overall recommendation engine quality. The criteria for measuring performance of a recommendation engine varies quite significantly. This is because recommendation engines serve potential items of interest to purchase. While the top item served might actually be the most relevant, the correct item chosen by the user might be the 3rd or 4th item recommended.

In the case of insurance, we want to know how diverse the recommendations are for the test set provided. High diversity indicates that a wider range of products have been suggested, as opposed to one or two very popular products that are always being recommended. For a test data set, we calculate the diversity using the Intra-List-Similarity (ILS) for a user using the following formula (Ziegler et al. 2005):

$$ILS_{User} = \frac{1}{2} \sum_{i_j \in L} \sum_{i_k \in L} S(i_j, i_k) \tag{5.2}$$

where L represents the list of recommendations provided to a particular user, and the similarity function S implemented is the cosine similarity between products j and k shown above, however other similarity functions can be implemented as well (ibid.). The cosine similarity uses the overall similarity of the vectors for product j and k based on the total users who have rated the products historically (see section 3.1 for item-based similarity details). Once the ILS for each user is calculated, this can then be averaged for all users. The ILS is measured on a scale of 0 to 1, from completely dissimilar (diverse) recommendations to highly similar recommendations respectively.

Catalogue coverage is another important indicator which shows what percentage of the full catalogue list was recommended to users. This is determined using the following formula:

$$Coverage_{Cat} = \frac{length(F)}{N \times U} \times 100 \tag{5.3}$$

where F represents the full list of all items recommended, N is the total number of users in the test set, and U is the total number of products.

The final measure of recommendation engine quality is novelty (Castells, Vargas, and J. Wang 2011). Novelty is a measure of uniqueness and specificity of a product to a particular user, instead of just recommending globally popular options (ibid.). The novelty is measured on a product basis, by determining the difference between the probability product i is recommended for a particular user and the probability that product i is recommended for any user. This is determined with the following function:

$$P_i(User) = \frac{n - rank_i(user)}{n-1} \tag{5.4}$$

$$P_i(allUsers) = \frac{n - rank_i(allusers)}{n-1} \tag{5.5}$$

Where $P_i(user)$ indicates the probability for product i being recommended for a particular user, and $P_i(allusers)$ is the probability of product i being recommended for all users. The difference between these 2 probabilities indicate the novelty.

6. Discussion and Analysis

This chapter involves analysing the final trained performance of the recommendation engines developed, both in isolation and as a combined system. The criteria for measuring performance of a recommendation engine varies quite significantly. This is because recommendation engines suggest potential items of interest to purchase. While the top item recommended might actually be the most relevant, the correct item chosen by the user might be the 3rd or 4th item recommended by the algorithm. This chapter aims to discuss some of the results of the models built, as well as future improvements which could be made to the system.

6.1 Model Performance

Following the model training and hyperparameter tuning process for the collaborative filtering and neural network models, the results indicate the following accuracies:

Model	**Accuracy Measure**	**Test**	**Training**
Neural network	Top-3 Classification Error	77.2%	88.9%
Collaborative Filtering	Top-3 Classification Error	83.8%	91.35%
Collaborative Filtering	RMSE	0.13	0.06

Table 6.1: Model Accuracy

Since collaborative filtering produces a predicted implicit rating, we can obtain both an RMSE as well as classification error. RMSE error obtained is based on predicting an implicit feedback rating of 0 to 1. Neural networks produce only a classification output. Both datasets had a 80-20 split in the dataset where 80% of the data is used for training the model, and 20% is used for evaluation. For the neural network, there was an additional filtering component for the training and evaluation dataset. For the chosen members used for evaluation, all products owned by the member were removed from the training set of neural networks. This avoided the issue of overfitting the members to products that they already had. This essentially treated all test data as new members. If any of the top 3 products recommended to the member corresponded to their true choice, then this was classified as a correct prediction.

The following results show the ILS measures for both the neural network and collaborative filtering models.

Model	**Average ILS**
Neural Network	0.624
Collaborative Filtering	0.723

Table 6.2: Diversity measures for recommendation engines

The following results are obtained for catalogue coverage.

Model	**Catalogue Coverage %**
Neural Network	83.12
Collaborative Filtering	94.4

Table 6.3: Catalogue coverage

The average novelty for all users shown in the results below for the 2 models tested.

Model	**Average Novelty**
Neural Network	0.21
Collaborative Filtering	0.27

Table 6.4: Novelty Results

6.2 Discussion of Results

The results above indicate a successful prediction for both the content-based model and collaborative filtering model. Considering that for new customers that join, there is a 77.2% accuracy in being able to predict the product that will be best suited for their needs. This is powerful from a marketing perspective. For existing customers, to predict cross-sell for new products with a 83.8% accuracy is extremely valuable for growing portfolio bases with existing customers. Hence the solution solves the problem of cold-start for new customers as well as cross-sell for existing customers. The RMSE is based on using the member time as implicit feedback to measure the affinity of customers for a particular product (on a rating scale of 0-1).

To compare the results obtained above to a simpler model, a multinomial logistic regression was conducted on the dataset used to predict new products for existing users. The product was modelled as a function of gender, policy time, habit, age, premium and occupation. The classification accuracy obtained from this model was 38.4% on a validation dataset. Details of this implementation are presented in Appendix E. This indicates that the methods explored in this research definitely outperform simpler regression techniques.

As shown in the ILS results, the neural network has a lower ILS and thus greater diversity than collaborative filtering. This means that the approaches complement each other, since new users joining the company will get diverse recommendations from the neural network, but once they become a more well established client with more products, the collaborative filtering will make more niche suggestions. The results for catalogue coverage indicate good coverage for both neural networks and collaborative filtering. However, the high coverage in this case might be due to the relatively small number of insurance products (22), in comparison to most recommender systems on movies or e-commerce systems with thousands of products. In these cases, this value is usually significantly lower (Arora et al. 2014).

As shown in the results, novelty is greater for collaborative filtering than for neural networks. This could be due to neural networks providing recommendations based on features (age, gender, occupation, etc.), hence similar products may be recommended for certain groups of people without much individual specificity. Collaborative filtering uses a rich selection history to make a recommendation instead of general features, and this could explain slightly more novel or unique suggestions in products. Once again, this does suit a complementary approach of using neural networks for new users, and collaborative filtering for existing users.

6.3 Combined System Analysis

As discussed previously, the system developed indicated a good performance on both the content based side of the recommendations with neural networks, as well as the collaborative filtering side. This indicates that the approach of using neural networks for new users, and collaborative filtering for existing users is powerful in providing holistic recommendations for all users. In production, this system could theoretically route a user based on the number of products they have to the correct recommender, and provide a classification. However, this is not a true hybrid system, and the method implemented here is more of an ensemble approach.

A true hybrid system uses the latent features learned from collaborative filtering as additional inputs into a neural network based model (Lee, Choi, and Woo 2002). The latent features represent hidden features of the products and users which are not directly obvious from observation or from explicit labels. The matrix factorization process thus learns these hidden dimensions and features for existing users. These hidden features can then be used as inputs to a neural network, in combination with the explicit features (gender, age, occupation, etc.). This architecture would be relevant for both new users and existing users. New users might not have data for latent features, so only the explicit fields would be used with null entries for latent features. This would be a potential research area that could build on the results obtained here.

Another approach in using hybrid models is incorporating a joint model or multitask learning model. Joint training or multitask learning refers to the ability of simultaneously optimizing an objective function with multiple models, reinforcing the strengths of each model and mitigating the weaknesses. Generally, with joint training the results of each model's training iteration is backpropogated to the other model during the training process (Ruder 2017). Joint training differs from ensemble approaches where models are trained independently, and models are integrated only at the prediction stage for new data (inference stage) (Cheng et al. 2016). The drawback of an ensemble approach is that it requires large models since each model independently requires sufficient training data to produce accurate predictions. Joint training, however, only requires each model to be trained for components which complement the weaknesses of other models (Q. Yang and Y. Zhang 2017).

In the context of recommendation engines building on this research, joint training approaches could be used for a hybrid collaborative filtering and neural architecture. A particular case study which has shown great results with recommendation engines is wide and deep learning. This combines the strengths of generalized linear models and deep neural networks.

6.4 Future Recommendations

Based on the results obtained above, there are a number of approaches which can be implemented to enhance the capabilities of the current recommendation engine. These recommendations will primarily aim to increase the model performance and provide a more production ready hosted version of the model.

The first approach would be to integrate the two models trained independently into a single hybrid model. As discussed in section 6.3 above, the WALS method which obtains the latent factors can be combined with a neural network approach to build an end to end hybrid system. Furthermore, in order to gain better hyperparameter values, a future suggestion would be to perform more exhaustive hyperparameter tuning strategies. For the system built, a bayesian optimization was conducted to tune the parameters for WALS, and a grid search technique was used for the neural network. Various approaches such as grid search, random search and bayesian optimization should all be tested with more trials. Furthermore, a joint training model could be incorporated for multitask learning to optimize both the neural network and WALS method simultaneously, instead of feeding the data separately in an ensemble approach. This would make the model much more efficient to deploy at scale in a production system, and when there is much more input data.

The next recommendation would be to host the model in a more production ready environment to test the model at scale. Although the statistical analysis of the data provided a good set of results, actual performance may vary on real-world data. For the system built on Tensorflow, the model would need to be packaged into an Application Programming Interface (API), which can provide recommendations to front end environments such as web or mobile sites. This tests the capabilities of the model with both real data and in a scalable environment with many users. Furthermore, for online training to dynamically update the results, we would need to incorporate a feedback mechanism to retrain the data.

7. Conclusion

The aim of this research was to design a recommendation engine for insurance customers which could suggest appropriate products for new or existing users based on their age, gender, occupation, smoking habits, premium amount, and policy time. Two major approaches of building recommendation engines involve collaborative filtering techniques and neural networks. Collaborative filtering techniques work well for long standing customers with a rich purchase history. Neural networks work well by using customer features (age, gender, etc.) to find suitable products.

A combined recommendation engine was therefore designed, implemented and tested on an insurance dataset. The two models built involved a collaborative filtering approach using Weighted Alternating Least Squares (WALS), and a Content based Model using neural networks. The Collaborative filtering approach achieved a 83.8% accuracy and the neural network achieved a 77.2% accuracy on classification. Test results are indicated that both models are essential for a successful recommendation engine, since the neural network helps with solving a cold start problem for new users with no data, and the matrix factorization approach is useful for existing users. Given enhancements to the system, like combining the two models into single system, and making the system more production ready, the designed system could certainly be implemented into a production insurance environment to enhance targeted marketing, cross-sell and up-sell approaches in the insurance industry.

Appendices

A. Further Data Exploration

This section involves showing further data exploration, with a number of different variables compared to test for statistical relevance. The visualizations are produced as support for the model building process to identify correlation and relationships in the data.

A.1 Correlation Tests

Correlation tests are applied on the data to determine relative importance of features to the target label "product". Considering the large amount of categorical columns in the data, we use the Chi Square test to determine whether there is possible association between the outcome variable and the input variables (McHugh 2013). Using the Chi Square test, we are trying to determine whether a relationship exists between two categorical variables. This involves defining a null hypothesis, and then testing this hypothesis. For the purpose of each categorical feature, this hypothesis is as follows:

H_0: There is no association between the target feature and the categorical feature being compared.

H_a: There is an association between the target feature and the categorical feature being compared.

To test this hypothesis, we need to compute a test statistic using the following Chi-squared test (ibid.):

$$\tilde{\chi}^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k} \tag{A.1}$$

where $\tilde{\chi}^2$ is the test statistic, O_k refers to instances observed for each categorical feature of the insurance data set, and E_k is the expected outcome assuming the null hypothesis is true (no association) across all cases n. For example, in the case of gender, for the null hypothesis it is expected that there will be an equal distribution of males and females across the product catalogue. Once $\tilde{\chi}^2$ is determined, this test statistic follows a particular Chi-square distribution, with degrees of freedom dependant on the variables being tested, and calculated as follows:

$$DOF = (Var - 1)(Products - 1) \tag{A.2}$$

where Var is the possible types of the variable being tested, and $Products$ is the possible types of the target variable Product. For example, when testing association between gender (2 types) and product (22 types), the total degrees of freedom is 21. We then use the standard Chi-square distribution with 21 degrees of freedom to find a critical value for $\tilde{\chi}^2$ at a significance level of 0.05. If the value determined from equation A.1 is greater than the critical value, this means that the probability ($p - value$) of experiencing conditions under which the null hypothesis holds true is greater than 0.05, and we thus accept the null hypothesis. Using this approach approach, we can calculate the $p - values$ for each of the categorical features against the target product feature, and the results are indicated in Table A.1. For continuous variables, we group the data into specific categorical ranges. For example, the variable age could be grouped into young, middle-aged and old. After grouping the continuous variables, we apply the Chi Square test in the same manner. As shown in Table A.1, for all variables the null hypothesis is rejected, indicating statistical relevance between the various fields and the target field "product".

Feature	$p - value$
Habit	<< 0.01
Gender	<< 0.01
Occupation	<< 0.01
Age	0.0023
Policy Time	0.018
Premium	<< 0.01

Table A.1: Results of significance tests after Chi Square is applied

A.2 Visualizations

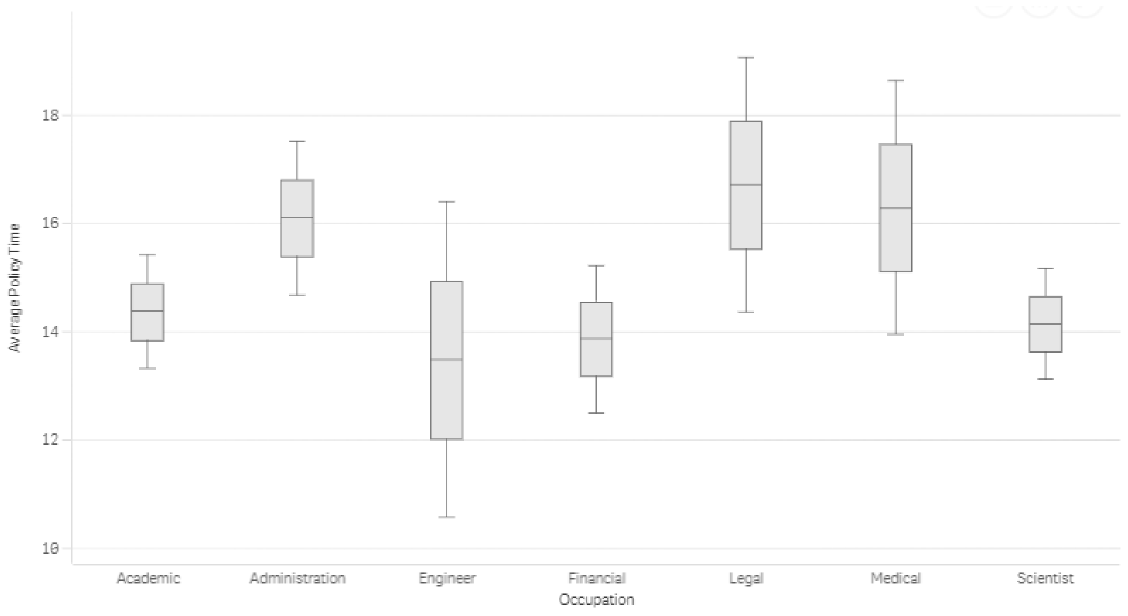


Figure A.1: Box Plot Showing the distribution of policy time for various Occupation Groupings

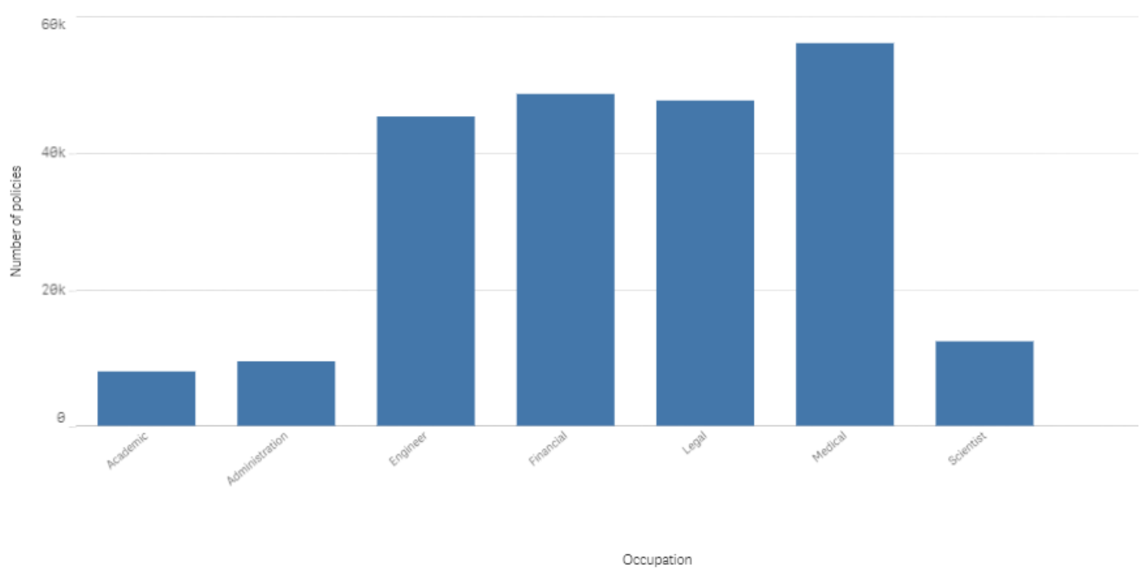


Figure A.2: Bar graph Showing the number of products per Occupation Group

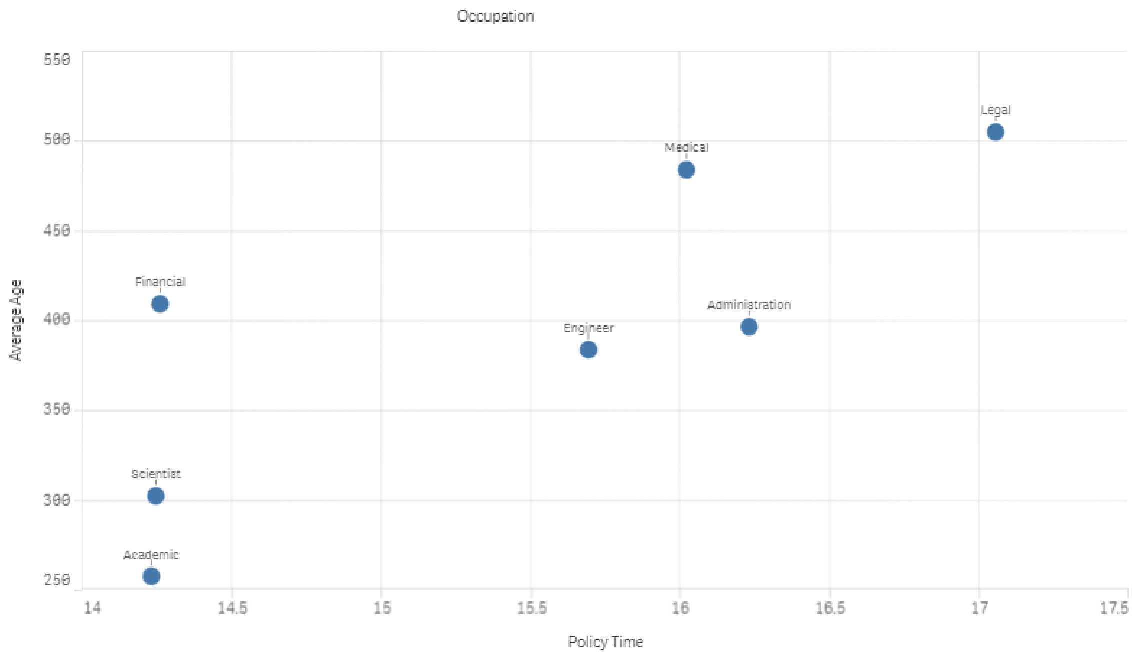


Figure A.3: Scatter plot showing the variation of clustering of occupation groups according to premium amount and age

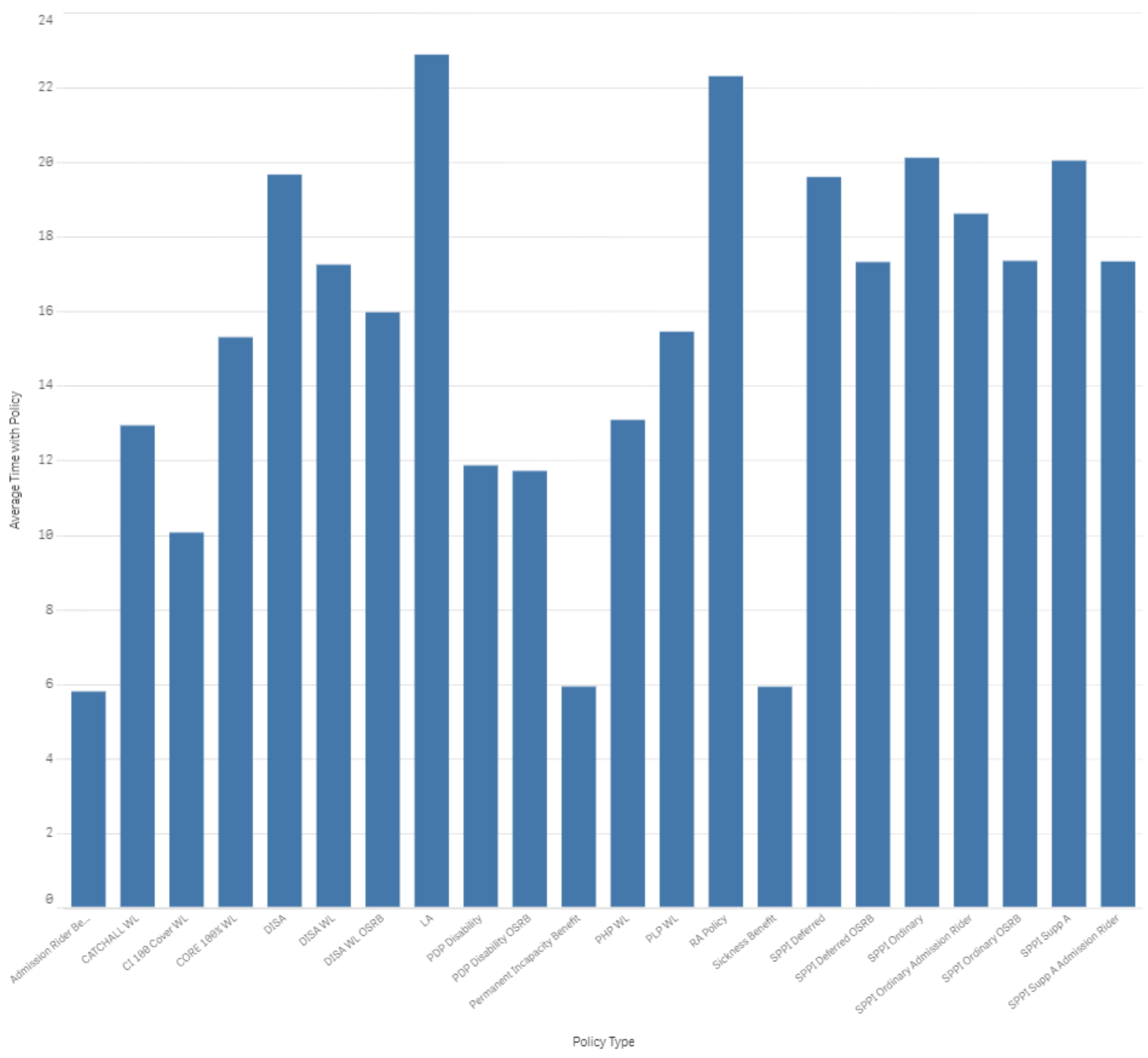


Figure A.4: Bar graph showing the average member time per product

B. Tensorflow Algorithms

The following sections provide an overview of key logic algorithms used in the model building process.

B.1 Generate input rows and Columns for WALS algorithm

This involves generating a Tensorflow Record (TFRecord) for all the rows (users) and another record for all the columns (products/items). Tensorflow records are specific storage objects which are optimized for efficient processing by the Tensorflow library. When feeding the list of rows into the TFRecord storage object, we specify a "key", "index" and "value" field for each row/column. In the case of dealing with rows (insurance users), the key in this case will be the user for the row, the indices would be the various item IDs, and the values would be the implicit feedback that the user provides each item (the policy time in this case). An example of this record structure is shown in Figure B.1. Similarly, for the columns TFRecord (products), the key would be the item, the indices would be the user IDs for that particular column, and the values would be the various implicit feedback ratings. These TFrecords can now be parsed and batched rows or columns can be sent to the WALS estimator for training.

For the input functions, a parser is developed for the TFrecords that store the user and item data. The objective of parsing is to store the item and products into a data storage object that efficiently handles sparse data points for computation efficiency. Typical arrays or matrices are not optimized for sparse data points. Within the tensorflow library, Sparse Tensors are storage objects which handle these datasets, where it stores the non-zero values and the corresponding coordinates from the original TFrecord.

The input function also specifies an epoch which repeats the parsing process for a group of users or items (rows or columns) a number of times to determine the optimal latent features during the training process. Batching is a technique used to group a set

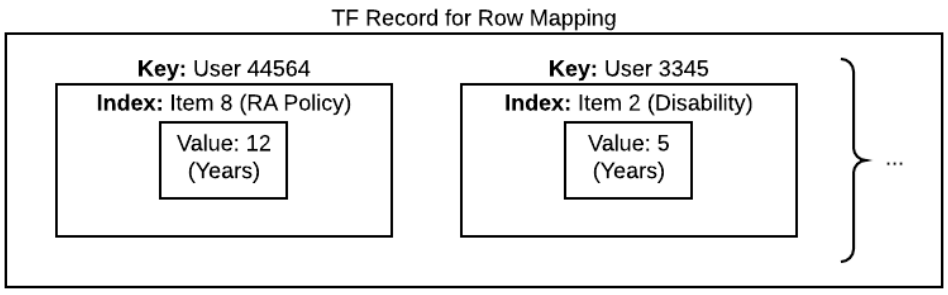

TF Record for Row Mapping
Key: User 44564
Index: Item 8 (RA Policy)
Value: 12 (Years)
Key: User 3345
Index: Item 2 (Disability)
Value: 5 (Years)
...

Figure B.1: TFRecord structure for mapping the rows of the user-item matrix into the appropriate storage structures in Tensorflow

of user and items that are processed at once through the WALS matrix factorization process. Figure B.2 shows the overall process.

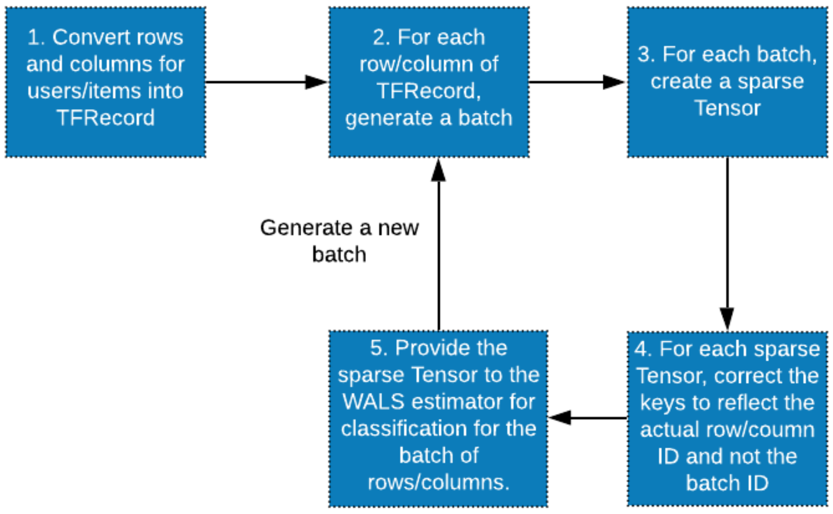

1. Convert rows and columns for users/items into TFRecord
2. For each row/column of TFRecord, generate a batch
3. For each batch, create a sparse Tensor
4. For each sparse Tensor, correct the keys to reflect the actual row/coumn ID and not the batch ID
5. Provide the sparse Tensor to the WALS estimator for classification for the batch of rows/columns.
Generate a new batch

Figure B.2: Process for generating sparse tensors to provide to WALS

As mentioned before, for a batch obtained from the user or product set, we need to generate a sparse tensor before feeding this into the WALS library for processing. Batching is important, since we want to work with alternating subsets of rows and columns as per the WALS algorithm (alternating). When we generate a batch of rows or columns in a sparse tensor, the row or column IDs of each record get overwritten by the batch ID. A remapping process then needs to occur, where we replace the batch ID for the particular value with the user ID (for rows) or item ID (for columns).

B.2 Key remapping algorithm

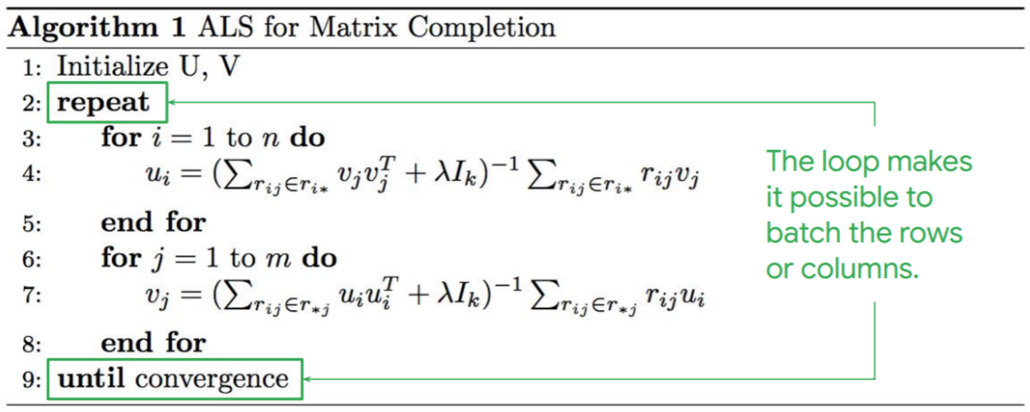
Specifically this refers to the key remapping algorithm used for Weighted Alternating Least Squares (WALS) in matrix factorization, and the alternating process to solve for the row and column latent features discussed in Chapter 5. In the WALS algorithm, for a batch obtained from the user or product set, we need to generate a sparse tensor before feeding this into the WALS library for processing. Batching is important, since we want to work with alternating subsets of rows and columns as per the WALS algorithm (alternating). When we generate a batch of rows or columns in a sparse tensor, the row or column IDs of each record get overwritten by the batch ID. A remapping process then needs to occur, where we replace the batch ID for the particular value with the user ID (for rows) or item ID (for columns). The specific algorithm for key remapping is explained with the following pseudocode.

Algorithm 1 WALS Key remapping

```
1: procedure PARSE TFRECORD(filename)              ▷ Input function
2:     Open File                                   ▷ Either row or column file
3:     Create a sparse tensor                      ▷ store in Better structure
4:     Save Key                                    ▷ remapping IDs
5:     Concatenate key with sparse Tensor          ▷ pass the keys back
6:     Generate a batch of rows or columns
7:     Remap the keys from batch overwriting
8:     return remapped batch                       ▷ Return and iterate
```

B.3 Weighted Alternating Least Squares Algorithm

For the algorithm used to solve iteratively for the user and item features, the following pseudocode indicates the process:



Algorithm 1 ALS for Matrix Completion

```
1: Initialize U, V
2: repeat
3:     for $i = 1$ to $n$ do
4:         $u_i = (\sum_{r_{ij} \in r_{i*}} v_j v_j^T + \lambda I_k)^{-1} \sum_{r_{ij} \in r_{i*}} r_{ij} v_j$
5:     end for
6:     for $j = 1$ to $m$ do
7:         $v_j = (\sum_{r_{ij} \in r_{*j}} u_i u_i^T + \lambda I_k)^{-1} \sum_{r_{ij} \in r_{*j}} r_{ij} u_i$
8:     end for
9: until convergence
```

The loop makes it possible to batch the rows or columns.

Figure B.3: Weighted Alternating Least Squares algorithm

This is an iterative process which obtains an optimal **U** and **V** matrix, to produce the best latent features for each customer and each product, such that when they are multiplied together the ordinary least squares error is minimized in the predicted matrix.

C. Insurance Products

The insurance product recommender references a number of insurance products which use codes and abbreviations which are not intuitive to interpret. This section provides a lookup table of all insurance products and codes and names used in the database.

Product Database name (code)	Full Name
SPPI	Sickness and Permanent Incapacity Benefit
DISA	Disability Cover
LA	Life Assurance
OSRB	Occupation Specific Rider Benefit
WL	Wellness Lifestyle
PLP	Accidental Death
Catchall	Comprehensive Medical Cover
PHP	Professional Health Provider
Core 100%	General Health Cover
PDP	Professional Disability Provider
Rider Benefit	Dependants on main member
RA	Retirement Annuity
Sickness Benefit	Income protection for sickness

Table C.1: Full list of insurance products and definitions

*Note: The full list of products includes variations of the above codes. For example, a product may be PDP OSRB which indicates that it is for a Professional Disability Provider (PDP), but with an additional dependant (rider benefit) on the insurance (spouse, child, etc.).

D. Hyperparameter Tuning

Hyperparameter tuning is a key element required to optimize the model for higher accuracies. This section discusses Bayesian optimization as a method of hyperparameter tuning. For implementation, the native Tensorflow library is used to perform Bayesian optimization (Swersky, Snoek, and R. P. Adams 2013). Bayesian optimization is powerful in optimizing parameters when the mathematical form of the objective function is difficult to compute (ibid.). In the case of a single hyperparameter, consider Figure D.1 below (Ravikumar 2018). This shows the actual output points for various values of the hyperparameter. The true objective function is shown, and the goal of hyperparameter tuning is to approximate this function as closely as possible. We can approximate the values between the known points with a Gaussian process, as shown in Figure D.2. As illustrated, the most likely function can be estimated as the curve with the mean of the Guassian process (ibid.).

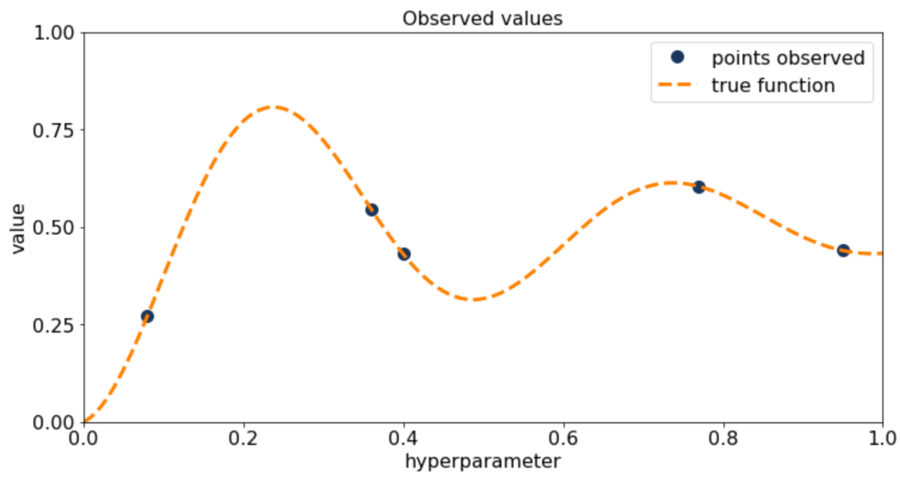


Figure D.1: True objective function mapped to observed points

To further refine the most likely function, regions of known maxima/minima as well as high Gaussian variance (uncertainty) are sampled. The points are sampled in these regions with an acquisition function (ibid.). A variety of different acquisition functions can be used, but the details are beyond the scope of this research. The

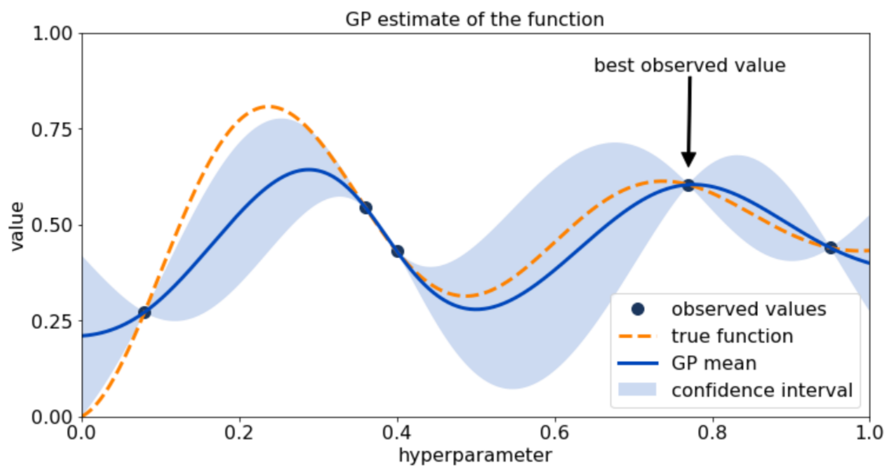


Figure D.2: Possible curves shown as a Gaussian process between known points

primary goal of the function is to find points which yield expected improvement in these regions, as shown in Figure D.3. The samples which yield the maximum value in the acquisition function are selected as new points on the most likely function.

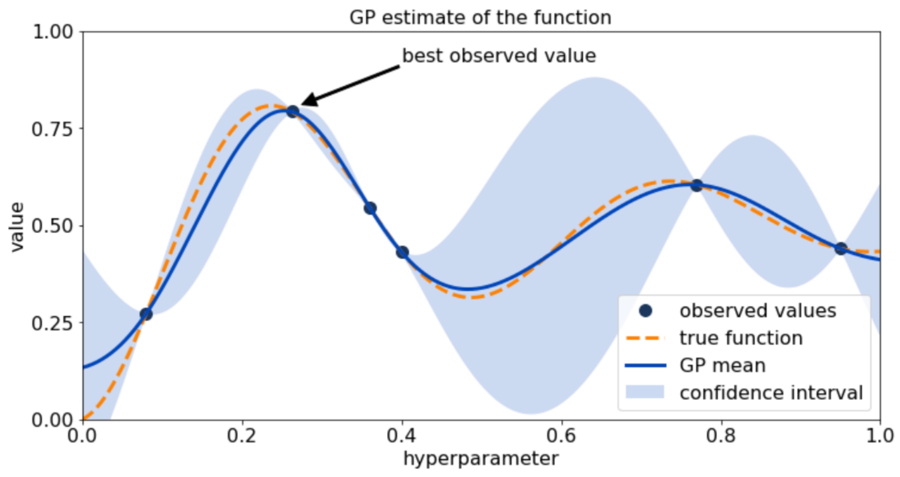


Figure D.3: Refined predicted function after sampling with acquisition function in regions of high uncertainty and known maxima/minima. The values which yield expected improvement are selected as likely points on the predicted curve

E. Linear Model Implementation

The recommendation engines designed are based on neural networks and collaborative filtering techniques. This section involves the implementation of a simpler model to compare. Specifically a multinomial logistic regression was implemented using the *R nnet* package. The source code for the model is shown in Appendix F. Multinomial logistic regression is used to test the significance of continuous variables on multi-class categorical variable outcomes. In a similar way to traditional logistic regression for binary outcomes, multinomial logistic regression predicts the probability that the outcome is equal to one of the possible categorical values using the formula below:

$$P(1), P(2), ..., P(K-1) = \frac{e^{\beta_{K-1}} \times x_i}{1 = \sum_{j=1}^{K-1} e^{\beta_j} \times x_i} \tag{E.1}$$

Where P(1), P(2), ..., P(K-1) represent the probabilities of the various outcomes. β represents the coefficients for linear combinations of the independent variables, and x_i represents the input data associated with outcome Y_i.

The insurance data is imported, and split into training (80%) and test (20%) datasets. A multinomial regression is performed by modelling the "product" column with all other columns as inputs. The validation obtained a classification accuracy of 38.4%. A deeper analysis of variance and other statistical features were not extracted, since a basic classification accuracy was required to compare with collaborative filtering and neural networks.

F. Source Code

All source code for the research is stored in a remote repository with the following link:

https://github.com/prinpillay/masters/

This repository contains the code and notebooks required for model development. The following notebooks and modules can be found:

1. GLM_insurance.ipynb - this contains a simple multinomial logistic regression on the insurance dataset.
2. Clustering.ipynb - data exploration with unsupervised clustering techniques
3. nn_preproc.ipynb - this is the preprocessing of data required for neural network model development
4. nn_model.ipynb - the neural network recommendation engine.
5. nn_htuning.ipynb - hyperparameter tuning for the neural network recommendation engine
6. nn_htuning.ipynb - hyperparameter tuning for the neural network recommendation engine
7. CF_wals.ipynb - collaborative filtering recommendation engine including preprocessing, model development and model tuning
8. wals_packaged - packaged python module version for CF_wals.ipynb.
9. wals_htune - packaged files required for hyperparameter tuning of WALS recommendation engine

Bibliography

Abadi, Martın et al. (2016). “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Alan, Shi Yue; Larson Martha; Hanjalic (2014). “Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges”. In: *ACM Computing Surveys (CSUR)* 47, pp. 1–45. DOI: `http://doi:10.1145/2556270`.

Aranganayagi, S and K Thangavel (2007). “Clustering categorical data using silhouette coefficient as a relocating measure”. In: *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*. Vol. 2. IEEE, pp. 13–17.

Arora, Gaurav et al. (2014). “Movie recommendation system based on users’ similarity”. In: *International Journal of Computer Science and Mobile Computing* 3.4, pp. 765–770.

Arpit, Devansh et al. (2017). “A closer look at memorization in deep networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 233–242.

Bherde, Prof Gajanan (2015). “Credit Card Fraud Detection”. In: *International Journal on Recent and Innovation Trends in Computing and Communication.* IJRITCC, pp. 2069–2071.

Bracha, Adomavicius Gediminas; Tuzhilin Alexander; Ricci Francesco; Rokach Lior; Shapira (2015). *Recommender Systems Handbook.* Springer.

Buda Andrzej; Jarynowski, Andrzej (2010). *Life time of correlations and its applications.* Wydawnictwo Niezależne.

Castells, Pablo, Saúl Vargas, and Jun Wang (2011). “Novelty and diversity metrics for recommender systems: choice, discovery and relevance”. In:

Charu, Aggarwal (2016). *Recommender Systems: The Textbook.* Springer.

Cheng, Heng-Tze et al. (2016). “Wide Deep Learning for Recommender Systems”. In: *arXiv:1606.07792*. URL: `http://arxiv.org/abs/1606.07792`.

Chris, Koren Yehuda; Bell Robert; Volinsky (2009). “Matrix Factorization Techniques for Recommender Systems”. In: *Computer* 42.8, pp. 30–37. DOI: `http://doi:10.1109/MC.2009.263`.

Covington, Paul, Jay Adams, and Emre Sargin (2016). "Deep neural networks for youtube recommendations". In: *Proceedings of the 10th ACM conference on recommender systems.* ACM, pp. 191–198.

Dahl, George E, Tara N Sainath, and Geoffrey E Hinton (2013). "Improving deep neural networks for LVCSR using rectified linear units and dropout". In: *2013 IEEE international conference on acoustics, speech and signal processing.* IEEE, pp. 8609–8613.

Dietterich, Thomas G (2000). "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems.* Springer, pp. 1–15.

Furnas, Will Hill; Mark Rosenstein; George (1995). "Recommending and evaluating choices in a virtual community of use". In: *Proceedings of the SIGCHI conference on Human factors in computing systems.* DOI: `http://pages.cpsc.ucalgary.ca/~saul/601.13/readings/Recommending/wch_bdy.htm`.

Gemulla, Rainer et al. (2011). "Large-scale matrix factorization with distributed stochastic gradient descent". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, pp. 69–77.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing.* IEEE, pp. 6645–6649.

Harris, David (2011). *Digital design and computer architecture (2nd ed.)* Morgan Kaufmann Publishers.

Haykin, Simon (1994). *Neural networks: a comprehensive foundation.* Prentice Hall PTR.

He, Xiangnan et al. (2016). "Fast matrix factorization for online recommendation with implicit feedback". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval.* ACM, pp. 549–558.

Jhalani, Tanisha, Vibhor Kant, and Pragya Dwivedi (2016). "A linear regression approach to multi-criteria recommender system". In: *International Conference on Data Mining and Big Data.* Springer, pp. 235–243.

Koren, Yehuda (2008). "The BellKor 2008 Solution to the Netflix Prize". In: *Netflix Prize Forum.*

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 8, pp. 30–37.

Krueger, David et al. (2017). "Deep Nets Don't Learn via Memorization". In:

Lee, Meehee, Pyungseok Choi, and Yongtae Woo (2002). "A hybrid recommender system combining collaborative filtering with neural network". In: *International conference on adaptive hypermedia and adaptive web-based systems.* Springer, pp. 531–534.

Liu, Weiyang et al. (2016). "Large-margin softmax loss for convolutional neural networks." In: *ICML.* Vol. 2. 3, p. 7.

Low, Jerry (2017). *Super Sales Strategies: Quick Tips for Upselling And Cross-Selling.* URL: `https://www.business.com/articles/quick-tips-for-upselling-and-cross-selling/`.

Lumb, Richard (2016). *Fintech Report.* URL: https://www.accenture.com/t20170411T170619Z__w__/id-en/_acnmedia/PDF-15/Accenture-Fintech-Report-London-Lab-News-Release.pdf.

Martın Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. URL: https://www.tensorflow.org/.

McHugh, Mary L (2013). "The chi-square test of independence". In: *Biochemia medica: Biochemia medica* 23.2, pp. 143–149.

Nguyen, Tien T et al. (2014). "Exploring the filter bubble: the effect of using recommender systems on content diversity". In: *Proceedings of the 23rd international conference on World wide web.* ACM, pp. 677–686.

Nielsen, Michael A. (2015). *Neural Networks and Deep Learning.* Determindation Press.

O'Shaughnessy, Baker J.; Deng Li; Glass Jim; Khudanpur S.; Lee C.-H.; Morgan N.;D. (2009). "Research Developments and Directions in Speech Recognition and Understanding, Part 1". In: *IEEE Signal Processing Magazine* 26, p. 3. DOI: http://doi:10.1109/msp.2009.932166..

Ravikumar, M (2018). *Let's Talk Bayesian Optimization.* URL: https://mlconf.com/blog/lets-talk-bayesian-optimization/.

Resnick, Paul et al. (1994). "GroupLens: an open architecture for collaborative filtering of netnews". In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work.* ACM, pp. 175–186.

Rosa, M. Montaner; B. Lopez; J. L. de la (2003). "A Taxonomy of Recommender Agents on the Internet". In: *Artificial Intelligence Review* 19.4, pp. 285–330. DOI: http://10.1023/A:1022850703159.

Ruder, Sebastian (2017). "An Overview of Multi-Task Learning in Deep Neural Networks". In: *CoRR* abs/1706.05098. arXiv: 1706.05098. URL: http://arxiv.org/abs/1706.05098.

Sarwar, Badrul Munir et al. (2001). "Item-based collaborative filtering recommendation algorithms." In: *Www* 1, pp. 285–295.

Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 65, pp. 85–117.

Spiliopoulos, Vassilis, George A Vouros, and Vangelis Karkaletsis (2007). "Mapping ontologies elements using features in a latent space". In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07).* IEEE, pp. 457–460.

Sweeney, Mack et al. (2016). "Next-term student performance prediction: A recommender systems approach". In: *arXiv preprint arXiv:1604.01840.*

Swersky, Kevin, Jasper Snoek, and Ryan P Adams (2013). "Multi-task bayesian optimization". In: *Advances in neural information processing systems*, pp. 2004–2012.

Takács, Gábor et al. (2009). "Scalable collaborative filtering approaches for large recommender systems". In: *Journal of machine learning research* 10.Mar, pp. 623–656.

Thai-Nghe, N. (2010). "Recommender system for predicting student performance". In: *Proceedings of the 1st Workshop on Recommender Systems for Technology Enhanced Learning.*

Wong, AKC and Wang Yang (1997). “High-order pattern discovery from discrete-valued data”. In: *IEEE Transactions on Knowledge and Data Engineering* 9.6, pp. 877–893. DOI: http://10.1109/69.649314.

Xing, Cao Jian; Hu Hengkui; Luo Tianyan; Wang Jia; Huang May; Wang Karl; Wu Zhonghai; Zhang (2015). “Distributed Design and Implementation of SVD++ Algorithm for E-commerce Personalized Recommender System”. In: *Communications in Computer and Information Science.* 572, pp. 30–44. DOI: doi:http://10.1007/978-981-10-0421-6_4.

Xu, Rui and Don Wunsch (2008). *Clustering.* Vol. 10. John Wiley & Sons.

Yang, Qiang and Yu Zhang (2017). “An overview of multi-task learning”. In: *National Science Review* 5.1, pp. 30–43. ISSN: 2095-5138. DOI: 10.1093/nsr/nwx105. eprint: http://oup.prod.sis.lan/nsr/article-pdf/5/1/30/24164435/nwx105.pdf. URL: https://doi.org/10.1093/nsr/nwx105.

Zhang, Lei, Shuai Wang, and Bing Liu (2018). “Deep learning for sentiment analysis: A survey”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4, e1253.

Ziegler, Cai-Nicolas et al. (2005). “Improving recommendation lists through topic diversification”. In: *Proceedings of the 14th international conference on World Wide Web.* ACM, pp. 22–32.

Zweig, Heck L.; Tur G.; Yu D.; G. (2015). “Using recurrent neural networks for slot filling in spoken language understanding”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 23, pp. 530–539. DOI: http://doi:10.1109/taslp.2014.2383614.