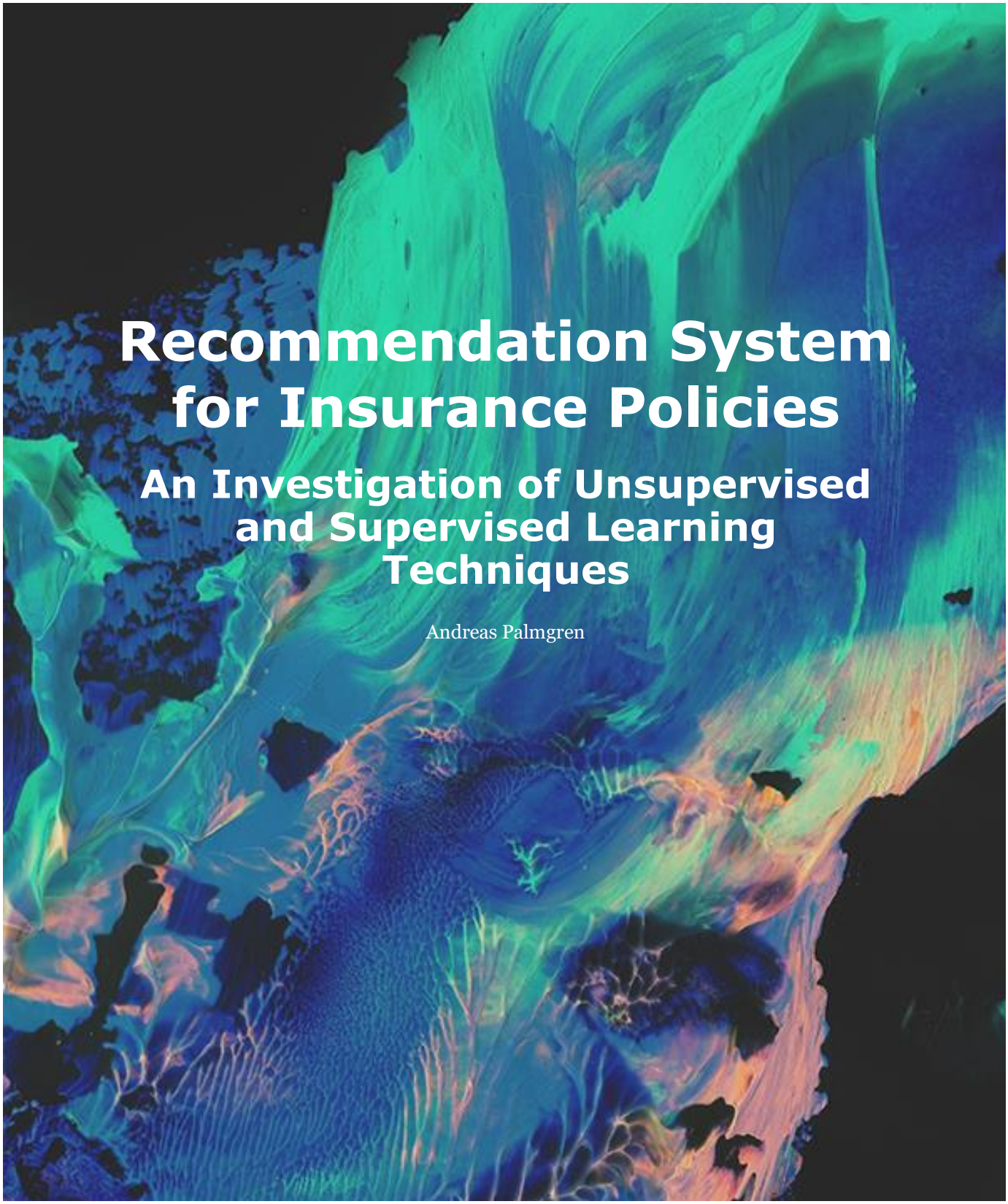
UMEÅ UNIVERSITY



Recommendation System for Insurance Policies

An Investigation of Unsupervised and Supervised Learning Techniques

Andreas Palmgren

Master thesis, 30 ECTS

M. Sc. In Industrial Engineering and Management, 300 ECTS

Spring term 2023

Recommendation System for Insurance Policies
An Investigation of Unsupervised and Supervised Learning Techniques

Andreas Palmgren, anpa0199@student.umu.se

Supervisors:	Frank Kody	Insurely
	Armin Eftekhari	Umeå University
Examiner:	Alp Yurtsever	Umeå University

Degree Project in Industrial Engineering and Management, 30 ECTS
Department of Mathematics and Mathematical Statistics
Umeå University
SE-901 87 Umeå, Sweden

Abstract

Recommendation systems have significantly influenced user experiences across various industries, yet their application in the insurance sector remains relatively unexplored. This thesis focuses on developing a car insurance recommendation system that implements a 'consumers like you' feature.

The study initially employs a clustering-based recommendation system due to missing labels in an offline environment. However, challenges emerge, such as determining the optimal number of clusters and managing complex data. Additionally, the inability to effectively update based on feedback and lower predictive performance compared to supervised methods necessitated exploring supervised alternatives.

In response, this thesis proposes a methodology where the unsupervised approach simulates consumer behavior in an offline environment. Supervised alternatives are pre-trained on the clustering-based system to replicate it and come with the ability to be fine-tuned based on live traffic. Three supervised alternatives — KNN, XGBoost, and a neural network — are developed and compared.

Given the supervised recommendation system adaptability based on feedback, supervised methods can provide more accurate, personalized recommendations in the insurance domain. The XGBoost and neural network-based recommendation systems were able to replicate the unsupervised approach, and their expressive power makes them valid candidate models to further evaluate on live traffic. The thesis concludes with the potential to both improve and adapt these recommendation systems to other insurance types, marking a significant step toward more personalized, user-friendly insurance services.

Keywords: *Recommendation System, Car Insurance, Machine Learning, Cluster Analysis, KNN, XGBoost, Neural Network*

Sammanfattning

Rekommendationssystem har fått ett stort inflytande på användarupplevelser inom många branscher, men dess tillämpning inom försäkringssektorn är fortfarande relativt outforskad. Denna avhandling fokuserar på att utveckla ett rekommendationssystem för bilförsäkring som implementerar en 'kunder som liknar dig'-funktion.

Studien använder initialt ett rekommendationssystem baserat på klustring på grund utav omärkt data i en offline-miljö. Utmaningar uppstår dock, såsom att bestämma det optimala antalet kluster och hantera den komplexa datan. Det ovägledda alternativet innebar även komplikationer för att effektivt uppdatera baserat på återkoppling och potentiellt en lägre prediktionsförmåga i jämförelse med som kan åstakommas med övervakad inlärning.

Som svar på detta föreslår denna avhandling en metodik där den oövervakade metoden används för att simulera kundbeteende i en offline-miljö. Övervakade rekommendationssystem återskapar den klusterbaserade lösningen och kan sedan fortsätta tränas baserat på konsumenters återkoppling för att öka den prediktiva förmågan. Tre övervakade alternativ - KNN, XGBoost och ett Neuronnät - utvecklades och jämfördes.

Med tanke på det övervakade alternativens anpassningsförmåga baserat på återkoppling och eventuell ökning i prestanda kan övervakade metoder ge mer exakta, personliga rekommendationer inom försäkringsområdet. Både rekommendationssystemet baserat på XGBoost och Neuronnätet kunde effektivt replikera klusteringen, och deras förmåga att fånga komplexa relationer gör dem till tänkbara modellar att fortsätta utforska mot riktiga kunder. Avhandlingen avslutas med tänkbara möjligheter att både utveckla och anpassa dessa rekommendationssystem till andra försäkringstyper, vilket markerar ett betydande steg mot mer personliga och användarvänliga försäkringstjänster.

Nyckelord: *Rekommendationssystem, Bilförsäkring, Maskininlärning, Klusteranalys, KNN, XGBoost, Neuronnät.*

Acknowledgements

I want to express my sincere gratitude to my supervisor at Insurely, Frank Kody, for providing me with the opportunity to undertake and complete this Master Thesis. His expertise, insightful feedback, and encouragement have been instrumental in shaping the course of my work.

I am also grateful to my supervisor at Umeå University, Armin Eftekhari, for his scholarly guidance and discussions on the theoretical foundations of my work. His expertise and insightful perspectives have been precious in enhancing the depth and quality of this thesis.

Furthermore, I extend my heartfelt thanks to my family and friends for their unwavering support, encouragement, and understanding throughout my academic journey over the past five years.

Andreas Palmgren

Stockholm, May 2023

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Networks
CBF	Content-Based Filtering
CF	Collaborative Filtering
CPU	Central processing unit
EMR	Exact Match Ratio
GDPR	General Data Protection Regulation
GPU	Graphics processing unit
KNN	K-Nearest Neighbors
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
RS	Recommendation System
RSS	Residual Sum of Squares
SMC	Simple Matching Coefficient
t-SNE	T-Distributed Stochastic Neighbor Embedding
XGBoost	Extreme Gradient Boosting

1 Introduction

This section provides the necessary foundation for the thesis and offers an overview of the topic, the problem definition, and its importance in the broader context.

1.1 Background

Insurely, a Stockholm-based company founded in 2018, provides open insurance solutions allowing data sharing through Application Programming Interfaces (APIs). As a subset of open finance, open insurance enables consumers to access and share their data, empowering them and enhancing the overall customer experience. Insurely's Business-to-Business strategy offers software and solutions to banks and insurance companies, intending to make insurance easier to understand and deal with for the end consumers. The company is committed to driving change and shaping the insurance industry's future. [13]

Working closely with open finance has allowed Insurely to gain a deeper understanding of the needs of both insurance providers and consumers. With the increased data flow of open finance, new opportunities arise to empower the consumer. One way of providing value to consumers is through giving personalized advice and demystifying their insurance. With this new data flow, the topic of Recommendation Systems (RSs) becomes increasingly attractive. RS technology has been successful in several industries and is an active research area. It has integrated itself into our daily lives, with examples such as Netflix movie suggestions or Spotify's 'Fans also like'. However, the insurance industry has yet to receive the same attention. The limited number and complexity of the products within the industry make it different from other domains where RSs have gained popularity. Nevertheless, as data become accessible from different insurance companies across several markets, it is possible to gain an in-depth understanding of consumers' needs.

The insurance industry lacks standardized criteria for defining coverage, making it difficult for consumers to fully comprehend their policies and compare them across various insurance providers. A uniform set of coverage parameters can be applied to all insurance policies to tackle this issue, as seen in Figure 1. These parameters are devised and mapped by industry experts, specifying what a specific insurance policy covers. For instance, a car insurance policy may include a parameter related to theft and another for rental coverage.

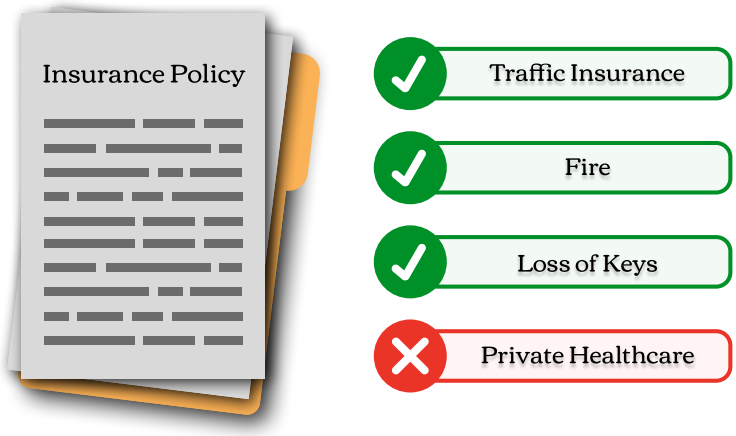


Figure 1: *Parameter mapping of insurance policies.*

A more comprehensive understanding of consumer insurance coverage can be achieved by assessing individuals based on consistent parameters. Furthermore, this method also enables a shift in focus from the insurance provider to the individual's specific needs. For example, the consumers' coverage taking the form of mapped parameters can be combined with consumer metadata, as seen in Figure 2. Metadata encompasses various characteristics, including the consumers' demographic information, such as their age and gender, as well as specific attributes related to the insured item in question. Throughout this thesis, consumers shall refer to the end-consumer, the person who owns the insurance.

Figure 2: *The underlying data set consists out of the consumers current coverage and metadata.*

1.2 Aim of Thesis

This thesis aims to provide an in-depth exploration and comprehensive overview of candidate recommendation systems for the insurance industry. The study will compare traditional approaches in recommendation technology with state-of-the-art alternatives, incorporating both theory and code implementations to supplement the analysis. Finally, the study aims to equip Insurely with a thorough understanding of the various options available, considering both performance and the trade-off between computational complexity and inference.

Recommendation systems in the insurance industry can enhance the consumer experience. This thesis also aims to offer valuable insights and recommendations on how to incorporate RSs in

this sector as effectively as possible while considering the particular challenges and requirements of the insurance industry.

RSs are commonly used for next-item predictions, such as Netflix recommending a movie to watch or Spotify proposing a new song. In this study, we aim to make tailored recommendations to benefit the consumer in the form of a 'consumers like you'-feature. The goal is to empower consumers by giving them a more in-depth understanding of their coverage by revealing the coverage choices of consumers with similar profiles. Consequently, recommendations shall be used to inform consumers if they are over or underinsured. Figure 3 display an overview of the recommendation system.

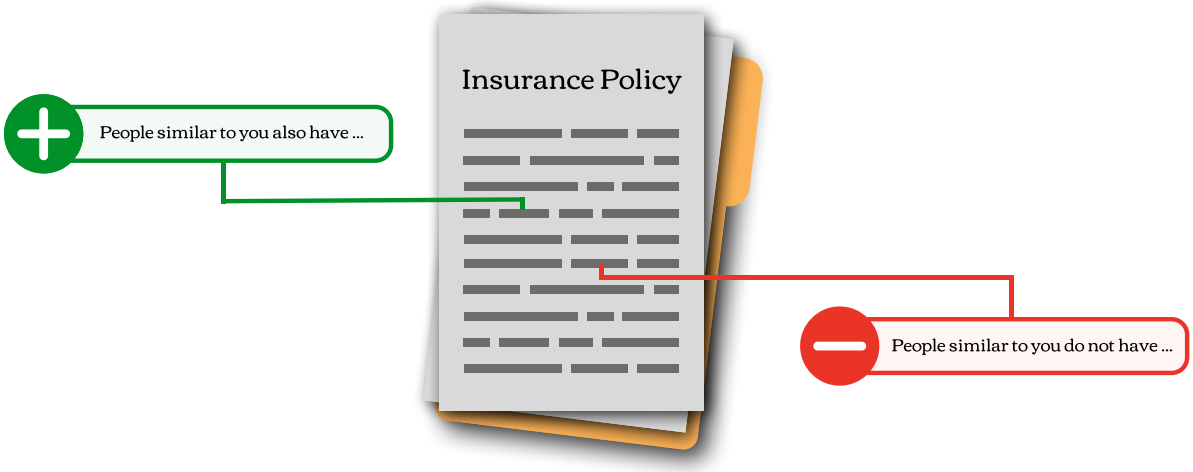


Figure 3: *Illustration of the recommendation system in practice, where it serves as a 'consumers like you'-feature.*

1.3 Delimitation

The thesis will only consider consumers who already have existing insurance coverage. Instead of offering recommendations on existing policies, the system will adopt a more granular approach and focus on standardized coverage parameters. The recommendation system shall inform consumers of a set of parameters explicitly tailored for them, determined by the preferences of similar consumers. The final recommendations will derive from how the consumer's coverage differs from the suggested parameters. The recommendations could be further developed to cater to individual insurance companies by considering their insurance policies.

The scope of the thesis is limited to vehicle insurance, particularly car insurance. Vehicle insurance has a specific set of coverage parameters mutually exclusive from other types of insurance, such as housing. Although the explored recommendation systems could be adapted to address other insurance types, this thesis will concentrate solely on car insurance.

The time period of the project does not allow for evaluation through online testing. Evaluation through live traffic will more accurately measure how well the recommendation systems perform, but it is both slow and expensive. The thesis is an initial exploration of potential alternatives and will be restricted to offline evaluation. A detailed discussion of the limitations associated with offline evaluation will be provided later in the thesis.

1.4 Brief Literature Review

The low amount of documented cases suggests that little attention has been paid to implementing recommendation systems in the insurance industry. A potential reason could be the insurance industry being inherently different from other domains where recommendation systems have proven successful. The goal of recommendations is to filter a large amount of data and is especially effective in domains where an abundance of products exist, which contrasts with the insurance industry.

Numerous recommendation systems within the insurance industry focus on suggesting one policy from a limited selection. For instance, [20] employed Bayesian networks to identify appropriate consumer policies. Similarly, the work in [17] explored the potential of XGBoost and the Apriori algorithm as recommendation systems for car insurance. Although both studies operate within the context of insurance recommendations, their primary objectives are up-selling or cross-selling. It seeks to identify consumers likely to buy additional coverage and what type to recommend. However, this thesis differentiates from these studies in terms of both its objective and the underlying data since it focuses on standardized coverage parameters.

In [4], a recommendation system was proposed for life insurance and endowment. This system aimed to identify the most suitable package for a given consumer based on the package attributes. These attributes bear a resemblance to the coverage parameters utilized in this thesis. However, the nature of the explored insurance packages allowed a few attributes to describe the insurance perfectly. The primary focus of this thesis is on car insurance, which exhibits a higher degree of complexity and flexibility in terms of insurance coverage.

1.5 Disposition

The thesis is organized as follows: Section 2 provides a comprehensive overview of the recommendation systems field and compares it to the thesis problem. Section 3 presents the theoretical foundation of neighborhood models, tree-based models, and artificial neural networks. It also provides a theoretical background for the unsupervised problem of clustering. Section 4 presents the data set for the thesis and the pre-processing required to use it. Section 5 presents the methodology employed for training and evaluating various recommendation systems. Results are presented in Section 5 and subsequently discussed in Section 6. Finally, Section 7 concludes the thesis by summarizing the findings and offering suggestions for further work.

2 Recommendation Systems

This section aims to provide a deeper understanding of the field of recommendation systems.

In recent years, the rapid growth of the digital landscape has resulted in an overwhelming amount of data and content available to users. This influx of information has made it increasingly difficult for users to find relevant, personalized content that caters to their interests and preferences. As a consequence, the importance of recommendation systems has grown substantially. Recommendation systems, designed to filter and present tailored content to users, have become an integral component in various industries, including e-commerce, entertainment, news, and social media platforms.

This chapter aims to provide a comprehensive overview of the field of recommendation technology, delving into the underlying methods, techniques, and algorithms employed to generate personalized suggestions. We will discuss the primary types of recommendation systems, namely content-based filtering, collaborative filtering, and hybrid approaches while highlighting their respective advantages and limitations. Furthermore, we will explore the evaluation metrics used to assess the performance of these systems, addressing the challenges and limitations associated with their development and implementation.

2.1 Types of Recommendation Systems

Recommendation Systems (RSs) filter vast quantities of information to provide the most relevant items to a specific user. The data filtering method and how to determine relevance are design choices influenced by the domain in which the system is to be implemented. The two critical methodologies utilized in most recommendation systems are collaborative filtering and content-based filtering. The available data and the system's objectives determine the choice between the two approaches. A third alternative is to combine the two methods.

2.1.1 Content-based filtering

Content-based filtering (CBF) generates a profile for each user based on their past behavior and interactions. Recommendations are made by analyzing the content of items and comparing it with the preference of individual users. CBF focuses on item features to discover similarities and is successful in areas with well-defined item attributes.

CBF benefits industries like medicine or finance, which require a good understanding of the items when providing recommendations. In the context of this thesis, CBF could be useful when recommending an entire insurance policy. The underlying item is complex and a lot of information is available to describe the item. Leveraging item attributes can significantly enhance recommendations' efficacy, highlighting the importance of content-based filtering. However, this is not the sole approach for providing recommendations. This consideration introduces us to another well-known technique - collaborative filtering.

2.1.2 Collaborative filtering

Collaborative filtering (CF) predicts a consumer's future behavior by analyzing the past behavior of other consumers. The technique builds on making recommendations based on users similar to themselves. As the name implies, collaborative filtering tries to filter information through the collaboration of numerous agents. If both users A and B have expressed interest in the same item, user A may prefer another item that user B prefers. Stored interactions in which the user expresses their view directly fall under *explicit feedback*, such as rating a movie. The other type of interaction is *implicit feedback*, which refers to interactions between the user and the system that are not dependent on the user's opinions, such as online shopping history. While we can see which items the user has engaged with, we do not know how satisfied they are with them. [3]

The entertainment industry frequently utilizes CF, especially in the music and film sectors, to deduce user preferences from behavior. However, CF may not always be as trustworthy as CBF for certain recommendations since users' prior behavior may not always accurately reflect their current or future demands.

Memory-Based Methods

Memory-based filtering is easy to implement and has succeeded in several commercial domains, like Amazon [18]. The approach directly makes use of the user-item interactions to make recommendations. In other words, the system searches for users with similar behavior or tastes as the target user and recommends items that these similar users have liked. The two categories of memory-based collaborative filtering are user-based and item-based filtering.

- **User-based filtering.** Recommendations are made based on the similarity between users' preferences. The method works by finding similar users based on their preferences and recommending items that these users have rated highly. User-based filtering can provide personalized recommendations and can capture complex user preferences. However, it can suffer from the sparsity problem, where users may have rated only a small subset of items, making it difficult to find similar users. User-based filtering can also be less scalable than item-based filtering, especially when the number of users is large.

- **Item-based filtering.** Recommendations are made based on the similarity between items that users have rated highly. The method creates a similarity matrix that measures the similarity between each pair of items based on the ratings given by users. The algorithm then recommends items most similar to those the user has already rated highly. Item-based filtering scales well as it only needs to compute similarities between items, and it effectively addresses the cold-start problem by providing recommendations for new items with no ratings. In addition, item-based filtering can provide more accurate recommendations in some cases, especially when the number of items is much larger than the number of users.

Model-Based Methods

This approach involves building a model of users and items based on their behavior and using this model to make recommendations. Typically, developers build the model using machine learning techniques such as matrix factorization, clustering, or deep learning. Unlike memory-based collaborative filtering, which requires calculating similarities between users or items for each recommendation, model-based collaborative filtering can make predictions based on the learned model. This approach can handle sparsity in the user-item matrix and perform better than memory-based methods in many cases, but it requires more computational resources for training the model.

2.1.3 Hybrid Approach

When developing a recommendation system, one should view collaborative and content-based filtering as complementary rather than competing approaches. Each strategy has advantages and disadvantages, and combining them can provide a more robust recommendation system. The third strategy, hybrid filtering, combines collaborative and content-based filtering to produce a robust and efficient recommendation system. It makes use of both user behavior and item attributes to generate recommendations. The design decision of how to merge the two approaches often requires an ad hoc solution suitable for the particular circumstance. For example, the system may first use content-based filtering to identify items that match user preferences based on past interactions. Then, the recommendation system may use collaborative filtering to identify similar users and their behavior to further refine the recommendations.

2.2 Challenges and Limitations

Different domains entail different data sets, leading to different possibilities and challenges. The performance of recommendation systems is not always transferable across different data sets. The characteristics of the underlying data determine what is an appropriate approach and what is not. For example, a model performing well on data with millions of items might not be the best choice when the number of features is limited.

The properties of the underlying data are thus critical when considering the design of a recommendation system. There are several key properties that data sets for recommendation systems should possess, including:

- **Sparsity**. Recommendation systems often deal with large, sparse data sets, meaning that the number of items or users is significantly greater than the number of interactions or ratings. This property can make it challenging to generate accurate recommendations, as limited data may be available for some users or items. However, the insurance industry typically features a small range of products, which mitigates the sparsity problem in this context.

- **Source of interactions**. Users may not provide explicit feedback in some domains, such as ratings or reviews. Instead, implicit feedback, such as clicks or purchase history, is used to infer user preferences. While implicit feedback may be noisy and less reliable than explicit feedback, it can still generate accurate recommendations. In this thesis, we treat a consumer's current coverage as implicit feedback.

- **Cold-start problem**. Recommendation systems can face challenges such as the cold-start problem, where there is little to none data available for new users or items. In these cases, it can be challenging to generate accurate recommendations, as there needs to be more information available about the user's preferences. However, in this study, the recommendation system focuses on assisting consumers who already possess coverage. The recommendations aim to guide consumers in adjusting their coverage to match those of similar consumers better. Consequently, the cold-start problem is not a significant concern for this thesis.

- **Item popularity**. Many recommendation systems deal with a long-tail distribution, where a few items are highly popular, and most items have low popularity. This property can pose challenges in generating recommendations for less popular items, and the problem intensifies as the number of items increases. However, the insurance industry typically offers a limited range of products, which mitigates the impact of long-tail distribution in our context.

2.3 Evaluation

Constructing a recommendation system requires crucial considerations regarding the choice of evaluation. There are various approaches to evaluation, and it is essential to select the appropriate method based on the system's purpose. For example, one common approach is creating a ranked list of recommendations or identifying all good items for the user [11].

Whatever the metric choice, measuring a RS performance in real-world situations is the only way to determine its true performance. In academic research, researchers commonly use offline evaluation methods due to the low cost of data and the ability to explore options quickly. However, offline evaluation methods only measure supposed reactions, which may differ from the true response in live traffic. Offline experiments aim to identify a set of candidate models that can be tested on live traffic [23].

Offline evaluation requires simulating user behavior, which can turn out to be difficult. A common approach is to hide a predetermined number of known interactions for a user and try to predict them [23]. This evaluation method has raised concern since one can question the generalization power. To address the generalization problem, more advanced user simulations can be employed.

An advanced simulation framework necessitates solutions tailored to specific problems, yet it can offer a more comprehensive understanding of the performance of recommendation systems. However, it is essential to ensure the simulated user behavior is representative of real-world behavior to obtain accurate and reliable results. The desired behavior to be studied must be clearly defined and simulated using appropriate methods. For example, in [25], recommendation systems were evaluated by simulating different types of user behaviors. The simulation framework allowed for the exploration of atypical user behavior, which could be of interest in some domains.

The offline stage ends when a set of candidate models has been identified for further exploration via online evaluation, specifically through A/B testing. Here, a random selection of users encounters varying versions of the recommendation system. The performance of these different systems can be analyzed based on metrics like conversion and revenue generation. This method provides an effective means to assess the performance of a recommendation system within a live environment and can help identify areas for improvement.

3 Theory

This section aims to construct a rigorous theoretical framework that serves as the foundation for thesis.

3.1 Data

Analyzing and modeling require a good understanding of the data types involved. Following is a formal description of data types encountered in this study. We also present relevant transformations and similarity metrics.

3.1.1 Data types

Categorical data can be divided into distinct groups. If the categories have an implied order, such as age groups, it is called *ordinal data*. The other type is *nominal data* which is categories without order, such as gender.

Dichotomous data is nominal data with two levels, allowing for a binary representation of the data. For example, it could indicate the presence or absence of a particular feature. *Binary data*, a subset of dichotomous data, have categories numerically encoded as 0 and 1. The encoding offers several advantages, like computational efficiency and compatibility with various algorithms and models.

3.1.2 Dummy encoding

Dummy encoding is a widely employed technique that transforms categorical variables into a binary representation. This conversion proves particularly advantageous when dealing with algorithms requiring numeric inputs. The method entails the construction of binary dummy variables corresponding to all but one distinct category within the categorical variable. In other words, dummy encoding will transform a categorical feature with k levels into $k - 1$ dummy variables. Figure 4 provides an example of dummy encoding on the categorical variable country.

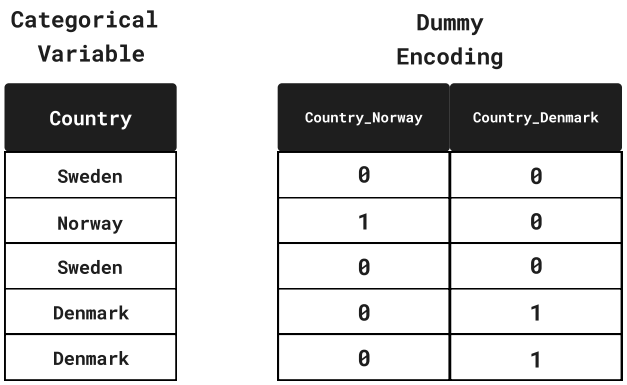


Figure 4: *Example of dummy encoding for the categorical feature country, creating two new binary variables with the same information.*

Another widespread transformation is *one-hot encoding* which does not omit a level. The rationale for selecting k-1 dummy variables instead of k stems from the motivation to circumvent multicollinearity.

3.1.3 Similarity measures

In statistics, similarity measures are a way to quantify the similarity between two observations. Measuring similarity requires taking the data sets characteristics into account. Consequently, many different metrics exist for different use cases and require careful consideration.

Simple matching coefficient

The *Simple Matching Coefficient* (SMC) represents a fundamental and intuitive approach to measuring similarity among dichotomous data. This similarity metric is particularly well-suited for comparing binary data, given its capacity to account for the presence and absence of attributes simultaneously. SMC is an appropriate choice of similarity metric in scenarios wherein positive and negative instances contribute information equivalently.

$$\text{SMC} = \frac{\text{number of matching attributes}}{\text{sum of all attributes}}$$

Exact Match Ratio

The *Exact Match Ratio* (EMR) can be interpreted as accuracy in a multilabel classification problem. It is a harsh evaluation metric since it does not account for partially correct labels. Formally, it is defined as:

$$\text{EMR} = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i)$$

where I is the indicator function.

3.2 Clustering

Clustering is an unsupervised learning paradigm for finding meaningful subgroups. The principal objective entails partitioning a data set into non-overlapping clusters based on the inherent similarity among the observations. We seek to find subgroups such that data points within the same cluster exhibit higher similarity to one another than to data points belonging to different clusters. Clustering algorithms aim to identify these inherent structures or patterns in the data

without relying on prior knowledge or labeled examples. However, finding a meaningful set of clusters requires a careful decision of metrics to quantify the similarity between observations. No universal choice of similarity metric exists and often requires domain-specific considerations. [14]

3.2.1 K-means

The K-means algorithm is a well-known and straightforward clustering technique that requires us to specify the number of desired clusters. Let us define K predefined number of sets $\mathbf{C} = \{C_1, C_2, ..., C_K\}$, where each set contains the indices of the observations within that cluster. These sets exhibit two essential properties. Firstly, each observation has been assigned to at least one of the clusters. Secondly, clusters are non-overlapping; no observation belongs to two or more clusters. Formally, we can express the two properties as follows:

1. $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, \ldots, n\}$
2. $C_i \cap C_j = \theta \quad \forall i \neq j$

The idea behind K-means is that a good partition minimizes the *within-cluster variation*, which measures the dissimilarity between observations within the same cluster. Let $W(C_k)$ be the within-cluster variation for cluster C_k, the K-means problem can thus be defined as:

$$\underset{C_1,\ldots,C_K}{\arg\min}\left\{\sum_{k=1}^{K} W(C_k)\right\} \tag{3.1}$$

To solve (3.1), the within-cluster variation need to be defined. The most common choice is to use the squared Euclidean distance:

$$W(C_k) = \frac{1}{|C_k|}\sum_{i,i' \in C_k}\sum_{j=1}^{p}(x_{ij} - x_{i'j})^2 \tag{3.2}$$

where $|C_k|$ is the number of observations within the kth cluster. By combining (3.1) and (3.2), we obtain following problem formulation:

$$\underset{C_1,\ldots,C_K}{\arg\min}\left\{\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{i,i' \in C_k}\sum_{j=1}^{p}(x_{ij} - x_{i'j})^2\right\}$$

If the number of clusters K and observations n are not small, we have too many possible partitions to consider every single one. The K-means algorithm presented in Algorithm 1 is an intuitive way of finding a local optimum. Since K-means initialize by randomly assigning observations as cluster centroids and the algorithm finds a local optimum, the result is sensitive to the initialization step. [14]

3.2.2 Metrics

Clustering is an unsupervised task with no labels to be measured against. However, several other metrics exist to verify the performance of clustering algorithms. This thesis utilizes both the *elbow method* and *silhouette analysis* to evaluate cluster quality.

Algorithm 1 Algorithm for K-means clustering

1. Initialization step: Specify the number of clusters K and randomly select K data points to be cluster centroids.
2. Iterate until the algorithm converge and the assignment step makes no new changes:
   (a) Assignment step: Each observation is assigned to the nearest cluster according to Euclidean distance.
   (b) Update step. Recalculate cluster centroids based on the new clusters.

3.2.3 Silhouette analysis

Silhouette analysis is a technique to assess the quality of clusters. It is handy for determining the optimal number of clusters, as it evaluates the generated clusters' internal cohesion and external separation.

Given an observation i, let us denote A as its assigned cluster. Let $a(i)$ represent the average dissimilarity of i to all other observations of A. Let us consider any cluster C different from A. As we did for observations in A, we can calculate the average dissimilarity of i to all observations of C given by $d(i, C)$. We denote B as the cluster with the lowest average dissimilarity given by $b(i) = \min_{C \neq A} d(i, C)$. With our notations, the *Silhouette coefficient* $S(i)$ is given by:

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

$S(i)$ is between -1 and 1, where a higher positive score indicates clear, distinguishable clusters. Negative scores suggest the observation has been assigned the wrong clusters. [21]

Silhouette analysis can help determine the optimal number of clusters by looking at the average silhouette. In practice, a score of at least 0.5 is often seen as acceptable. One option is to use the average silhouette method, which compares the average silhouette score across different numbers of clusters. Another alternative is to utilize a silhouette plot to assess the silhouette coefficient for each cluster. The visual representation allows for investigation of fluctuation in size between clusters and if any have below-average silhouette scores.

3.2.4 Elbow method

One of the most common techniques used in cluster analysis is the Elbow method, also known as scree plot. The method utilize the *inertia*, which is the calculated by:

$$\text{Inertia} = \sum_{i=1}^{N} (x_i - C_k)^2 \tag{3.3}$$

where N is the number of observations and C is cluster centroid. In other words, Equation (3.3) sum the squared distance between each observation to its cluster centroid. Consequently, a lower value means more compact clusters. The method is used to tune hyperparameters, for example the number of clusters as in Figure 5.

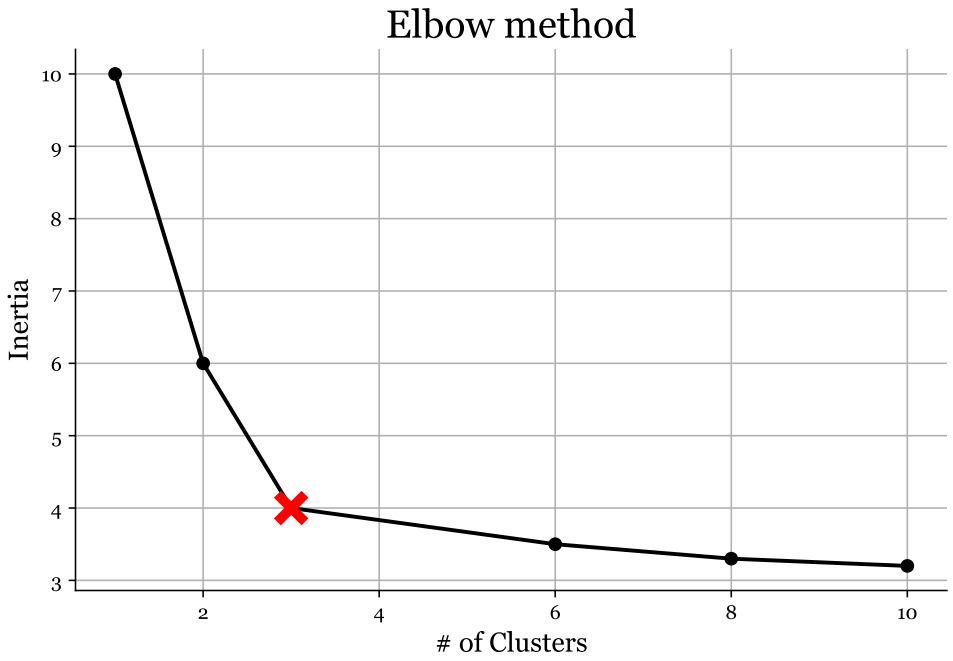


Figure 5: *Example of the elbow method to assess the cluster quality and tune the number of clusters. The elbow point is indicated by the cross which is when decrease in inertia begins to slow down.*

3.3 Supervised models

Three different methods will be explored in terms of supervised learning: neighborhood models, tree-based models, and neural networks. Each offers a unique perspective to approach decision making, with their own advantages and drawbacks.

3.4 K-Nearest Neighbors

Nearest neighbor algorithms are one of the most fundamental algorithms in machine learning. The idea is simply comparing a new instance with the memorized training data and predicting based on the closest neighbors. The *K-Nearest Neighbors* algorithm, also known as KNN, is a nearest neighbor algorithm commonly used for statistical classification. It is a non-parametric, instance-based learning method for classification and regression tasks. Classifications are made by looking at the K most similar observations.

Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a sequence of the training examples. The Euclidean distance between two observations is defined as $\rho(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}|| = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2}$. For each input $\mathbf{x}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a reordering of the training data indices $\{1, \ldots, m\}$ in ascending distance to $\mathbf{x}$. In other words, the distance $\rho(\mathbf{x}, \mathbf{x}_{\pi_i(\mathbf{x})}) \leq \rho(\mathbf{x}, \mathbf{x}_{\pi_{i+1}(\mathbf{x})})$. For a new instance $\mathbf{x}$, the algorithm selects the K closest neighbors from the training data $(\mathbf{x}_{\pi_1(\mathbf{x})}, y_{\pi_1(\mathbf{x})}), \ldots (\mathbf{x}_{\pi_k(\mathbf{x})}, y_{\pi_k(\mathbf{x})})$. The label for the new instance is determined through majority voting based on the closest neighbors. [22]

To better understand the algorithm, an example of the decision boundary for the 1-NN rule can be seen in Figure 6. In this case, a new instance is assigned the label corresponding to its closest neighbor.

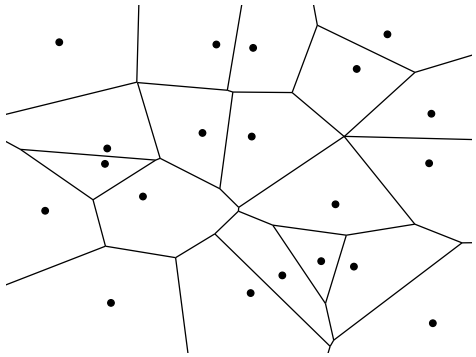


Figure 6: *Illustration of the 1-NN decision boundaries through a Voronoi diagram. The points represents the training data and a new instance would be assigned the label from the training sample found in the corresponding cell.*

Since KNN is an algorithm and a non-parametric, instance-based learning method, the *training phase* does not construct a model but rather stores all training data. Consequently, the training time for the algorithm is hard to beat compared to other alternatives. All heavy lifting is done during the *classification phase*. Prediction speed is correlated to the number of training examples and features since each new instance requires a pairwise similarity score with all training examples. Due to KNN being a lazy learner, the algorithm is more appropriate when working with a smaller data set. [7]

The KNN algorithm is commonly used for recommendation systems based on collaborative filtering. By considering the explicit or implicit information, one makes recommendations based on others with similar information. One example could be movie recommendations based on users' historical reviews. The explicit information collected from a particular user can thus be matched with other users who have seen the same movies and responded similarly. With the approach's simplistic nature, the interpretation of the recommendations becomes understandable and transparent.

3.5 Ensemble methods

Ensemble methods are a class of machine learning techniques that aim to improve predictive models' overall performance and robustness by combining multiple base models or learners. These methods capitalize on the wisdom of the crowd, integrating diverse perspectives and reducing the impact of individual model biases and errors. The fundamental assumption underpinning ensemble methods is that the collective output of multiple models delivers more accurate and stable predictions than the output of any single model. [14]

3.5.1 Bootstrap Aggregating

Bootstrap aggregating (Bagging) is an ensemble method that involves training multiple base models independently, typically with the same algorithm, on different subsets of the original training data. Generating these subsets involves using bootstrapping, which entails random sampling with replacement. After training the base models, they aggregate their predictions, typically through a majority vote for classification tasks or by averaging for regression tasks. This aggregation process reduces variance and mitigates overfitting, enhancing the overall model performance. [14]

3.5.2 Boosting

Boosting is an iterative ensemble method in which base models are trained sequentially, with each subsequent model attempting to correct the errors of its predecessor. The approach involves assigning weights to the training instances, updated iteratively based on the model's performance. Misclassified instances or instances with larger errors are assigned greater weights, prompting the subsequent model to focus on these more challenging instances. The final ensemble model is constructed by taking a weighted combination of the base models, where the models' performances typically determine weights. Boosting can effectively reduce bias and variance, resulting in improved predictive accuracy. Common boosting algorithms include AdaBoost and Gradient Boosting. [14]

3.6 Decision Tree

Decision trees are a simple but widely used supervised learning algorithm for classification and regression problems. The method recursively partitions the predictor space. A decision tree can be visualized through a flow chart. Decision trees can arguably be seen as closely related to how humans make decisions. Because of this nature, they are often easy to understand and explain to others.

Constructing the tree comprise of two main steps

1. Partition of the predictor space into M distinct and non-overlapping regions $R_1, ..., R_M$.
2. For each observation in region R_j, the prediction is the mean (or mode for classification) for all training observations in region R_j.

For regression, to find M regions we minimize the Residual Sum of Squares (RSS), given by:

$$\sum_{m=1}^{M} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean within the j^{th} region. As one can imagine, considering every possible partition is often practically impossible. Instead, one starts from the top (all observations are in the same region) and successively splits the predictor space into two new branches by making the best possible split at each step. This top-down, greedy approach is also known as recursive binary splitting. [14]

In other words, for each predictor X_j, we aim to find a split point s such that the resulting tree minimize the RSS. For any j and s, we define following two regions

$$R_1(j, s) = \{X \mid X_j < s\} \text{ and } R_2(j, s) = \{X \mid X_j \geq s\}$$

and we aim to find j and s such that we minimize

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1}) + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean response for the training observations in $R_1(j, s)$ and $R_2(j, s)$, respectively. The resulting regions are once again split until a predetermined stopping criterion as been met. For instance, the process stop when no region contain more than K observations. [14]

Decision trees for classification are very similar to the regression setting. Instead, the prediction for each observation is based on the most frequent class of training observations in a region. Since RSS can not be used for categorical data, an alternative could be the *classification error rate*, simply the fraction of observations that differ from the most common class. However, this metric is not sufficiently sensitive for tree-growing. In practice, a preferable choice over the classification error is the *Gini index*. Formally, we can define the Gini index G as:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where $\hat{p}_{mk}$ is the proportion of training observations in the m^{th} region from the k^{th} class. It looks at the total variance across all K classes. [14]

3.7 Random Forest

A single decision tree generally results in high variance. To address the problem, one can utilize the bagging algorithm to create a large collection of trees, also known as *bagged trees* or a forest of trees. *Random Forest* is a further improvement by decorrelating the trees in the forest by introducing an additional layer of randomness since only a subset of all features is considered when finding the best split. This adjustment will create a broader variety in the collection of trees, improving generalization and reducing overfitting. [14]

The number of predictors to consider at a given split is a hyperparameter that must be predetermined. A common rule is to consider the square root of the total number of predictors. Consequently, most predictors are hidden during a split, minimizing the effect a strong predictor might have when building trees. If all predictors are considered at each step, and one predictor is participle strong, all bagged trees will look similar, thus defeating the purpose of bagging. [14]

3.8 Extreme Gradient Boosting

Extreme Gradient Boosting, or XGBoost, is a robust machine learning algorithm predominantly used in structured or tabular data sets. As a member of the gradient-boosted trees family, it exhibits high expressive power and has demonstrated its proficiency across multiple domains.

XGBoost, like Random Forest, builds on the concept of decision tree ensembles. While Random Forest builds independent trees in parallel and combines them by averaging their predictions, XGBoost uses an additive strategy instead. Trees are constructed sequentially, where each tree is built to correct the errors of its predecessor.

3.8.1 Regularization of Ensemble Model

Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ with m features and a cardinality of n, the tree ensemble model is defined as:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}$$

where $\mathcal{F}$ is the space of classification and regression trees known as CART. $\mathcal{F}$ is defined as $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$. Let us break it down by first looking at q which represents the tree structure for each tree that maps a data point to corresponding leaf index. T represents the number of leaves in corresponding tree with leaf weights w. Consequently, each f_k corresponds to a unique tree structure and leaf weights.

We minimize the following regularized objective function to learn the set of K functions:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{3.4}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$. The differentiable convex loss function l measures the difference between our predictor and target. The objective function is regularized due to the second term Ω, which penalizes complexity. With this parameter set to zero, the objective function would be equivalent to traditional gradient tree boosting.

3.8.2 Gradient Boosting

Since Equation (3.4) contain functions as parameters it requires the model to be trained in an additive manner rather than traditional optimization in Euclidean space. Consequently, let $\hat{y}_i^{(t)}$ represent the prediction of the i-th instance at the t-th iteration. By adding f_t, we obtain following objective function where we greedily add the f_t according to the improvement on the model obtained in Equation (3.4):

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \sum_k \Omega(f_k)$$

To optimize the objective function, we apply a second order approximation.

$$\mathcal{L} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ are the first and second order gradient statistics on the loss function. A simplified form is obtained by removing the constant term.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{3.5}$$

Let the set of observations at leaf j be defined as $I_j = \{i \mid q(\mathbf{x}_i) = j\}$. Consequently, Equation (3.5) can be rewritten as:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_i^2 + \gamma T]$$

For a given tree structure $q(\mathbf{x})$, the optimal weight w_j^* of leaf j is given by:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

and the corresponding optimal value is given by:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{3.6}$$

Equation (3.6) serve the same function as impurity for evaluating decision trees. It is a scoring function used to measure the quality of a given tree structure.

In practice, it is impossible to consider every single tree structure. However, it is possible

to employ a greedy algorithm instead. It starts with a single leaf and iteratively grows by adding branches. After a split, let I_L and I_R be the instance sets for the left and right nodes. By letting $I = I_L \cup I_R$, the loss reduction after the split is given by:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \lambda$$

In addition to the regularized objective function, XGBoost utilizes two more techniques to prevent overfitting. Firstly, shrinkage introduces weight to newly added trees. One can interpret the weight as the model's learning rate, where shrinkage reduces the weight of individual trees. The second technique is column subsampling, also used in Random Forest as discussed in Section 3.7. [6]

3.9 Artificial Neural Networks

In recent years, *Artificial Neural Networks* (ANNs) have proven successful in several domains. Neural networks are an attempt at recreating a simpler version of the biological neural networks in the brain. A large collection of simple classifiers (neurons) are connected like synapses in the brain. Before jumping to the network architectures, we must understand the fundamental building blocks. We shall start with our artificial neurons.

3.9.1 Artificial Neurons

Constructing a simple neural network involves arranging artificial neurons hierarchically, where the output from one neuron serves as the input for the next layer. All layers between the input and output are called *hidden layers*, where the connections to and from the hidden layers are the trainable weights. Figure 7 displays a simple neural network with fully connected layers, also known as a *fully dense neural network*.

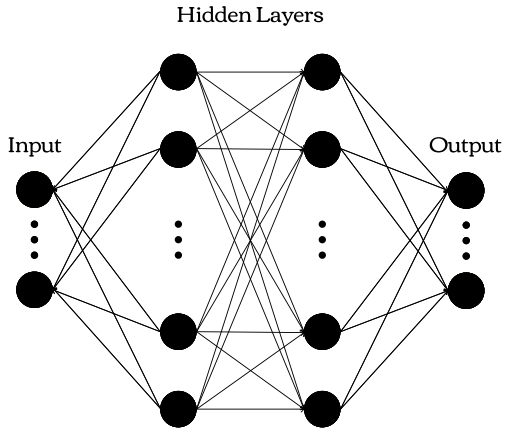


Figure 7: *An example of a fully dense neural network with 2 hidden layers.*

Given a single neuron, let $\mathbf{x}$ be the output from the previous layers and $\mathbf{w}$ be the weights

connected to the neuron. Formally, one can define the output of a given neuron as:

$$y = \sigma\left(\sum_{j=0}^{m} w_j x_j\right)$$

where σ is called the *activation function.*

3.9.2 Activation Functions

One of the core building blocks of the neural network is the *activation functions.* The output of a neuron is passed through an activation function which can be interpreted as a gate that determines if the output shall pass or not. The activation function also allows for a desired transformation, which can both add non-linearity to the network and control the neural network's output. A commonly used activation function is the rectified linear unit (ReLU).

$$\sigma(x) = \max(0, x) = \begin{cases} 0, & \text{if } x < 0. \\ x, & \text{otherwise.} \end{cases}$$

The output layer often employs a different activation function than the hidden layers. For regression problems, one can use the output directly without passing it through an activation function since no transformation is needed. For a binary classification problem, using a Sigmoid activation function in the output layer is common practice since it predicts the probability of a given output. The Sigmoid function used in deep learning architectures is the logistic function defined by:

$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

Figure 8 provides a visualization of both the ReLu and Sigmoid activation functions.

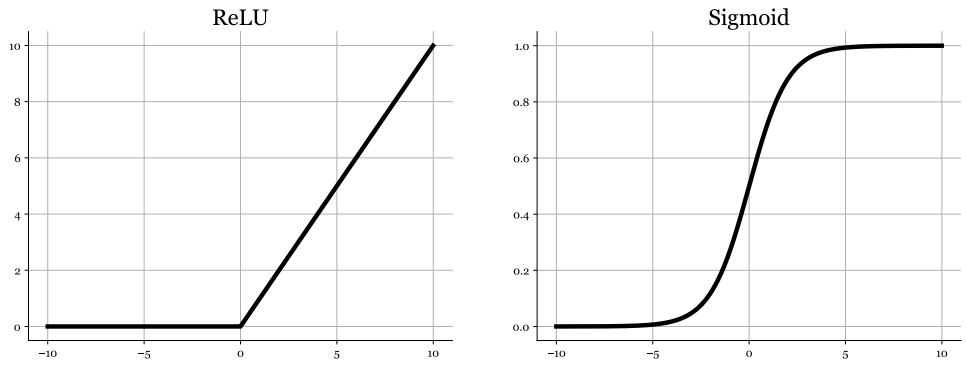


Figure 8: *Two commonly used activation functions: ReLU (left) and Sigmoid (right)*

3.9.3 Feed-forward network

A feed-forward neural network is one of the more straightforward neural network types since information travels in one direction, and the network does not contain any cycles. In other words, it can be described by a directed acyclic graph $G = (VE)$. The graph includes a weight function over the edges $w : E \to \mathbb{R}$. Each node in the graph is represented as a neuron, which is defined as a simple scalar function $\sigma : \mathbb{R} \to \mathbb{R}$, where σ is called an *activation function.*

Let $v_{t,i}$ denote the ith neuron in the tth layer. The output of $v_{t,i}$ is denoted as $o_{t,i}(\mathbf{x})$ where $\mathbf{x}$ is the input vector. Calculations in a feed-forward network is in a layer by layer manner. In other

words, the outputs from layer t is used to calculate the outputs from layer $t+1$. More specifically, the inputs to $v_{t+1,j}$ is the weighted sum of the output from the neurons in the previous layer t. The connection between layers are the trainable weights w and the output of a layer is the activation function σ applied to its input. This layer-wise propagation can be expressed in equations if we fix some $v_{t+1,j} \in V_{t+1}$ and let $a_{t+1,j}(\mathbf{x})$ denote the input to $v_{t+1,j}$. Consequently,

$$a_{t+1,j}(\mathbf{x}) = \sum_{r:((v_{t,r},v_{t+1,j}))\in E} w(v_{t,r}, v_{t+1,j}) o_{t,r}(\mathbf{x}) \tag{3.7}$$

$$o_{t,r}(\mathbf{x}) = \sigma(a_{t+1,j}(\mathbf{x})) \tag{3.8}$$

The weighted sum in Equation (3.7) considers all nodes in layer t connected to nodes in layer $t+1$. Equation (3.8) shows the output of layer t, which is once again simply the input passed through its activation function. [22]

3.9.4 Training of Neural Networks

Given a specific neural network $f_{V,E,\sigma,\mathbf{w}}$, one obtain a function $h_{V,E,\sigma,\mathbf{w}} : \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$. With V, E and σ fixed, we define the set of possible neural network predictors to be $h_{V,E,\sigma,\mathbf{w}}$ for some $w : E \to \mathbb{R}$. Consequently, the parameters specifying a hypothesis in our hypothesis class are the weights.

A network with n input nodes, k output nodes, and weight function $\mathbf{w}$, has a network function denoted by $h_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}^k$. The loss of predicting $h_{\mathbf{w}}(\mathbf{x})$ is given by $\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$, where $\mathbf{y}$ is the target. Given an unknown joint distribution $\mathcal{D}$ over our predictors (input) and response (output), the *population risk* of the network can be defined by:

$$L_{\mathcal{D}}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}[\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})] \tag{3.9}$$

Training a neural network entails minimizing the risk function given in Equation (3.9). Unfortunately, $\mathcal{D}$ is unknown which in turn makes it impossible to calculate $\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$. However, we have access to a set of training observations which allows to calculate the *empirical risk*, also called training error.

Given an example $(\mathbf{x}, \mathbf{y})$, the loss function is given by $l(\mathbf{w}) = \Delta(h_{\mathbf{w}}(\mathbf{x}))$. We must somehow minimize the loss function with respect to $\mathbf{w}$. *Gradient descent* is a simple and intuitive approach to estimate the empirical risk numerically. It is an iterative approach for minimizing a differentiable convex function $f(\mathbf{w})$. Let $\Lambda f(\mathbf{w}$ denote the gradient of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{w}$. Given an initial value of $\mathbf{w}$, the algorithm will at each time step update in the direction of the negative of the gradient. The update step within gradient descent can be formulated as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \Delta f(\mathbf{w}^{(t)})$$

where η is the step size, also referred to as the *learning rate*. The learning rate is an important hyperparameter which must be tuned.

In practice, gradient descent is inefficient since it considers the entire training set at each time step. *Stochastic Gradient Descent*, also called SGD, tries to solve the shortcomings of gradient descent by only considering a single observation. However, only considering a single observation also has its shortcomings. Combining the two, *mini-batch SGD*, is a more practical approach considering only a subset of the training data. Consequently, the update step is no longer based on the gradient but rather an estimate of the gradient by considering a randomly selected subset of the training data.

Neural networks employ *backpropagation* to calculate the gradient. After an input has traversed the network and a loss has been calculated, the error is propagated backward throughout the network. By the chain rule, one can calculate the partial derivative of the loss function with respect to weights and biases. The weights and biases are then updated based on the computed gradients. [22]

3.9.5 Loss Function

The choice of loss function requires careful consideration. Depending on the objective and underlying data, drastically different loss functions should be employed. For regression, a common choice is the *Mean Squared Error* (MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

For classification, MSE would generally not be an appropriate loss function. A better alternative for multi-class classification problems would be *categorical cross-entropy* since it compares the predicted probabilities against the ground truth. This thesis predicts a feature's presence (1) or absence (0), thus finding itself in a binary classification problem. In such cases, *binary cross-entropy* is a good alternative:

$$-(y\ \log(p) + (1-y)\ \log(1-p))$$

3.9.6 Embeddings

Neural networks have great expressive power, but it does face limitations in the case of approximating any arbitrary non-continuous function. Revealing the underlying continuity in the data can thus increase the power of the neural network. This is true for many types of neural networks, for instance, how convolutional neural networks will group pixels to increase continuity rather than representing images as a single flattened vector.

Since networks assume a certain level of continuity, dealing with categorical variables is problematic. A common way of feeding categorical variables into models is through one-hot encodings, as discussed in Section 3.1.2. However, the method comes with two significant drawbacks. High-cardinality variables result in many sparse features, and the transformation is uniform, resulting in the similarity between categories not impacting the vector representation. [9]

Dummy encoding can be seen as an unsupervised task since it does not try to minimize any error based on the transformation. However, as we have transitioned into the world of neural networks, one can utilize the network's training to learn embeddings. The vector space which defines the embedding for a categorical feature becomes part of the network's weights. Since the network is trained in a supervised manner to minimize loss, the resulting feature space will ensure that more similar categories are placed closer to each other.

3.10 Dimensionality Reduction

Dimensionality reduction transforms high-dimensional data into a low-dimensional space where the new representation retains important structure from the original data. Working in a high-dimensional space can be challenging since it does not allow for visualization, and one needs to face the curse of dimensionality.

3.10.1 t-SNE

T-distributed stochastic neighbor embedding, or t-SNE, is a nonlinear dimensionality reduction algorithm used to visualize high-dimensional data. The methods aims to convert a high-dimensional data set $X = \{x_1, x_2, \ldots, x_n\}$ into a low-dimensional map $\mathcal{Y}$. The goal is to preserve significant structure of the high-dimensional data in a lower space that can be visualized.

The method begins by converting the high-dimensional Euclidean distances between observations into conditional probabilities and represent the similarity between observations. The similarity between observation x_j to x_i is the conditional probability that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. The conditional probability $p_{j|i}$ is given by:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2}$$

where σ_i is the variance of the Gaussian centered on observation x_i. To ensure each observation makes a significant contribution to the cost function, we define the similarity score in the high-dimensional space to be the symmetrized conditional probabilities.

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

As for the high-dimensional data, we measure the conditional probability for the low-dimensional counterpart, denoted as $q_{j|i}$. t-SNE aims to find a low dimensional representation of the data which minimizes the mismatch between $p_{j|i}$ and $q_{j|i}$. t-SNE employ the Student t-distribution with one degree of freedom (also known as Cauchy distribution) as the heavy-tailed distribution in the low-dimensional map. The joint probabilities q_{ij} are defined as:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

Since we are only interested in the pairwise similarity, the joint probability for $i = j$ is set to zero. The Kullback-Leibler divergence is minimized using gradient descent to find the map $\mathcal{Y}$. Consequently, we obtain the cost function C. [24]

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

3.10.2 Autoencoder

An autoencoder is a network trained to recreate its input. The idea is to force the network to compress the input into a lower dimensional latent space through an encoder. The encoded representation is then fed into a decoder which reconstructs the input. The network learns a latent data representation by minimizing the reconstruction error.

Given an *encoder* f_θ, for each observation $x^{(t)}$ from the data set $\{x^{(1)}, \ldots, x^{(T)}\}$, the *feature-vector* or *latent representation* is defined as

$$h^{(t)} = f_\theta(x^{(t)})$$

The *decoder* g_θ maps the latent representation back into the input space, thus producing a reconstruction $r = g_\theta(h)$.

Autoencoders are parameterized through their two primary components: the encoder and the

decoder. It is the encoder and decoder's weights and biases that determine the structure and capacity to learn of an autoencoder. The set of parameters θ are learned simultaneously by minimizing the reconstruction error $L(x, r)$. If the latent representation computed from $x^{(t)}$ is just as high dimensional as the input, the encoder can copy the input into the representation. To prevent the autoencoder from learning the identity function, one can ensure the latent representation is of a lower dimension than the input space, in other words, bottlenecked. [2]

In summary, autoencoders aim to find a set of parameters θ minimizing the reconstruction error

$$\mathcal{J}_{AE}(\theta) = \sum_{t} L(x^{(t)}, g_\theta(f_\theta(x^{(t)}))$$

where $x^{(t)}$ is a training example.

4 | Data

This section provides a detailed description of the available data and pre-processing steps required for modeling.

4.1 Description

The metadata includes demographic, socioeconomic, and geographical attributes. Let us denote metadata as $M \in \mathbb{R}^{n \times k}$, where n is the number of consumers, and k is the number of metadata attributes. Thus $m_{i,j}$ corresponds to the ith consumer and the jth attribute.

Insurance coverage consists of binary information, whether a consumer's current coverage includes the parameter. Let us define coverage data as $X \in \mathbb{R}^{n \times d}$, where n is the number of consumers, and d is the number of insurance parameters. The current coverage for the ith consumer is defined by the vector $X_i = \{x_1, x_2, \ldots, x_d\}$, where each entry represents the presence (1) or absence (0) of the corresponding insurance parameter. This thesis will refer to these insurance coverage parameters as *coverage parameters*.

Our latent variable is $Y \in \mathbb{R}^{n \times d}$, thus having the exact dimensions as the insurance coverage since they share the same features. Therefore, recommendations will be made for every coverage parameter, thus proposing the ith consumer a set $Y_i = \{y_1, y_2, \ldots, y_d\}$ parameters.

4.1.1 Descriptive Statistics

We should ensure that the idea behind our latent variable is evident since the thesis is based on these recommendations. The 20+ different coverage parameters specify different types of coverage. Take an example by looking at four coverage parameters: traffic insurance, fire, loss of keys, and private healthcare. Figure 9 illustrates the age distribution, one of our metadata features, across the four chosen coverage parameters. Age is one factor that is often seen as necessary when looking at insurance since it affects the price. However, our perspective differs from the usual insurance setting since our features try to explain what the consumer should be covered by. The age distribution is quite similar between most coverage parameters. However, more unique alternatives, such as the add-on for private care in emergencies, seem to be more popular with older consumers.

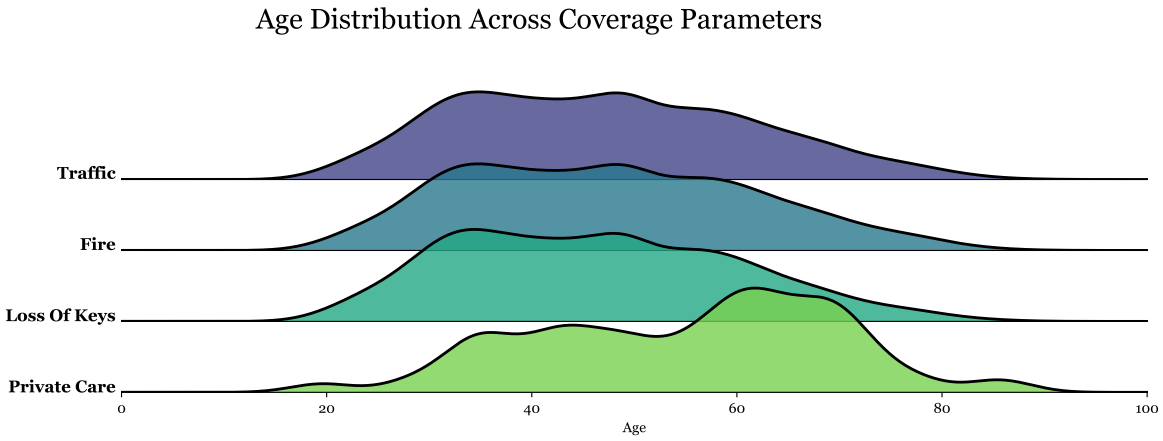


Figure 9: *Age distribution across different coverage parameters.*

Another essential feature is the consumers' premium, where the distributions of the normalized premium can be seen in Figure 10. The long right tale is due to some consumers spending way more than average. These outliers are luxurious car owners who generally pay a higher premium than the average consumer.

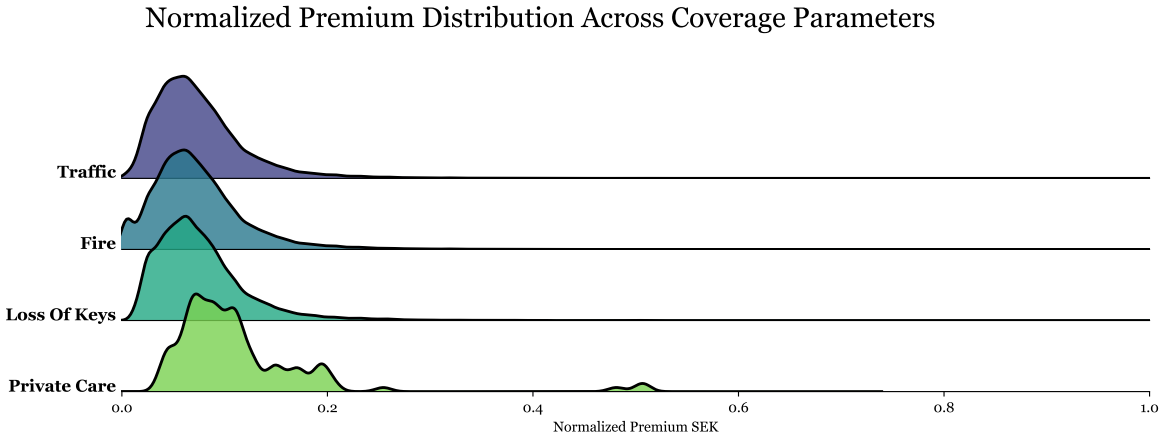


Figure 10: *Premium distribution across different coverage parameters.*

These were examples of two features for only a few coverage parameters, but the result is similar across all other parameters. The underlying product is complex, and more than individual features is needed to base recommendations on. Therefore, we must capture complex relationships in our data to make valid recommendations.

Figure 11 displays the pairwise linear correlation, also known as *Pearson correlation*, between a subset of the coverage parameters. As one can see, some of the parameters are highly correlated because insurance policies often come packaged with others. However, this thesis does not consider different packaging of coverage parameters since it allows us to remove the insurance companies from the picture and focus on the coverage provided.

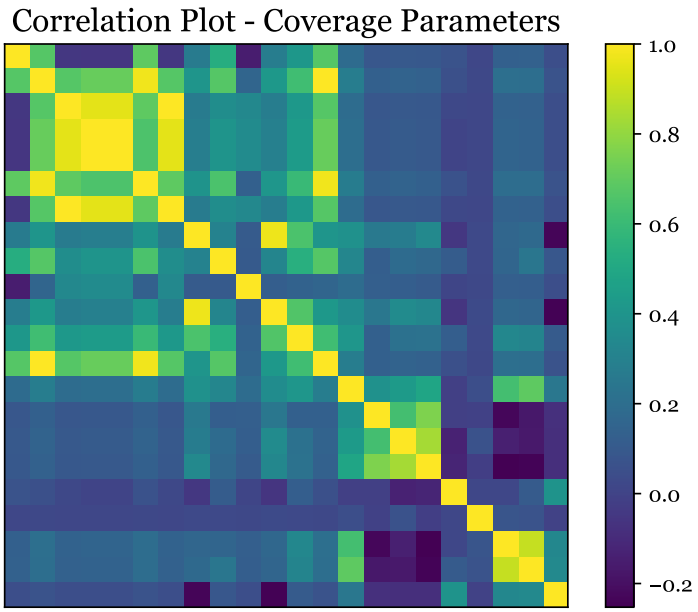


Figure 11: *Correlation plot using Pearson correlation between a subset of the coverage parameters.*

4.2 Pre-processing

Section 4.1 provided an overview the available data sources: consumer metadata and their current insurance coverage. However, before analyzing the data, the information must be pre-processed to ensure its quality and completeness. This section describes the pre-processing steps required to refine the data set.

Data exploration and cleaning allowed a better understanding of available information. The data set is organized as individual policies, with each stored instance representing when a consumer fetched their data. For example, if a consumer fetched their data multiple times, each retrieval was stored as a unique policy. Only the most recent collection from a consumer for each insurance company was considered to avoid duplicates of the same consumer.

The collected information on insurance policies exhibited significant variability regarding missing values. Therefore, we removed any policies with missing information. More recently collected policies also contain more information, making them more valuable for analysis.

4.3 Feature Engineering

The first step of feature engineering is feature selection, where we identify which features to keep and which to drop. Since the data is stored in a database, some features related to the storing process can be dropped without further investigation. Other features were dropped to avoid fully correlated features.

The most crucial features for car insurance is attributes regarding the insured product, which is the car. Car brands significantly influence the determination of suitable insurance coverage, as brand variations affect factors such as repair costs and the likelihood of mechanical damage. Engine size and fuel type are among the factors that can impact the frequency of car breakdowns, as evidenced by a study conducted by Länsförsäkringarna [16]. As a result, car brands, such as Porsche, tend to have a higher frequency of damages than others, such as Lexus. These differences between car brands underscore their importance in the context of insurance coverage

and the need to consider these factors when developing recommendation systems for insurance products.

The quality of the available car brand feature needs to be higher for its independent use. Consequently, a new feature, *brand value*, is constructed by examining the average premium associated with each brand and categorizing the results into three levels: cheap, intermediate, and luxurious. A visual representation of the feature can be seen in Figure 12. The bottom 20% is considered cheap, while the top 20% is considered luxurious. This proxy feature captures information about cars with similar costs, which is particularly relevant when selecting an insurance coverage plan. Although the new feature effectively represents general insurance costs for various brands, it inevitably leads to a simplification that obscures brand-specific nuances.

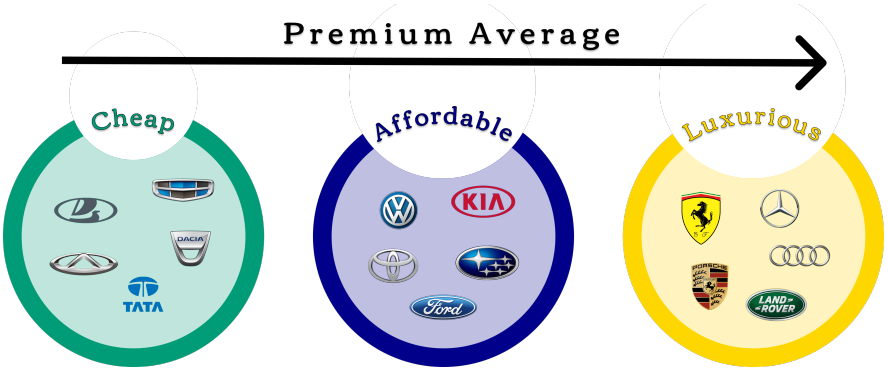


Figure 12: *A new feature is constructed to represent the brand value by looking at the average premium for each brand.*

The new proxy feature only has three dimensions, with one level for each price tier. With the original brand feature, which encompasses all brands and has higher dimensionality, it is reasonable to expect an increase in recommendation accuracy. Therefore, any recommendation system developed should account for the potential increased dimensionality due to utilizing a clean version of the original brand feature.

5 | Methodology

5.1 Problem definition

This thesis addresses a multilabel classification problem to develop a 'consumers like you'-feature. Predictions can thus be defined as $\hat{Y} \in \mathbb{R}^{n \times d}$, where $\hat{y}_{ij}$ is jth insurance parameter for the ith consumer. Figure 13 visually represents the system flow.

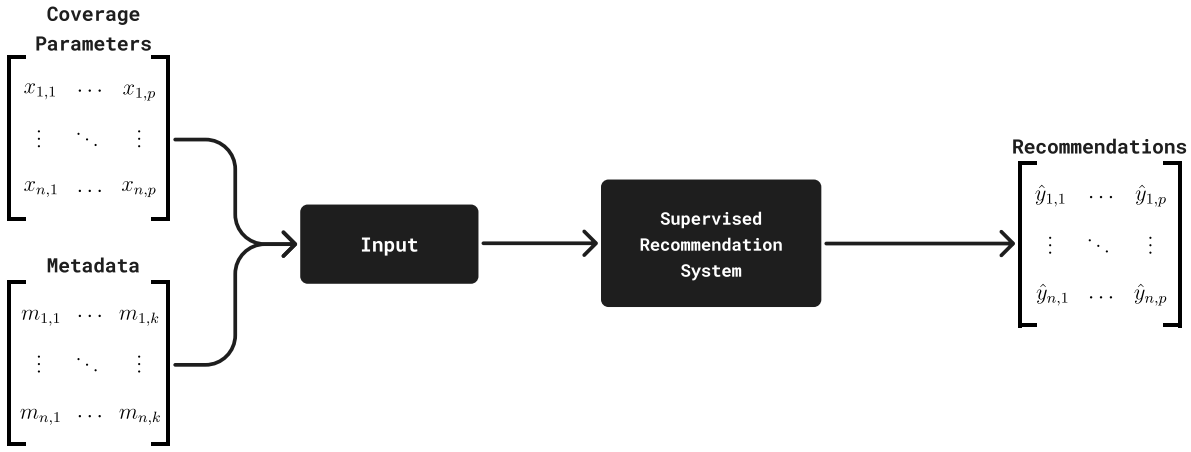


Figure 13: *Overview of the recommendation system flow.*

Typically, recommendation systems are designed for next-item prediction, which only considers the addition of items. In such cases, binary positive-only information would characterize a consumer's current coverage. In the context of recommendations, treating it as a supervised task often involves providing consumers with a recommendation and gathering feedback based on their response. However, online evaluation is necessary to receive this feedback. Online evaluation is expensive, which favors offline evaluation for conducting an initial exploration.

Not only is the primary objective up-selling in the conventional recommendation systems, but they usually deal with an abundance of items, resulting in a sparse interaction matrix. In such cases, one emphasizes positive interactions while interpreting a lack of interactions (negative information) as possible recommendations. However, in this thesis, we encounter a unique scenario with a limited number of items. Given this limitation, we assume that consumers are likely aware of most, if not all, items. The common practice of concealing an interaction to simulate consumer feedback is not feasible. It would fundamentally alter the consumer's profile, leading to inaccurate simulations and recommendations. Based on the premise that every interaction, whether possessing an item or not, carries significant information, it contributes to a comprehensive understanding of the consumer's preferences. Consequently, we cannot afford to ignore or hide any known interactions in our system.

This thesis introduces a cluster-based simulation of consumer behavior to generate pseudo-labels

for assessing the recommendation system's performance. By examining the average insurance coverage within a specific cluster, the system aims to help individuals achieve insurance coverage similar to others with comparable profiles. This clustering approach is an unsupervised method of constructing the 'consumers like you'-feature. Suppose we have k clusters denoted by $C = \{C_1, C_2, \ldots, C_k\}$, where C_j represents the consumers in the jth cluster. The clustering process entails assigning each consumer to a cluster j, where $1 \leq j \leq k$. For each cluster C_j, the average coverage parameters are calculated by

$$\bar{\mathbf{x}}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i$$

where $|C_i|$ is the cardinality of C_i and $\mathbf{x}_i$ is the ith consumers current coverage. The elements of $\bar{\mathbf{x}}_i$ are rounded to obtain a binary representation and can be interpreted as a label for each cluster. Since labels are derived from the average coverage, the label is not unique, and several clusters can end up with the same average coverage.

For each consumer i, we assign the average coverage parameters of their corresponding cluster as their pseudo-label $\widetilde{y}_i = \bar{\mathbf{x}}_j$. In other words, $\widetilde{y}_i$ represents the pseudo-label for the ith consumer and is defined by the average coverage parameters from the consumer's corresponding cluster. Figure 14 displays the connection between the supervised and unsupervised parts of the recommendation system. We feed the pseudo-labels obtained from our clustering back to the supervised recommendation systems.

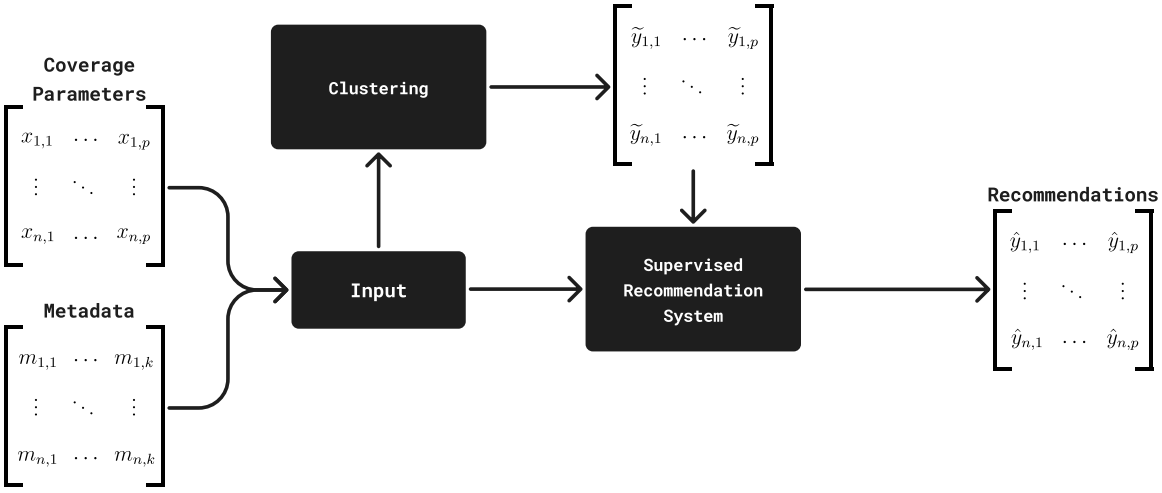


Figure 14: *Illustration of how the simulation framework ties into the recommendation flow.*

It is essential to acknowledge the prevalence of unsupervised recommendation systems in practice. Consequently, one may question the necessity of a supervised approach when the unsupervised method could be employed exclusively for generating recommendations. Nonetheless, supervised models have distinct advantages, primarily attributable to their superior predictive performance [8]. The performance disparity arises from the fact that supervised models are explicitly designed to optimize predictions by utilizing labeled observations, while unsupervised methods lack such explicit optimization objectives. Predictive performance is paramount in our context, primarily driven by the nature of the underlying product - insurance. Insurance entails complex and critical considerations, where an erroneous recommendation could have severe ramifications.

Moreover, we must emphasize that the absence of labels in our problem domain is limited to the offline stage. In a deployed recommendation system, gathering feedback is possible and can

be utilized to enhance the performance of the supervised system. Although an unsupervised approach also could benefit from consumer feedback, one can anticipate the predictive performance of a supervised approach to outperform, as it explicitly optimizes model parameters based on the labeled data. Consequently, if our supervised models can demonstrate comparable performance to the unsupervised approach, employing them from the outset is justifiable, leveraging their ability to optimize predictions based on labeled data and thereby maximizing predictive accuracy.

5.2 Clustering-based Recommendation System

The assessment of cluster quality in this study deviated from conventional cluster analysis. We did use conventional methods such as Scree plots and Silhouette analysis to tune the clustering. Since the number of clusters determined the granularity of identifiable consumer types, finding a set of highly distinguishable clusters may not capture enough consumer types. Consequently, we utilize custom metrics for our domain to evaluate the clustering further. We ensure recommendations only make reasonable big changes to a consumer's coverage. For instance, if recommendations often suggest altering 25% of a consumer's policy, it may indicate that the clustering needs to be more granular. Another important metric is the number of consumers who suggested changing their coverage.

5.2.1 Dimensionality Reduction

The similarity metric is one of the most critical decisions during clustering analysis. In the case of the binary coverage parameters, both positive and negative information carry equal value. Consequently, SMC, see Equation (3.1.3), is an appropriate way to measure similarity. However, dealing with two distinct sources of information to describe consumers raise additional concern. For example, if one measures the two sources independently, how should they be combined into a single measure?

We obtained a compressed latent representation of the initial data using an autoencoder. This transformation allowed us to circumvent the curse of dimensionality, where we have data sparsely scattered in a high dimensional space. In addition, the original feature was transformed into a numerical representation, which allowed for Euclidean distance as our chosen distance metric. While a simple statistical measurement such as SMC offers simplicity and interpretability advantages, it must still produce valuable results. This study has no obvious choice in similarity metric if one seeks to use the original data. Instead, the numerical representation obtained from the autoencoder solves the problem with mixed data types and the concern with our data coming from two different sources.

The transformation introduced complexity, necessitating an analysis to verify that the initial data sets' characteristics were preserved in the new latent representation. To achieve this, we employed another dimensionality reduction technique called t-SNE on the latent space, which enabled visualization of the transformed data. By examining the visualizations, we ensured that the information from the original features had been successfully retained.

5.2.2 K-means

One of the key requirements for the recommendation system is the ability to assign new instances to a cluster in real time. K-means clustering is a suitable algorithm for this purpose, as it utilizes a set of cluster centroids that can be updated incrementally as new data becomes available.

K-means is also advantageous in terms of scalability, as it can handle many observations. However, it is essential to note how the K-means algorithm is sensitive to the initialization of centroids, which can affect the quality of the clusters [14]. We ran the algorithm five times to address this issue and kept the set of clusters with the highest silhouette score.

5.3 Supervised Recommendation Systems

Three alternatives for supervised approaches were explored and assessed regarding their potential to replace the cluster-based recommendation system. The first approach was a neighborhood model, specifically the KNN algorithm. The algorithm is similar to the clustering approach, where we identify similar consumers and base our prediction on them. The second approach was a tree-based approach called XGBoost, which makes decisions similar to human decision-making. The third and final approach was an Artificial Neural Network (ANN) aimed at creating a simplified model of the human biological neural networks. The ability to replicate the cluster-based recommendation system was measured with EMR, see Equation (3.1.3).

5.3.1 KNN

The KNN algorithm make predictions based on similar consumers. Consequently, one needs to decide on a similarity metric. We utilized the same approach as we used for the cluster-based recommendation system, where an autoencoder was applied and then the Euclidean distance. Working in a lower dimension is preferable since the algorithm can struggle in high-dimensional data.

We tuned the hyperparameter K through 5-fold cross-validation. It was essential to find a balance, as few neighbors can lead to underfitting while too many can lead to overfitting. Additionally, a heatmap was used to investigate cases where recommendations were incorrect.

5.3.2 XGBoost

XGBoost is a model-based approach and a more complex alternative than the KNN algorithm, but the implementation of XGBoost is straightforward with the use of the XGBoost package in Python [5]. In the context of this thesis, XGBoost was used as a tree-based approach to create a recommendation system. The approach was of interest due to the underlying data as well as the recent success during a Kaggle competition to construct a multi-objective recommendation system featuring XGBoost in many top solutions. [1]

The data required some pre-processing before being fed to the XGBoost model. One-hot encoding was applied to the categorical metadata, followed by horizontal concatenation to combine the metadata and coverage parameters. Figure 15 illustrates how the XGBoost-based recommendation system handled the multi-label problem by creating a separate model for each label. Since predictions are binary (either 0 for absence or 1 for presence), the objective function used for all models was binary logistic regression.

[1] The Multi-Objective Recommender System Competition hosted early 2023 can be found on Kaggle at: https://www.kaggle.com/competitions/otto-recommender-system/overview

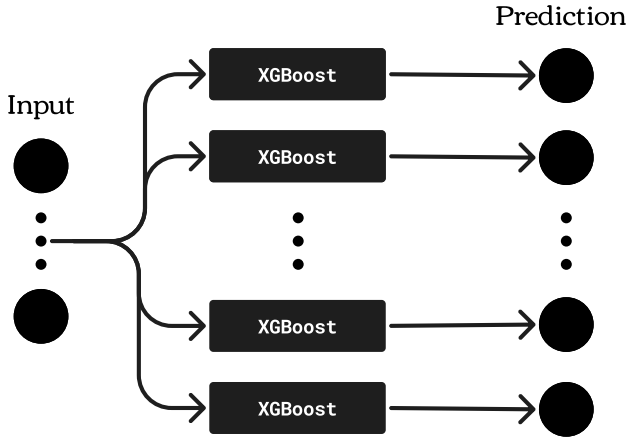


Figure 15: *Illustration of the XGBoost-based recommendation system. Each label is assigned a separate XGBoost model.*

5.3.3 Artificial Neural Network

The final approach to building a recommendation system was based on deep learning by using a fully dense neural network. One alternative would have been to use a dummy metadata encoding and concatenate the transformation with the binary coverage parameters, allowing the input to be fed directly to an ANN. The simplicity of this method is advantageous, but the drawbacks of this design outweigh the benefits.

Dummy encoding leads to an increase in dimensionality, resulting in higher memory usage and longer training times. The resulting sparseness of the encoding also results in most entries being zero, which could pose a challenge when training an ANN. Another challenge is ensuring the network can learn the inherent differences between levels in the categorical features. We helped the ANN to identify the categorical features by assigning a separate embedding layer to each. Categorical embeddings contain weights used to learn how to represent these features better. In theory, given unlimited time and resources, using dummy encoding to feed data into an ANN should yield comparable results. However, assuming training does not get stuck in a local minimum, it would also take longer for the simpler design to converge. The architecture can be seen in Figure 16.

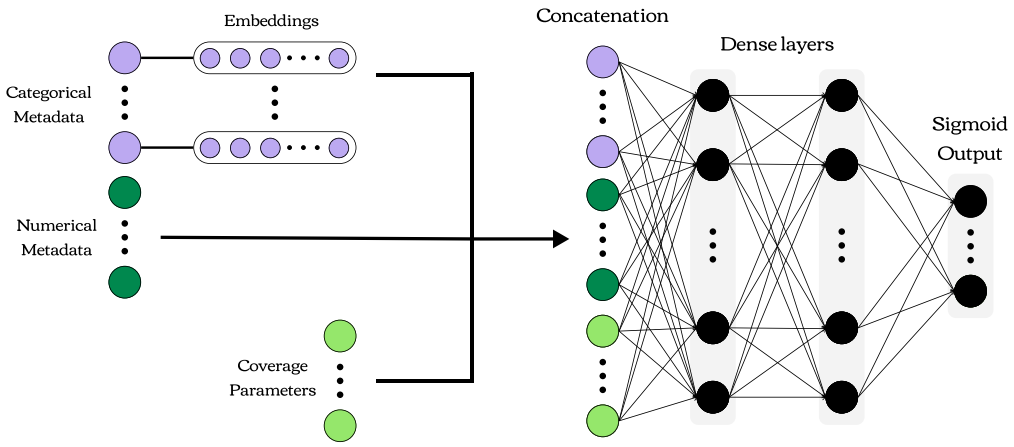


Figure 16: *Illustration of the network architecture. Each categorical feature is individually processed through distinct embedding layers. The embedded features are concatenated with the rest of the input and passed through two dense hidden layers.*

5.4 Software

The data set was stored in a data warehouse where SQL was employed for data extraction and manipulation. All programming was conducted using Python, where the following were the most heavily utilized libraries for this study:

- NumPY: Utilized for matrix operations [10]. Extended upon with the Numba package, which is used to lower runtime by translating functions into machine code [15].
- Pandas: Applied for data manipulation and data analysis [19].
- Matplotlib: Data visualization [12].
- Tensorflow: Implementation of ANN [1].

Local machines were used primarily with a single CPU, including data pre-processing to model exploration. Any additional GPU power was obtained through cloud computing.

6 | Result

This section presents the results of creating a cluster-based recommendation system and the supervised alternatives.

6.1 Simulation Framework

Figure 17 displays a non-directional graph representing the consumers within the data set. It is generated by calculating the Euclidean distance between consumers based on the latent representation obtained from the autoencoder. The resulting graph reveals a highly complex network characterized by numerous distinct neighborhoods. This intricacy underscores the need for many clusters to accurately capture the variety of consumer types within the data set.

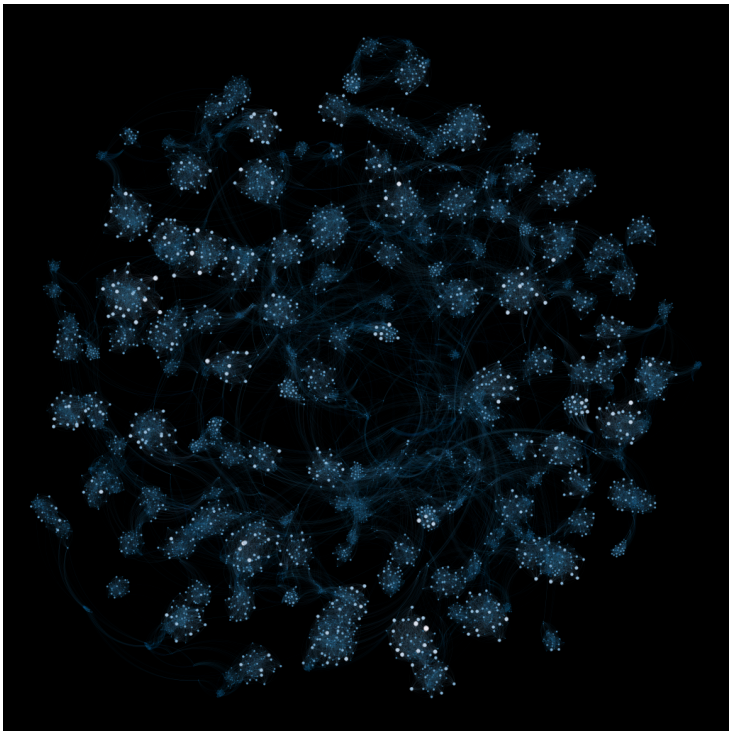


Figure 17: *Graph representation of the transformed data. Each node is connected to its 25 nearest neighbors, where size is related to the number of attached edges to a given node.*

Figure 18 displays the results of t-SNE dimensionality reduction with perplexity 50. We aim to ascertain whether the new latent representation preserves the characteristics of the original features.

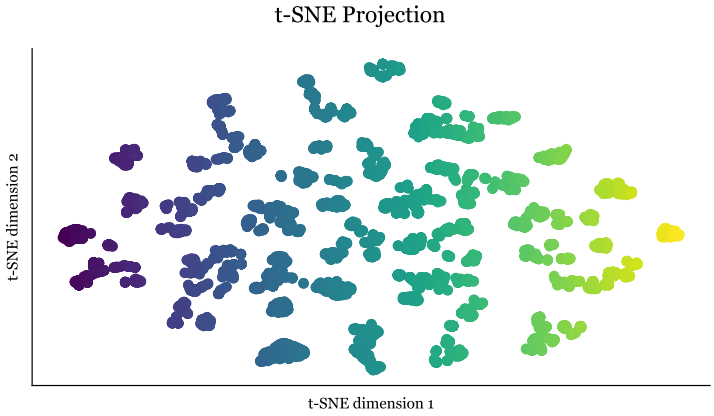


Figure 18: *Visualization of the latent vector space obtained from the autoencoder through t-SNE.*

Figure 19 employs the original feature regarding the Urban-Rural typology [1] of the consumer to color the new latent representation. The presence of distinct groupings suggests that the new representation effectively retains the original complexity. However, not all features lend themselves to clear interpretation. For example, the original feature *age* and the *gender* resulted in less distinct groupings. However, this observation does not necessarily warrant concern, as groups within groups may require some features to appear in multiple locations.

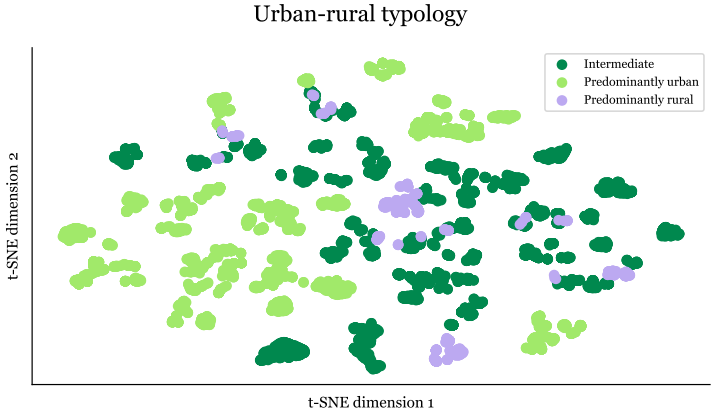


Figure 19: *Visualization of the latent vector space through t-SNE. Colors correspond to the value of the original feature regarding Urban-Rural topology. As clear groupings can be seen, it is evidence how the new representation is able to retain information from the original features.*

6.1.1 K-Means

Figure 20 contains two metrics, the inertia (left) and silhouette score (right). The silhouette score reaches a value of 0.5 at 90 clusters, suggesting the optimal number of clusters for distinguishable clusters. The inertia displays an exponential decrease and looks to plan out near 120 clusters.

[1] The Urban-Rural feature offer insights into the NUTS-3 region's urbanization level and rural characteristics.

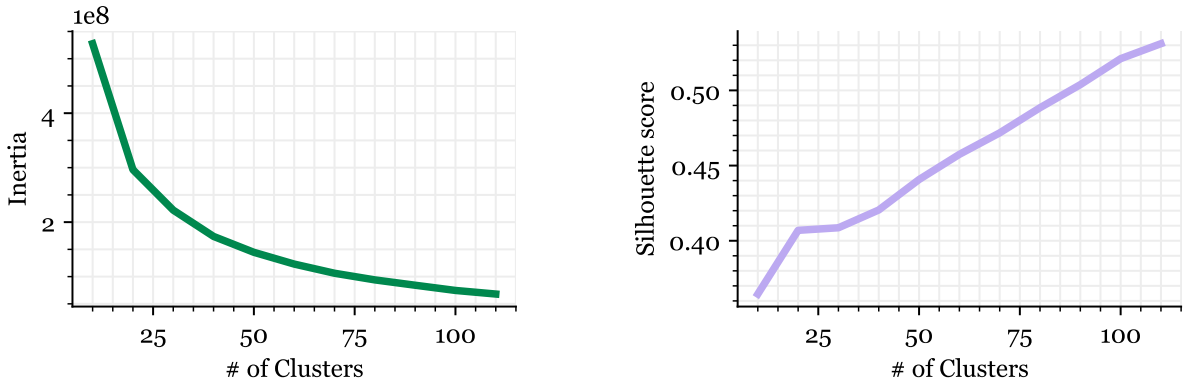


Figure 20: *Metrics to assess cluster quality: inertia (left) and silhouette score (right).*

Figure 21 shows the percentage of consumers who should maintain their current coverage depending on the number of clusters. The K-means algorithm captures a more granular data representation as the number of clusters increases. The presence of insufficient clusters results in more than half of all consumers being identified as having incorrect coverage. Increasing the number of clusters leads to the attainment of more accurate representations. The graph's trajectory suggests that it reaches a maximum of 80% and plateaus at approximately 130 clusters.

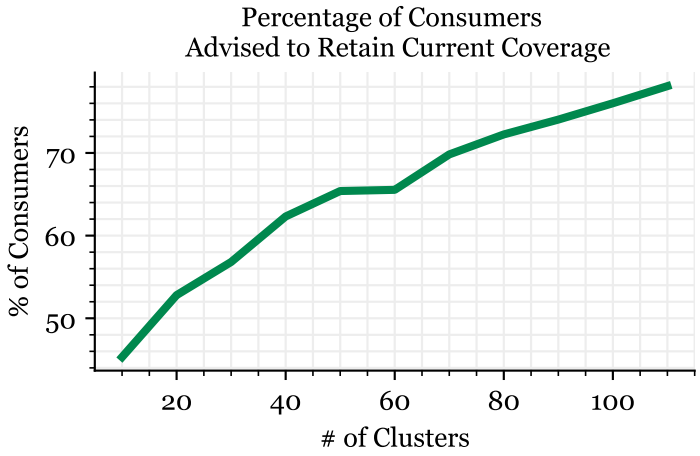


Figure 21: *The figure displays the percentage of consumers recommended to update their current coverage. An increase in granularity (more clusters) results in fewer consumers recommending changing their current coverage. However, the trajectory suggests the percentage stagnates at 80%.*

In Figure 22, we present what percentage of consumers' current coverage aligns with the recommendations provided. To maintain the practicality of recommendations, they should not propose significantly different coverage than a consumer already possesses. The green line illustrates the percentage of consumers whose recommendations match at least 80% of their current coverage. It seems that 10 clusters are enough to make recommendations on this level since more clusters only slightly increase the percentage. In contrast, the purple line represents a 90% match and indicates to peak at 90 clusters.

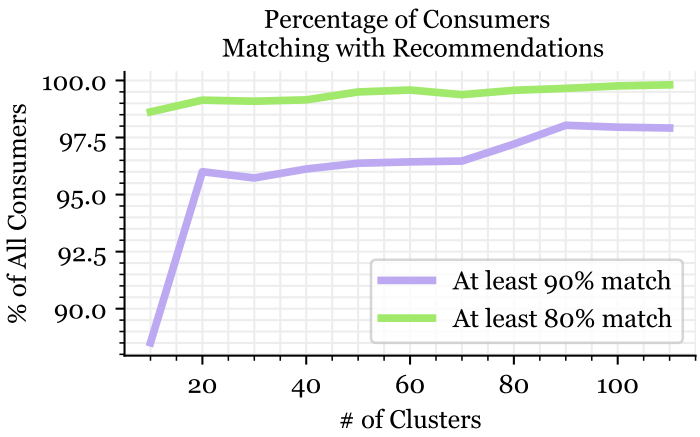


Figure 22: *The figure shows how many consumers must update their coverage over a certain percentage to match the recommendation. In other words, at least 90% of the consumers' current coverage matches the recommendation.*

6.1.2 Investigation of Recommendations

Since 90 clusters achieved a silhouette score of 0.5, we used it as the final choice for the number of clusters. The underlying complexity behind the data set and the reason why no less than 90 clusters were able to achieve an acceptable performance can be hard to grasp. Figure 23 shows a non-directional graph representation of the underlying data, where each node represents a consumer connected to its 25 nearest neighbors. The colors represent the different clusters obtained from the k-means algorithm.

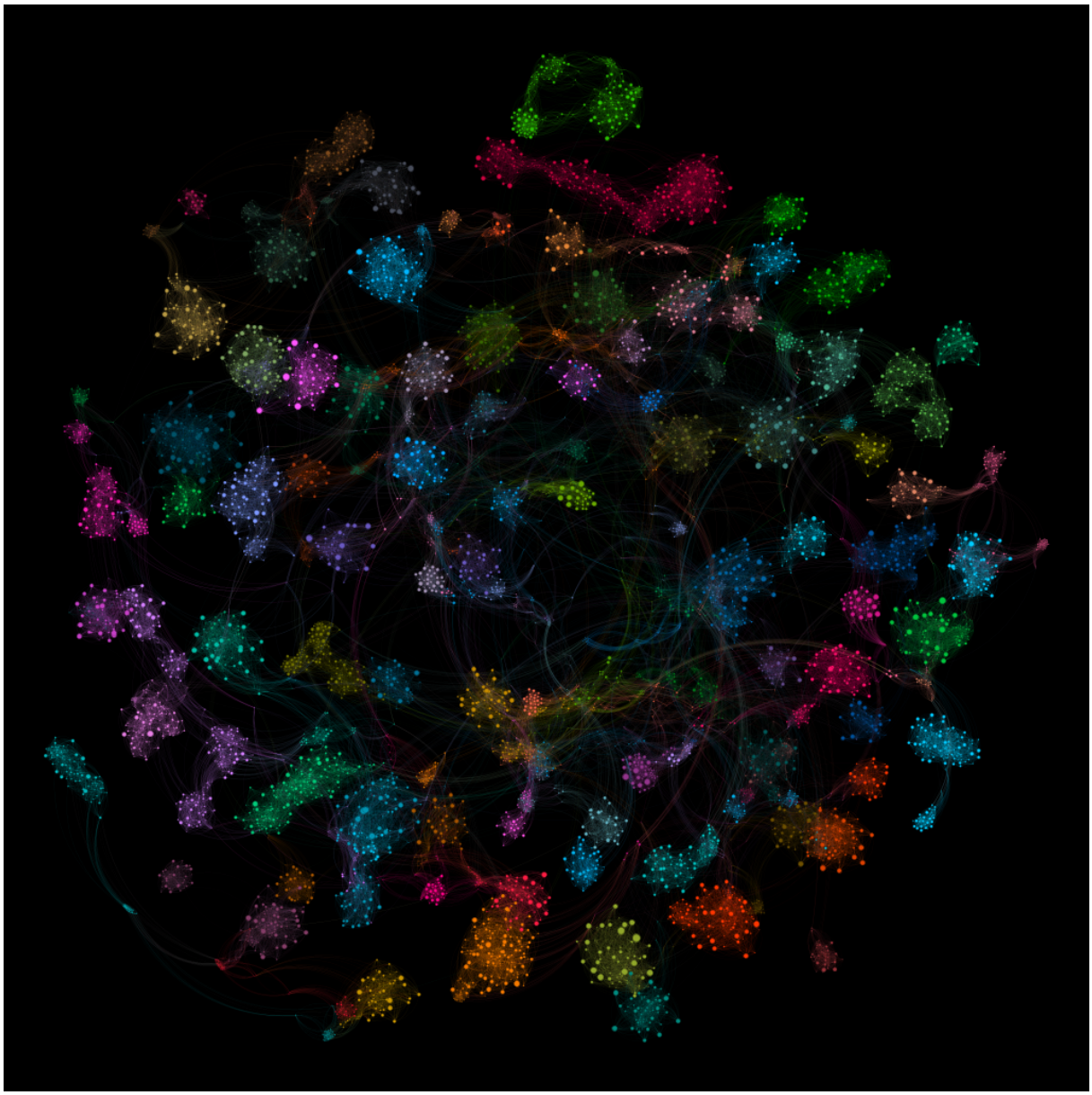


Figure 23: *Graph representation of the transformed data with coloring based on K-means clustering for 90 clusters. Each node is connected to its 25 nearest neighbor, where size is related to the number of attached edges to a given node.*

The similarities between consumers are based on the pairwise Euclidean distance on the latent features obtained from the autoencoder. The resulting clusters from the K-means algorithms are only used to color the graph and have no effect on the structure. The key takeaway from Figure 23 is to understand the complexity in trying to determine the optimal number of clusters.

Clustering consumers is a good way of creating the 'consumers like you'-feature. However, the recommendation should still be sensible. The actual recommendations should be investigated to highlight the limitations of the clustering approach. Based on 90 clusters, Figure 24 show the recommendations made. The coverage parameters are coded. However, we can obtain several interesting takeaways from a manual inspection of the recommendations.

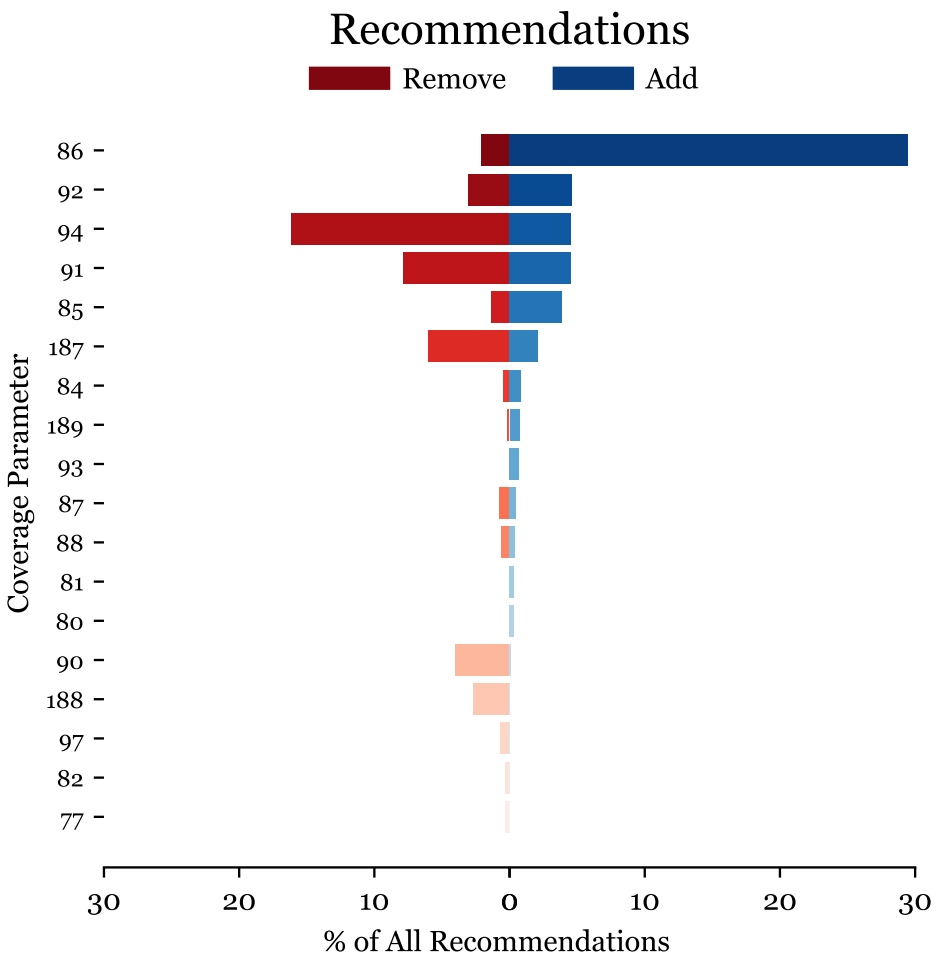


Figure 24: *Count of the actual recommendations made for each coverage parameter. The most likely parameter to be recommended is parameter 86 which is coverage of damages on your own car.*

The most common recommendation to add is coverage parameter 86, which concerns coverage of damages to your car. The interpretation of this recommendation is that many consumers appear only to be traffic insured, whereas others like them also have their cars insured.

Some limitations of the clustering approach can also be highlighted by analyzing the individual recommendations. For instance, coverage parameter 97 is an add-on regarding private healthcare in an emergency. It is an uncommon coverage parameter that around only 0.25% of consumers have. However, the recommendations suggested that all these consumers should remove it. These consumers may have been assigned their cluster, but either the granularity was too high, or the cluster was not found. Another problematic recommendation is that a few consumers were recommended to remove parameter 77, which is traffic insurance required by law. Recommendations are not based on enough features to make this type of recommendation.

6.2 KNN

Figure 25 shows the exact match ratio for different numbers of neighbors for the KNN algorithm when trying to recreate the clustering-based recommendation system. It indicates an optimal performance when considering only a few neighbors, somewhere around five neighbors.

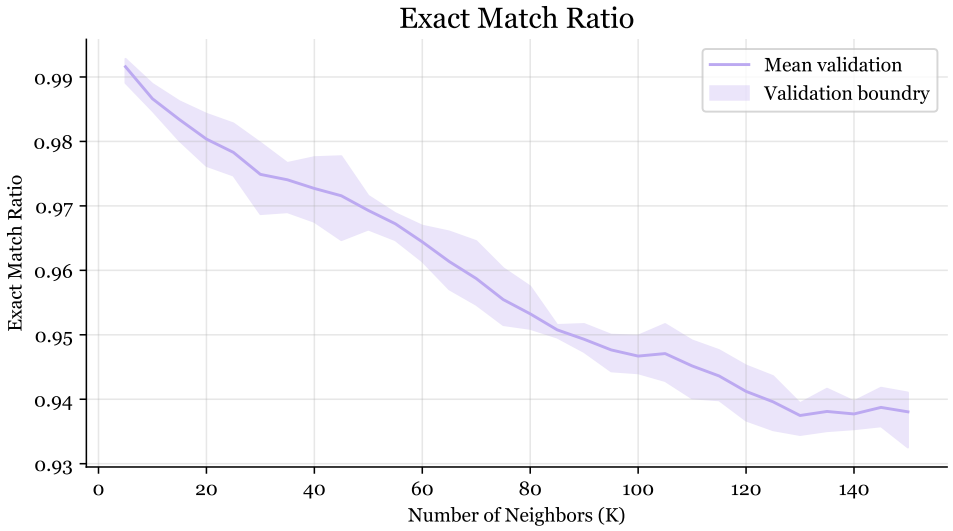


Figure 25: *Exact match ratio for different numbers of neighbors for KNN. The validation boundary shows the worst and best EMR obtained across all validation folds*

In Figure 26, the heatmap presents a detailed analysis of the K parameter's impact on the KNN algorithm, focusing on the accuracy of misclassified observations. The data displayed is drawn from an arbitrary validation fold and concentrates on cases with fewer than ten neighbors. One notable limitation of the KNN algorithm is the risk of overfitting when using a low value for K. This issue is evident in the initial rows of the heatmap, where the performance improves as more neighbors are considered. Conversely, underfitting occurs when an excessive number of neighbors are considered, leading to a decreased accuracy for observations. This effect is also visible in the heatmap, with several instances demonstrating lower accuracy as neighbors increase.

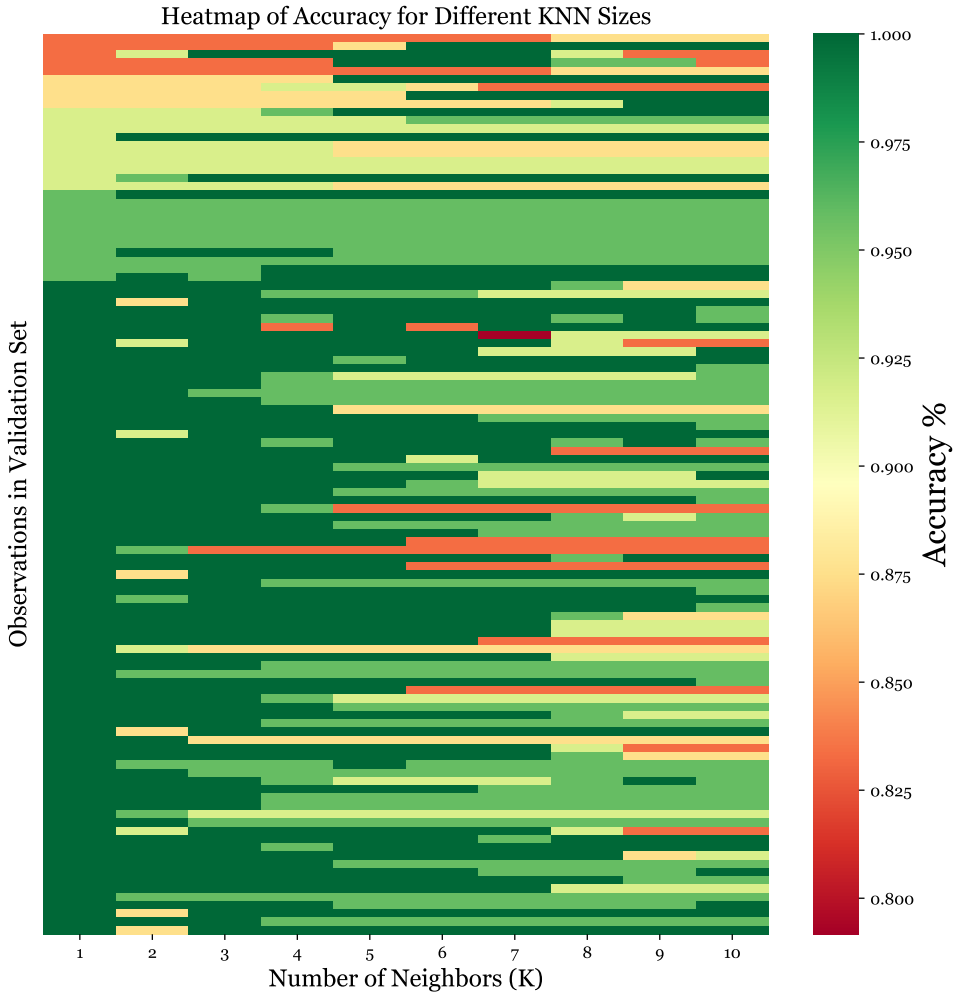


Figure 26: *Heatmap of accuracy for one of the validation sets obtained from 5-fold validation. Observations which obtained 100% accuracy across all sizes have been removed.*

K was set to 5 for the final model to avoid overfitting and underfitting. With $K = 5$, the KNN algorithm obtained an exact matching ratio of 96.9% on the test set. The result would suggest that the neighborhood model can replicate the cluster-based recommendation system.

Figure 27 display the label-wise misclassifications for the 10 coverage parameters with highest miclassification rate from the KNN. *False positives* represent inaccurate suggestions to include a specific coverage parameter, while *false negatives* correspond to incorrect predictions advocating for the removal of one. The result indicates possible failure over a wide range of coverage parameters. Three coverage parameters appear to be especially difficult to classify: deductible discount animal collision, vehicle damage, and crisis. The results show a higher amount of false positives, which results in consumers being overinsured if they follow the recommendations.

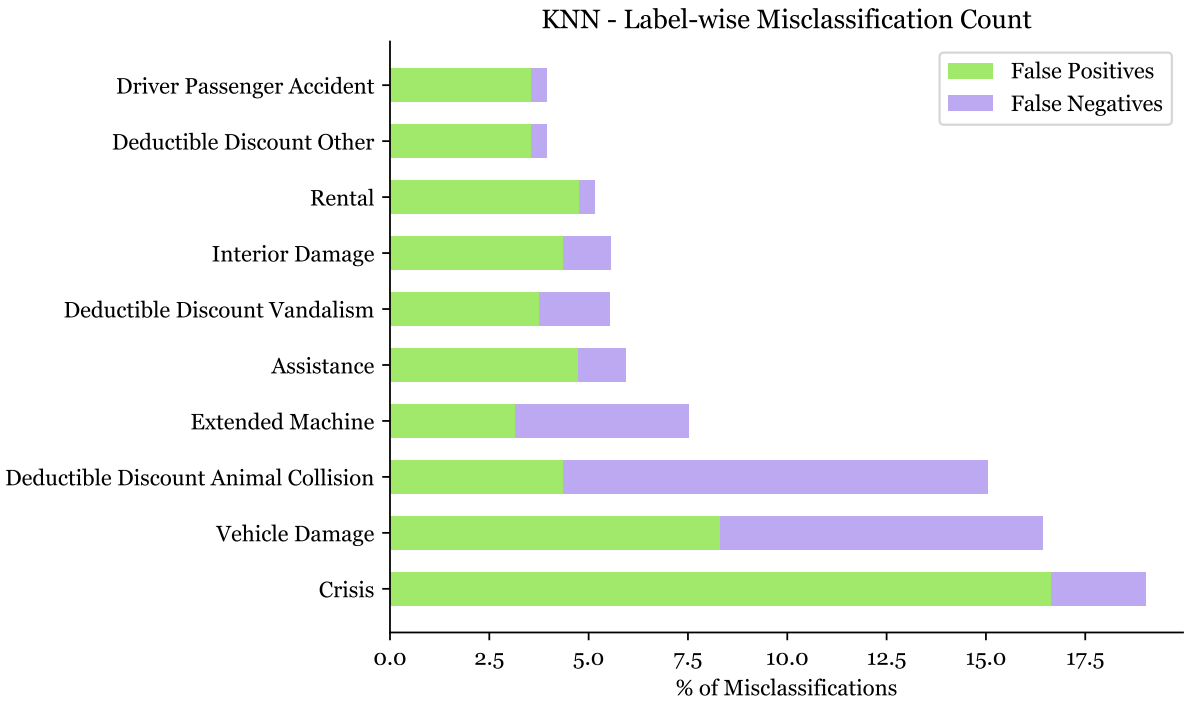


Figure 27: *The label-wise misclassification count for the KNN-based recommendation system. The figure display the 10 coverage parameters with highest misclassification rate.*

Figure 28 display the number of misclassified labels for each consumer. When the algorithm fails, it appears the algorithm can fail quite badly. To change six coverage parameters is equivalent to a substantial amount of the consumers current policy.

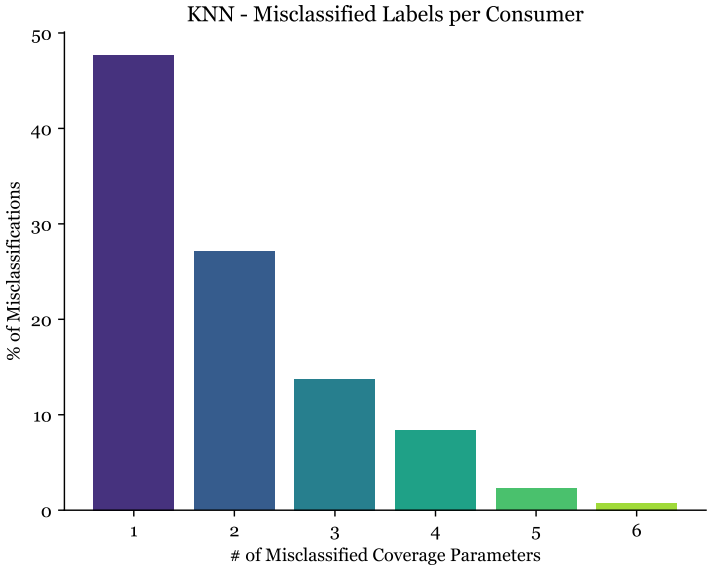


Figure 28: *Distribution of the number of wrong predictions for each consumer from the KNN-based recommendation system.*

6.3 XGBoost

The XGBoost-based recommendation system obtained an exact matching ratio of 99.4% on the test set. The result suggests that the model can almost perfectly replicate the cluster-based recommendation system.

Figure 29 display the label-wise misclassifications for the 10 coverage parameters with highest miclassification rate from the XGBoost model. The top three coverage parameters in terms of difficulty appear to be the same as for the KNN. The results also shows how false negatives are more common than false positives.

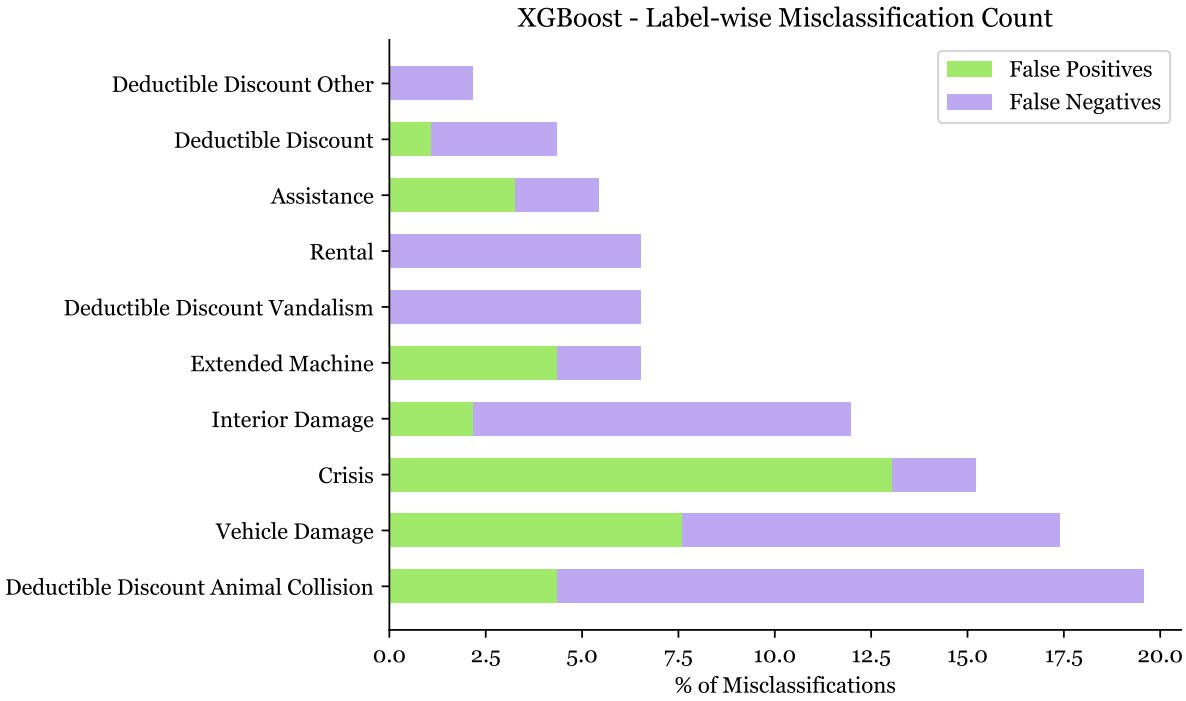


Figure 29: *The label-wise misclassification count for the XGBoost-based recommendation system. The figure display the 10 coverage parameters with highest misclassification rate.*

Figure 30 display the number of misclassified labels for each consumer from the XGBoost model. When the model fails, the result suggests it will not fail as badly as the KNN algorithm on average.

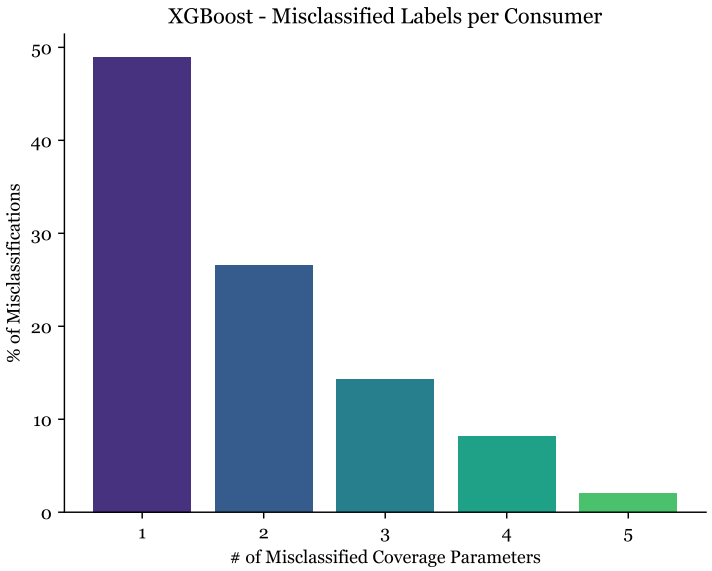


Figure 30: *Distribution of the number of wrong predictions for each consumer from the XGBoost-based recommendation system.*

6.4 Artificial Neural Network

The ANN obtained an exact matching ratio of 99.3% on the test set. Thus, the model could replicate the clustering-based recommendation system to a very high degree, similar to the XGBoost model. Figure 31 displays the label-wise misclassifications for the 10 coverage parameters with highest miclassification rate from the ANN. The results suggest a fair balanced ratio between false positives and false negatives.

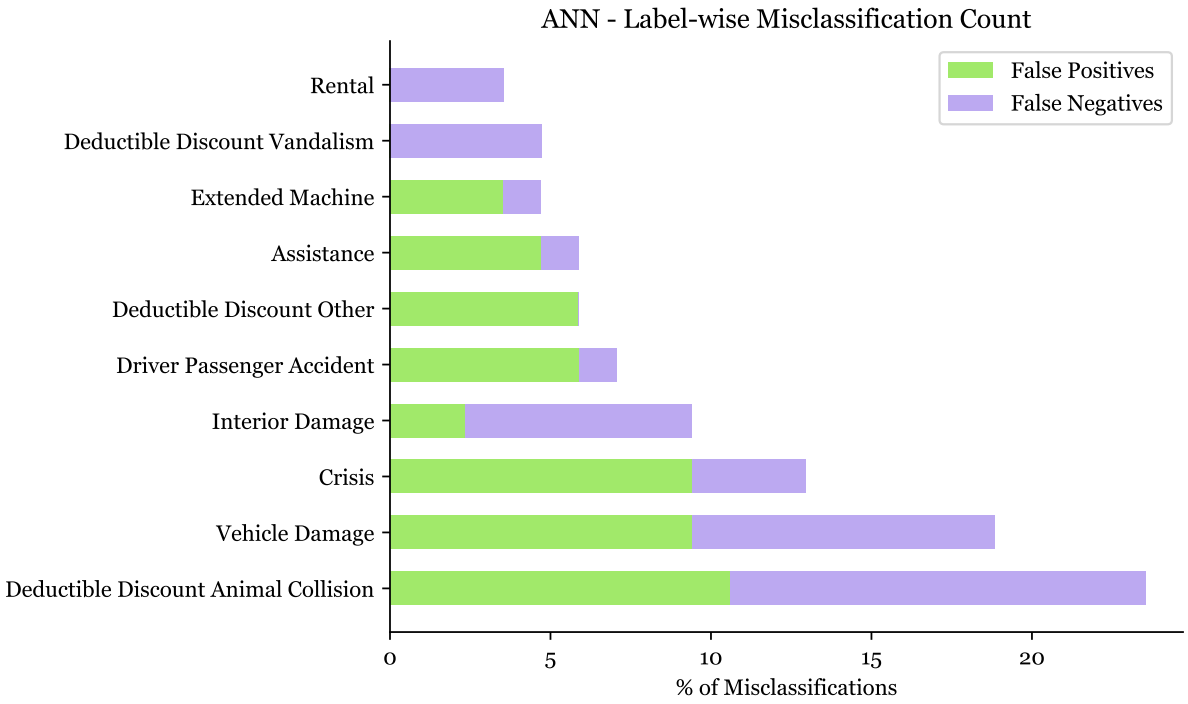


Figure 31: *The label-wise misclassification count for the ANN-based recommendation system. The figure display the 10 coverage parameters with highest misclassification rate.*

Figure 32 display the number of misclassified labels for each consumer from the ANN. When the model fails, the result suggests it will not fail as badly as the KNN algorithm. The distribution also differs from the XGBoost model, where the ANN exhibits a steeper exponential decrease. This would suggest that the ANN would in general not fail as badly as the XGBoost model.

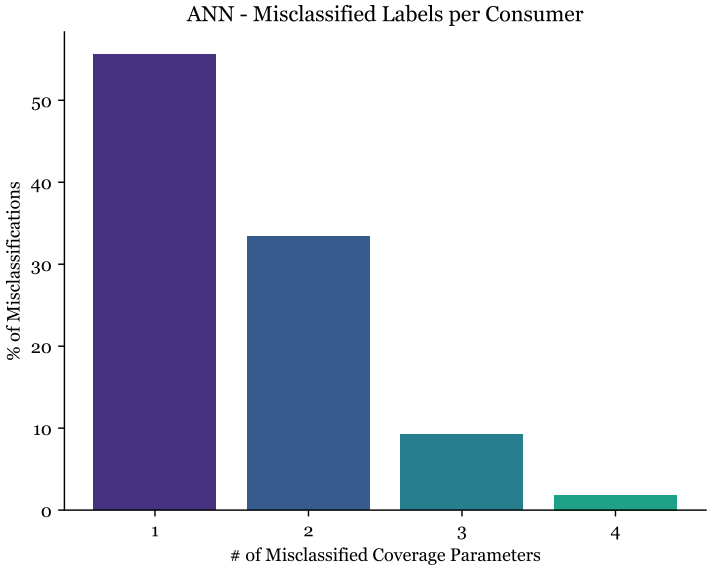


Figure 32: *Distribution of the number of wrong predictions for each consumer from the ANN-based recommendation system.*

6.5 Comparison between Supervised Recommendation Systems

Table 6.1 shows the EMR for all three supervised recommendation systems. Both XGBoost and ANN achieved an above 99% EMR, which suggests they are highly capable of replicating the cluster-based recommendation system. The KNN algorithm falls slightly behind with an EMR of 96.9%, which still suggests it can replicate the unsupervised approach but not to the same degree.

Table 6.1: *Exact Match Ratio (EMR) for all three supervised recommendation systems in their ability to replicate the cluster-based recommendation system.*

Method	KNN	XGBoost	ANN
EMR	96.9%	99.4%	99.3%

Figure 33 provides a direct comparison of the number of misclassified labels per consumer between the three supervised methods. Looking at when the supervised recommendation systems failed, the KNN-based alternative failed worse than the other two.

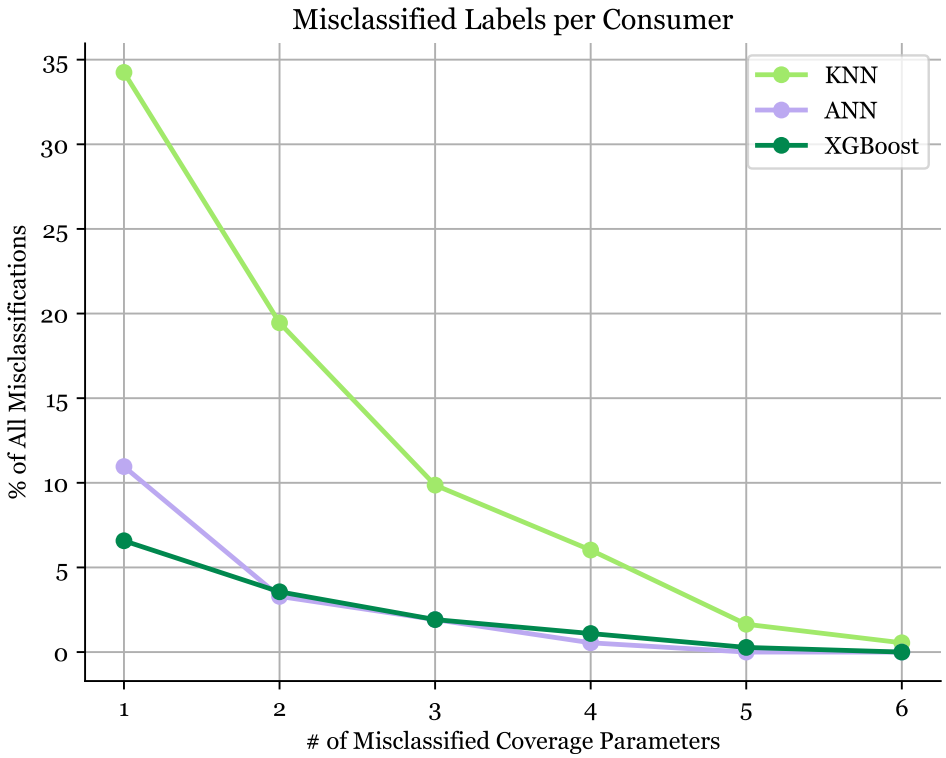


Figure 33: *The percentage of all faulty recommendations over the number of misclassified labels. When the recommendation system fails, the KNN algorithm will generally fail worse than the other two supervised alternatives.*

7 | Discussion

This section discusses the clustering-based recommendation system and the potential replacement with a supervised alternative.

The primary objective of this thesis is to investigate the development of a recommendation system for insurance policies. However, we must recognize how identifying a universally superior method is impossible in an offline setting. Nevertheless, we can still pinpoint several candidate models that warrant further examination. Let us begin the discussion by focusing on the clustering-based recommendation system, which supplied the pseudo labels utilized to construct the supervised approach.

7.1 Clustering-Based Recommendation System

The clustering-based recommendation system demonstrates potential in developing a 'consumers like you'-feature. However, the study also exposes the inherent complexity of the data set. Capturing various consumer types necessitates a large number of clusters, and the granularity of our recommendations relies on the number of clusters. A significant challenge lies in determining and verifying the optimal number of clusters in such an environment. The graph representation of consumers in Figure 23 helps illustrate consumer complexity. The network contains several densely connected neighborhoods of consumers, indicating that the optimal number of clusters is ambiguous. Data complexity impacts the clarity of clustering methods, where evaluating if individual clusters are meaningful is challenging.

To improve recommendations, granularity can be increased by using more clusters. This study found 90 clusters suitable due to a satisfactory silhouette score. Nevertheless, there might be merit in considering 130 clusters, as this was when the number of customers who should retain their existing coverage stabilized. Notably, the silhouette score alone cannot identify the optimum cluster count.

Despite its challenges, the clustering-based recommendation system serves as a robust base for personalizing insurance recommendations. Incorporating consumer feedback via an app or sales agent can improve the system. Recommendations are educational, not purely sales-driven, so a negative consumer response does not imply a flawed recommendation. However, persistent negative responses suggest the need for improvement. One method to incorporate feedback is by updating cluster labels, though it may oversimplify and degrade cluster quality. Alternatively, reclustering post-feedback could be considered, but more is needed to resolve the issue of selecting an optimal number of clusters or effective feedback integration.

7.2 Supervised Recommendation Systems

This thesis has investigated three distinct supervised models to replace the unsupervised approach. We will briefly discuss each supervised approach's results and then proceed to a more

comprehensive comparison.

7.2.1 Predictive Performance

The KNN-based recommendation system effectively mirrored the clustering-based system with an exact match ratio exceeding 97%. Achieving a similar ratio to other supervised models necessitates around five neighbors (K=5), though this low value risks overfitting. This hyperparameter can be adjusted with more data.

Both the XGBoost and ANN obtained an exact match ratio of above 99%. This means the neighborhood model falls short in terms of accuracy compared to the other two supervised models. The higher predictive performance of the XGBoost and ANN indicates the importance of expressive power and capturing more complex relationships.

While the focus remains on minimizing misclassifications, limiting the extent of erroneous recommendations when they do occur is also important. The findings demonstrate that in the event of recommendation system failure, the KNN model tends to falter more severely than the XGBoost and ANN models. Upon model failure, XGBoost and ANN display similar performances. The ANN misclassified more consumers than the XGBoost model, but these misclassifications commonly involved only a single coverage parameter.

Our study also encompassed an analysis of false positives and negatives, wherein the ANN model displayed a balanced distribution of both. KNN produced more false positives, incorrectly recommending unneeded coverage parameters. This could be detrimental to consumer satisfaction despite not being a significant concern from the insurer's perspective. Conversely, XGBoost resulted in a higher rate of false negatives, risking instances of underinsured consumers. However, the overall low misclassification rate of XGBoost may mitigate this concern.

7.2.2 Prediction time

The KNN approach primary distinction is that it is a memory-based method wherein all calculations are executed during the prediction stage. Depending on the system's implementation, it may constrain the allowed response time, and prediction time could pose a problem. In comparison, both the XGBoost and ANN are model-based approaches. While their computational demands still depend on the model size, the size required to achieve an EMR of over 99% still makes these models the clear winners in prediction time.

7.2.3 Interpretability

In our domain, it is more important to consider the interpretability of the actual recommendation rather than looking at feature importance. Even if the XGBoost model and ANN can provide feature importance, it does not provide information behind individual recommendations. The cluster-based recommendation system and the KNN algorithm allow extracting information from the consumers on whom the recommendation is based. This could prove helpful since recommendations can be complemented with information to provide trust and transparency. For example, one could phrase a recommendation to specify how the other consumers are similar. The value to be gained here is situational since the knowledge of how recommendations are already based on others similar to themselves should be sufficient.

7.2.4 Choosing Wisely: The Evaluation of Candidate Models

All four approaches explored in this thesis could be used to develop a recommendation system. It is up to Insurely to weigh the pros and cons to determine the most suitable system for their

needs. For example, A/B testing could be employed to compare all four against each other, depending on the online traffic available.

Considering the substantial weight of the underlying product, we argue that the recommendations should not be treated lightly and favor predictive performance. One should only suggest consumers alter their coverage if we are confident in the recommendation. Therefore, minimizing the number of faulty recommendations should be a top priority. Consequently, both the XGBoost and ANN-based recommendation systems are valid candidate models. They should be further explored and evaluated against live traffic through A/B testing.

Regardless of the approach, A/B testing should be used to refine the recommendation system further. For example, if the clustering-based recommendation system is preferred, one could analyze different numbers of clusters. Alternatively, fine-tuning the learning rate could be interesting if the ANN is preferred. Transitioning to the online environment enables exploring possibilities not feasible offline.

7.3 Future Work

7.3.1 Confidence in Predictions

All recommendation systems currently predict the absence or presence based on which is most likely. In practical implementations, one may have a stricter criterion before suggesting a consumer coverage change. For instance, a threshold of 95% confidence could be set in place. This would prevent recommendations from being made if they lack substantial evidence.

Another important factor is considering the implication of recommending a particular coverage parameter. Recommending to sign traffic insurance makes little sense since this information can be assumed to have been considered by the consumer already. This work requires manual inspection and consideration of which coverage parameter should be able to be recommended.

7.3.2 Similarity Between Consumers

The entire work of recommendations is based on how well one can measure the similarity between consumers. The most crucial feature when considering how similar consumers are should be their similarity on the insured item. More information about the consumer means we can ensure the consumers match relevant features for specific coverage parameters.

Apart from the underlying data, the chosen similarity metric has a significant impact. Investigating, defining, and working with different similarity metrics are incorporated in many data-oriented projects. If Insurely finds a similarity measure during the efforts of another work, it can be combined with the work of this thesis to improve the recommendation system. Having a well-defined similarity metric will improve the cluster-based recommendation system. Consequently, it would improve the pseudo-labels used to pre-train the supervised recommendation systems.

7.3.3 Implementation in Other Domains

This thesis has investigated the development of a car insurance recommendation system, with findings and methodologies not solely confined to this domain. The techniques applied could be adapted and expanded to diverse areas of insurance, demonstrating a comprehensive cross-domain utility. For example, a comparable recommendation system could benefit home insurance, where information from others in similar living situations could help determine insurance needs.

This thesis did not utilize information from insurance policies other than vehicle insurance, but there is also potential for *cross-selling* strategies. Cross-selling refers to selling different products or services to existing customers, which is common in the insurance sector. This approach enables companies to leverage an already established risk profile of the consumers to understand their needs and preferences for other policies.

In the context of this study, if a consumer has a specific type of car insurance, the system could recommend a compatible home insurance policy, considering the consumer's risk perception and coverage preference. Such an approach has the potential to significantly enhance the system's predictive power by creating a personalized insurance portfolio for each consumer.

8 Conclusion

This thesis explored potential recommendation systems for the insurance sector that functions as a 'consumers like you'-feature. The research contrasted a clustering-based recommendation system with potential supervised alternatives in an offline environment, intending to identify a set of candidate models for further exploration with live traffic evaluation.

8.1 Consolidation of Findings

Our research revealed both the potential and constraints of the clustering-based and supervised recommendation systems. Despite challenges such as determining the optimal number of clusters, the clustering-based recommendation system created a 'consumers like you'-feature. The supervised recommendation systems, which included the neighborhood model, XGBoost, and ANN, emulated the results of the clustering-based system with varying success rates. Critical considerations of each system's effectiveness were performance, prediction time, and interpretability. The important nature of the underlying product would entail a favor in predictive performance. Consequently, both the XGBoost and ANN are valid candidate models to be further explored. However, high-quality data is vital for valuable recommendations regardless of the system's methodology.

The practical implications of these findings are considerable for Insurely, demonstrating the future possibilities of open insurance solutions. The research provides a foundation to be further built upon, resulting in candidate models that can be evaluated on live traffic using A/B testing methods. It also highlights the crucial role of incorporating consumer feedback to refine and enhance the recommendations since predictive performance is vital for this industry.

8.2 Suggestions for Future Research

This thesis serves as a robust foundation for future exploration. Potential areas of interest include improving the similarity measure between consumers and establishing more rigorous criteria for recommendations. Further examination into the adaptation of these systems to other domains and their applicability for other uses, such as next-item prediction, could also offer valuable insights.

8.3 Closing Remarks

In conclusion, this thesis marks a significant stride toward developing personalized recommendation systems in the insurance sector. It emphasizes these systems' essential role in empowering consumers and improving their decision-making abilities, achievable through increased transparency and data flow inherent in open insurance. This approach can help simplify the complexities of insurance for the average consumer.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. & Zheng, X. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Bengio, Y., Courville, A. & Vincent, P. "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives". In: *CoRR* abs/1206.5538 (2012). arXiv: 1206.5538. URL: http://arxiv.org/abs/1206.5538.

[3] Breese, J. S., Heckerman, D. & Kadie, C. "Empirical analysis of predictive algorithms for collaborative filtering". In: *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA* 98052 (1998).

[4] Chawuthai, R., Choosak, C. & Weerayuthwattana, C. "A Recommender System for Insurance Packages Based on Item-Attribute-Value Prediction". In: *Recent Challenges in Intelligent Information and Database Systems: 13th Asian Conference, ACIIDS 2021, Phuket, Thailand, April 7–10, 2021, Proceedings 13*. Springer. 2021, pp. 73–84.

[5] Chen, T. & Guestrin, C. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[6] Chen, T. & Guestrin, C. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[7] Cover, T. & Hart, P. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[8] Delua, J. *Supervised vs. Unsupervised Learning: What's the Difference?* 2021. URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning (Accessed on 05/15/2023).

[9] Guo C. & Berkhahn, F. "Entity Embeddings of Categorical Variables". In: *CoRR* abs/1604.06737 (2016). arXiv: 1604.06737. URL: http://arxiv.org/abs/1604.06737.

[10] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T.E. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[11] Herlocker, J. L., Konstan, J. A., Terveen, L. G. & Riedl, J. T. "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 5–53.

[12] Hunter, J. D. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[13] Insurely. *About Us*. 2021. URL: https://www.insurely.com/about-insurely/about-us (Accessed on 03/16/2023).

[14] James, G., Witten, D., Hastie, T. & Tibshirani, R. *An Introduction to Statistical Learning: with Applications in R*. New York: Springer, 2013. ISBN: 978-1-4614-7137-0.

[15] Lam, S. K., Pitrou, A. & Seibert, S. "Numba: A llvm-based python jit compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6.

[16] Länsförsäkringar. "Maskinskaderapport 2020". In: (2020). URL: https://www.lansforsakringar.se/49aedf/globalassets/aa-global/dokument/ovrigt/aa-om-oss/rapporter-och-undersokningar/lf_maskinskaderapport_2020.pdf (Accessed on 04/19/2023).

[17] Lesage, L., Deaconu, M., Lejay, A., Meira, J. A., Nichil, G. & State, R. "A recommendation system for car insurance". In: *European Actuarial Journal* 10 (2020), pp. 377–398.

[18] Linden, G., Smith, B. & York, J. "Amazon.com recommendations: Item-to-item collaborative filtering". In: *IEEE Internet computing* 7.1 (2003), pp. 76–80.

[19] McKinney, W. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[20] Qazi M., Fung G. M., Meissner K. J. & Fontes E. R. "An insurance recommendation system using bayesian networks". In: (2017), pp. 274–278.

[21] Rousseeuw, P. J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[22] Shalev-Shwartz, S. & Ben-David, S. *Understanding machine learning: from theory to algorithms*. New York, NY, USA: Cambridge University Press, 2014, p. 397. ISBN: 978-1-107-05713-5.

[23] Shani, G. & Gunawardana, A. "Evaluating recommendation systems". In: *Recommender systems handbook* (2011), pp. 257–297.

[24] Van Der Maaten, L. & Hinton, G. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[25] Yao, S., Halpern, Y., Thain, N., Wang, X., Lee, K., Prost, F., Chi, E. H., Chen, J. & Beutel, A. "Measuring recommender system effects with simulated users". In: *arXiv preprint arXiv:2101.04526* (2021).