TREND FEATURES AND ADDITIVE FEATURE SELECTION METHODS FOR CHURN MODELS UNDER PROPERTY AND CASUALTY INSURANCE BUSINESS PARADIGMS

by

Jacob Foster Anderson

M.B.A University of Colorado, Colorado Springs, 2012

B.A. Thomas Edison State University, 2016

B.A. American Military University, 2009

A thesis submitted to the Department of Mathematics and Statistics
Hal Marcus College of Science and Engineering
The University of West Florida
In partial fulfillment of the requirements for the degree of
Masters of Science in Mathematical Sciences

2018

The thesis of Jacob F. Anderson is approved:

Subhash C. Bagui, Ph.D., Committee Chairman	Date
Sikha Bagui, Ed.D., Committee Member	Date
Rohan Hemasinha, Ph.D., Committee Member	Date

Accepted for the Department/Division:

Jia Liu, Ph.D., Chair	Date

Accepted for the University:

Kuiyuan Li, Ph.D., Interim Dean, Graduate School	Date

ACKNOWLEDGMENTS

First, I would like to thank Dr. Subhash Bagui of the Hal Marcus College of Science and Engineering at the University of West Florida (UWF). Prof. Bagui's encouragement and mentorship provided much needed motivation while wrangling with this dataset. His flexibility, reassurance, and guidance during the final months was crucial.

I would also like to thank the committee members Dr. Sikha Bagui and Dr. Rohan Hemasinha. Their expertise and candid input was invaluable for the final completion of this analysis.

I'd be remiss not to acknowledge Dr. Anthony Okafor for affording me the unique and exciting opportunity to perform additional research in the UWF Data Lab as a graduate research assistant. Prof. Okafor's patience and advice in the lab, fostered the type of environment needed for the creative exploration of difficult real-world datasets.

I'd like to express my gratitude to Company XYZ for allowing me the opportunity to intern. This thesis topic was entirely made possible by Company XYZ's encouragement of creative data solutions and readiness to explore emerging data mining methods. During the internship, I had the pleasure of working under one of their upper-level managers whose input was instrumental in determining the thesis direction.

Lastly, I'd like to thank my friends and family for their continued support throughout my academic pursuits.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

ABSTRACT

TREND FEATURES AND ADDITIVE
FEATURE SELECTION METHODS FOR
CHURN MODELS UNDER PROPERTY
AND CASUALTY INSURANCE BUSINESS
PARADIGMS

Jacob Foster Anderson

The P&C Small Commercial Insurance industry presents a significant policy retention challenge given the presence of independent intermediaries. Current retention models generally follow traditional relational marketing paradigms and therefore do not account for the complexities introduced by the presence of an independent intermediary. Given an anonymous policy dataset consisting of correlated, high-dimensional, and high-cardinality categorical/numeric data; current data mining methods are used to construct a retention model with practical applications. Additionally, predictive features that capture intermediary-related information are engineered and designated as candidate features. Candidate features are selected for final model inclusion using various data mining approaches to feature importance measurement and feature selection.

CHAPTER I

INTRODUCTION

Customer retention across industries has been strongly advocated in marketing strategies since the emergence of relational marketing in the 1990s; the cost of obtaining new customers is frequently cited as significantly higher than the cost of retaining existing clients [49, 74]. The business landscape of today has been transformed by vast repositories of consumer data with learning algorithms becoming increasingly adept at database information extraction for knowledge verification and discovery [43]. As a natural consequence, retention-focused predictive models have received attention from both business practitioners and academics. Models of this type are typically referred to as *churn* models. Churn, within most business domains, is defined as a customer discontinuing service with a business.

The majority of customer-centric marketing disciplines intrinsically assume a certain degree of business-client interactions when considering retention strategies [49]. Churn[1] models therefore have historically focused on customer demographics, product information, and data related to business-client interactions. While this assumption holds for a large percentage of the business arena, some industries have a different business-customer dynamic. For Property and Casualty (P&C) insurance firms (carriers) using business paradigms involving independent intermediaries (agents), there exists an additional filter between carriers and customers that impacts retention-based strategies. For carriers in this segment, customer churn has an added dimension that has resulted in a scarcity of P&C industry-specific retention models.

Customers in the P&C industry are referred to as policyholders and are the end-users of insurance products originating from the carrier. For the purposes of this analysis,

[1] Used interchangeably with retention models

policyholder churn[2] will be defined as an annual policy non-renewal. Non-renewal occurs when a policyholder does not renew during the annual renewal month. The cooperation of an anonymous insurance carrier, Company XYZ, allowed for the use of an anonymized small commercial policy-level dataset.

Section 1.1 discusses the current state of the industry segment with a focus on the role and influence of independent agents. Churn models consideration for insurance carriers using an independent agent paradigm are described in Section 1.2. The chapter concludes with Section 1.3 problem statement and chapter outline.

1.1 P&C Industry

The current P&C small commercial industry life cycle stage is considered mature with a high level of competition and market saturation; regulation is considered to be heavy with low customer loyalty [20, 44]. Within the P&C industry, carriers provide products to policyholders through agents; agents are an additional layer between carriers and policyholders of varying degrees of independence from the carrier. Agents act as market matchers who assist policyholders in choosing the best carrier given their business needs [10]. They generally represent numerous carriers and recommend policies based off several factors with evolving levels of importance. On average, agents have 11 different carrier relationships and prefer carriers that can offer more types of coverage [51].

Despite the relatively recent rise of online insurance product offerings, independent agents remain the preferred method for obtaining commercial coverage. According to J.D. Power [51], as of 2018, independent agents wrote 83% of commercial premiums. Furthermore, the sheer complexity of P&C industry coverage per se necessitates the

[2]Policyholder churn should not be confused with agent churn. The latter has negative connotations in the P&C insurance segment

continued dominance of carrier-agent-policyholder channels over the next five years [20]. That being said, a 2016 McKinsey&Company online survey found that 70% of small commercial buyers polled are gathering insurance information through means outside of agents, only 53% are "actively loyal," and 60% would be open to direct policyholder-carrier channels [56].

Characteristic of highly saturated markets, Gambardella et al. [20] found that renewals rather than new business comprise the majority of sales revenue. For small commercial insurance, the decision to renew can be influenced heavily by the agent. A level of influence is to be expected as the primary function of the agent is to essentially match carriers to the needs of the policyholder [57] and facilitate underwriting[3], the accomplishment of which requires the agent to obtain a detailed understanding of the policyholder coverage needs. While this depth of knowledge regarding the policyholder allows for the agent to better perform as a market matcher, it also affords the agent additional opportunities. For example, the agent must reasonably assess policyholder sophistication levels when determining coverage needs. This assessment is additional knowledge that an agent can then leverage to influence carrier-related decisions for reasons outside of the knowledge sphere of the policyholder [61]. Detailed knowledge about policyholders can also be used to influence settlements and even switch carriers [35]. Carriers understand potential agent influence over policyholders and attempt to align interests through numerous means. In addition, the mere fact that agents are able to perform functions on behalf of *both* the policyholder and the carrier obscures the line that divides representing the policyholder and representing the carrier. Deeper analysis is usually conducted using a framework derived from principle-agent theory [16] and therefore outside the scope of this analysis. It merits acknowledgment, however, that the ability of agents to act in their own best interest raises moral hazard concerns.

[3]Carrier acceptance of liability documented as a policy

Compensation strategies generally aim to counter this hazard by aligning carrier-agent interests [10]. For the purpose of this analysis, it is sufficient to acknowledge the possible impact of agents on policyholder churn as well as the existence of carrier compensatory strategies.

1.1.1 Company XYZ

Company XYZ is carrier that writes small commercial multi-line P&C insurance primarily through agents. The small commercial market serves businesses with less than 100 employees and generally less than $100,000 in annual premiums, although the majority of the market share consists of businesses with fewer than 29 employees [56]. Annual premiums in the policy dataset generally conformed to this definition with a small fraction of outliers ($\approx 2.8 \times 10^{-4}$) above the industry defined upper premium threshold. The dataset policy non-renewal percentage spanning 10 fiscal quarters was $\approx$ 17% excluding the presence of new policies. This presented an imbalanced dataset as shown in Figure 1.

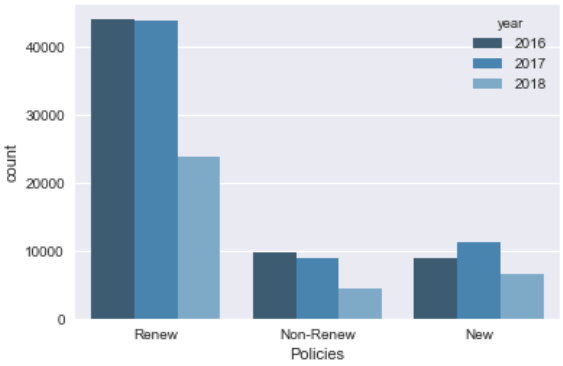


Figure 1: *Imbalanced Dataset*

Aside from reflecting policyholder retention, the dataset was used (by Company XYZ) for general purpose corporate assessment of factors that affect or may affect future profitable growth. Therefore, the data generally covered premium, agents, policyholders, policies, and new policies/retention. While the carrier has contact with the agent, direct contact between the policyholder and carrier is limited. Furthermore, the actions of the agent are not influenced in the same manner as a policyholder. Despite a possible small commercial shift towards direct carrier-policyholder channels as indicated by the survey

conducted by [56], the dataset used for this analysis was exclusive to policies obtained through and "owned" by agents.

1.2 Modeling Considerations

Existing retention/churn models serve as indicators for practitioners. If a policy within a sample dataset has a certain probability of non-renewal, a company can take preemptive measures to increase the probability of policy renewal. Preemptive measures are manifold and can range from telephone calls or emails to more aggressive measures such as price adjustments or compensation [61]. Churn models generally attempt to predict non-renewals or otherwise defined customer disengagement behavior. However, due to the complicated carrier-agent-policyholder dynamic in the P&C industry, there are comparatively fewer churn models that attempt to predict policyholder non-renewal. When policyholder churn is caused by carrier-agent relationship termination, it is sometimes referred to as *agent disengagement* [64].

When addressing policyholder churn, carriers and agents are often seen as partners with a mutual interest to work together in order to retain the business of a policyholder. While this description is mainly accurate, both business entities are still driven by corporate best interests; it may then be more appropriate to relegate the relationship to a business arrangement. Furthermore, it can be assumed that any opportunities afforded to either the carrier or agent that may enhance their relative position in the arrangement, will be leveraged. As such, there are a variety of factors that further define the agent-carrier relationship, to include a factor that carries legal implications. Generally, carriers offer competing bids for policies with the agents ultimately deciding or recommending the best policy for the policyholder [35]. While the carrier performs the underwriting, the agent has direct contact with the policyholder and maintains *ownership* of the client (policyholder) list. The concept of ownership can be defined as a

legal right of the agent wherein the carrier cannot directly solicit an agent's client and cannot attempt to replace the agent for that client [57]. This concept extends to policy expirations and therefore further increases possible agent influence at the time of renewal. This legally defined aspect of the carrier-agent relationship distinguishes it from more traditional business-to-client channels referenced in relational marketing [49]. Within the P&C industry, while the agent directs retention efforts toward the policyholder, the carrier must direct retention efforts towards *both* the policyholder and the agent.

There are four generally agreed upon policy attributes that influence retention; two main attributes and two ancillary attributes. In almost all cases, the policyholder is very aware of the main attributes: premium and coverage [20, 10, 16]. As a large degree of homogeny exists among coverage, premium generally takes priority for the small commercial client and is often referenced as policyholder *price sensitivity* [20]. However, there are numerous other policy attributes that may impact renewal, especially in a soft market[4]. The two primary ancillary attributes identified are: ease of doing business and compensation [20, 51] (see Appendix A.1). While the agent is cognizant of the additional attributes, oftentimes the policyholder is not - particularly of the latter ancillary attribute. Compensation is a complex and somewhat controversial dimension in the P&C industry that includes various forms of commissions and perks provide by the carrier to the agent. Company XYZ's compensation practices are pursuant with industry norms and are consistent with their main competitors. As such, engineering of compensation related features were not attempted.

[4]an industry term for a market cyclical stage characterized by lower premiums and an increased importance of other policy traits

1.3 Problem Statement

If new policies were considered along with existing policies, policyholder churn seemed to fluctuate annually around 14% for Company XYZ. The [56] survey on small commercial insurance suggested the industry norm to be considerably lower ($\approx$ 6%), though the survey authors acknowledged actual churn percentages were likely quite a bit higher than 6%. Additionally, the survey addressed agent impact on policyholder segmentation; identifying the primary polarizing segmentation force aside from price sensitivity as the "extent and nature of customer reliance on agent." The general analyst consensus indicated a larger shift to direct carrier-policyholder relationships may be on the horizon. Though the policy dataset is focused only on agent managed policyholders, general increases in policyholder churn could be a result of a larger paradigm shift away from the traditional carrier-agent-policyholder channel and not necessarily a trend that carriers can attempt to mitigate through traditional agent-based marketing strategies.

1.3.1 Research Questions

Main Research Question Given the intermediate filter between carrier and policyholder, can a churn model be constructed using the popular tree-based ensemble XGBoost system with implications for practical application?

Sub-research Question 1: Given a limited feature space, can additional features be engineered based off current data mining techniques and domain knowledge to capture agent-related information that may influence policy churn?

Sub-research Question 2: Can an effective performance-based feature selection wrapper method be constructed to determine engineered feature contribution and improve model performance?

1.3.2 Outline

Chapter 2 A literature overview of 2018 advances in churn prediction as well as current data mining techniques relevant to the analysis are covered.

Chapter 3 Dataset features are introduced and discussed. Basic engineering methods are reviewed to include feature extraction and transformation. Based off information presented in Literature Review, an approach to engineering trend features is proposed.

Chapter 4 Data mining methods are reviewed. Performance-based feature selection methods are proposed to assist in both feature selection through performance contributions and model performance. Lastly, the XGBoost [76] model is described.

Chapter 5 Engineered features as well as feature selection methods are tested and compared. The top performing method is selected and features are used by a hyper-parameter optimized XGBoost for comparison against popular baseline models. Test results are analyzed for practical use and conclusions/recommendations made

CHAPTER II

LITERATURE REVIEW

The importance of retaining existing customers is evidenced by the expanding field of customer relationship management (CRM) across industries [49]. The prevailing heuristic that acquisitions of new clients are typically 5 to 6 times more expensive than retaining existing clients is a motivation used often in consumer retention literature [74, 68, 78]. Consumer-centric marketing has transformed from a priority to an absolute necessity and was described as a marketing approach ”paradigmatic shift” by scholars around the turn of the 20th century [49]. As discussed in the opening chapter, policyholder churn in the small commercial P&C industry space has an additional layer that increases the complexity of traditional business-customer relationships. The presence of an agent with varying degrees of influence over the policyholder and interests that do not always align with carrier (or the policyholder) adds a human dimension that can influence policyholder churn. While predictive modeling efforts that account for independent agents are rare, there has been considerable research on direct business-customer churn. As companies throughout various business domains continue to refine data collection efforts, increasingly complex predictive models are being leveraged to increase retention, especially in the telecommunications industry. Relatively recently, neural networks and tree-based ensemble models tackling highly imbalanced datasets have demonstrated strong improvements in churn prediction.

A review of 2018 churn models in other industries can give insight into the latest data mining advancements for business practitioners. In section 2.1, current inter-industry churn models are reviewed with a focus on contributions to the field of data mining to include feature engineering. Section 2.2 reviews literature relevant to the methods used on the policy dataset. References to the policy dataset are made throughout the chapter when relevant.

2.1 Churn models in 2018

A Seycor Consulting [23] team constructed a highly accurate churn classification model utilizing an `XGBoost` (XBG) base and numerous temporal features. The model achieved a Log-loss of .07974 and placed first in the Web Search and Data Mining (WSDM) 2018 Cup Customer Churn Challenge. [23] applied a systematic approach to feature selection using an iterative python function; the resulting model was a stacked XGB and LightGPM model. Several of the temporal features were trailing time windows (lags) and numerous features were engineered to capture additional predictive information. The results of the competition indicated that the emphasis on engineered features was paramount for the stacked model's success.

XGB has outperformed Neural Nets in numerous Kaggle[5] competitions, to include challenges involving customer behavior prediction, and has demonstrated advantages over other models through scalability[6] and flexibility [7, 23]. The tree-based construct of the model brings the added capability of ranking feature importance as well as feature interactions [43, 28]. While by design, the XGB is able to leverage CPU computing power through parallel processing [7], Mitchell et al. [46] described an implementation of XGB executed entirely on the graphics processing unit.

Data scientists from Samsung Research and several large US universities built a semi-supervised and inductive embedding model for large scale mobile game churn [39]. The model learned both the prediction and embedding function for user-app relationships using deep neural networks capable of capturing contextual and relational information. Additionally, the model incorporated several loss functions to include supervised, unsupervised, contextual, and temporal aspects. Given a highly unbalanced

[5] A website that hosts machine learning competitions

[6] Model ability to handle increases in sample and/or feature space as well as high-cardinality categorical features [43]

dataset (roughly 15% minority class), the model outperformed base models in both recall and AUC. As opposed to training data random-sampled (hyper-parameter) tuning used for this analysis, [39] tuned hyper-parameters using the test set.

Churn models undoubtedly reduce business marketing expenses and have demonstrated predictive ability. However, the expense reduction is not only a function of churn model fidelity, but also the expenses saved for a particular would-be churner. Therefore, detecting would-be churners who are of greater value to the company would contribute significantly to the overall cost savings in a retention campaign. Stripling et al. [68] recently proposed a technique for churn classification that uses a profit-based performance metric. The model was run as an enhanced lasso-regularized logistic regression against other linear base models on nine separate data sets. Given inherent weaknesses in AUC as a single numerical metric, [25] proposed the H metric as an alternative or addition to AUC. The H metric uses a prior representing a cost uncertainty and was included by [68] along with AUC, recall, and F1. While the mentioned metric results were mixed with the model performing generally well, the profit gain obtained from the model was on average 1.43 pounds higher per customer than base models. So while the model was only slightly better than average at predicting churn using establish metrics for imbalance data sets, it was able to select higher value would-be churners than the base models.

2.1.1 Feature Engineering

Various forms of feature engineering exist to maximize the information gain from a given base feature. Frequency counts of categorical variables are often used for text classification models. Forman [17] described the practice of *stemming*, which involves merging certain word forms in order to ultimately reduce the feature space. [17] also used frequency word counts within a document as a means of gaining information. Similarly, the concept of equal-frequency binning groups continuous features by

intervals and assigns a frequency count, therefore extracting additional distribution related information [34]. Frequency counts can also serve as a means of combining categories values for computational efficiency; similar to *stemming* for text classification shown by [17]. [34] highlights the computational efficiency of equal-frequency binning for tree-based models given their use of order statistics (for comparison at decision points). Less common categorical values can be identified by their frequency count and merged to increase data tractability, especially in the instance of high-cardinality[7] categorical features. Kang et al. [33] introduced several data aggregation techniques involving merging homogenous values of different predictors. The consolidation of categorical data based off relative frequency was also reviewed. For the purposes of this analysis, aggregation methods for low relative frequency values for high-cardinality features are used on the policy dataset, resulting in the use of categorical features that would otherwise cause severe overfitting.

In addition to outlining the teams general approach to the WSDM 2018 Cup Customer Churn Challenge, [23] discussed some of the features engineered. Temporal features were engineered using the relative refactoring method and an absolute method. Relative refactoring involves designing temporal features base off a fixed point in time while absolute temporal features are invariant to any particular point in time and are generally nominal categorical features. The policy dataset used in this analysis had limited temporal features. However, numerous additional features were extracted and engineered based off techniques mentioned by [23]. Furthermore, Gregory made brief mention of a wrapper method for feature selection utilized by the Seycor Consulting team; the method used a python script to iteratively add candidate features to a set of base features and measure accuracy. This is a form of a *Sequential Forward Selection* (SFS)

[7] Categorical feature with feature values > 100

method as described by Pudil et al. [52]. Modified SFS methods similar to the type described by [23] are used later in this analysis.

In a recent analysis on early churn prediction in the telecommunications industry using time-series data, Oskarsdottira et al. [48] proposed a method of feature engineering which involved developing call networks and then designing features based off the customers interactions within that network. The network generally included prior churners, the count of which was included as a feature for the customer. If the network concept is transferable to a more general grouping, this method can be extended to create additional observation-level information related to other members of a group. The engineered observation-level information could then be given a temporal version in the form of the previous time window. With the policy dataset, the concept introduced by [48] was extended to policies under a given agent and used for engineering trend features. Sharma [64] also briefly discussed the construction of a General Linear Model (GLM) using trailing aggregate agent premium windows to predict churn resulting from agent disengagement. [64] stated the GLM achieved an 86% accuracy for agents with $> 70\%$ probability of churn. [64] model performance is discussed further in Chapter 5. However, both previous trends in performance and the number of certain types of accounts, could be significant predictors of non-renewal [48, 64]. The concepts of trailing agent premium and within-network influence as predictors, are built upon in Chapter 3.

Strobe et al. [69] discussed the importance of relevant predictive feature identification when using black box models. Tree-based ensemble methods of ranking feature importance were investigated using high dimensional correlated data; furthermore, the tendency for such models to show preference for correlated features was explored. Using simulated data, [69] demonstrated a tree-based ensemble inflating a feature importance due to correlation. Tolosi et al. [72] expanded on correlation concerns

related to high dimensional datasets and tree-based ensemble methods. Given that correlated features often arise in groups within a feature space, [72] reviewed *feature representative* strategies and compared results with baseline methods to demonstrate a reduction in correlation bias. [72] also demonstrated the same negative effect on tree-based ensemble methods as [69]. Numerous correlated features existed in the policy dataset. Therefore, correlation analysis and feature derivative engineering were conducted to identify the possible inflation effect discovered by [69] and to use representational features as described by [72].

2.2 Techniques

Generally, the main objective of a churn model is to predict a minority class. Therefore, for strong generalization[8], appropriate metrics and model behavior are required. The concept of over-fitting and under-fitting are machine learning extensions of the well known variance-bias tradeoff [43]. Given the potential high dimensionality of the policy dataset and the tree-based ensemble model used, the risk of overfitting was high. A mitigation measure taken to counter over-fitting by observing training scores (error) relative to prediction scores is covered in chapter 4. However, an imbalanced dataset without adequate data preparation can easily exacerbate issues related to model performance and overfitting. Current data-level and algorithmic approaches for handling imbalanced and anomalous data are reviewed below.

2.2.1 Data Preparation

The policy dataset included field collected features and therefore introduced the possibility of outliers. Outliers can surface for different reasons, to include an incorrect recording of data [31]. An outlier can be defined as the following:

An observation in a data set which appears inconsistent with the remainder of that set of data [43].

[8]Model induction abilities

Removal of outliers that would otherwise detract from model performance is a preprocessing step of debatable importance when constructing an effective model [43]. Liu et al. [38] preformed an empirical evaluation using the Isolation Forest method to isolate and identify anomalies in a large high dimension data set. The method is compared against a popular clustering method, a distance-based method, and a Random Forest method using AUC and processing time. Isolation forest showed superior performance against the base methods in both AUC and processing time for larger datasets. [38] implemented a version of Isolated Forest that was able to maintain low processing time as dimensionality increased and has clear applications for high volume real-time data.

The `rate_adq` feature introduced in the next chapter was an ordinal (integer) ranking assigned to policies that reflected the current desirability of the rate. The higher a score, the less acceptable a current rate was and the greater the subsequent rate adjustment would be at the next renewal. Approximately 10% of `rate_adq` feature values were missing. Identified by domain experts to be of high importance for policyholder churn prediction, `rate_adq` represented a prime candidate for imputation. In 2009, the *matrix completion problem* gained sudden popularity by its performance with high dimension datasets with a high percentage of missing values during the Netflix competition [27]. The `Soft Impute` algorithm introduced by Hastie et al. [45] computes the low-rank singular value decomposition (SVD) of a dense matrix and iteratively replaces missing matrix elements from the low-rank SVDs during optimization. The rank constraint for the optimization algorithm is substituted with the nuclear norm[9]. The python algorithm [26] was used to impute missing `rate_adq` values in chapter 3.

[9]Sum of matrix singular values

2.2.2 Data Imbalance

The *class imbalance problem* refers to issues that occur when an under-represented class exists in a classification model [41, 43]. It has been ranked as one of the top 10 data mining problems [37] and is a common issue for models in churn, fraud, medical diagnosis, and many other fields [63, 41]. Nitesh et al. [6] introduced the Synthetic Minority Over-sampling Technique (SMOTE) as a method to address class imbalance. SMOTE is a widely used and established data-level solution that over-samples the minority class by interpolating synthetic minority class samples along line segments joining k nearest minority neighbors [6, 41, 63]. The additional synthetic samples enlarge and obfuscate the decision region, preventing over-fitting and forcing the classifier model to be more generalized. Sima et al. [63] applied enhanced versions of SMOTE using 1:36 imbalanced transportation dataset in a case study of an Iranian highway. Using different classifiers, improvements were seen in almost all datasets. In an effort to improve recall without a large decline in precision, a hybrid principle component analysis (PCA) and SMOTE method was proposed by Naseriparsa et al. [47]; wherein the application of the unsupervised dimensionality reduction method immediately prior to synthetic oversampling showed promising empirical results on a Lung Cancer dataset.

Some of the most successful models used in retention modeling can achieve greater performance given an imbalanced dataset through an algorithmic solution approach in the form of hyper-parameter capabilities. Probst et al. [50] defined the amount of performance gain as *tunability* and performed a benchmarking study using six popular models, to include XGB. While greater tunability is desirable, it typically comes with an attendant increase in the number of hyper-parameters. Learning algorithms with a higher number of hyper-parameters can make tuning difficult and even infeasible using baseline algorithms such as grid and random search [50, 15]. Thornton et al. [71] used 21

popular datasets to compare classifier performance using various hyper-parameter search methods. Bayesian optimization outperformed other optimization algorithms including base line algorithms and, in some cases, performed on a level comparable with or exceeding domain experts [15]. XGB has numerous hyper-parameters that support algorithmic-level adjustment for model complexity and performance. Combinations of hyper-parameter adjustments have been shown to improve model training and performance as measured by AUC for imbalanced data [50]. Bayesian optimization of selected hyper-parameters using defined intervals is performed on the XGB and baseline models in chapter 5.

CHAPTER III

DATA

The raw policy dataset is a roughly $N = 15 \times 10^4$ sample set of policy-level information. For this analysis, policies are often referred to as samples or data points within a sample space. Policy information is represented by $D = 67$ original variables referred to as features within a feature space. Features that are being assessed for inclusion in the final policy dataset are referred to as *candidate* features. The notation used throughout this paper is a combination of machine learning notation [21], standard math notation [31], and set/matrix notation [29]. The combined notation was used to facilitate understanding given the convergence of several mathematical disciplines. The policy dataset is presented in various stages of preprocessing and is generally represented as $\mathbf{X} \in \mathbb{R}^{N \times D}$. Additionally, the policy dataset is often subsetted for different phases: training, testing, and validating. Subsets are represented as a set $(\mathbf{X}^{(\text{phase})}, \mathbf{y}^{(\text{phase})})$ such that $\mathbf{X}$ is a matrix of features and $\mathbf{y}$ is the response vector. Models $\mathcal{P}$ treats samples as vectors of features which can be represented as sets $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$. Feature vectors are then vectors of samples represented as $\{\mathbf{x}_1, ..., \mathbf{x}_D\}$ and can also be expressed with matrix notation $\mathbf{X}_{:,j}$. Therefore, a feature value j for sample i is represented $x_j^{(i)}$.

The chapter begins with several data processing techniques introduced during Literature Review. Section 3.2 covers the policy dataset in more detail, introducing and briefly discussing available features while covering some basic feature engineering methods. Additional information regarding original and engineered features as well as explanations of general concepts mentioned in Literature Review can be found in Appendix A.1. The section segues to 3.3 where the proposed engineering methods based off ideas advanced by [48, 64] are explained. Section 3.3 concludes the chapter with a description of the policy dataset correlation concerns and an analysis of

numerical/categorical feature correlation using standard statistical tests is performed. Additional correlation tables (charts) can be found in Appendix A.2.

3.1 Data Processing

Initial preprocessing involved screening samples for usability, removing null values, identifying missing values, and (in some cases) removing extreme or error values. Duplicate values were removed and features with missing values were assessed on a case-by-base basis. The term *data leakage* is not formally defined and is used in a variety of contexts. For this analysis, leakage was defined as the unintentional injection of response data into a training set or predictor. The inclusion of features that unintentionally have response-related patterns often leads to misleading performance metrics that will ultimately reflect in poor generalization. Numerous potentially useful features in the policy dataset unfortunately took on zero or null values once a given policy failed to renew. While imputation could have been attempted, data leakage risk would have increased to an unacceptable level. For the majority of the missing values in the policy dataset, if the missing values were less than a certain threshold and there was not a discernible pattern, the samples containing the missing values were removed entirely. The `rate_adq` feature was identified as a numerical feature of significant importance by domain experts with roughly 10% of values missing. The matrix completion algorithm introduced in the previous chapter was used to impute missing values and is reviewed.

3.1.1 Missing Value Imputation

The `Soft Impute` algorithm uses nuclear norm $||\cdot||_*$ regularization to approximate and fit a low rank matrix to a matrix with missing values [45]. The algorithm uses approximations to fill in missing values while minimizing the rank of the matrix. For matrix completion, the nuclear norm behaves similar to a ℓ_1 penalty by shrinking

singular values. This allows for more dimensions without the cost of higher variance. Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be a numerical matrix with missing values, rank $= r$, singular value decomposition (SVD) $= UDV'$, and the set of indices of observed entries $\Omega = \{(i,j) : \mathbf{X}_{ij} \text{ observed}\}$. [27] defined a projection matrix $P_\Omega(\mathbf{X})$ with the observed elements of $\mathbf{X}$ preserved and missing values = 0. Conversely, $P_\Omega^\perp(\mathbf{X})$ was the defined projection matrix onto the complement of the set Ω. Therefore, observed elements from $\mathbf{X}$ are set at zero and placeholders (later to be approximated) values replace missing values. Since the placeholders are essentially the same as missing values, the following holds: $\mathbf{X} = P_\Omega^\perp(\mathbf{X}) + P_\Omega(\mathbf{X})$. We can then define the SVD *soft-thresholding* function $\mathbf{S}_\lambda(\mathbf{X})$ as:

(i) Given SVD **s.t.** $\mathbf{X} = UDV^T$

(ii) $\mathbf{S}_\lambda(\mathbf{X}) \equiv U\,\mathbf{S}_\lambda(D)V^T$ **s.t.** $\mathbf{S}_\lambda(D) = diag[(d^{(1)} - \lambda)_{[+]}, ..., (d^{(r)} - \lambda)_{[+]}]$

We can then begin updating the matrix $\hat{\mathbf{X}}$ with approximated missing values using an iteratively updating matrix $\hat{\mathbf{Z}}$; which can be initialized as a zero matrix:

(i) $\hat{\mathbf{X}} \leftarrow P_\Omega(\mathbf{X}) + P_\Omega^\perp(\hat{\mathbf{Z}})$

(ii) $\hat{\mathbf{X}} = UDV^T$

(iii) $\hat{\mathbf{Z}} \leftarrow U\mathbf{S}_\lambda(D)V^T$

[45] defined the following convex optimization function:

$$f_\lambda(Z) := \underset{Z}{\text{minimize}} \frac{1}{2}||P_\Omega(X) - P_\Omega(Z)||_F^2 + \lambda||Z||_* \tag{1}$$

Equation (1) can be solved algorithmically as follows:

Algorithm 1 Soft-Impute

Require: Grid of warm starts λ **s.t.** $\lambda_1 > \lambda_2 > ..... > \lambda_k$

function SOFT IMPUTE(X)
  solutions $\leftarrow \{\}$
  Initialize Z with a matrix of zeros, $Z_{old} \leftarrow 0$
  for i from 1 to k (for λ values) **do**
    $\hat{X} \leftarrow P_\Omega(X) + P_\Omega^\perp(Z_{old})$
    $Z^{new} \leftarrow U\mathbf{S}_\lambda(D)V^T \leftarrow \mathbf{S}_\lambda(\hat{X})$
    Define the error threshold
    while $\frac{||Z^{new} - Z^{old}||_F^2}{||Z^{old}||_F^2} > \tau$ **do**
      $Z_{old} \leftarrow Z_{new}$
      $\hat{X} \leftarrow P_\Omega(X) + P_\Omega^\perp(Z_{old})$
      $Z_{new} \leftarrow U\mathbf{S}_\lambda(D)V^T \leftarrow \mathbf{S}_\lambda(\hat{X})$
    The matrix Z with imputed missing values attains the desired error
    $\hat{Z}_{\lambda_i} \leftarrow Z_{new}$
    solutions.append($\hat{Z}_{\lambda_i}$)
  return solutions

The algorithm takes in a value λ for regularization of the nuclear norm and solves the optimization problem in equation (1). If a certain threshold τ is achieved, a matrix containing imputed missing values for the respective λ is appended to a solution set for all λs. Using the `Soft Impute` algorithm, the missing values for `rate_adq` were imputed and the feature was included as a candidate feature. The policy dataset was processed for imputation to avoid data leakage and confounding variables by removing the response and categorical features. The `rate_adq` feature became the response feature for the algorithm and all other numerical values were normalized using a preprocessing pipeline [53]. Due to the null values, a custom normalizing script was written to normalize existing values while bypassing (and keeping) null values for the response feature. The python translated `softImpute` version of the original R package authored by Hastie et al. [26] was obtained through github.

3.1.2 Data Sample Reduction

Multivariate methods for identifying outliers are traditionally categorized as parametric or non-parametric [43]. Detection methods involve identifying data points that differ significantly based off density assumptions regarding the data distribution (parametric), distance-based measures, and clustering (non-parametric). Methods that involve assumptions about data distribution are often regarded as unsuitable for high dimension datasets. Furthermore, masking and swamping effects[10] can reduce the effectiveness of non-parametric methods.

`Isolation Forests` algorithms (IFA) leverage decision tree model partitioning with the fact that outliers are "few and different" [38]. This method reduces swamping and masking effects, is less computationally complex, and scales well for high dimensional datasets. Using an ensemble of decision trees, IFA identifies outliers by number of partitions. Given the large presence of dependent categorical features and the attendant (possibly) confounding effects they introduced, only numerical features were used for the policy dataset. Let $\mathbf{X} \in \mathbb{R}^{N \times D*}$ **s.t.** $x_j^{(i)} = x$ be a reduced D^* matrix of features consisting of only numerical features. Since outliers are highly distinguishable from other data points, decision trees will partition them early on, requiring noticeably less partitions or a shorter path length $h(x)$ of a given point x. Outlier scores are ultimately calculated by averaging the path lengths for data points across an ensemble of trees. Let $c(n)$ be the average path length $h(x)$ given n data points and $E(h(x))$ be the expected value of $h(x)$ from a subset of isolation trees, the outlier score $s(x)$ can then be defined:

$$s(x) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2}$$

[10]Outlier clusters skew the data distribution and confound distance-based methods of identification [43]

s.t.

$E(h(x)) \to c(n)$, $s \to .5$ indicates non-outlier

$E(h(x)) \to 0$, $s \to 1$ indicates strongly as an outlier

$E(h(x)) \to n - 1$, $s \to 0$ indicates strongly as non-outlier

Algorithm 2 Outlier Identification

Require: $s(x, n)$, score function
Require: $\phi(\texttt{set}, \rho)$, filter function with contamination parameter ρ
  function ISOLATION FOREST($\mathbf{X}$, ρ)
    for $x_j^{(i)} \in \mathbf{X}$ **do** $s(x_j^{(i)})$
      Each data point is scored and appended to a set
      $\texttt{set} \leftarrow \texttt{set.append}(s(x_j^{(i)}))$
    The scores are then ranked according to outlier potential
    $\texttt{ScoreSet} \leftarrow s(x_j^{(i)}) > \cdot\cdot > s(x_j^{(k)}) \leftarrow \texttt{rank}(\texttt{set})$
    The matrix is reduced according to ρ
    $\mathbf{X}^{N(1-\rho)} \leftarrow \phi(\mathbf{X}, \texttt{ScoreSet}, \rho)$
    return $\mathbf{X}^{N(1-\rho)}$

The IFA was obtained from the `sklearn` library [30]. The entire policy dataset was used during feature processing and engineering. Outlier reduction of 5% was performed immediately prior to any further modeling, to include feature selection. From a practical standpoint, the proposed features and methods will not necessarily be used with the same tree-based ensemble method by practitioners. Therefore, applying a standard outlier reduction will help control for variations in model performance given the presence of extreme values.

3.1.3 Data Imbalance

Imbalance problems occur when an imbalanced dataset causes standard classification models to focus on the majority class while attempting to minimize error, leading to deceivingly accurate appearing results [43]. Often times, if consideration is not given to data imbalance, models will simply categorize all responses as the majority class. If the

minority class comprises only 15% of the data, this model would return a misleading accuracy score of 85%.

Solutions to imbalanced datasets can be categorized as data-level or algorithm-level [78, 77]. Data-level solutions are applied during the preprocessing phase and involve techniques aimed to correct for imbalances by over and under-sampling the minority and majority classes respectively. Algorithm-level solutions are model specific and often involve optimizing or adjusting hyper-parameters in order to focus learning on the minority class. `XGBoost` (XGB), with proper hyper-parameter tuning, is an example of an ensemble model adept at handling imbalanced datasets. XGB employs boosting (Appendix B.3) and can adjust for data imbalance through class weight rebalancing [41].

Data-level solutions are model agnostic and generally preferred by practitioners [78], especially if the model being employed lacks algorithmic-level adjustments available for imbalanced sets. Over-sampling with replacement tends to cause over-fitting and little improvement in minority class recognition [5, 61]. For large datasets, majority class under-sampling clearly can make the data more tractable through overall size reduction with balancing. Such a reduction, however, comes at the expense of a decrease in majority class information and typically is unsuitable for big data real world use [63].

As mentioned in Literature Review, the Synthetic Minority Over-sampling Technique (SMOTE) is a well known data-level solution that over-samples the minority class by synthetically generating new minority data points using interpolation and the spatial relationship between minority class members [41, 63]. Chawla et al. [6] described the randomness of the process and attendant obfuscation of decision regions which facilitates more generalized learning.

Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be an imbalanced dataset with $\mathbf{x}^{(\hat{i})} =$ minority class samples, $\mathbf{X}^{k(\hat{i}) \times D}$ represent a sub-matrix for each minority sample consisting of k nearest minority sample neighbors, and $\mathbf{x}^{(\hat{i}*)}$be a synthesized datapoint for $\mathbf{x}^{(\hat{i})}$.

Algorithm 3 Oversampling Method

Require: `knn( )`, function for identifying k nearest neighbors
function SMOTE($\mathbf{X}$, Z, k)
for $\mathbf{x}^{(\hat{i})} \in \mathbf{X}$ **do** $\mathbf{X}^{k(\hat{i})\times D} \leftarrow \texttt{knn}(\mathbf{x}^{(\hat{i})})$
Randomly extract $z \leq k$ samples from $\mathbf{X}^{k(\hat{i})\times D}$ to form yet another sub-matrix.
$\mathbf{X}^{z(\hat{i})\times D} \leftarrow \texttt{random}(\mathbf{X}^{k(\hat{i})\times D})$
Let $\mathbf{x}^{(\hat{i})}$ be the minority sample and $\mathbf{X}^{z(\hat{i})\times D}$ be its $z \leq k$ nearest minority neighbors subset
for $\mathbf{x}^{z(\hat{i})} \in \mathbf{X}^{z(\hat{i})\times D}$ **do**
$\Delta^{z(\hat{i})} \leftarrow (\mathbf{x}^{(\hat{i})} - \mathbf{x}^{z(\hat{i})}) \times \texttt{random}(0:1)$
$\mathbf{x}^{(\hat{i}*)} \leftarrow \mathbf{x}^{(\hat{i})} + \Delta^{z(\hat{i})}$
Then for each nearest neighbor in sub-matrix $\mathbf{X}^{z(\hat{i})\times D}$, a synthetic data point is created for $\mathbf{x}^{(\hat{i})}$
return augmented matrix

The `smote` module available in python `imblearn` library over-samples the minority class [37, 66] according to a defined ratio or automatically. The imbalance for the policy dataset was discussed in Chapter 1.

3.2 Features

The initial features were selected based off a combination of field practitioner knowledge and current literature. A continual dialogue with domain experts enabled enhanced feature selection that would not otherwise have been possible.

The initial features were grouped as policyholder[11], agent, carrier, and policy. Features can be further sub-divided by datatype as timestamp, numeric, and categorical. Numerical data that was dichotomous is referred to as an indicator. The following tables display initial feature programming abbreviations, datatypes, and brief descriptions. Additional descriptions and analysis can be found in Appendix A.1. The features displayed in the tables were actively collected and used by practitioners in the P&C

[11] Also referred to as an account

industry for analysis and decision making. While some of the features were engineered by the carrier, the majority were collected through independent agents and updated monthly.

3.2.1 Initial Feature Review

Policyholder-related features and *policy-related* features were not easily distinguishable (from each other) and their categorization was therefore moderately subjective. For the purposes of this analysis, policyholder features were any features that could be considered attributes of the *actual* policyholder. Unlike traditional churn models that focus on numerous end-user attributes, the policyholder data representing end-users was very limited, as seen in Table 1. This was likely a consequence of the independent agent maintaining the majority of the contact with the policyholder and the concept of *agent ownership* [10] discussed in Literature Review.

Table 1: *Policyholder/Agent Features*

Policyholder		
Account ID	`pol_num`	categorical, unique anonymized policy identifier
Industry	`industry`	categorical, covering business sectors
Segment	`segment`	categorical, also covering broad business sectors; but with modified definition
Program	`prgm`	high cardinality categorical further subdividing industry
Multi-policy	`multi_ind`	dichotomous, multi-policy accounts
Agent		
Agent	`agnt`	high-cardinality categorical
Master Agent	`mstr_agnt`	categorical
Agent state	`agnt_state`	categorical, US state agent operates out of

Agent-related features in Table 1 were also relatively few. Features included the title, location of the independent agent, and master agent for each policy sample. `mstr_agnt` is simply the larger independent business entity that the referenced `agnt` is subsumed under. Often times, the two features were equal.

The samples in the dataset were at the policy level[12]. The *policy-related* features were any features that contained information about the policy and not necessarily the policyholder. Aside from price-related features, policy features in Table 2 covered insurance product types with some overlap. The feature `line` is a broad categorization of insurance type. Other features, such as geographic and agent-related features, may be specific to certain `lines` and potentially likely caused higher dependency (correlation) between features. Additionally, `risk_complex` and `price_complex` were company-internal engineered indicators determined by number of locations, total insured value, account complexity, and line asset specifics. The feature `tenure` was a relative refactoring [23] method based feature while `eff_dt` was an absolute timestamp. Geographic features were included in this category and were related to the assets or entities covered by the policy.

Table 2: *Policy Features*

Policy		
Premium	`prem`	numeric, the annual premium
Ins. Line	`line`	categorical, the main insurance lines
Cons. Type	`con_type`	categorical, construction type, CMP policies only
Renewal Month	`eff_dt`	timestamp, indicating the month/year of renewal
Tenure	`tenure`	numeric, years policy has been with carrier
Geographic	`risk_state`	categorical, zip code, state, coastal, and county
	`zip_code`	
	`county`	
	`costal_ind`	
Rate Adequacy	`rate_adq`	numeric, score of current adequacy of rate
Complexity	`risk_complex`	risk and price complexity indicators
	`price_complex`	
Rate change	`rate_change`	numeric, positive or negative

Premium generally reflects policyholder price sensitivity and is a primary predictor of policy churn in the P&C industry [1]. Therefore, `prem` was used as the base feature for

[12]A policyholder (or account) can have several policies

the performance-based feature selection methods introduced in Chapter 4 and served as a comparative benchmark for evaluating contributions of other candidate features.

Several of the *carrier-related* features in Table 3 were anonymized employee titles. Given that these features were not high-cardinality, they were included to capture additional policy/policyholder information related to the presence of certain underwriters specializing in policyholders of a given size or type. Additionally, such features potentially added information related to the human interaction aspect of the policy. For example, some underwriters may have better relationships with certain independent agents therefore increasing a given agent's preference for the carrier, a preference that may influence policyholders. The `s_r` feature operated in a similar manner. Processing centers were considered representative, to an extent, of carrier-policyholder interactions absent the influence of the independent agent. `seg_strategy` was a company internal designation representing carrier management approaches for different account types.

Table 3: *Carrier Features*

Carrier		
Sales Region	`s_r`	categorical, the carrier sales representative title (anonymized)
Underwriter	`UW`	categorical, UW title (anonymized)
Processing center	`PC_cat`	categorical, varies depending on policyholder
Processing interaction	`PC_num`	numeric, carrier-policyholder interaction rank based off `PC_cat`
Segment Strategy	`seg_strategy`	categorical, strategy category used by carrier depending on policy attributes

3.2.2 Basic Engineered Features

Basic feature engineering methods used on the policy dataset included: frequency binning, feature extraction, and feature transformation. The features `zip_code` and `county` were high-cardinality categorical features and therefore invoked the often cited

(and dreaded) *curse of dimensionality*[13]. Even with aggregation of less common categories, the feature categories were numerous and caused over-fitting with additional computational complexity. As mentioned in Literature Review, equal-frequency binning [34] is typically used to convert numerical features into categorical features. A continuous interval is divided into K intervals with approximately equal occurrence frequency counts. The interval is then defined as a bin and assigned to the continuous feature as a category. This is not necessarily a replacement of the continuous feature, but rather a method to obtain more spatial information in the form of frequency distributed categories. Although commonly used for continuous features, equal frequency binning can be applied to high-cardinality categorical features using a slight modification. Frequency counts were *first* assigned to the high-cardinality categorical features, converting them into numerical values. The above mentioned equal-frequency method was then applied for binning.

Frequency counts of categorical features were necessary to implement this form of equal-frequency binning as well as to construct subsequent features not yet introduced. A frequency count assigning function f was defined using an indicator function followed by a summation. Let $\mathbf{X} \in \mathbb{R}^{N \times D}$, the indicator function I_k was used to count the occurrences of a particular value p_k for a categorical feature $\mathbf{x}_p$ with values $p_k \in \{p_1, ...p_K\}$. For a given sample $\mathbf{x}^{(i)}$, Feature value p_k cardinality was expressed as the following:

$$I_k(\mathbf{x}^{(i)}) = \begin{cases} 1 & \text{if } x_p^{(i)} = p_k \\ 0 & \text{else} \end{cases}$$

[13]Problems arising from high-dimension feature spaces

s.t. the cardinality of a specific categorical feature value was then:

$$|p_k| = \sum_{i=1}^{N} I_k(\mathbf{x}^{(i)}) \tag{3}$$

The domain for $\mathbf{x}_p$ categorical values $p_k \in \{p_1, ...p_K\}$ then has a codomain of integer values representing cardinality $|p_k| \in \{|p_1|, ...|p_K|\}$. A one-to-one function mapping feature values to frequency counts was then defined $f : p_k \mapsto |p_k|$ and an additional feature of frequency counts was constructed $f : \mathbf{X}_{:,p} \mapsto \mathbf{X}_{:,p^*}$.

`UW` and `s_r` were both assigned frequency counts and equal-frequency binning to represent information related to policyholder/policy grouping, policy complexity, interactions, and even carrier employee performance.

For some features, even with aggregation of less common categories, the feature categories were too numerous and caused severe overfitting with unacceptable increases in computational complexity. Features that would otherwise have been dropped, primarily geographic, were therefore able to be transformed into the above mentioned frequency form and included for feature evaluation. The frequency transformations applied to `zip_code` and `county` for example, may have provided additional spatial information for each policy through the relative frequency of the geographic locations with different boundaries. This transformation technique was applied to all dataset features with cardinalities > 100, although the application of this technique to create derived features did not automatically exclude parent features that were within a tolerated cardinality (i.e. `agnt`, `prgm`).

Feature extraction was used primarily with timestamp data. Derived features, such as month and year, were engineered from renewal dates. Using the derived features, additional temporal features were constructed for quarter, absolute quarter, and monthly progression. Indicator functions for state/agent information, the presence of

competitive state funds, and differences between other engineered features were constructed to describe feature relationships, external influences, and possible interactions. The indicator function features also supported dimensionality reduction.

Feature transformation can take many forms. Equal-frequency binning, for example, is a form of transformation, albeit slightly more complicated than other more common transformations. Data features in the policy dataset, especially high-cardinality geographic, could provide a unique transformation opportunity if the feature values $x_c^{(i)} = c_k \in \{c_1, ...c_K\}$ had a one-to-one relationship with external data $d_r \in \{d_1, ..., d_k, ..., d_R\}$ **s.t.** $K \leq R$. In which case, an assignment function was defined: $f : c_k \mapsto d_r$ and additional features were constructed $f : \mathbf{X}_{:,c} \mapsto \mathbf{X}_{:,d}$. Features representing competitive state funds, tech index, and education index were created with `risk_state`. Urban Influence codes (UIC) obtained from the department of Agriculture website [76] were mapped to FIPS[14] county codes and then to policy dataset zip codes. Transformed features using one-to-one functions are described in Appendix A.1.

3.3 Proposed Features

RQ2: *Given a limited feature space, can additional features be engineered based off current data mining techniques and domain knowledge to capture agent-related information that may influence policy churn?*

As mentioned, for Property and Casualty (P&C) insurance carriers employing independent intermediaries, there exists a potentially influential additional filter between carriers and policyholders. Attempts at constructing conventional client-centric churn models are complicated by agent ownership of the policyholder and the resulting lack of policyholder information. Churn prediction is further hampered by overall low-loyalty and cyclical industry trends. Additional policyholder features can be

[14]Federal Information Processing Standards

engineered from features such as `prem` and geographic information. However, such features do not address the elephant in the room, the influence of the independent agent. Even if carriers were able to collect more policyholder data, churn models would still struggle to account for the actions and/or motivations of the agent.

In Literature Review, [48] proposed a method of feature engineering which involved the development of networks and subsequent construction of features based off the customer interactions within that network. The concept was extended to independent agents (networks) and policyholders (customers). A feature engineering approach was designed to capture information related to agent-policyholder-carrier loyalty and trends. The network approach described by [48] used information about other users within the network to predict behavior of the end-user. In a similar fashion, information about other policyholder behavior under a particular agent may be leveraged to predict policyholder behavior. Specific to the policy dataset, the number of previous quarterly non-renewals and new policies for a given agent may hold latent information pertaining to a given policyholder under the same agent. Additionally, [64] briefly described the construction of trailing agent premium features as a predictor of churn resulting from agent disengagement. The concept was extended to the policy dataset and trailing aggregated agent premium features were engineered as possible additional churn predictors. Therefore, this analysis proposed the construction of higher-level features containing agent-related information for the previous quarter(s).

3.3.1 Policy Trend Features

A matrix reduction function was defined for a given matrix of features $\mathbf{X} \in \mathbb{R}^{N \times D}$. For a categorical feature $\mathbf{x}_p$ with values $p_k \in \{p_1, ...p_K\}$, indicator function I_k, and equation (1) (Section 3.3.2) were applied to obtain the feature value cardinalities. The

reduction function for feature values p_k was then defined:

$$f_{p_k}(\mathbf{X}) = \mathbf{X}^{|p_k| \times D} \textbf{ s.t } x_p^{(i)} = p_k \quad \text{or } \Omega_{p_k} \tag{4}$$

Using equation (4), the dataset was reduced by quarter for feature $\mathbf{x}_q = \text{quarter}_k = q_k$, followed by feature $\mathbf{x}_a = \text{agent}_k = a_k$. The composite function was then defined:

(i) $f_{q_k}(\mathbf{X}) = \mathbf{X}^{|q_k| \times D} \textbf{ s.t } x_q^{(i)} = q_k \quad \text{or } \Omega_{q_k}$

(ii) $f_{a_k} \circ f_{q_k}(\mathbf{X}) = f_{a_k}(f_{q_k}(\mathbf{X})) = (\mathbf{X}^{|q_k| \times D})^{|a_q| \times D} \textbf{ s.t. } x_a^{(i)} = a_k \quad \text{or } \Omega_{(a_k:q_k)}$

The obtained subset of agent_k quarterly policies $\Omega_{(a_k|q_k)}$, indicator function I_k, equation (1), and a one-to-one function for the policy status feature[15] were applied to create the policy trend feature $f : \mathbf{X}_{:,a_k} \mapsto \mathbf{X}_{:,a_{k^*}}$ **s.t.** $\mathbf{x}_{a_k^*}$ was an integer feature value $a_{k^*} \in \mathbb{Z}^+$ representing the agent number of quarterly new policies. The same approach was applied for number of quarterly non-renewals per agent.

Algorithm 4 Policy Trend Features

```
Reduce X^{N×D} to sub-matrices for each quarter
Create sub-matrices for quarterly feature values x_q equal to quarter q_k
for q_k ∈ {q_1, ..., q_K} do X^{|q_k|×D} s.t. x_q^{(i)} = q_k
    Reduce X^{|q_k|×D} to sub-matrices for each agent
    Create sub-matrices for agent feature values x_a equal to agent a_k
    for a_k ∈ {a_1, ..., a_K} do X^{|a_k|×D} s.t. x_a^{(i)} = a_k
        Reduce X^{|a_k|×D} to sub-matrices for new policies
        Create sub-matrices for policy feature values x_c = new policies c_np
        for c_k = c_np do X^{|c_np|×D}
            Create dict(|c_np_k|, a_k)
            Map |c_np_k| to dataset X by agent a_k to create new feature x_p*
            x_p* ← x_a dict(|c_np_k|, a_k)
    return x_p*
```

[15] Categorical feature with values new policy, renew, or non-renew

There were feature engineering hurdles given the *quarterly* breakdown of the dataset and *annual* renewals. Fortunately, the policy trend feature was unaffected by the time interval differences. Previous quarterly counts of new policies and non-renews for an independent agent[16] were obtained using Python pivot tables with results mapped to the policyholder level. With a larger dataset, new policies and non-renew features could lag for numerous quarters. However, given the constraints of the policy dataset, only one lag quarter was used.

3.3.2 Premium Trend Features

A similar technique was also applied to engineer lag features for agent quarterly premium, although several adjustments needed to be made to compensate for the quarter-based feature given annual appearances of policyholder information. First, a function was defined that takes in two matrices of equal sample size and performs an append:

$$f_{[x,x]}(\mathbf{X}_1^{N\times D}, \mathbf{X}_2^{N\times p}) = \mathbf{X}^{N\times(D+p)} = \mathbf{X}^{N\times D*} \tag{5}$$

Additionally, the proposed method used a base sub-matrix approximated from the earliest year of available data. The base year $\mathbf{X}^B$ dataset containing all policies for that year was approximated by using the previous year policies minus new policies (renewal and non-renewal only) :

$\Omega_B = \mathbf{X}^B \approx \mathbf{X}^{B+1}$ **s.t.** $x_c \neq c_{np}$

If said data already exists, then the above approximation method would not be needed. The base represented all active policies as of the previous year Q4. Ω_B was updated quarterly (using equation (5)) with the reduced quarterly matrix *minus* non-renewals (nr) but *with* new policies (np):

(i) $f_{q_k}(\mathbf{X}) = \mathbf{X}^{|q_k|\times D}$ **s.t.** $x_q^{(i)} = q_k$ or Ω_{q_k}

[16]There are over 12k agents reflected in the dataset

(ii) $f_{c_k} \circ f_{q_k}(\mathbf{X}) = f_{c_k}(f_{q_k}(\mathbf{X})) = \left[(\mathbf{X}^{|q_k|\times D})^{|c_k|\times D} \textbf{ s.t. } x_c^{(i)} \neq c_{nr}\right] = \mathbf{X}_{q_k}^{|c_k|\times D}$ or $\Omega_{(c_k|q_k)}$

(iii) $f_{q_k}(\mathbf{X}^B) = (\mathbf{X}^B)^{(N-|q_k|)\times D}$ **s.t.** $x_q^{(i)} \neq q_k = \mathbf{X}^{(B|(-)q_k)}$ or Ω_{B^*}

(iv) $f_{[x,x]}(\Omega_{B^*}, \Omega_{(c_k|q_k)}) = \Omega_B^{\text{updated}}$

The quarterly sample space of agent$_k$ for all annual policies was $\Omega_{(a_k|q_k)}$. As opposed to the policy trend features, each subset reflected *all* agent$_k$ annual policies for that quarter, to include those *not* up for renewal. The indicator function was not needed and instead the policy premiums feature $\mathbf{x}_v$ with positive integer values was aggregated for the reduced sample space to create the new feature value $v_{k^*} \in \mathbb{Z}^+$ of agent quarterly premium for all active policies.

$f(\Omega_{(a_k|q_k)}) = \sum_{i=1}^{(|a_k|:|q_k|)} x_v^{(i)} = v_k^*$

The one-to-one function was then $f_v(\mathbf{X}_{:,a_k}) \mapsto v_k^*$ creating the new feature $\mathbf{X}_{:,v*}$ for agent quarterly total premium. All programming was performed in Python using primarily `numpy` and `pandas`.

Algorithm 5 Premium Trend Features

1: Approximate base year if necessary
2: $\Omega_B \leftarrow \mathbf{X}^B \approx \mathbf{X}^{B+1}$ **s.t.** $x_c \neq c_{np}$
3: Update quarterly
4: $\Omega_{B^*} \leftarrow f_{q_k}(\Omega_B)$
5: Follow Algorithm 1 steps 3 to 6 to find sample space $\Omega_{a_k|q_k}$
6: Sum premiums for given agent$_k$ $\sum_{i=1}^{\Omega} x_v^{(i)}$
7: Assign aggregated premium v_k^* to occurrences of a_k in the main dataset
8: $f_v(\mathbf{X}_{:,a_k}) \mapsto v_k^*$

3.4 Correlation

Gradient boosted tree-based ensemble models, such as XGB, are generally considered robust to correlated features. Therefore a detailed analysis of correlated feature impact on model performance would normally seem excessive. However, the methods

introduced in the following chapter attempt to provide deeper insight into performance driven feature selection, specifically for the premium and policy trend features, thereby necessitating some considerations for correlation. Furthermore, given the high correlation between features, there was likely numerous instances of underlying[17] features that affected intra-feature group correlation as well as feature-response correlation. Therefore, a general analysis of policy dataset feature correlation was performed.

The policy dataset consisted of numerous high-cardinality categorical features that introduced challenges related to overfitting and complexity. In addition, the categorical features describing policy attributes had significant overlap and were therefore highly dependent (correlated). For example, often times certain values for one categorical feature dictated the range of values for another categorical feature.

(i) For $\mathbf{X} \in \mathbb{R}^{N \times D}$ and categorical features $\mathbf{x}_p, \mathbf{x}_q \in \mathbf{X}$ **s.t.** $\mathrm{x}_p^{(i)} = p_k \in \{p_1, ...p_K\}$ or set P_k and $\mathrm{x}_q^{(i)} = q_k \in \{q_1, ...q_K\}$ or set Q_k

(ii) **if** $\mathrm{x}_p^{(i)} = p_k \implies \mathrm{x}_q^{(i)} = q_k \in Q \subset Q_k$.

Underlying correlations described by [69] were likely present within the policy dataset as well. In which case, a feature $\mathbf{x}_a$ may have appeared to have predictive importance. However, the later inclusion of an additional feature $\mathbf{x}_b$ revealed $\mathbf{x}_a$ had little to no predictive importance and was simply correlated with $\mathbf{x}_b$, a feature that had predictive importance.

3.4.1 Spearman

Given the mix of continuous and ordinal feature values, Spearman's rank correlation ρ was used to test numerical features [14]. Let $\texttt{rank}(\mathrm{x}_j^{(i)})$ denote the rank of a numerical

[17] Lurking [69]

feature $x_j^{(i)}$ and $d_i = \texttt{rank}(x_j^{(i)}) - \texttt{rank}(x_k^{(i)})$, then:

$$\rho = 1 - \frac{6 \sum_{i=1}^{N} d_i^2}{N(N^2 - 1)} \tag{6}$$

The Python's `pandas` library correlation function with parameter `spearman` generated a correlation matrix for analysis. Possibly impactful correlations were observed prior to testing.

The `UI_code` feature was assigned based off zip code and was intended to provide amplifying geographic information. In theory, it should have some correlation with other geographic features, though preferably not *strong* correlation. As seen in Figure 2(a), `UI_code` showed a moderate to strong negative correlation with `county_freq` (-.66). Given the definition of `UI_code` , it was inferred that more frequently occurring counties have lower urbanity scores and therefore tend to be from larger metropolitan areas.

Additionally, there was a strong correlation between `agnt_freq` and both `prev_non` and `prev_NB` (new policy). There was also a moderate to strong correlation between `prev_non` and `prev_NB`. This triangle of stronger correlations was likely partially the result of an underlying variable, volume[18]. `agnt_freq` was intended to represent agencies that are used more frequently by the carrier spanning the entire dataset. However, this feature also captured information pertaining to the number of new policies and policies up for renewal. If broken down by quarter; new policies, non-renewals, and renewals would add up to the quarterly agent frequency count. This is discussed further in recommendations with representational (derivative) features as a possible option to counter correlation issues.

Given the definition of small commercial policies in Chapter 1, there was an expected strong negative correlation between `prem` and `prem_freq`, demonstrating the infrequency

[18]The amount of business a given agency holds with the carrier

of higher premiums in the dataset (Figure 2(b)). A weak correlation between `prem` and `rate_chng` given the `rate_chng` feature was in actual dollar amounts, was a probable reflection of larger premiums seeing larger attendant dollar amount changes in response to rate changes. A slightly stronger correlation between `tenure` and `prem` may be indicative of higher-premium policyholder loyalty.

The complete Spearman correlation matrix can be found in Appendix A.2. Many of the trend-related group of features showed heavy intra-group correlations and other moderate correlations. Additional volume-related information was captured by the agent premium trend features wherein agencies with higher total premiums in previous quarters have more new polices, renewals, and non-renewals. Therefore, as with `agnt_freq`, correlation was influenced by an underlying (volume) variable.

	UI_code	county_freq	agnt_freq	prev_non	prev_NB
UI_code	1.000000	-0.663693	-0.068297	-0.080250	-0.061345
county_freq	-0.663693	1.000000	0.060796	0.071606	0.087678
agnt_freq	-0.068297	0.060796	1.000000	0.795838	0.700670
prev_non	-0.080250	0.071606	0.795838	1.000000	0.603419
prev_NB	-0.061345	0.087678	0.700670	0.603419	1.000000

(a) *Matrix 1*

	prem_	prem_freq	rate_chng	tenure_
prem_	1.000000	-0.769889	0.280922	0.129098
prem_freq	-0.769889	1.000000	-0.243719	-0.074090
rate_chng	0.280922	-0.243719	1.000000	0.398074
tenure_	0.129098	-0.074090	0.398074	1.000000

(b) *Matrix 2*

Figure 2: *Correlation Matrices*

3.4.2 χ^2 Test

For categorical features, contingency tables with χ^2 tests were used to check correlations [10]. The majority of the categorical features showed fairly strong correlation using χ^2 statistic and contingency table frequency counts. The χ^2 statistic is given by the following equation using frequency count observed values $x_j^{(i)} = x_i$ and expected values E_i.

$$\chi^2 = \sum \frac{(x_i - E_i)^2}{E_i} \tag{7}$$

The following hypothesis test was used:

H_0: The features are independent

H_1: The features are dependent

The few categorical feature pairs that did not reject the H_0 involved absolute temporal features and some of the indicator features. The categorical features were expected to be correlated in most cases, since they are used to segment/describe policies. For example, `pgrm` correlated highly with other categorical features as the specific program describing a given policy dictates `seg_stragey`, `industry`, and `UW`. There were numerous examples in the policy dataset wherein correlations (or dependencies) should exist. A table of p-values for categorical features can be found in Appendix A.2.

In order to verify that all engineered features showed correlation with the response, numerical engineered features were discretized using 5 interval bins and tested against the response using χ^2. All features returned p-values < 0.05 with the derivative premium trend feature `lag_diff_abs`[19] having the greatest p-value ≈ 0.0165.

[19] Absolute value of the difference between lag_1 and lag_2

CHAPTER IV

METHOD

The saying "garbage in, garbage out" as it applies to the data inputed into a model, is a truism that is directly tied to the features selected. The features ultimately selected to serve as predictors are so impactful, they generally determine the actual type of model used. While more features represent more information for a model to render predictions from, the attendant increased dimensionality comes at a cost. [43] succinctly summarized this tradeoff as a problem associated with minimizing theoretical costs h while keeping model error rates below a certain threshold[20]. The additional risk of over-fitting given variance-bias tradeoff in machine learning is reviewed in 4.2.4. Guyon et al. [24] described the benefits of feature selection as including enhanced interpretability, reduced complexity, and increased generalization. While [65] primarily defined the purpose of feature selection as the elimination of otherwise irrelevant and/or redundant features in order to improve model performance, [24] gave a more appropriate definition while describing the focus of her research. For the purposes of this analysis, feature selection is defined as the following:

The process of constructing and selecting appropriate subsets of features that will optimize model generalization and overall performance.

For tree-based methods in machine learning, the process by which predictions are derived gives users the added advantage of assessing feature importance using multiple metrics. A more in depth review of tree-based methods is available in Appendix B. Additionally, [34] offers a short review of relevant concepts with intuitive illustrations. In Figure 3, the 2-D visual of a typical tree index partitioning a feature space is given by Hastie et al. [28]. The generic tree-based algorithm is shown divideing the feature space X into 4 different regions $R_1, R_2, ..., R_4$ at $t_1, ..., t_4$ decision points. The tree-based model is

[20]Referred to as the efficiency frontier

of the form:

$$f(X) = \sum_{m=1}^{J} c_m I(x \in R_j) \tag{8}$$

with indicator function $I(x)$ and constant = c_m. There are several types of feature

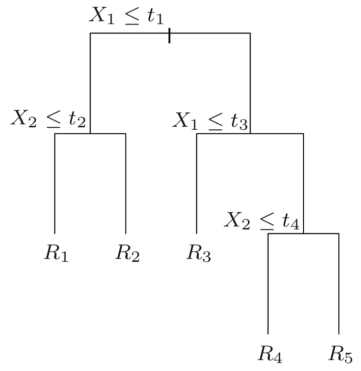


Figure 3: *Tree-based Partitioning* [28]

selection approaches. For this analysis, three main feature selection approaches are reviewed: wrapper, filter, and embedded methods.

Filter methods are based off the *dataset* rather than the model and are typically well suited for high dimensional data when computation time is a factor [43]. The Fisher Criterion is an example of a filter method that assesses feature importance independent of the learning algorithm and is therefore model agnostic [24]. *Embedded* methods differ from filter methods in that supervised learning algorithms are used to score features. Embedded methods are run internally by the model and are often offered as additional model functions. For example, `sklearn` modules for both `Random Forest` and `Decision Tree Classifier` have the added function of ranking inputted features after training. For the purpose of feature space reduction, embedded methods assign importance weights in the form of coefficients to features. Follow on algorithms referred to as `transformers`, are then able to use the coefficient weights to reduce the feature space according to feature importance and a given threshold. *Wrapper* methods are brute-force

algorithms that construct subsets of features for recursive model testing. As such, wrapper methods are generally time-intensive, performance-based (or driven), and highly dependent on the model type [65]. Candidate features are added or eliminated based on performance or feature importance changes until an optimal subset is created.

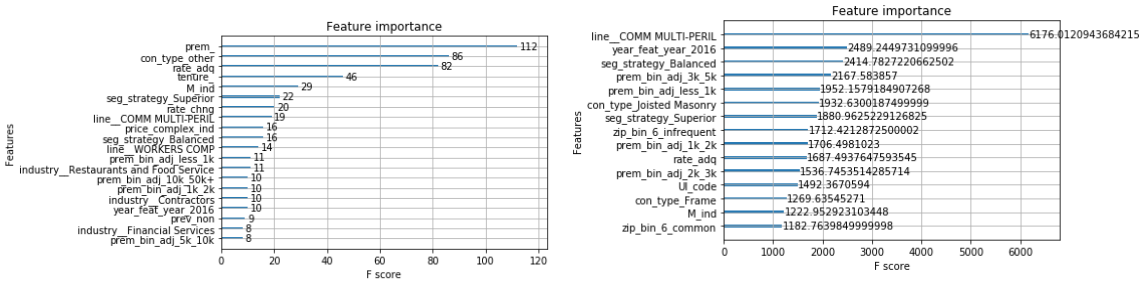
In Section 4.1, the embedded method used for this analysis is discussed along with two transformers. Section 4.2 describes the metrics and validation process used for evaluating relative feature importance as well as feature impact on model performance. Two basic wrapper method algorithms are constructed in Section 4.3 to allow additional analysis of engineered feature effects on model performance. The chapter concludes in Section 4.4 with a brief overview of the `XGBoost` model (XGB).

4.1 Feature Selection Metrics

XGB [76] features an embedded method with several options. The Fscore metric is the default and simplest feature importance metric available; it assigns coefficient weights to features according to the attention a given feature received (from the model) during the training process. For plotting purposes, raw scores are used to visualize the relative importance of features. Given feature $\mathbf{x}_p$, the Fscore (for tree-based methods) is the sum of the number of splits for a given feature $s_i(\mathbf{x}_p)$ across all base tree models M and represents the overall model's use of the feature $\mathbf{x}_p$.

$$Fscore(\mathbf{x}_p) = \sum_{i}^{M} s_i(\mathbf{x}_p) \tag{9}$$

The embedded method for XGB is called using the `feature importance` tool available through [76]. The tool offers a built-in plotting function (Figure 4) and list of ranked features with weight coefficient (Figure 23).



(a) *Fscore* (b) *Gain*

Figure 4: *Metric Comparison*

Another common metric used with embedded methods is *Information Gain* [43]. While Fscores were used for this analysis due to the complexity of the model and the emphasis on wrapper methods, Gain was used as an additional analysis tool primarily for identifying significant feature interactions.

James et al. [31] provided an introduction to tree-based methods using measurements of *entropy*[21] to determine splits (or decisions) for a given feature (or node). Entropy $\texttt{Ent}(X)$ can be broadly defined within the field of information theory as a measurement of information content. When training a model in a supervised manner, the higher the uncertainty (or impurity), the higher the contained information. Therefore, it follows that a *gain* in information would be the difference between entropy prior to the decision split and entropy after the decision split. In other words, the reduction in uncertainty resulting from a decision. Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the training set with feature $\mathbf{x}_p$ as a given decision point, Gain is then defined as:

$$\texttt{Gain}(\mathbf{X}) = \texttt{Ent}(\mathbf{X}) - \texttt{Ent}(\mathbf{X}|\mathbf{x}_p) \tag{10}$$

Information Gain can then be summed for all nodes for feature $\mathbf{x}_p$ and used as a metric for comparison and subsequent ranking [43]. Additionally, feature *interactions* take on a

[21] or Gini index [31]

different meaning for tree-based methods than for General Linear Models (GLM) [28]. Since a tree structure is essentially connected feature nodes branching toward predictions (leaves), the importance of a given feature *interaction* can be measured using Gain as well. Gain for a given feature interaction is measured as the entropy difference from before the first feature (of the interaction) split and after the final feature (of the interaction) split [75]. The difference gives the interaction's reduction in entropy which can then be used for gauging importance. In machine learning, Gain can often present bias towards features with larger domains and was therefore used as a secondary metric for feature importance [43].

Additional details on individual and interaction metrics for each candidate feature were obtained using the `XGBoost Feature Interaction Reshaped` (XGBFIR) [75] algorithm. XGBFIR is a model output parser that records embedded feature importance information, to include 2nd and 3rd level interactions, into an excel file (Figures 27 & 28).

Figure 4 displays a clear difference in ranking between metrics. Differences between Fscores and Gain are explainable given the type of information returned for the metrics. For example, a categorical feature with a high Gain score will not necessarily be used more often for decision nodes, and vice versa. While the different metrics did assign importance to some of the same features, many of the continuous features did not score comparatively well for Gain. The absence of `prem` in particular raised Gain bias concerns. However, this analysis was primarily concerned with model performance changes given engineered features and the feasibility of constructing a useful retention model given current modeling techniques. As such, a deeper analysis of embedded feature importance metrics was outside the scope of this analysis. It was sufficient to acknowledge the existence of Gain bias towards larger domain categorical features and the simplicity of the Fscore for determining model preference for a given feature.

An alternate approach to embedded feature metrics that accounts for the differences between Fscore, Gain, and other metrics[22] by [42] can be found in Appendix B.4.

4.1.2 Transformers

The `SelectFromModel` (SFM) [62] algorithm performs automated feature space reduction based off coefficient weights with a set threshold τ_w. Reduction is typically performed after model training and prior to performance testing, resulting in a reduced matrix of features (dimensionality) with increased generalization. The SFM algorithm works well with models that assigned feature importance metrics in the form of coefficient weights to features, such as XGB. τ_w is determined by a percentage of the mean feature weights. For $0 < c \leq 1$, matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, and w_j = feature weight coefficient for feature $\mathbf{x}_j$.

$$\tau_w = c \times \frac{1}{D} \sum_{j=1}^{D} w_j \tag{11}$$

Algorithm 6 Reduced Features

function SELECT FROM MODEL($\mathbf{X}$, $c_{\%}$)
  $\tau_w \leftarrow c \times \frac{1}{D} \sum_{j=1}^{D} w_j$
  for i from 1 to D **do**
    if $w_j < \tau_w$ **then** discard $\mathbf{x}_j$
  return Reduced form $\mathbf{X}^{N \times R}$

`Recursive Feature Elimination` (RFE) is a transformer that uses a hybrid embedded/wrapper feature selection method to recursively train diminishing subsets of data until achieving a predetermined feature space dimensionality [22]. The RFE algorithm available in the `sklearn` library [59] utilizes a given model's embedded selection method (one that assigns coefficient weights to features) to drop features ranked as less important when selecting a subset to train. While *top-down* sequential

[22] A metric called Cover is also offered by the XGB embedded method

selection methods are generally considered suboptimal due to *nesting effects*[23] [52], the use of a tree-based model embedded method to rank features prior to every elimination mitigates the chances of predictive features being eliminated early on due to correlation bias. However, the RFE transformer is limited by the effectiveness of the embedded importance metric and features are eliminated based off their individual scores without taking interactions into account. Additionally, the transformer does not use performance criteria for determining final reduced feature space dimensionality and will continue eliminating features until the desired end-state rank is achieved. Furthermore, it is a brute force method that requires considerable[24] processing time. RFE was initially included in this analysis as a comparison for the sequential forward selection (SFS) wrapper methods.

4.2 Performance Metrics

Given the imbalance of the dataset and emphasis on practical application, certain metrics were given emphasis during performance evaluation. The metrics used for wrapper method feature selection, model comparison, and overall model performance against a test set are discussed in the following section.

4.2.1 Confusion Matrix

A confusion matrix is a common metric used for measuring classifier performance [28, 43]. It provides a practical way to monitor classifier raw scores and is used to compute common model performance metrics. In Figure 5, predicted values are columns and actual values are represented by rows. For the policy dataset, negative values = 0 are assigned to non-churn while positive values =1 are assigned to churned policyholders. Therefore when viewing confusion matrices in Chapter 5, the correct

[23] Eliminated features cannot be selected later on

[24] Using the hardware detailed in Chapter 5, the transformer had to be run overnight

predictions are diagonal from upper left to bottom right with the bottom right partition containing correct predictions for the minority class (churn).

The binary response feature left the model susceptible to two main errors: False Positive (FP) or *Type I* errors and False Negative (FN) or *Type II* errors. FP is the number of non-churn policyholders that the model classified as churn. FN is the number of churn policyholders that the model classified as non-churn. It follows then, that True Positives (TP) were correctly classified churn and True Negatives (TN) were correctly classified non-churn. The probability of making a Type II error is the False Discovery Rate (FDR), therefore the probability of not making a Type II error is the *power*. Power is also often referred to as sensitivity and *recall*.

$$\texttt{FDR} = \frac{FP}{TP+FP} \qquad \texttt{Power} = 1 - \frac{FP}{TP+FP} \qquad \texttt{Recall} = \frac{TP}{TP+FN}$$

The conventional metric for model performance most often used in machine learning, *accuracy*, is the ratio of correct predictions over all predictions.

$$\texttt{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Given the class imbalance problem inherent in the policy dataset and discussed in Section 3.1.3, the accuracy metric was not well suited as a performance metric [6]. The recall metric specifically measures the model's ability to identify the minority class.

$$\texttt{Precision} = \frac{TP}{TP+FP} \qquad \texttt{F-measure} = \frac{2TP}{2TP+FP+FN} \text{ [38, 42]}$$

However, while recall returns a ratio of correct minority class predictions over all minority class samples, it can introduce excessive FP thereby reducing the *precision* (or Positive Predicted Value) and overall utility of the model for practitioners. The F-measure is the harmonic mean of precision and recall and was preferable to either metric alone given the dataset. Harmonic means are generally preferable for use with rates and are the reciprocal of the arithmetic mean of the reciprocals of values [63].

As pointed out by Lopez et al. [41], minority classes in an imbalance dataset are usually of greatest interest and cost to practitioners. The policy dataset did not deviate

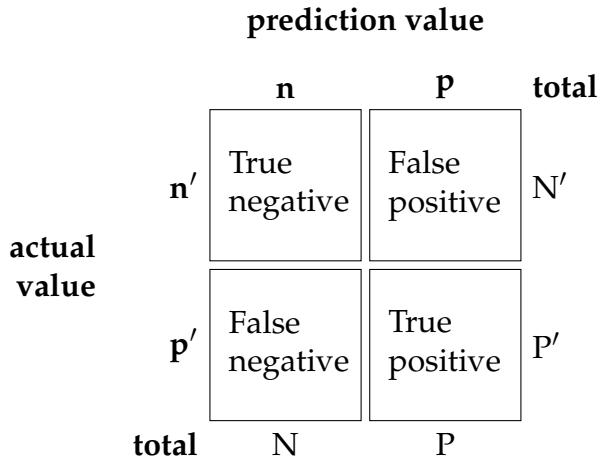


Figure 5: *Confusion Matrix diagram*

from the norm in this respect. Accurately predicting or failing to predict a would-be churner will not carry equal error penalties in practical application. As such, Type I errors (FP) and Type II errors (FN) did not carry the same weight. For example, given a confusion matrix output for a retention model, a FN is much more costly to business than a FP. The FN identified the policy as a non-churn, when in actuality the policy churns. The FP identities the policy as a churn when it is actually a non-churn. That being said, if a given model is optimized for recall, it may return an output with an impressive percentage of TP but an unacceptable 50% FP.

4.2.2 ROC Curve & Log-loss

Another popular metric used for classifiers, particularly on imbalanced datasets, is the Receiver Operating Characteristic (ROC) curve [78] shown in Figure 6. The ROC graph plots TP rate (TPR) on the y-axis vs. FP rate (FPR) on the x-axis relative to a 45 degree x = y line.

$$\texttt{TPR} = \frac{TP}{TP+FN} \qquad \texttt{FPR} = \frac{FP}{TN+FP}$$

The area under the line x = y line is .5 and equivalent to random guessing. The area under the curve (AUC) for the ROC quantifies the performance of a model relative to random

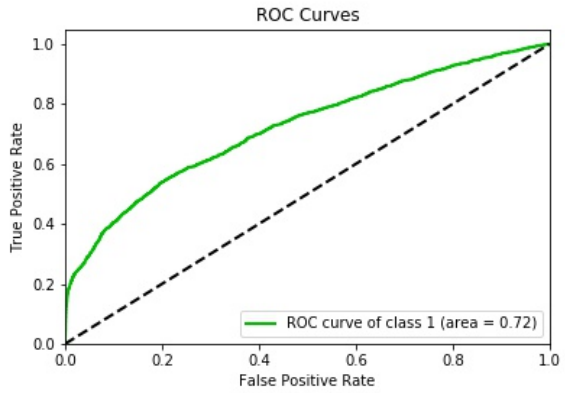


Figure 6: *ROC Curve*

chance represented by the x = y line. The ROC curve captures the trade off between TPR and FPR with optimal curves steeply moving up the y axis prior to curving along the x-axis. The AUC represents the performance of the model for all possible threshold values τ_{cm}[25]. The integral is computed in a standard way treating the ROC curve as parametric with dependent variable τ, therefore TPR $= y(\tau)$ and FPR $= x(\tau)$. Using substitution gives:

$$\int_0^1 y(\tau)x'(\tau)d\tau \text{ or } \int_0^1 \texttt{TPR}(\tau)\texttt{FPR}'(\tau)d\tau \tag{12}$$

While the AUC presents a convenient scalar value for comparison between models, the ROC curve is needed to observe actual TPR and FPR (also referred to as *fallout*) tradeoff. From a practical cost-sensitive standpoint [41], the XGB churn model in this study was measured by its ability to control False Discovery Rate (FDR) while increasing the recall (TPR). This can be interpreted as the ability to identify an acceptable subset of predicted churn containing the highest percentage of actual churners. Log-loss is related to cross entropy and is used to judge the reduction in uncertainty the model achieves [28]. The model outputs probabilities for $\hat{y}^{(i)}$ that are assigned to a binary class depending on τ_{cm}, which defaults to 0.50. Log-loss captures how close the actual prediction was to the test

[25] τ_{cm} was the setting that assigns predictions to the positive class based off their probability values

value. If $\hat{p}$ is the predicted probability for $\hat{y}$ and $y \in \{0, 1\}$ then the difference between the predicted probability and test value for each sample follows a Bernoulli distribution $\hat{p}^y(1-\hat{p})^{1-y}$. The Log-loss is simply the averaged logarithm of the test sample:

$$\texttt{Log-loss} = \frac{1}{K}\sum_{i=1}^{K} -(y\texttt{ln}(\hat{p}) + (1-y)\texttt{ln}(1-\hat{p})) \tag{13}$$

4.2.3 Cross-validation

Cross-validation is an established wrapper approach that obtains performance metrics using a random sampling of data from a given test set (or validation set) to repeatedly train and subsequently test model performance [43]. Performance metrics are then averaged to give a better estimate of model generalization. In this sense, cross-validation can reveal if a given model was over-fitting the training data.

The K-fold cross validation method uses k randomly and equally partitioned disjoint sets *or folds* of the validation set in order to measure model performance. The policy validation set was a subset of the training data (fiscal Q1 2018) identified early in the preprocessing phase, subsequent steps were taken to prevent possible data leakage into the validation set. Stratified versions of K-fold attempt to preserve the class balance of the larger set in each of the folds, a method frequently used with imbalanced datasets [58]. Given a validation set $\mathbf{X}^{(\text{valid})}$, model $\mathcal{P}$ iteratively trains on $n-1$ folds and validates on the nth fold. Defining $\hat{f}^{-\mathcal{K}}(x)$ as fitted function with omitted k^{th} part, the misclassification error is averaged for all data points [31].

$$CV_{(k)} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y^{(i)}, \hat{f}^{-\mathcal{K}(i)}(\mathbf{x}^{(i)})) \tag{14}$$

The `sklearn` algorithm `RepeatedStratifiedKfold` (RSK) [58] version of the following algorithm was used. RSK preserves class proportions in the random samples (or folds) and is generally recommended for imbalanced datasets.

Algorithm 7 K-Fold Stratified Cross Validation

Require: $\mathcal{K} : \{(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(n)}, y^{(n)})\} \mapsto \{\mathbf{X}^{(1)}_{fold}, ..., \mathbf{X}^{(k)}_{fold}\}$

function K-FOLD STATIFIED($\mathcal{K}, \mathcal{P}, \mathbf{X}^{(\text{valid})}$, number of folds = k)
  The defined function $\mathcal{K}$ maps the validation set to the k intervals.
  $\dot{\bigcup_{i \in k}} \mathbf{X}^{(i)}_{folds}$ for $1 \leq i \leq k \leftarrow \mathcal{K}(\mathbf{X})$
  for i from 1 to k **do**
    $\hat{f}^{-\mathcal{K}(i)}(\mathbf{x}^{(i)}) = \mathcal{P}(\mathbf{X} \backslash \mathbf{X}^{(i)}_{folds})$
    for $(\mathbf{x}^{(j)}, y^{(j)}) \in \mathbf{X}^{(i)}_{folds}$ **do**
      $Err^{(j)} \leftarrow \mathcal{L}(y^{(j)}, \hat{f}^{-\mathcal{K}(j)}(\mathbf{x}^{(j)}))$
  return Err

4.2.4 Learning Curve

The variance-bias tradeoff in machine learning describes the dilemma in which predictive models with higher variance tend to have lower bias and vis-versa. This tradeoff is referred to as model over-fitting and under-fitting [43, 28] and is observable for high dimensional data using a Learning Curve shown in Figure 7. When visualizing a line fitting on a 2-D surface, the smoother low-variance fit is an example of an under-fitting model. Conversely, the less smooth (erratic looking) low-bias fit characterizes a model that is over-fitting the training data. Ideally, a model is preferable if it is able to fully leverage training data while maintaining the ability to *generalize* predictions for test data. High-variance, low-bias models tend to over-fit (essentially memorize) the training data and fail to generalize well for the test data. Black box methods, such as ensembles and neural networks, are examples of models more susceptible to over-fitting. Conversely, high-bias, low-variance models tend to under-fit the training data (essentially ignore) the information embedded in the training data and

again fail to perform well on test data. Models that tend to under-fit include regression and basic statistical models given non-linear datasets. Decomposing the variance-bias relationship gives the following equations:

(i) $\texttt{bias}(\hat{y}^{(i)}) = E(\hat{y}^{(i)} - y^{(i)})$

(ii) $\texttt{var}(\hat{y}^{(i)}) = E((\hat{y}^{(i)})^2) - (E(\hat{y}^{(i)}))^2$

(iii) $\texttt{Expected Error} = E[(y^{(i)} - \hat{y}^{(i)}] = (\texttt{bias}(\hat{y}^{(i)}))^2 + var(\hat{y}^{(i)}) + \texttt{Irreducible Error}$

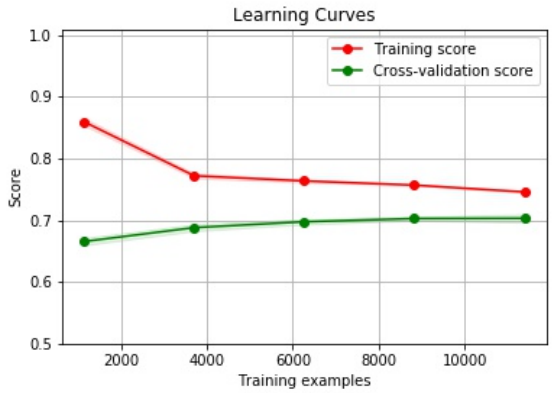


Figure 7: *Learning Curve*

The XGB model used with the policy dataset was an example of a black box model using high dimensionality data. Therefore, special consideration was given for model tendency to over-fit. There is no single metric that will score the variance-bias tradeoff. However, visually monitoring the relative difference in a given model's training and cross-validation scores as shown in Figure 7, can assist in identifying over-fitting. Learning curves plot training scores along with cross-validation scores and were the primary method used to observe for over-fitting [36].

4.3 Wrapper Methods

RQ3: *Can an effective performance-based feature selection wrapper method be constructed to determine engineered feature contribution and improve model performance?*

The following Sequentially Forward Selection (SFS) [52] wrapper methods attempted to reduce feature space dimensionality and improve performance by systematically

adding only candidate features that contribute to model performance. In addition, the algorithms were simple enough to allow for closer observation of candidate feature selection relative to the presence or absence of other candidate features.

4.3.1 Base + 1 Method

The Base + 1 (B1) method was an SFS method based off the feature selection algorithm used by [23]. B1 established a base set of features then iteratively tested candidate features individually using the base feature set. Candidate features that improved model performance were added to the dataset and used for future modeling. Metric thresholds for F-measure and AUC replaced the accuracy metric used by [23].

The B1 method presupposed the use of an appropriate model $\mathcal{P}$, training dataset $\mathbf{X}^{(\text{train})}$ correlated to a response vector $\mathbf{y}^{(\text{train})}$, and the selection of an appropriate base set of features $\mathbf{X}^{N\times B} = \mathbf{X}^{B}$. Candidate feature vectors $\mathbf{x}_p^*$ (or $\mathbf{X}_{:,p}$) from a matrix of features $\mathbf{X}^{N\times P^*} = \mathbf{X}^{P^*}$ **s.t.** $\mathbf{x}^* \in \{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$ were iteratively added to the base set. The base set was then tested using cross-validation on the validation set $\mathbf{X}^{(\text{valid})}$. Metrics m varied depending on the dataset and the threshold τ_m was adjusted as the base set of features was built for testing the remaining candidate features. Features that increase $m > \tau_m$ were selected for future modeling. First, a function was defined to append selected candidate features to the policy dataset $\mathbf{X} \in \mathbb{R}^{N\times D}$:

$$g(m^*, \tau_m, \mathbf{X}_{:,p}) = \begin{cases} \left[\mathbf{X}, \mathbf{X}_{:,p}\right] = \mathbf{X} & \text{if } m^* > \tau_m \\ \mathbf{X} & \text{o.w.} \end{cases}$$

Given a set of candidate features $\mathbf{X}^{P^*}$, validation dataset $\mathbf{X}^{(\text{valid})}$, model $\mathcal{P}$ with parameters θ, metric m, and threshold τ_m, the append function defined by equation (5) was used to define the following sequence for updating the policy dataset with performance enhancing candidate features. Given the matrix of candidate features $\mathbf{X}^{P^*}$

then $\forall \mathbf{x}^* \in \{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$:

(i) $f_{[x,x]}(\mathbf{X}^B, \mathbf{X}_{:,p}) = \left[\mathbf{X}^B, \mathbf{X}_{:,p}\right] = \mathbf{X}^{N\times(B+1)} = \mathbf{X}^{B^*}$

(ii) $f_{\text{kfold}}(\mathcal{P}(\mathbf{X}^{B^*}), \mathbf{X}^{(\text{valid})}, \theta) = m^*$

(iii) $g(m^*, \tau, \mathbf{X}_{:,p}) = \mathbf{X}$

Algorithm 8 B+1 Wrapper Method

1: Determine a set of candidate features $\{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$
2: Append each $\mathbf{x}^*$ to the training set $\mathbf{X}^{(\text{train})}$ and run the model $\mathcal{P}$
3: **for** $\mathbf{x}^* \in \{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$ **do**
4: $\quad \mathbf{X}^* \leftarrow \left[\mathbf{X}^{(\text{train})}, \mathbf{x}^*\right]$
5: $\quad$ Train $\mathcal{P}$ and obtain performance metrics m^* using cross-validation
6: $\quad m^* = f_{\text{kfold}}(\mathcal{P}(\mathbf{X}^*), \mathbf{X}^{(\text{valid})})$
7: $\quad$ **if** $m^* > \tau$ **then** $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{x}^*]$
8: $\quad\quad m^*$ meets the threshold score τ, candidate $\mathbf{x}^*$ is added to the dataset
9: $\quad$ **else** $\mathbf{x}^*$ is not added

4.3.2 Strictly Additive Method

The Strictly Additive (SA) method was a SFS wrapper method similar to B1, save a few key differences. Initially, the base feature set $\mathbf{X}^B$ consisted of only one feature, `prem`. Candidate features were added to the base set as before. However, if the candidate feature performance metric exceeded the threshold $m^* > \tau_m$, the candidate feature was added to the *base* set and the threshold was updated with the improved metric score $\tau_m = m^*$. An adjustment was made to the above B1 equations and algorithm with all notation remaining the same. First, a function was defined that augmented the base set $\mathbf{X}^B$ depending on the candidate feature contribution to model performance:

$$g(m^*, \tau_m, \mathbf{X}_{:,p}) = \begin{cases} [\mathbf{X}^B, \mathbf{X}_{:,p}] = \mathbf{X}^B \text{ and } \tau_m = m^* & \text{if } m^* > \tau_m \\ \mathbf{X}^B, \tau_m & \text{o.w.} \end{cases}$$

Given a matrix of candidate features $\mathbf{X}^{P^*}$, validation dataset $\mathbf{X}^{(\text{valid})}$, model $\mathcal{P}$, metric m, and threshold τ_m, and the append function defined by equation (5), a sequence is defined for updating the base set of features. Given a matrix of candidate features $\mathbf{X}^{P^*}$ then $\forall \mathbf{x}^* \in \{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$:

(i) $f_{[x,x]}(\mathbf{X}^B, \mathbf{X}_{:,p}) = [\mathbf{X}^B, \mathbf{X}_{:,p}] = \mathbf{X}^{N\times(B+1)} = \mathbf{X}^{B^*}$

(ii) $f_{\text{kfold}}(\mathcal{P}(\mathbf{X}^{B^*}), \mathbf{X}^{(\text{valid})}, \theta) = m^*$

(iii) $g(m^*, \tau, \mathbf{X}_{:,p}) = \mathbf{X}^B$

Algorithm 9 Strictly Additive Method

1: Determine a set of candidate features $\{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$
2: Append each $\mathbf{x}^*$ to the base set $\mathbf{X}^B$ and run the model $\mathcal{P}$
3: **for** $\mathbf{x}^* \in \{\mathbf{x}_1^*, ...\mathbf{x}_P^*\}$ **do**
4: $\quad \mathbf{X}^* \leftarrow \left[\mathbf{X}^B, \mathbf{x}^*\right]$
5: $\quad$ Train $\mathcal{P}$ and obtain performance metrics m^* using cross-validation
6: $\quad m^* = f_{\text{kfold}}(\mathcal{P}(\mathbf{X}^*), \mathbf{X}^{(\text{valid})})$
7: $\quad$ **if** $m^* > \tau$ **then** $\mathbf{X}^B \leftarrow [\mathbf{X}^B, \mathbf{x}^*]$ **and** $\tau \leftarrow m^*$
8: $\quad$ **else** $\mathbf{x}^*$ is not added and τ is not updated

4.4 Model

The XGB model has many aspects that build on existing tree-based methods, to include: gradient boosting, bagging, shrinkage, and ensembles. A more thorough breakdown of the XGB process can be found in Appendix B.5. Additionally, Chen et al. [7] authored a detailed description of the XGB model and its capabilities. The below review assumes an understanding of the aforementioned aspects and primarily follows the mathematical framework presented by [7]. Appendix B.5 reviews requisite methods and provides an illustrative example of the scoring system described by [7], intermediate equations are also included for clarity.

[7] introduced the XGB as a gradient boosting decision tree *system* to develop fast and accurate supervised learning algorithms. At its root, XGB is a generalized tree boosting algorithm that uses a regularization term to penalize complexity [46]. XGB has been highly successful in competitions in part due to its scalability, enabling practitioners to run large high-dimension datasets using limited CPUs. Bagging is incorporated along with a modified form of boosting using a weight shrinkage[26] technique. An approximated exact greedy algorithm variant is used for split decisions as well as a weighted quantile sketch [7] algorithm. XGB is sparsity aware and actually exploits sparsity such that computation complexity is controlled, allowing for superior run times relative to non-sparsity aware models. Where tree-based models are often burdened with sorting data, XGB uses column blocks [7] for parallel learning to harness all machine resources. As opposed to other tree-based ensemble classifiers, the XGB system uses an ensemble of parameterized regression trees rather than decision trees as base learners. Regression trees are non-parametric models that use a recursive binary splitting [31] greedy approach to predictive modeling.

For a given dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, [7] defines the structure q of a given tree as a mapping from a root node through multiple internal nodes, concluding at terminal nodes called *leaves*. Also referred to as the leaves *index*, structure q contains all the decision rules for a given tree F_m of M additive base learners and determines the final leaf values. Thus $q(\mathbf{X}) = T$, the total number of leaves for $F_m(\mathbf{X})$. For the XGB system, leaves contain scores $w_q \in \mathbb{R}^T$ that are summed for the data point final prediction. Therefore, a given base learner $F(\mathbf{X}) = w_{q(\mathbf{X})}$ and it follows that $\hat{y} = \sum w_{q(\mathbf{X})}$. Given $\mathbf{X} \in \mathbb{R}^{N \times D}$, an XGB prediction can be defined:

$$\hat{y}^{(i)} = \psi(\mathbf{x}^{(i)}) = \sum_{m=1}^{K} F_m(\mathbf{x}^{(i)}),\ F_m \in \mathbb{F} \tag{15}$$

[26] Reduces base learner influence to allow model improvement using subsequent base learners [28]

The differentiable convex loss function regularized to penalize model complexity is defined:

$$\mathcal{L}(\psi) = \sum \mathscr{L}(\hat{y}^{(i)}, y^{(i)})_i + \Omega(F_m) \tag{16}$$

$$\textbf{s.t. } \Omega(F) = \gamma T + \tfrac{1}{2}\lambda||w||^2$$

To expedite the scoring method, a greedy algorithm adds branches outward from a starting node. Splits are determined by a loss function that uses scores before and after the split with I_L and I_R as instance sets of left and right leaves respectively s.t. $I = I_L \cup I_R$.

The greedy algorithm used, termed *exact greedy algorithm*, is greedy in the sense that it only constructs the tree to the extent required for a index score. However, enumerating over all possible splits for a node is also computationally costly, especially given possible memory constraints. Therefore, another greedy algorithm is utilized called the *approximate algorithm*. The *approximate algorithm* assigns possible splitting points based on the feature distribution percentiles. Continuous features are then binned accordingly and aggregated gradient statistics are compared relative to other possible splitting points. There are variants to this algorithm as well as additional strategies for determining split points from feature distributions.

The XGB also accounts for the sparsity of a given dataset when making split decisions. In the case of the policy dataset, this is especially useful given the number of high-cardinality categorical features and resultant one-hot encoded sparse vectors. As the *approximate algorithm* sorts the feature distribution for binning, if missing values are encountered, values are enumerated such that all the missing feature values are pushed in one direction. The algorithm then focuses only on binning non-missing values for aggregate statistic comparison (for split decision). This method has demonstrated significant improvement in computational run times.

Tree-based methods in general are considered robust to collinearity or, at a minimum, significantly better at handling irrelevant and/or highly correlated data. The sequential base model construction process implemented by XGB per se, acts as an internal feature selection. Thereby ignoring irrelevant (often times correlated) features [28]. Gradient boosted methods in general reduce the risk of collinearity through shrinkage and by the sequential addition of base learners that target the previous learner(s) weaknesses. XGB takes this a step farther by adding a penalty for model complexity, encouraging an increased focus on excluding irrelevant or highly correlated features. However, aside from the policy dataset containing numerous correlated categorical features, the dataset has an added complication arising from *high-cardinality* correlated categorical variables. As such, the arguments advanced by Tolosi et al. [72] regarding the presence of some correlation bias in tree-based ensemble models were held into consideration during analysis.

CHAPTER V

RESULTS

Section 5.1 describes the programming modules used and metric settings. The feature selection processes are outlined in section 5.2 with subsequent feature selection performed using the Sequential Forward Selection (SFS) wrapper methods; performance results on the validation set are compared. Section 5.3 begins with feature selection methods compared using the test set. Hyper-parameter optimization is performed on XGB using the best performing reduced matrix of features. XGB is then compared with two hyper-parameter optimized baseline models using the test set. Section 5.4 provides an analysis of feature stability and selection methods. Additional details on the analysis can be found in Appendix C.3. Model applications given the cost-sensitivity of the dataset are explored in Section 5.5 with prediction-response distributions observed for custom threshold settings. Section 5.6 revisits the research questions and provides final analysis and conclusions. The analysis concludes in Section 5.7 with recommendations for model/feature use, future research, and model interpretation.

5.1 Test Setup

5.1.1 Python libraries

An Anaconda [2] distribution of Python using Jupyter Notebook [32] was used for all programming. A virtual environment was set up with standard machine learning packages. The following less common machine learning packages were used with relevant hyper-parameters $\boldsymbol{\lambda}$:

- `sklearn.ensemble.IsolationForest` [30] (IFA) $\boldsymbol{\lambda}$: contamination = 0.05
- `imblearn.over_sampling` [66] (SMOTE) $\boldsymbol{\lambda}$: ratio determined automatically
- `xgboost.XGBClassifier` [76] (XGB) $\boldsymbol{\lambda}$: objective = binary logistic, metric = auc

- `RepeatedStratifiedKFold` [58] (RSK) $\boldsymbol{\lambda}$: k = 5 folds
- `sklearn.model_selection.learning_curve` [36] with `matplotlib.pyplot`
- `sklearn.feature_selection.SelectFromModel` [62] (SFM) $\boldsymbol{\lambda}$: $\tau_w = 0.5$
- `sklearn.feature_selection.RFE` [59] (RFE) $\boldsymbol{\lambda}$: dimension = 65
- `skopt.BayesSearchCV` [3] (BCV) (Appendix C.1)
- `xgbfir` [75] (XGBFIR)

5.1.2 Metrics

AUC and F1[27] scores for τ_m (method) metrics were obtained using RSK. AUC was the area metric for all τ_{cm} and was the primary metric for feature selection. F1 was used as a secondary metric to monitor recall vs precision changes for different thresholds. While there exists debate regarding AUC vs. F1, given that the practical output of this model was a *probabilty* of churn rather than simply a binary prediction, a metric that captures all possible thresholds was preferred. Often times a feature was selected by the SFS wrapper methods based off AUC, seemingly at the expense of F1. This was not necessarily the case however, since F1 was calculated for a default threshold of 0.5 and AUC was calculated for all thresholds. The following additional table abbreviations were used:

LL = Log-loss

TPR = True positive rate (recall)

PPV = Positive predictive value (precision)

ACC = Accuracy

$\boldsymbol{p}$ = number of features.

[27] Also referred to as F measure; not to be confused with Fscore

In addition to performing RSK on the validation set, the models were tested on the entire validation set. The latter intent was to provide a confusion matrix in order to observe model behavior given an imbalanced dataset, observe the ROC curve, and compare scores with RSK obtained scores.

The $\boldsymbol{T(p)}$ metric was used to reflect computational complexity. All model run time T metrics were taken using a 2.9 GHz Intel Core i7 processor with 2 cores x 4 hyper-threads and a 64-bit architecture. Given that factors such as processors, architecture, and algorithms were held constant for the methods, complexity was defined as equal to a function T that takes input $\mathbf{X}$. Since the different methods used different subsets of features p, complexity[28] was defined:

$$\texttt{Complexity} = T(p) \tag{17}$$

As mentioned, the XGB embedded feature selection method assigned coefficient weights to features. The SFM [62] transformer algorithm introduced in chapter 4.1.2 created the reduced feature space with a $\tau_w = 0.5 \times \frac{1}{D}\sum_{j=1}^{D} w_j$. τ_w settings were tested for various values. The original intent was to preserve roughly a quarter of the original dimensions. However, dropping the τ_w down as far as 0.01 returned approximately the same feature dimension as 0.5. Therefore, the default $\tau_w = 0.5$ was used.

5.2 Selection Methods

The XGB model was used for embedded and wrapper feature selection methods. Models using certain methods for determining reduced feature sets are referenced as models (i.e. SA model, B1 model). For the feature selection methods, two performance-metric sets were compared:

[28]There were a few instances where models were run on different machines. In which case, the $\boldsymbol{T(p)}$ metric was left blank.

1. The SFS models were tested against a model using the full feature set (Full model). The three models were trained using 7 previous quarters and tested using RSK on the first fiscal quarter (Q1) 2018 test set. Metrics collected from the validation/test set for the three models were the *method* metrics.

2. The SFM transformer with threshold τ_w used the XGB embedded method to reduce the feature spaces of the three models. The three models re-trained on the reduced feature set and performance metrics were again obtained. The RFE transformer (RFE model) reduced the entire feature space and also outputted performance metrics. Therefore, four models produced sets of performance metrics from the validation/test set called *final* metrics.

5.2.1 SFS Wrapper + Embedded

The SFS wrapper methods were *bottom up* approaches that are generally considered suboptimal due to the previously mentioned *nesting* effect [52]. The SA method (as opposed to the B1 method) must add features in a particular order, an order which could affect the rejection/acceptance of candidate features based off the presence of correlated or positive interaction features in the base set. However, the SFS methods in this paper were strictly based on the performance changes of XGB, a gradient boost ensemble considered robust to correlated features. Therefore, whether or not a feature was selected depended solely on its predictive contribution to the XGB model. Due to dimensionality and overfitting concerns given the policy dataset, after features were selected by the wrapper method, the XGB was allowed to use its embedded method to again reduce the dataset prior to final performance testing.

Regarding dimensionality concerns, the policy dataset contained numerous categorical variables, many of which were high-cardinality. As a result, the preprocessing conversion of categorical variables to one-hot encoded vectors caused a

large increase in the feature space dimension. The added dimensions were sparse and handled well by the XGB model. However, an increase from dimension $N \times 55$ to $N \times 1497$ was considerable and computational time increased from 4.75 seconds to 97 seconds. While this may be acceptable for the policy dataset with roughly 150k samples, it would rapidly become unacceptable for practitioners if the sample space increased to million + samples. Therefore, the SFS methods were further reduced using the XGB embedded[29] method.

For $\mathbf{X} \in \mathbb{R}^{N \times D}$ and model $\mathcal{P}$ given the features selected by the SFS methods $\mathbf{X}^*$ and method metrics m^*, the feature space was further reduced $\mathbf{X}^{**}$ using the model embedded method and transformer SFM. The model outputted the final metrics m^{**}.

(i) Wrapper method $\rightarrow$ embedded method $\rightarrow$ model output

(ii) $\mathcal{P} \circ \texttt{B1}(\mathcal{P}(\mathbf{X})) \rightarrow \mathbf{X}^*, m^*$ then $\texttt{SFM}(\mathcal{P}(\mathbf{X}^*)) \rightarrow \mathbf{X}^{**}$ finally $\mathcal{P}(\mathbf{X}^{**}) \rightarrow m^{**}$

(iii) $\mathcal{P} \circ \texttt{SA}(\mathcal{P}(\mathbf{X})) \rightarrow \mathbf{X}^*, m^*$ then $\texttt{SFM}(\mathcal{P}(\mathbf{X}^*)) \rightarrow \mathbf{X}^{**}$ finally $\mathcal{P}(\mathbf{X}^{**}) \rightarrow m^{**}$

While the simplicity of the SFS methods allowed observation of engineered feature performance contribution, the methods were still performing feature space reduction. The SFS methods were brute force performance-based algorithms meant to identify and permanently remove non-contributing features thereby reducing overall dimensionality for *all* current and future training sets. The subsequent embedded method depended only on the *current* training set and was based off Fscores obtained during the fitting of that particular training set, further reducing the feature space by temporarily removing non-contributing features. This provided the model the opportunity to adjust feature selection for changes in training data thereby lowering complexity (run time) and the risk of overfitting.

[29] Using Fscore

SFS method thresholds τ_m were allowed to be more liberal with features that demonstrated as least *some* performance contribution. In practical use, given a particular training set, features that only slightly contribute to the model performance or that perhaps enhance one metric at the expense of another will generally have correspondingly light feature importance weights (Fscores) assigned by the embedded method. Such border-line irrelevant features will then be identified and removed by transformers prior to prediction.

The Full model included for comparison used only the model embedded method and SFM transformer to produce both the method and final metrics.

(i) Embedded method $\rightarrow$ model output

(ii) $\mathcal{P}(\mathbf{X}) \rightarrow \mathbf{X}^*, m^*$ then $\texttt{SFM}(\mathcal{P}(\mathbf{X}^*)) \rightarrow \mathbf{X}^{**}$ finally $\mathcal{P}(\mathbf{X}^{**}) \rightarrow m^{**}$

The RFE model was included as a comparison method that utilizes a top-down elimination approach rather than an SFS (bottom-up) approach. Since the RFE method required a pre-specified end-state feature space dimension, it was run after the SFS and Full models. Additionally, the RFE method is a transformer that uses the model embedded method *combined* with feature subset performance to determine eliminations. Therefore, RFE only provided final metrics.

(i) Embedded/wrapper method $\rightarrow$ model output

(ii) $\texttt{RFE}(\mathcal{P}(\mathbf{X})) \rightarrow \mathbf{X}^{**}$ finally $\mathcal{P}(\mathbf{X}^{**}) \rightarrow m^{**}$

The validation set consisted of policies from Q1 2018. Q2 2018 was the final test set and was excluded from the feature selection process. Additionally, learning curves were observed to check for over-fitting and confusion matrices calculated to ensure appropriate[30] model behavior.

[30]With unbalanced dataset, models can become bias towards a class and display large swings in FP values

5.2.2 Base + 1 Method

As mentioned in early chapters, `prem` is the strongest predictor of policyholder churn. The base feature selection process began with training a model using only `prem` in order to establish a baseline m^*. The candidate base features consisted only of original features that, when paired with `prem`, increased metrics m^* by an undetermined threshold τ_b. After all the base candidate features were tested, the candidate base features that showed the largest improvement in performance were selected as the base features (Table 4).

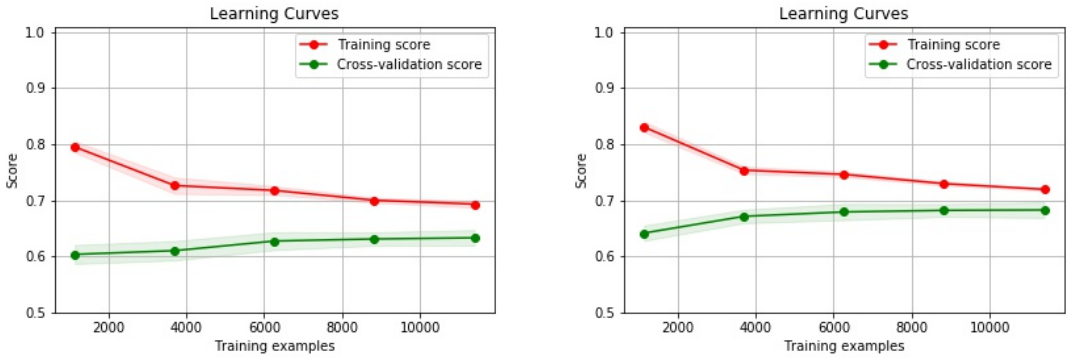
Additional preliminary feature selection due to correlation between similar engineered features was also performed. Similar equal-frequency bins, for example `UW_3` and `UW_6`, were tested using the B1 method. The better performing feature was then selected for use by all three methods. Premium trend features (`lags`) that were previously broken down by `line` showed mixed results during preliminary testing and were removed in favor of their aggregate. Line specific premium trend features are discussed in recommendations.

Table 4: *Base Feature Selection*

Metric	**AUC**	**TPR**	**PPV**	**F1**	**LL**	**ACC**
prem	0.6334	0.1119	0.7037	0.1119	0.4208	0.8418
`rate_adq`	0.6767	0.1830	0.7406	0.2931	0.4029	0.8511
`seg_strgty`	0.6707	0.1543	0.7249	0.2543	0.4104	0.8473
`industry`	0.6830	0.1800	0.7551	0.2906	0.4036	0.8517
base	**0.7031**	0.2062	0.7719	**0.3251**	0.3917	0.8557

After collecting performance metrics for each candidate base feature, τ_b was set at 0.67 for AUC and 0.25 for F1. Additionally, all three selected features showed a noticeable narrowing of the learning curve gap along with an upward shift indicating a increased generalization. Figure 8 shows the improvement from the addition of the `industry` feature with an upward shift in both curves. The shift was noticeably more for the cross-validation score further reducing the separation between training score and RSK

prediction score.



(a) *Premium* (b) *Industry*

Figure 8: *Learning Curve Improvement*

After the base features were selected, thresholds were set for AUC and F1 at 0.7031 and 0.3251 respectively. A Python script was then used to iterate through the remaining original and engineered features. As described in the previous section, candidate features were individually added to the set of base figures. The model then trained and performed RSK for metrics. If the model's performance improved for either AUC or F1, the candidate feature was added to the policy dataset for future modeling. Unlike the SA method, candidate features were *not* added to the base set and τ_m was *not* updated. Table 5 gives a summary of some of the candidate features.

5.2.3 Strictly Additive Method

As seen in Table 6, the SA method began with only the `prem` feature representing the base feature set. τ_m was updated as candidate features that improve XGB performance were added to the base feature set. The updated base set and τ_m were then used to assess the next candidate feature. A Python script was used to perform the method and a log (Figure 26) was generated to show base set and τ_m updates. Feature order was determined according to datatype in order to minimize processing time. The following order was used to assess features from first to last: numerical, indicator, low-cardinality categorical,

Table 5: *B1 Feature Selection Method*

Metric	**AUC**	**F1**	**LL**	**Dim**	**Adds**
base	0.7031	0.3251	0.3917	$N \times 25$	X
`tenure`	0.7058	0.3523	0.3879	$N \times 26$	X
`rate_chng`	0.7096	0.3272	0.3895	$N \times 26$	X
`UI_code`	-	-	-	$N \times 26$	
.	.	.	.	.	.
.	.	.	.	.	.
↓	.	↓	.	↓	.
`prev_non`	0.7061	0.3257	0.3918	$N \times 26$	X
`prev_np`	0.7039	0.3264	0.3922	$N \times 26$	X
`lag_1`	0.7036	-	0.3924	$N \times 26$	X
`lag_2`	0.7033	-	0.3922	$N \times 26$	X
`lag_diff_abs`	0.7038	-	0.3922	$N \times 26$	X
`lag_diff_ind`	0.7035	-	0.3918	$N \times 26$	X
`county_bin_3`	0.7039	0.3271	0.3916	$N \times 28$	X
`se_bin_3`	0.7038	-	0.3915	$N \times 28$	X
.	.	.	.	.	.
↓	.	↓	.	↓	.

and high-cardinality categorical.

5.2.4 Method Comparison and Results

XGB performance results using the features sets selected by the two SFS methods were compared with each other as well as with a model using all candidate features. The B1 method selected 41 of 55 candidate features compared to the SA method selection of 39. Model performance results and learning curves using the method selected features and all features are displayed in Table 7 and Figure 9 respectively.

Of note, results in Table 7 were the result of cross-validation using the validation set. Therefore, there was minor data leakage as wrapper methods were allowed to select candidate features using the same validation set later used for performance metric comparison. Therefore, the subsequent Q2 test set carried greater weight regarding practical consideration when comparing performance metrics. Using the validation set, the SA method outperformed both other models in all *method* metrics. The SA model

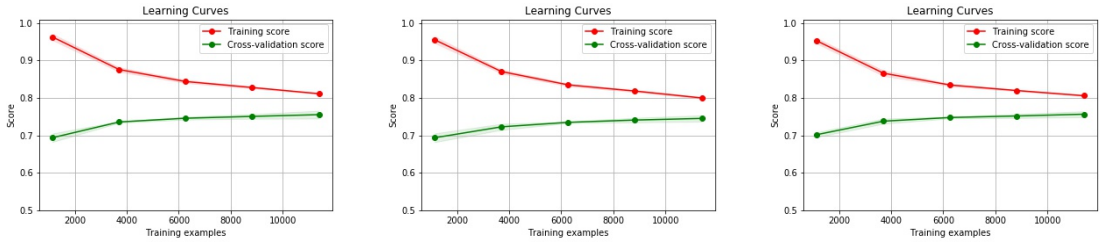
Table 6: *SA Feature Selection Method*

Metric	**AUC**	**F1**	**LL**	**Dim**	**Adds**
prem_	0.6334	0.1119	0.4208	$N \times 1$	X
tenure_	0.6461	0.2175	0.4173	$N \times 2$	X
rate_chng	0.6780	0.2494	0.4080	$N \times 3$	X
UI_code	0.6797	0.2584	0.4075	$N \times 4$	X
PC_num	-	-		$N \times 9$	
.	.	.	.	.	.
↓	.	↓	.	↓	.
.	0.7056	0.3355	0.3886	-	.
lag_1	0.7059	-	0.3893	$N \times 10$	X
lag_2	-	-	-	-	
prev_non	0.7092	-	0.3884	$N \times 11$	X
prev_np	-	-	-	-	
.	.	.	.	.	.
↓	.	↓	.	↓	.
.	0.7275	0.3707	0.3784	-	.
lag_diff_ind	0.7280	-	0.3786	$N \times 24$	X
diff_state_ind	0.7284	-	0.3644	$N \times 25$	X
csf_ind	0.7284	-	0.3781	$N \times 26$	X
zip_bin_3	-	-	-	$N \times 29$	
.	.	.	.	.	.
↓	.	↓	.	↓	.

Table 7: *Model Performance*

Method	**AUC**	**TPR**	**PPV**	**F1**	**LL**	**ACC**	***p***	**Dim**	**T(p)**
Method Metrics									
Full	0.7556	0.2545	0.8894	0.3955	0.3661	0.8688	55	$N \times 1497$	97s
B1	0.7455	0.2091	0.8749	0.3374	0.3755	0.8614	41	$N \times 1433$	92s
SA	0.7567	0.2603	0.8941	0.4032	0.3656	0.8699	39	$N \times 1159$	55.4s
Final Metrics									
Full	0.7511	0.2682	0.8824	0.4111	0.3659	0.8704	-	$N \times 68$	5.78s
B1	0.7423	0.2250	0.8646	0.3569	0.3734	0.8632	-	$N \times 55$	4.81s
SA	0.7538	0.2624	0.8657	0.4025	0.3656	0.8686	-	$N \times 60$	5.21s
RFE	0.7530	0.2690	0.8852	0.4124	0.3641	0.8707	-	$N \times 65$	5.48s

outperformed the Full model using $\approx$ 70% of the features; indicating the potential utility of the SA selection method. The learning curves for all three methods demonstrated appropriate model training behavior and indicated a sufficient ability to generalize.



(a) *Full* (b) *B1* (c) *SA*

Figure 9: *Method Learning Curves*

From the proposed engineered features and derivative features for premium and policy trends, the SA method selected `lag_1`, `lag_diff_ind` and `prev_non`. From the remaining engineered features, the method selected 9 of 14 frequency-based features, 4 of 6 time derivative features, all 5 mapped features, and 2 of the 3 added indicator features. All of the base features were also selected.

The B1 method selected `lag_1`, `lag_2`, `lag_diff_ind`, `lag_diff`,`lag_diff_abs`, `prev_np`, and `prev_non`. From the remaining engineered features, the method selected 10 of 14 frequency-based features, 4 of 6 time derivative features, 3 of 5 mapped features, and 2 of the 3 added indicator features.

5.3 Test Set

The same methodology described in 5.2 was utilized for the test set minus the use of cross-validation wrapper RSK. Models were trained on the entire previous 8 quarters and tested against the Q2 2018 test set. As before, method and final metrics were compared.

B1 and SA models as well as the RFE model were at a slight (though realistically necessary) disadvantage relative to the Full model as the mentioned models were not able to perform method feature selection using the additional quarter. However, by design, both SFS models required a validation set in order to perform feature selection, while the Full model only required a training set to performed embedded feature

selection. Technically, the RFE model also only required a training set to perform selection. However, its inclusion was primarily for comparison with the SFS methods. Therefore, the RFE model was forced to use features selected without the additional quarter. Moreover, since the SFS models were able to use model embedded selection that incorporates the additional quarter, the RFE model was actually at the greatest disadvantage and forced to generalize predictions the most.

5.3.1 Results

Table 8: *Test Set: Q2 2018*

Method	**AUC**	**TPR**	**PPV**	**F1**	**LL**	**ACC**	$\boldsymbol{p}$	**Dim**	**T(p)**
Method Metrics									
`Full`	0.7752	0.3244	0.7900	0.4597	0.3494	0.8901	55	$N \times 1497$	116s
`B1`	0.7617	0.3193	0.7699	0.4514	0.3649	0.8882	41	$N \times 1433$	110s
`SA`	0.7680	0.3310	0.7118	0.4518	0.3657	0.8843	39	$N \times 1159$	88s
Final Metrics									
`Full`	0.7702	0.3188	0.7512	0.4476	0.3526	0.8867	-	$N \times 64$	5.68s
`B1`	0.7620	0.3188	0.7269	0.4432	0.3690	0.8846	-	$N \times 62$	6.26s
`SA`	0.7681	0.3279	0.7083	0.4483	0.3644	0.8837	-	$N \times 62$	5.98s
`RFE`	0.7716	0.3188	0.7631	0.4497	0.3533	0.8876	-	$N \times 65$	6.2s

While the performance metrics for the SA and Full models were mixed, the test results confirmed the cross-validation results from the previous section. The SA method was able to reduce the feature set by 30% while maintaining comparable model performance to the Full model.

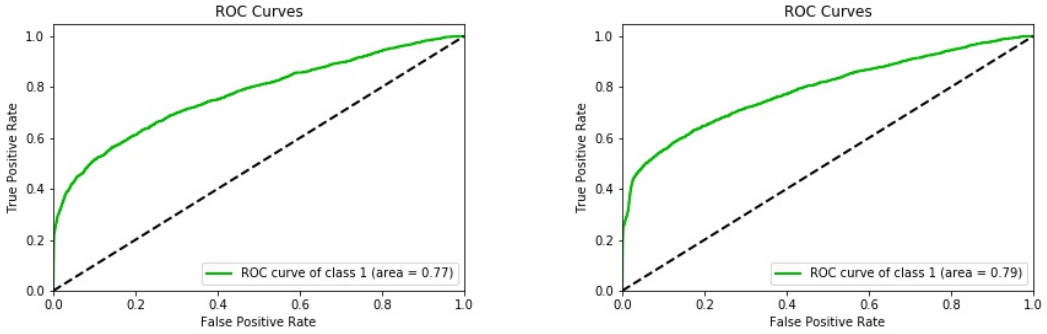
5.3.1 Model Comparison

Despite being at a disadvantage, RFE was identified as the best performing model by final metrics and therefore selected to provide the reduced matrix of features for XGB comparison with competing baseline models. In order to optimize performance for all models, selected hyper-parameters $\boldsymbol{\lambda}$ were optimized using a Bayesian Optimizer (Appendix C.1).

The XGB $\boldsymbol{\lambda}$ for learning rate, base model numbers, and tree (base model) depth were optimized to give the model performance metric improvements shown in Table 9 and Figure 10.

Table 9: $\boldsymbol{\lambda}$ *Optimized*

Method	**AUC**	**TPR**	**PPV**	**F1**	**LL**	**ACC**	$\boldsymbol{p}$	**Dim**	**T(p)**
Final-Reduced Performance									
RFE	0.7716	0.3188	0.7631	0.4497	0.3533	0.8876	-	$N \times 65$	6.2s
RFE w/λ	0.7905	0.3838	0.7675	0.5117	0.3176	0.8945	-	$N \times 65$	-



(a) XGB (b) XGB w/λ

Figure 10: $\boldsymbol{\lambda}$ *Optimized ROC Curve*

The XGB Model performance improved significantly by optimizing only a few key hyper-parameters. The largest improvement was for the F1 metric which consisted of an 6.5% increase in recall with a slight improvement in precision. Improvements to both AUC and Log-loss were also notable.

Logistic Regression (LR) and Random Forest (RF) (Appendix C.2) models with hyper-parameters optimized for AUC, were selected for comparison with the XGB model. LR $\boldsymbol{\lambda}$ for regularization weight was optimized. RF $\boldsymbol{\lambda}$ for base model numbers and tree (base model) depth were optimized. All three models were trained/tested using the RFE selected matrix of features with results recorded in Table 10.

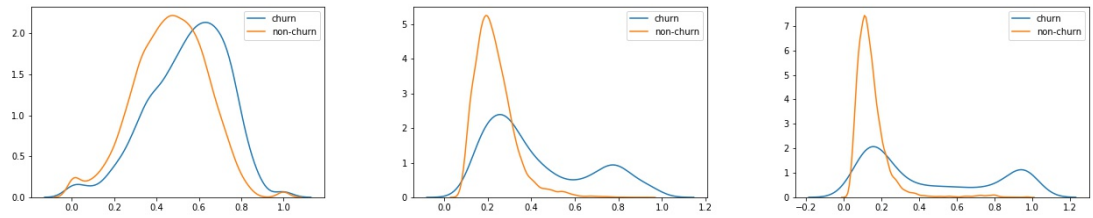
Table 10: *Model Comparison*

Method	**AUC**	**TPR**	**PPV**	**F1**	**LL**	**ACC**	p	**Dim**	**T(p)**
Final-Reduced Performance									
XGB	0.7905	0.3838	0.7675	0.5117	0.3176	0.8945	-	$N \times 65$	-
RF	0.7618	0.3071	0.7194	0.4305	0.3803	0.8829	-	$N \times 65$	-
LR	0.6424	0.6360	0.1988	0.3029	0.7050	0.5782	-	$N \times 65$	-

As shown, the XGB model returned superior performance results for all metrics. The only additional hyper-parameter used by XGB compared to RF, was learning rate (or shrinkage). For boosted tree-based methods, shrinkage is distinct from learning rate as it normally applies to stochastic optimization [43]. Shrinkage is used to control weights similar to a stochastic optimization, however, the weights are assigned to existing base learners to reduce influence and allow improvement through the sequential addition of base learners constructed based off a scoring function incorporating previous learner error [7]. Since RF models randomly choose feature and sample subsets to create base learners, shrinkage is not utilized. As shown in Appendix C.2, given similar optimization intervals for number of base learners and tree depth, the XGB used almost double the amount of base learners (175 learners) compared to RF (99 learners) and shallower tree depths of 7 compared to 14. Therefore, XGB showed a preference for a larger number of weaker base learners than RF. The LR had limited hyper-parameters and performance was likely negatively impacted by the large presence of high-cardinality and/or correlated features, supporting the assertion by [69] regarding performance issues that emerge when LR (with Lasso) encounters groups of correlated features. Furthermore, LR hyper-parameters were optimized for AUC (similar to the tree-based models). While the tree-ensembles responded well to the specific metric-driven optimization, optimizing a General Linear Model (GLM) [28] with a highly correlated and primarily categorical dataset for AUC, may explain the large number of FP and poor PPV score.

Analyzing model probability-response distributions in Figure 11 shows the advantage of using a non-parametric model with the policy dataset. The GLM was able to provide some separation between the populations (churn vs non-churn). However, for all τ_m there did not exist a clear threshold value that would result in useful predictions without ignoring a large population of FNs. That being said, if FN values are ignored, similar performance results to those presented by [64] could be derived from Figure 11(a) The derived performance results would technically be correct, albeit misleading to an extent. The RF (11(b)) improved upon the GLM by providing areas of discernible separation between the classes that, given an adjusted τ_m, would improve model performance. The XGB further defined the separation in Figure 11(c).

The clear separation shown in Figures 11(b) and 11(c) introduced some concerns. While there was clear practical applications for the model, the fact that the model was able to distinguish a certain amount of churners with such high accuracy indicated the possibility of data leakage. For example, certain insurance products may have been discontinued. In which case, the model would be able to identify all such products as non-renews with perfect accuracy. This is explored further in applications.



(a) GLM (b) RF (c) XGB

Figure 11: *Response Distributions*

5.4 Analysis

5.4.1 Feature Stability

Given the presence of a large number of correlated high-cardinality categorical features, the policy dataset posed not only an over-fitting risk but also an ill-conditioning [72] risk. Ill-conditioning in machine learning can occur with groups of correlated features. For many models, the presence of large groups of correlated features cause the model to choose a representative feature and drop the remaining correlated features. This choice of correlated feature within the group of correlated features can vary depending on relatively minor training data changes. Therefore, minor changes in training data can cause completely different features to be selected which in turn can have a larger affect on model outcome. Regularized GLMs using Lasso for feature selection are highly susceptible to this effect.

The conditioning of a model can reflect through the stability of feature importance weights whereby changes in the training data should not cause large swings in feature weights and rankings. While performing cross-validation on Q1 2018, embedded method feature importance scores for Q1 2018 were recorded to compare with Q2 2018 scores. Parenthetically, all the models performed better on the Q2 data. This may simply have been the result of more training data or other factors outside the scope of this analysis. However, the addition of an additional quarter for model fitting did represent an example of an adjusted training set. Using feature weights from both training sets, weights were compared to determine feature stability. The feature importance weights for each model trained on Q2 2016 through Q4 2017 were compared with feature importance weights obtained from training Q2 2016 through Q1 2018 (Figure 23). None of the model feature importance weights showed dramatic changes. The 5 heaviest weighted features fluctuated around 1% without rank change. The remaining 20 heaviest weighted features behaved roughly the same with some rank adjustments.

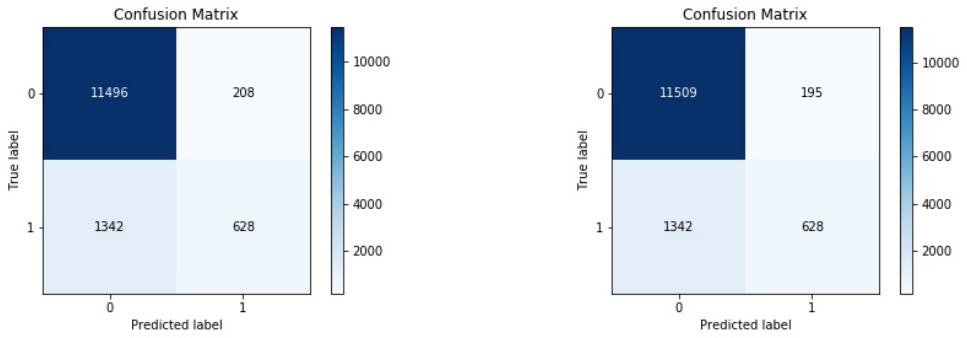
From this comparison, it was concluded that the models did not suffer from ill-conditioning as represented by feature importance weights.

5.4.2 Feature Selection Analysis

Key differences in SFS method feature selection included `risk_complex_ind`, `con_type`, and `UI_code`. The mentioned features all appeared in the embedded method selected (final) set and were examples of the SFS methods differing in their selection of certain features. The features demonstrated predictive properties and reasons for rejection were likely related to feature interactions and/or correlation. See Appendix C.5 for a more detailed analysis.

5.5 Application

The models trained to optimize AUC using a synthetically balanced training[31] dataset. The resulting confusion matrices in Figure 12 are from the performance test used to calculate scores.



(a) Full (b) RFE

Figure 12: *Confusion Matrices (CM)*

To demonstrate practical application, the top performing feature selection method from Table 8 (RFE), was used. Comparing the RFE confusion matrix with the Full feature set model showed a clear reduction in FPs. The same model using RFE with optimized

[31] The test/validation sets were not synthetically balanced

hyper-parameter adjustment gave less readily comparable, but still improved, results in Figure 13. From the perspective of a practitioner trying to improve retention by engaging policyholders with a higher probability of churn, being able to identify roughly 750 would-be churners while missing over 1200 with only around 230 FP represents a reasonable cost-sensitive opportunity through targeted retention marketing. Practitioners can focus their retention efforts on 985 policyholders, $\approx$ 77% of which would have churned. The 77% would-be churners account for about 38% percent of all churners for the quarter; representing significant cost reductions for the carrier. Moreover, segmenting the policy dataset further may help in reducing FP while maximizing the TPR, especially in the case of higher premium policyholders. The default threshold for classifying samples as churn $\mathbf{x}^{(i)} = 1$ was 0.5. In order to determine an optimal threshold, prediction-response distributions were observed. Since AUC represents the area for all possible threshold values, any adjustment that improved recall would be at the expense of precision and vis-versa. However, as indicated by the distributions in Figure 14(a), a reduced threshold for classifying churners would be of added benefit. Adjusting the threshold down to .3 resulted in the following confusion matrix results in Figure 14(b).

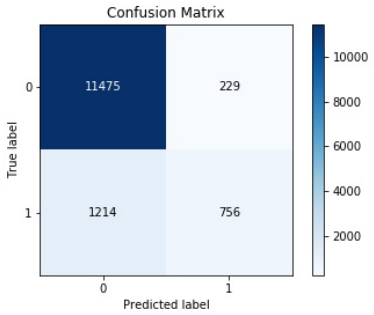
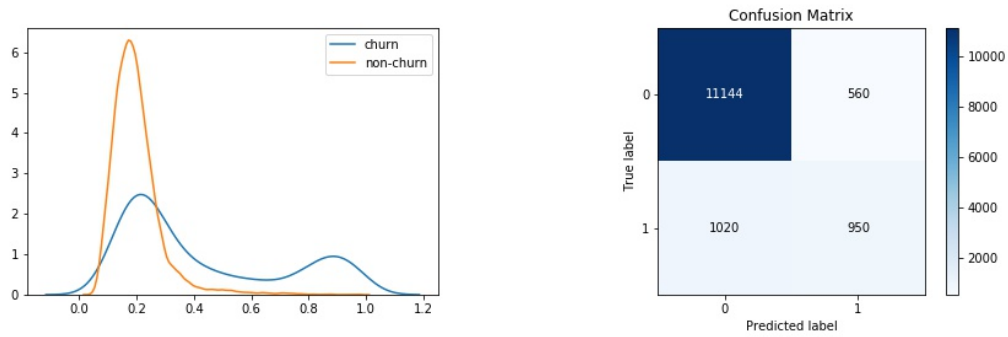


Figure 13: $\boldsymbol{\lambda}$ *Optimized CM*

Therefore, if a given practitioner was able to tolerate a greater FPR, and if retention efforts were therefore focused on 1510 identified policyholders, $\approx$ 63% of which are would-be churners. Than retention efforts would reach $\approx$ 48% of all would-be churners for the quarter. In addition, all FP were policies with a higher probability of churn. Though the model was incorrect for certain Q2 2018 policies, such policies may churn in Q3 2018 or later quarters. Therefore, engaging the (FP) policy early on may preclude



(a) *Distributions* (b) *Adjusted CM*

Figure 14: *Threshold Adjustment*

policy churn in quarters subsequent to Q2 2018. In other words, marketing retention strategies directed at FPs in the identified subset of policyholders (with a high probability of churn) is not necessarily wasted effort.

In addressing the concerns mentioned in 5.3.1, the features used by the model were further analyzed for data leakage by removing categorical feature values belonging to higher cardinality features, there were no noticeable changes to the distribution shapes shown in Figure 14(a). Strong categorical feature predictors `industry` and `seg_strategy`, were re-analyzed for churn value distribution and then removed from the model. Again, there was no dramatic changes in prediction-response distribution shapes. Features were then systematically removed until only `prem` remained. The resulting distribution in Figure 15 shows the model's ability to begin separating the minority class early on. Therefore, the possible concerns mentioned in 5.3.1 did not appear to be influencing the model's prediction ability. However, a subset of the dataset with prediction probabilities above 60% should still be explored for trends. Model interpretability techniques for XGB and other black box models are discussed in recommendations.

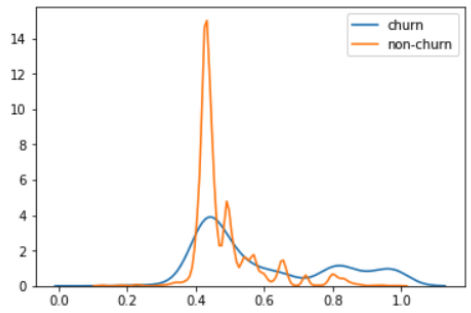


Figure 15: *Premium Only*

5.6 Conclusion

In this study, the research questions stated in Chapter 1 were addressed. This section revisits the questions and provides relevant summaries. Additional comments are made in the recommendations section.

5.6.1 Main Research Question

Given the intermediate filter between carrier and policyholder, can a churn model be constructed using the popular tree-based ensemble XGBoost system with implications for practical application?

Using current data mining techniques, the policy dataset was preprocessed. IFA [38] was used to reduce outliers, [26] to impute missing values of important features, and [6] to synthetically balance the training set. Several feature selection methods were compared using XGB with a validation set and results were confirmed on the test set. RSK provided performance metrics for comparison and learning curves were routinely observed for possible model overfitting. Using a RFE method reduced matrix of features, XGB demonstrated the best performance metrics and was therefore selected for comparison with baseline models on the test set. Basic hyper-parameters for all models were optimized using a random sample of training data and Bayesian Optimizer [3]. Hyper-parameter adjustment of XGB showed significant improvement in recall and Log-loss. The XGB outperformed the baseline models in all performance metrics. The XGB model was further analyzed using probability prediction-response distributions

and confusion matrices, τ_m values that may strengthen the model's utility for practitioners within the P&C industry were noted. Despite the higher dimensionality of the policy dataset, learning curves using cross-validation showed appropriate convergence between training and prediction scores. The convergence trends in Figure 9 indicate increased convergence with additional data and strong support for the model's scalability. Therefore, the XGB constructed with the mentioned methods demonstrated prediction generalization as well as strong implications for scalability and practical application. Common concerns involving black box interpretability are discussed in recommendations.

5.6.2 Sub-Research Question 1

Given a limited feature space, can additional features be engineered based off current data mining techniques and domain knowledge to capture agent-related information that may influence policy churn?

The concepts introduced by [48, 64] were extended to independent agents in order to construct higher-level informative features for agent influence over policyholder churn. Given the policy dataset, existing features for premium and policy types were used to construct features for previous quarters. The constructed features attempted to leverage previous quarter agent-specific information to capture possible policy and premium trends that may affect current quarter policyholder churn. As the constructed trend features were constructed from existing features, correlation was a concern. Derived *representative* trend features, as discussed by [72], were constructed to mitigate correlation bias with mixed results.

Premium Trend Features: The premium trend features described in chapter 3 and introduced by [64] were built using quarterly time frames given annual policyholder information. Therefore, dataset manipulation with certain assumptions was required. In order to provide information related to agency trends, the features were constructed for

two quarters prior to the test/validation quarter.

It was proposed that if agency X is experiencing overall previous quarterly declines in premium, then a higher-level feature representing those declines could carry information related to agency X policyholders' probability of churn. Given the size of premium, this could be further broken down by insurance line. For example, agency X experienced a decline in auto premium from the previous quarter, therefore auto policyholders of agency X in the current quarter have a higher probability of churn.

During preliminary performance testing, line-specific premium trend features showed mixed results indicating the necessity of further engineering to include representative premium trend features for line percentages, ratios, or using additional transformations to assist with correlation bias mitigation. Additional engineering of the premium trend line-specific features was therefore determined to be outside of the scope of this analysis and only aggregated premium trend features with possible representative features were assessed. The following premium trend features were tested:

- `lag_1`: Aggregated agent premium for the previous quarter.
- `lag_2`: Aggregated agent premium for `lag_1` previous quarter.
- `lag_diff`: Difference between lag quarters, `lag_1` - `lag_2`.
- `lag_diff_abs`: Absolute value of the difference between lag quarters, `lag_1` - `lag_2`.
- `lag_diff_ind`: Indicator for positive or negative values given `lag_1` - `lag_2`.

As shown in Table 5, aside from `lag_diff`, all of the premium trend features were selected by the B1 method. Additionally, correlation between the features was less of a concern with the B1 method. Increases in AUC were small however, and often at the expense of F1. As opposed to the B1 method, correlation was a concern for the SA method as seen in Table 6. The method was run two additional times with different permutations

of premium trend features. Regardless of permutation, `lag_1` and `lag_diff_ind` were consistently selected with only `lag_diff_ind` used in the embedded-selected final set.

Premium trend feature importance as it applies to the policy dataset, was not clearly demonstrated using the SFS methods introduced in this analysis. However, continued experimentation should explore different representative features with the possibility of additional lag quarters. Furthermore, increased historical data would preclude the need for assumptions[32] and could also enhance premium trend feature prediction value.

Policy Trend Features: In chapter 3, it was proposed that if agency X is experiencing overall changes in number of new policies and/or non-renewals, then higher-level features representing those changes could carry information related to agency X policyholders' probability of churn. The policy trend features were built to represent types of policies from the previous quarter only. Therefore, adjustments and assumptions regarding all other agent annual policies were not needed. The following policy trend features were tested:

- `prev_np`: Total count of agency new policies from the previous quarter.
- `prev_non`: Total count of agency non-renewals from the previous quarter.

As shown in Table 5, both policy trend features were selected by the B1 method. Increases in AUC were notable with attendant increases in F1. Using the SA method in Table 6, correlation was again a concern: $\texttt{Corr}(\texttt{prev\_np}, \texttt{prev\_non}) \approx 0.60$ and policy trend feature correlation with `agnt_freq` even greater (Figure 17). Only `prev_non` was selected with an increase of AUC at the expense of F1. Both policy trend features were introduced to the SA method before `agnt_freq`. However, `agnt_freq` was selected (along with `prev_non`) with an increase in AUC. Both `agnt_freq` and `prev_non` were present in the embedded-selected set with `prev_non` ranking individually higher on Gain and

[32]Regarding the base year dataset, Chapter 3

Fscore.

Suspecting that feature correlation may be causing the SA method to select prev_non and reject prev_np, two smaller SA method trials were conducted using different permutations of the three correlated features. prev_np was still consistently rejected and prev_non selected. Additionally, the prev_non feature was used in all embedded method-selected sets with a persistent presence in the top percentile of features as ranked by the model.

As discussed in chapter 3, the correlations between the policy trend features, prem, and agnt_freq were explainable to an extent given the underlying volume variable. Larger agencies mean more policies which equates to larger policy and premium trend features. Additionally, the linear combination:

(i) prev_np + prev_non + renewed policies = total agent quarterly policies

strongly indicated the need for representative features discussed in recommendations. Of note, total agent quarterly policies was *not* included as a feature. agnt_freq reflected all agent accounts for the entire dataset, not a particular quarter.

Given both SFS method selections of prev_non as well as its persistent presence in the embedded method-selected (final) sets with interactions showing notable Gain scores, the policy trend feature prev_non demonstrated predictive value and is recommended for inclusion in future policy datasets.

5.6.3 Sub-Research Question 2

Can an effective performance-based feature selection wrapper method be constructed to determine engineered feature contribution and improve model performance?

SA Method: Despite performing comparatively well against the Full set and B1 method, the SA model displayed some noticeable flaws. The primary flaw involved the inconsistent expansion of a base set of features for assessing subsequent candidate

feature contribution. While the threshold updating seemed to perform reasonably well at selecting only those features contributing to the XGB's current performance, there were two primary concerns:

1. The possible rejection of a candidate feature $\mathbf{x}_a$ due to the presence of a strongly correlated candidate feature $\mathbf{x}_b$ in the base set.

2. The possible rejection of a candidate feature $\mathbf{x}_a$ due to the absence of candidate feature $\mathbf{x}_c$ in the base set given that feature $\mathbf{x}_a$ and $\mathbf{x}_c$ are positively interactive.

If correlation was a contributing factor in method selection of candidate features, the following argument shows that the order of features introduced using the SA method will affect model performance:
Given numerical features $\mathbf{x}_a \not\perp\!\!\!\perp \mathbf{x}_b$ with moderate to strong correlation < 1,

1. $\mathbf{x}_a$ is tested first s.t. $\mathcal{P}([\mathbf{X}^B, \mathbf{x}_a]) = m > \tau \implies \mathbf{x}_a \in \mathbf{X}^B$ and $\tau^* = m$.

2. $\mathbf{x}_b$ is tested s.t. $\mathcal{P}([\mathbf{X}^B, \mathbf{x}_b, \mathbf{x}_a]) = m < \tau^*$ due to correlation $\mathbf{x}_a \not\perp\!\!\!\perp \mathbf{x}_b$.

3. $\therefore \mathbf{x}_a \in \mathbf{X}^B$ and $\mathbf{x}_b \notin \mathbf{X}^B$

4. Repeat steps 1 through 3 with $\mathbf{x}_b$ tested first and the result is $\mathbf{x}_a \notin \mathbf{X}^B$ and $\mathbf{x}_b \in \mathbf{X}^B$

Therefore we can conclude that if features were correlated and, despite the XGB robustness to correlated features, the strongly correlated features caused any decrease in model performance then the order of candidate features assessed affected the selection of candidate features and subsequent performance of the model. The effect on model performance may have varied and been minor, but the mere existence of the effect compromises the feature selection method. The same argument can be applied for positive interactions. Based on these concerns, the SA method is not recommended as a wrapper feature selection method for the policy dataset.

B1 Method: The B1 method was a modified version of the SFS feature selection method used by [23]. From a purely performance perspective, the base feature set used in the B1 method consisted of relatively strong predictors (Table 4) with a learning curve indicative of acceptable generalization. While there was some correlation between features, none of the base features were high-cardinality. Therefore, B1 represented a viable base set to assess individual engineered feature performance contribution to an extent. However, some engineered feature contributions to model performance were so slight, than it is reasonable to assume that the addition of other features would have caused the presence of the selected weak engineered feature to quickly become noise rather than value added. Furthermore, the possibility of feature interactions affecting model performance evinces further weakness in the method. Given the high correlation between many of the categorical features, the B1 method is also not recommended as a wrapper feature selection method for the policy dataset.

RFE: As mentioned in Chapter 4.1.2, the RFE algorithm [59] utilizes a given model's embedded selection method to drop features ranked as less important when selecting a subset to train. RFE is limited, therefore, by the effectiveness of the embedded importance metric and susceptible to correlation/interaction issues. For example, features $\mathbf{x}_a$ and $\mathbf{x}_b$ have performance enhancing interaction effects. However, $\mathbf{x}_b$ scores lower individually using the embedded method metric and is consequently eliminated. The positive interaction is also eliminated and the model performance suffers. This is exacerbated by the RFE parameter requirement for the final reduced feature space dimension. The transformer does not use performance criteria for determining final reduced feature space dimensionality and will continue eliminating features until the desired dimensionality is achieved.

While non-parametric models are generally considered robust to correlated features, [69, 72] demonstrated that ensemble tree-based models can use correlated features

interchangeably with less predictive features often replacing their more predictive correlated counterpart(s). Since XGBoost implements a weighted score based off Fscores, this subtle substitution may inflate the importance score for the weaker feature. As RFE relies on embedded method importance scores, the possibility of selecting weaker features based off inflated scores could compromise subsequent model performance.

It merits mention that the study by [69, 72] was conducted using Random Forests. Gradient boosted tree-based algorithms are generally considered even more robust to correlated features. Boosted tree-based ensembles, such as XGB, tend to predominately select one of the dominate correlated features since learners are sequentially added to the ensemble that effectively learn from the previous learner. This likely further diminishes the negative effect of correlated features, but does not necessarily eliminate it completely.

5.7 Recommendations

Feature Selection: Given the feature correlation and interaction concerns involving both SFS methods, an elimination method (such as RFE) for the policy dataset is recommended to reduce the feature space and test constructed features. Both wrapper methods are considered brute-force, bottom-up methods that systematically test feature subsets while monitoring model performance. Unlike the SFS methods, the top-down elimination method used (RFE) began with all features and eliminated features based on the desired feature space dimension end-state and recursively updated embedded importance weights/scores. As stated, this algorithm does require an end-state dimensionality. Therefore, running a preliminary model with a transformer (such as SFM) would provide an idea of the preferred feature space dimension end-state prior to utilizing the RFE algorithm.

Trend Features: The networking concept as introduced by [48] and used to engineer policy trend features for the dataset, should be explored further given the potential

impact independent agents have on policyholder churn. Additionally, the premium trend features made mention by [64] and constructed from aggregated agent premiums for the policy dataset, may identify agencies that have begun to shift recommendations to other carriers, agencies that are seeing a decline in certain types of policies due to forces specific to their business (i.e. competing agencies), and/or general trends (shifts to online services). Available data permitting, trend features can be engineered as monthly, quarterly, or semi-annual and lag for several time periods. As evidenced by this analysis however, any lagging (trend) features will show moderate to strong correlation to other trend features as well as certain other agency features. Therefore, representative features introduced by [72] should be considered and segmentation of trend features using other categorical features should also be explored. The linear relationship between `prev_np`, `prev_non`, renewed policies, and total accounts for the quarter as well as the line-specific premium trend features, introduces several feature construction opportunities. Trend features expressed as percentages, ratios, or other transformations may contribute to model performance while mitigating correlation issues.

Model Performance: The XGB has an impressive and possibly even intimidating number of hyper-parameters. `scale_pos_weight` [77] specifically addresses minority classes in an imbalanced dataset. However, using a bayesian optimizer with numerous XGB hyper-parameters can quickly become complicated. Many of the XGB hyper-parameters are highly technical and affect the core processes involved in base learner construction. Slight manipulations can cause dramatic swings in other hyper-parameters and undesirable model behavior. Therefore, many of the more technical hyper-parameters are better left at the default value unless extensive study and trial-and-error are able to be conducted. Detailed hyper-parameter information can be found at [77]. In this analysis, the utility of hyper-parameter bayesian optimization was demonstrated in Table 9. Additionally, [71] further demonstrated the added advantages

XGB hyper-parameters bring when used in conjunction with a bayesian optimizer. As described, given the presence of numerous high-cardinality and dependent categorical variables in the policy dataset, an XGB model with feature selection using a top-down elimination wrapper/embedded method is recommended. The RFE transformer is available at [59] and works well with embedded methods. The main drawback with XGB, and other black box models, involves prediction interpretability.

There are several options available to assist in XGB prediction interpretation. Locally Interpretable Model-Agnostic Explanations (LIME) perturb the feature space around the prediction in order to approximate a linear description [60], LIME is however restricted to the individual sample level (locally). [42] introduces a framework (SHapley Additive exPlanation or SHAP) for interpreting black box predictions at the local and global level (Appendix B.4). SHAP values are calculated based on a conditional expectation function that accounts for dependence which can be used to represent feature importance. Figure 16 demonstrates SHAP local output which is similar in scope to LIME local output. Given the inconsistencies between current embedded methods discussed in Chapter 4 (Gain vs. Fscore), SHAP feature importance values could provide score metrics of higher fidelity and should be explored for use with the policy dataset.

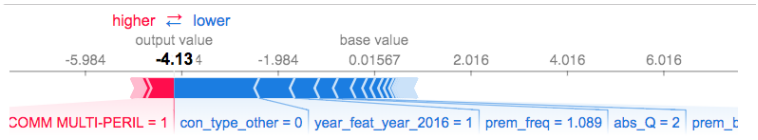


Figure 16: *SHAP*

REFERENCES

[1] Kumaresan A, Saurav Swaraj, and Raghunandan Kothamasu. *Top 10 Trends in Property & Casualty Insurance 2018*, Capgemini. (2017).

[2] Anaconda Website.
`www.anaconda.com/`

[3] Bayes Search CV.
`scikit-optimize.github.io/#skopt.BayesSearchCV`

[4] Avrim L. Blum and Pat Langley. *Selection of Relevant Features and Examples in Machine Learning*, Artificial Intelligence. **97** (1997), 245-271, DOI 10.1016/S0004-3702(97)00063-5.

[5] Eric Brochu, Vlad M. Cora, Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, CoRR. (2010), arXiv:1012.2599.

[6] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. *SMOTE: Synthetic Minority Over-sampling Technique*, J. Artificial Intelligence Res. **16** (2002), 321-357, DOI 10.1613
/jair.953.

[7] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*, KDD. (2016), DOI 10.1145/2939672.2939785. arXiv:1603.02754v3

[8] Marc Claesen, and Bart De Moor. *Hyperparameter Search in Machine Learning*, CoRR. (2015). arXiv:1502.02127v2.

[9] William G. Cochran *The χ^2 Test of Goodness of Fit*, Ann. Math. Statist. **23** (1952), no. 3,

315-345.

[10] J. David Cummins and Neil A. Doherty. *The Economics of Insurance Intermediaries*, J. Risk and Insurance. **73** (2006), no. 3, 359-396.

[11] Department of Agriculture, United States.
`www.ers.usda.gov/data-products/urban-influence-codes.aspx`

[12] Ross DeVol, Joe Lee, and Minoli Ratnatunga. *2016 State Technology and Science Index*, Milken Institute. (2016). `www.statetechandscience.org`

[13] Ian Dewancker, Michael McCourt, Scott Clark. *Bayesian Optimization for Machine Learning : A Practical Guidebook*, CoRR. (2016), arXiv:1612.04858v.

[14] Yadolah Dodge. *The Concise Encyclopedia of Statistics*, Springer-Verlag, New York, 2010.

[15] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H. Hoos, and Kevin Leyton-Brown. *Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters*, NIPS workshop on Bayesian Optimization in Theory and Practice. **10** (2013), 3.

[16] Michael Etgar. *Channel Domination and Countervailing Power in Distributive Channels*, J. Marketing Research **3** (1976), no. 3, 254-262.

[17] George Forman. *An Extensive Empirical Study of Feature Selection Metrics for Text Classification*, J. Mach. Learn. Res. **3** (2003), 1289-1305.

[18] Yoav Freund, and Robert E. Schapire. *Experiments with a New Boosting Algorithm*,

ICML. (1996).

[19] Jerome H. Friedman. *Greedy function approximation: A gradient boosting machine*, Ann. Statist. **29** (2001), 1189-1232, MRMR1873328.

[20] Anthony Gambardella. *IBISWorld Industry Report: Property, Casualty and Direct Insurance in the US*, IBIS World. (2018).

[21] Ian Goodfellow, Yoshua Bengio, annd Aaron Courville. *Deep Learning*, MIT Press. (2016). `www.deeplearningbook.org`

[22] Pablo M. Granitto, Cesare Furlanello, Franco Biasioli, and Flavia Gasperi. *Recursive Feature Elimination with Random Forest for PTR-MS Analysis of Agroindustrial Products*, Chemometrics and Intelligent Laboratory Systems. **83** (2006), 83-90, DOI 10.1016/j.chemolab
.2006.01.007.

[23] Bryan Gregory. *Predicting Customer Churn: Extreme Gradient Boosting with Temporal Data*, WSDM 2018. (2018), arXiv:1802.03396v1.

[24] Isabelle Guyon and Andre Elisseeff. *An Introduction to Variable and Feature Selection*, J. Mach. Learn. Res. **3** (2003), 1157-1182.

[25] David J. Hand. *Measuring classifier performance: a coherent alternative to the area under the ROC curve*, Mach. Learn. **77** (2009), 103-123, DOI 10.1007/s10994-009-5119-5.

[26] Trevor Hastie and Rahul Mazumder. *Matrix Completion via Iterative Soft-Thresholded SVD*, Repository CRAN. (2015).

github.com/travisbrady/py-soft-impute

[27] Trevor Hastie, Rahul Mazumder, Jason D. Lee, and Reza Zadeh. *Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares*, J. Mach. Learn. Res. **16** (2015), 3367-3402.

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* 2^{nd} *ed.*, Springer, New York, 2009.

[29] Roger A. Horn, Charles R. Johnson. *Matrix Analysis* 2^{nd} *ed.*, Cambridge, New York, 2013.

[30] Isolation Forest.
scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[31] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*, Springer, New York, 2013.

[32] Jupyter Notebook.
jupyter.org/.

[33] Suyeon Kang and Jongwoo Song. *Feature Selection for Continuous Aggregate Response and its Application to Auto Insurance Data*, Expert Systems With Applications. **93** (2018), 104-117, DOI 10.1016/j.eswa.2017.10.007.

[34] Thomas Keck. *FastBDT: A speed-optimized and cache-friendly implementation of stochastic gradient-boosted decision trees for multivariate classification*, CoRR. (2016), arXiv:1609.06119.

[35] Won-Joong Kim, David Mayers, Clifford W. Smith and Jr. *On the Choice of Insurance*

Distribution Systems, J. Risk and Insurance. **63** (1996), no. 2, 207-227.

[36] Learning curve
`scikit-learn.org/stable/modules/learning_curve.html`

[37] Guillaume Lemaitre, Fernando Nogueira, and Christos K. Aridas. *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning*, J. Mach. Learn. Res. **18** (2017), 1-5.

[38] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. *Isolation Forest*, ICDM '08. (2008), 413-422.

[39] Xi Liu, Muhe Xie, Xidao Wen, Rui Chen, Yong Ge, Nick Duffield, and Na Wang. *A Semi-Supervised and Inductive Embedding Model for Churn Prediction of Large-Scale Mobile Games*, ICDM 2018, (2018), arXiv:1808.06573v3.

[40] Logistic Regression
`scikit-learn.org/stable/modules/generated/sklearn.linear_model`
`.LogisticRegression`

[41] Victoria Lopez, Alberto Fernandez, Salvador Garcia, Vasile Palade, and Francisco Herrera. *An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics*, Information Sciences. **250** (2013), 113-141, DOI 10.1016/j.ins.2013.07.007.

[42] Scott Lundberg, Su-In Lee. *A Unified Approach to Interpreting Model Predictions*, NIPS. (2017), arXiv:1705.07874v2.

[43] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook 2nd*

ed, Springer, New York, 2010, DOI 10.1007/978-0-387-09823-4.

[44] *MarketLine Industry Profile: Non-Life Insurance in North America*, MarketLine. (2017). www.marketline.com

[45] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. *Spectral Regularization Algorithms for Learning Large Incomplete Matrices*, J. Mach. Learn. Res. **11** (2010), 2287-2322.

[46] Rory Mitchell and Eibe Frank. *Accelerating the XGBoost algorithm using GPU computing*, PeerJ Comput. Sci. **3** (2017), DOI 10.7717/peerj-cs.127.

[47] Mehdi Naseriparsa and Mohammad Mansour Riahi Kashani. *Combination of PCA with SMOTE Resampling to Boost the Prediction Rate in Lung Cancer Dataset*, International J. Computer Applications. **7** (2013), no.3, 33-38, arXiv:1403.1949v1.

[48] Maria Oskarsdottir, Tine Van Calster, Bart Baesens, Wilfried Lemahieu, and Jan Vanthienen. *Time series for early churn detection: Using similarity based classification for dynamic networks*, Expert Systems With Applications. **106** (2018), 55-65, DOI: 10.1016/j.eswa
.2018.04.003.

[49] Adrian Payne and Pennie Frow. *Customer Relationship Management: from Strategy to Implementation*, J. Marketing Management. **22** (2006), 135-168.

[50] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. *Tunability: Importance of Hyperparameters of Machine Learning Algorithms*, (2018), arXiv:1802.09596.

[51] *Property & Casualty Insurers That Most Satisfy Independent Agents Have Best Overall Financial Performance and Profitability, J.D. Power Finds*, J.D. Power, 2018 U.S. Independent

Insurance Agent Satisfaction Study.
www.jdpower.com/business/press-releases/jd-power-2018-us-independent-insurance
-agent-satisfaction-study

[52] P. Pudil, J. NovoviEova, and J. Kittler. *Pattern Recognition Letters: Floating Search Methods in Feature Selection*, Department of Electronic and Electrical Engineering. **15** (1994), 1119-1125.

[53] Quickpipeline
github.com/Mottl/quickpipeline

[54] Random Forest Classifier
scikit-learn.org/stable/modules/generated/sklearn.ensemble
.RandomForestClassifier

[55] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*, MIT Press. (2006). www.gaussianprocess.org/gpml/chapters/

[56] Sharmila Ray. *Small Commercial Insurance: A Bright Spot In the U.S. Property Casualty Market*, McKinsey & Company. (2016).

[57] Laureen Regan and Sharon Tennyson. *Agent Discretion and the Choice of Insurance Marketing System*, J. Law and Economics. **39** (1996), no. 2, 637-666.

[58] Repeated Stratified K-Fold cross validator algorithm.
scikit-learn.org/stable/modules/generated/sklearn.model_selection
.RepeatedStratifiedKFold

[59] RFE
http://scikit-learn.org/stable/modules/generated/sklearn

.feature_selection.RFE.html

[60] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, HLT-NAACL. (2016). arXiv:1602.04938v3

[61] Daniel Schwarcz. *Differential Compensation and the "Race to the Bottom" in Consumer Insurance Markets*, Connecticut Insurance Law Journal. **15** (2009), no. 2.

[62] Select From Model algorithm
scikit-learn.org/stable/modules/feature_selection

[63] Sima Sharifirad, Azra Nazari, Mehdi Ghatee. *Modified SMOTE Using Mutual Information and Different Sorts of Entropies*, (2018), arXiv:1803.11002.

[64] Yogesh Sharma *Churn Prediction: Five Success Stories*: (2017)
www.zs.com/publications/whitepapers/churn-prediction-five-success-stories

[65] D. Asir Antony Gnana Singh, S. Appavu alias Balamurugan, and E. Jebamalar Leavline. *Literature Review on Feature Selection Methods for High-Dimensional Data*, International J. Computer Applications. **136** (2016), no. 1.

[66] SMOTE
imbalanced-learn.org/en/stable/generated/imblearn.over_sampling.SMOTE

[67] Jasper Snoek, Hugo Larochelle, Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*, ArXiv. (2012), arXiv:1206.2944v2.

[68] Eugen Stripling, Seppe vanden Broucke, Katrien Antonio, Bart Baesens, Monique Snoeck. *Profit maximizing logistic model for customer churn prediction using genetic algorithms*,

Swarm and Evolutionary Computation. **40** (2018), 116-130.

[69] Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, and Thomas Augustin. *Conditional variable importance for random forests*, BMC Bioinformatics. **9** (2008), 307, DOI: 10.1186/1471-2105-9-307.

[70] Scott D. Szymendera. *Workers? Compensation: Overview and Issues*, Congressional Research Service. (2017). `www.crs.gov`

[71] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*, KDD. (2013), DOI: 10.1145/2487575.2487629.

[72] Laura Tolosi and Thomas Lengauer. *Classification with Correlated Features: unreliability of feature ranking and solutions*, Department of Computational Biology and Applied Algorithmics. **27** (2008), no. 14, 1986-1994, DOI: 10.1093/bioinformatics/btr300.

[73] U.S. Census Bureau.
`www.census.gov/topics/education/educational-attainment/data/tables.html`

[74] J. Vijaya, E. Sivasankar. *Computing efficient features using rough set theory combined with ensemble classification techniques to improve the customer churn prediction in telecommunication sector*, Computing. **100** (2018), no. 8, 839-860, DOI: 10.1007/s00607-018-0633-6.

[75] XGBFIR.
`github.com/limexp/xgbfir`

[76] XGBoost.

xgboost.readthedocs.io/en/latest/

[77] XGBoost Tuning.
www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[78] Bing Zhu, Bart Baesens, Aimee Backiel, and Seppe K. L. M. vanden Broucke. *Benchmarking sampling techniques for imbalance learning in churn prediction*, J. Operational Res. Society. (2017), DOI: 10.1057/s41274-016-0176-1.

APPENDICES

APPENDIX A

Additional Feature Information

A.1 Feature Information

This section includes additional information regarding the concepts of complexity and Ease of doing business. Engineered features created using one-to-one assignment functions are also described.

Ease of doing business: This attribute can be further divided into personal attention and coverage flexibility. The `processing` features proxy personal attention by distinguishing between claim processing conduits while `sale_serv_ind` indicates in-person sales support vs call center. Coverage flexibility is represented by `risk_complex_ind` as complex policies are handled differently by the carriers and agents. Features `rate_adq`, `risk_complex_ind`, and `price_complex_ind` were engineered by Company XYZ. Additionally, `prem` and `rate_change` were adjusted by XYZ consistent with industry norms to account for certain industry factors to include mid-term policy cancellations.

Complexity: The account of a given policyholder is more complex if it involves multiple policies than span different lines of insurance, requires individual underwriting attention, and/or has coverage that extends to assets in different states. More complex accounts tend to belong to policyholders with larger businesses within the small commercial space and greater total premiums paid to the carrier. In regards to determining coverage, such policyholders are considered more sophisticated [35] and often use in-house or hired risk managers to purchase the appropriate coverage [10]. Risk managers have a studied understanding of a given company's insurance needs and are able to distinguish between the competing competitor quotes on factors other than simply premium and coverage. Depending on the state, risk managers can also request that agents disclose information regarding compensation [61]. This additional expertise can affect the influence of the independent agent.

Smaller business that lack sophisticated insurance purchasers are likely to possess limited knowledge of compensation practices and the influence such practices might

have over a given agent's recommendations. It is also likely that the business uses a given insurance agency out of familiarity [10]. In other words, the small business owner may be sole proprietor whom uses the same local insurance agent that they use for personal insurance. This type of client would be considered unsophisticated and therefore easily influenced by the insurance agent. Therefore, policyholders with more complex accounts and insurance needs tend to qualify as more sophisticated policyholders. Policyholders with less complex accounts tend to be less sophisticated.

There are other dimensions of the smaller policyholder that could impact their level of sophistication. Given that the majority of smaller policyholders are collocated geographically with the covered assets, some additional geographic aggregate information could be assigned at the policy level in order to represent complexity information.

Education index: Scores by state obtained through the U.S. Census Bureau [73] were used to map a given states ranking in per capita highest education level by state. This feature was intended for small commercial owners/employees that do not employ risk managers and ultimately must determine the coverage needed. This feature assumed that a better educated owner/employee will investigate the coverage in more detail and therefore lessen the agents influence over policyholder churn.

Urban Influence Codes: (UIC) Dataset published by the US Department of Agriculture (2013) assigns an rank describing the level of urban influence. Metro areas are assigned values based off their largest city. Non-metro areas are assigned a code based off the density of the largest town and proximity to nearest metro areas [11]. UICs are available for Federal Information Processing Standards code (FIPS) which were downloaded and mapped to policy `zip_code` to add policy-level spatial and density information.

Competitive state funds: (CSF) for workers compensation exists in 18 states [70]. Business within those states can choose between private insurance and state funds. The

presence of competitive state funds changes the competitive landscape for private carriers in that state offering workers compensation [70]. The presence of competitive state funds for the WC line is captured as a dichotomous feature and mapped to the existing feature `risk_state`. Competitive state fund market share and dividends per state were obtained by an independent analytical company working for Company XYZ and mapped to `risk_state`.

Tech score: The recent acceleration in data analysis, automation, and connected devices has impacted the insurance industry and will continue to do so [1]. This effect is generally coined *InsurTech* and is anticipated to increasingly effect the industry and redefine ease of use. Effects are occurring at the carrier, agency, and policyholder level. The most recent State Technology and Science Index scores states by forecasted future return on science and technology assets [12]. This return is largely affected by human capital within geographic boundaries. Tech scores were mapped to policy-level observation by `risk_state`.

A.2 Correlation Charts

	tenure_	rate_chng	prem_	UI_code	csf_market_	csf_div	tech_score	ed_score	PC_num	month_prog	prev_non	prev_NB	lag_1_auto	lag_1_cmp	lag_1_wc	lag_2_auto	lag_2_cmp	lag_2_wc
tenure_	1	0.39807408	0.12909806	0.05909097	0.01812281	0.02919389	0.05648565	0.07761704	0.17064956	0.00300178	0.06266662	-0.1049175	0.1991522	0.1644873	0.03037544	0.21013477	0.17233895	0.03849758
rate_chng	0.39807408	1	0.28092199	0.02410023	0.04053799	0.04129303	0.07631359	0.1026361	0.1348513	0.02781728	0.03812491	-0.0474919	0.146316	0.12669435	0.02331837	0.15032863	0.12889585	0.02633812
prem_	0.12909806	0.28092199	1	-0.0008537	0.00095106	0.00541676	0.04067371	0.03844054	0.03334295	0.01057598	0.05037126	0.01743684	0.10245809	0.17471396	0.07702026	0.10245884	0.17473483	0.07886436
UI_code	0.05909097	0.02410023	-0.0008537	1	0.2198715	0.10026416	-0.1383663	-0.1192871	-0.0471103	-0.0050067	-0.08025	-0.0613448	0.02941541	-0.0814314	0.02300322	0.03742886	-0.0798119	0.01955205
csf_market_	0.01812281	0.04053799	0.00095106	0.2198715	1	0.8733229	0.2737518	0.28087054	-0.0215979	-0.0037936	-0.0375257	-0.0144056	-0.0296813	-0.0199364	0.17369327	-0.0286366	-0.0207068	0.16937521
csf_div	0.02919389	0.04129303	0.00541676	0.10026416	0.8733229	1	0.21742393	0.30708211	0.02513623	-0.0017219	-0.0238547	-0.0275411	-0.0048183	-0.0099307	0.14163001	-0.0042858	-0.0103027	0.13934024
tech_score	0.05648565	0.07631359	0.04067371	-0.1383663	0.2737518	0.21742393	1	0.94159979	0.0089945	0.00856346	0.01843137	-0.0278031	0.08424606	0.08497825	-0.0716842	0.08237532	0.08444484	-0.068608
ed_score	0.07761704	0.1026361	0.03844054	-0.1192871	0.28087054	0.30708211	0.94159979	1	0.05135399	0.00903826	0.01405761	-0.0425723	0.10944782	0.07597231	-0.08513	0.10836726	0.07554168	-0.0820112
PC_num	0.17064956	0.1348513	0.03334295	-0.0471103	-0.0215979	0.02513623	0.0089945	0.05135399	1	0.00637671	0.16263719	0.02236685	0.20497913	0.18976724	0.0876524	0.20691317	0.19161067	0.09116221
month_prog	0.00300178	0.02781728	0.01057598	-0.0050067	-0.0037936	-0.0017219	0.00856346	0.00903826	0.00637671	1	0.00858058	0.00237452	-0.0061382	0.00776155	0.00433932	-0.0096019	0.01009555	0.00290683
prev_non	0.06266662	0.03812491	0.05037126	-0.08025	-0.0375257	-0.0238547	0.01843137	0.01405761	0.16263719	0.00858058	1	0.60341922	0.50427555	0.73988112	0.61850876	0.50834419	0.74546321	0.62958362
prev_NB	-0.1049175	-0.0474919	0.01743684	-0.0613448	-0.0144056	-0.0275411	-0.0278031	-0.0425723	0.02236685	0.00237452	0.60341922	1	0.36076254	0.59209285	0.56647468	0.33460771	0.57230402	0.54581613
lag_1_auto	0.1991522	0.146316	0.10245809	0.02941541	-0.0296813	-0.0048183	0.08424606	0.10944782	0.20497913	-0.0061382	0.50427555	0.36076254	1	0.60137405	0.46339269	0.96700988	0.60057957	0.46401533
lag_1_cmp	0.1644873	0.12669435	0.17471396	-0.0814314	-0.0199364	-0.0099307	0.08497825	0.07597231	0.18976724	0.00776155	0.73988112	0.59209285	0.60137405	1	0.6391754	0.59949008	0.45781864	0.64267534
lag_1_wc	0.03037544	0.02331837	0.07702026	0.02300322	0.17369327	0.14163001	-0.0716842	-0.08513	0.0876524	0.00433932	0.61850876	0.56647468	0.46339269	0.6391754	1	0.45781864	0.6362707	0.97660536
lag_2_auto	0.21013477	0.15032863	0.10245884	0.03742886	-0.0286366	-0.0042858	0.08237532	0.10836726	0.20691317	-0.0096019	0.50834419	0.33460771	0.96700988	0.59949008	0.45781864	1	0.60474297	0.4630757
lag_2_cmp	0.17233895	0.12889585	0.17473483	-0.0798119	-0.0207068	-0.0103027	0.08444484	0.07554168	0.19161067	0.01009555	0.74546321	0.57230402	0.60057957	0.99334385	0.6362707	0.60474297	1	0.64369297
lag_2_wc	0.03849758	0.02633812	0.07886436	0.01955205	0.16937521	0.13934024	-0.068608	-0.0820112	0.09116221	0.00290683	0.62958362	0.54581613	0.46401533	0.64267534	0.97660536	0.4630757	0.64369297	1
UW_freq	0.06823032	0.01831163	-0.0369683	0.06133359	0.00025841	0.04847834	0.0190899	0.03105687	0.0363104	-0.2547682	-0.0739807	-0.0796134	0.11492833	-0.0702746	-0.1171018	0.11682991	-0.0701904	-0.1180346
SE_freq	-0.0658219	-0.0606149	-0.0197024	-0.0485229	0.10851529	0.05234161	-0.0992844	-0.2080186	-0.0940655	-0.2931955	0.04623486	0.12704868	-0.0526184	0.03967446	0.13501524	-0.0588737	0.03568219	0.13484146
agnt_freq	0.09086462	0.05425394	0.05693446	-0.0682973	-0.0327267	-0.0259684	0.03835626	0.03228246	0.16653208	-0.0435606	0.79583814	0.70067031	0.58675771	0.86639259	0.6917939	0.58205018	0.86219312	0.69285316
mstr_freq	0.09566422	0.0602708	0.04995542	0.02944971	0.04298636	0.057456	0.0293727	0.0634521	0.23620838	-0.0383922	0.46010057	0.39880322	0.34627442	0.51152571	0.41571381	0.34615453	0.51001885	0.41630294
zip_freq	0.01055033	0.0549102	0.01030949	-0.0602946	0.22631833	0.12592944	-0.0594432	-0.0550111	-0.0266854	-0.0473286	0.02201902	0.05105001	-0.0199267	0.03887801	0.11046791	-0.0201618	0.03805789	0.10858858
county_freq	-0.0653066	-0.0008633	-0.0012727	-0.0636928	-0.1407612	-0.115371	-0.0921102	-0.1039911	-0.00243	-0.0396777	0.07160603	0.08767822	-0.0789739	0.05790381	0.04972924	-0.0846625	0.05693533	0.05119199
prem_freq	-0.0740904	-0.243719	-0.7698889	0.01354311	-0.0280107	-0.0296507	-0.0296407	-0.028916	-0.0429496	-0.1903072	-0.0457534	-0.0082434	-0.1070851	-0.1533668	-0.1015515	-0.1089376	-0.1543692	-0.1024856
rate_adq	-0.1875517	-0.0285047	-0.0128996	-0.0400141	-0.0901505	-0.0758287	0.04675254	0.04046802	-0.0328896	0.00020267	0.00521225	0.03329297	-0.0488749	-0.0134414	-0.0173272	-0.0517219	-0.0152307	-0.0185884
lag_1	0.15500555	0.12398802	0.16937301	-0.0719331	-0.0109336	-0.0017663	0.07095108	0.06622318	0.17664965	0.00602568	0.75896338	0.61427569	0.66860249	0.97818483	0.7288464	0.6627781	0.97234635	0.72685647
lag_2	0.16378018	0.12646796	0.16949246	-0.0708566	-0.013203	-0.0031811	0.07077827	0.0661051	0.17906682	0.00803821	0.76586485	0.59054575	0.66386561	0.97278578	0.72030127	0.67041909	0.97844257	0.73171258
lag_diff	-0.073607	-0.0354704	-0.0075528	0.00737516	0.01938428	0.00871979	-0.0029663	-0.0082446	-0.0650057	-0.0088727	-0.1233628	0.1512866	-0.0222336	-0.0188473	0.01585915	-0.0992653	-0.0846185	-0.0714783
lag_diff_auto	-0.0466292	-0.0276369	-0.0048499	-0.0147728	-0.0155579	-0.0273937	0.01031989	-0.0015225	-0.0344886	0.01087793	-0.0627879	0.08426935	0.03821766	-0.0262365	-0.0116718	-0.1269107	-0.0453693	-0.0241208
lag_diff_cmp	-0.0808431	-0.0373787	-0.0091753	-0.0072967	0.00059519	-0.0057596	-0.0040457	-0.010466	-0.0616332	-0.0209655	-0.1167715	0.12887403	-0.0535242	-0.0188885	-0.0266625	-0.0926921	-0.0955178	-0.0512672
lag_diff_wc	-0.0376888	-0.0149315	-0.0098525	0.01790056	0.02263851	0.01285928	-0.0002499	-0.0019578	-0.0201942	0.0192135	-0.0790608	0.08095043	-0.026004	-0.0354945	0.05232422	-0.044477	-0.0498986	-0.0973238
lag_diff_abs	0.03345583	0.04467613	0.11017463	-0.0793158	0.01159188	0.01837574	0.04090207	0.03961355	0.10260033	-0.0268266	0.49125596	0.4678765	0.41715813	0.58348887	0.49795297	0.3991664	0.57412103	0.48817041

Figure 17: *Correlation Matrix 1*

UW_freq	SE_freq	agnt_freq	mstr_freq	zip_freq	county_freq	prem_freq	rate_adq	lag_1	lag_2	lag_diff	lag_diff_aut	lag_diff_cm	lag_diff_wc	lag_diff_abs
0.06823032	-0.0658219	0.09086462	0.09566422	0.01055033	-0.0653066	-0.0740904	-0.1875517	0.15500555	0.16378018	-0.073607	-0.0466292	-0.0808431	-0.0376888	0.03345583
0.01831163	-0.0606149	0.05425394	0.0602708	0.0549102	-0.0008633	-0.243719	-0.0285047	0.12398802	0.12646796	-0.0354704	-0.0276369	-0.0373787	-0.0149315	0.04467613
-0.0369683	-0.0197024	0.05693446	0.04995542	0.01030949	-0.0012727	-0.7698889	-0.0128996	0.16937301	0.16949246	-0.0075528	-0.0048499	-0.0091753	-0.0098525	0.11017463
0.06133359	-0.0485229	-0.0682973	0.02944971	-0.0602946	-0.6636928	0.01354311	-0.0400141	-0.0719331	-0.0708566	0.00737516	-0.0147728	-0.0072967	0.01790056	-0.0793158
0.00025841	0.10851529	-0.0327267	0.04298636	0.22631833	-0.1407612	-0.0280107	-0.0901505	-0.0109336	-0.013203	0.01938428	-0.0155579	0.00059519	0.02263851	0.01159188
0.04847834	0.05234161	-0.0259684	0.057456	0.12592944	-0.115371	-0.0296507	-0.0758287	-0.0017663	-0.0031811	0.00871979	-0.0273937	-0.0057596	0.01285928	0.01837574
0.0190899	-0.0992844	0.03835626	0.0293727	-0.0594432	-0.0921102	-0.0296407	0.04675254	0.07095108	0.07077827	-0.0029663	0.01031989	-0.0040457	-0.0002499	0.04090207
0.03105687	-0.2080186	0.03228246	0.0634521	-0.0550111	-0.1039911	-0.028916	0.04046802	0.06622318	0.0661051	-0.0082446	-0.0015225	-0.010466	-0.0019578	0.03961355
0.0363104	-0.0940655	0.16653208	0.23620838	-0.0266854	-0.00243	-0.0429496	-0.0328896	0.17664965	0.17906682	-0.0650057	-0.0344886	-0.0616332	-0.0201942	0.10260033
-0.2547682	-0.2931955	-0.0435606	-0.0383922	-0.0473286	-0.0396777	-0.1903072	0.00020267	0.00602568	0.00803821	-0.0088727	0.01087793	-0.0209655	0.0192135	-0.0268266
-0.0739807	0.04623486	0.79583814	0.46010057	0.02201902	0.07160603	-0.0457534	0.00521225	0.75896338	0.76586485	-0.1233628	-0.0627879	-0.1167715	-0.0790608	0.49125596
-0.0796134	0.12704868	0.70067031	0.39880322	0.05105001	0.08767822	-0.0082434	0.03329297	0.61427569	0.59054575	0.1512866	0.08426935	0.12887403	0.08095043	0.4678765
0.11492833	-0.0526184	0.58675771	0.34627442	-0.0199267	-0.0789739	-0.1070851	-0.0488749	0.66860249	0.66386561	-0.0222336	0.03821766	-0.0535242	-0.026004	0.41715813
-0.0702746	0.03967446	0.86639259	0.51152571	0.03887801	0.05790381	-0.1533668	-0.0134414	0.97818483	0.97278578	-0.0188473	-0.0262365	-0.0188885	-0.0354945	0.58348887
-0.1171018	0.13501524	0.6917939	0.41571381	0.11046791	0.04972924	-0.1015515	-0.0173272	0.7288464	0.72030127	0.01585915	-0.0116718	-0.0266625	0.05232422	0.49795297
0.11682991	-0.0588737	0.58205018	0.34615453	-0.0201618	-0.0846625	-0.1089376	-0.0517219	0.6627781	0.67041909	-0.0992653	-0.1269107	-0.0926921	-0.044477	0.3991664
-0.0701904	0.03568219	0.86219312	0.51001885	0.03805789	0.05693533	-0.1543692	-0.0152307	0.97234635	0.97844257	-0.0846185	-0.0453693	-0.0955178	-0.0498986	0.57412103
-0.1180346	0.13484146	0.69285316	0.41630294	0.10858858	0.05119199	-0.1024856	-0.0185884	0.72685647	0.73171258	-0.0714783	-0.0241208	-0.0512672	-0.0973238	0.48817041
1	0.160908	-0.0135039	-0.0015028	-0.052744	-0.0940915	0.09447235	-0.068977	-0.0570731	-0.0567711	0.01364131	-0.0044796	-0.0104883	0.00786889	-0.0664827
0.160908	1	0.1004542	0.02243432	0.1365593	0.14015016	0.07636028	-0.0455489	0.04131286	0.03776654	0.0484878	0.0138767	0.03556128	-0.0034388	0.0629182
-0.0135039	0.1004542	1	0.57554988	0.03795705	0.06079613	-0.0399739	-0.0015063	0.88798977	0.88383604	-0.0189929	-0.021314	-0.0215253	-0.0256969	0.52111478
-0.0015028	0.02243432	0.57554988	1	0.03183587	-0.011184	-0.0294525	-0.0005662	0.51379594	0.51263987	-0.0220862	-0.0222709	-0.0196002	-0.0165265	0.29579422
-0.052744	0.1365593	0.03795705	0.03183587	1	0.40122196	-0.0286315	-0.0386079	0.04003409	0.03812386	0.00663606	5.15E-05	4.02E-05	0.00300692	0.0464429
-0.0940915	0.14015016	0.06079613	-0.011184	0.40122196	1	-0.0007266	0.02491096	0.05683815	0.05552237	-0.0046017	0.01444317	0.00527845	-0.013243	0.07546393
0.09447235	0.07636028	-0.0399739	-0.0294525	-0.0286315	-0.0007266	1	0.00741036	-0.1538974	-0.1549461	0.012792	0.01595069	0.01537165	0.00184426	-0.0877531
-0.068977	-0.0455489	-0.0015063	-0.0005662	-0.0386079	0.02491096	0.00741036	1	-0.009254	-0.0112373	0.0173403	0.01104162	0.02268109	0.00663242	0.0138628
-0.0570731	0.04131286	0.88798977	0.51379594	0.04003409	0.05683815	-0.1538974	-0.009254	1	0.99261886	-0.0080697	-0.0180678	-0.0225004	-0.0169859	0.60741686
-0.0567711	0.03776654	0.88383604	0.51263987	0.03812386	0.05552237	-0.1549461	-0.0112373	0.99261886	1	-0.0926303	-0.0557213	-0.0907974	-0.0644881	0.59481357
0.01364131	0.0484878	-0.0189929	-0.0220862	0.00663606	-0.0046017	0.012792	0.0173403	-0.0080697	-0.0926303	1	0.35757148	0.80498487	0.43485557	0.01776014
-0.0044796	0.0138767	-0.021314	-0.0222709	5.15E-05	0.01444317	0.01595069	0.01104162	-0.0180678	-0.0557213	0.35757148	1	0.18265352	0.07205573	0.01691651
-0.0104883	0.03556128	-0.0215253	-0.0196002	4.02E-05	0.00527845	0.01537165	0.02268109	-0.0225004	-0.0907974	0.80498487	0.18265352	1	0.08016799	0.00838403
0.00786889	-0.0034388	-0.0256969	-0.0165265	0.00300692	-0.013243	0.00184426	0.00663242	-0.0169859	-0.0644881	0.43485557	0.07205573	0.08016799	1	0.01097452
-0.0664827	0.0629182	0.52111478	0.29579422	0.0464429	0.07546393	-0.0877531	0.0138628	0.60741686	0.59481357	0.01776014	0.01691651	0.00838403	0.01097452	1

Figure 18: *Correlation Matrix 2*

	churn	line_	risk_state	coast_ind	risk_comple	M_ind	seg_strategy	industry_	price_compl	prgm_	diff_mst_ag	diff_state	csf_ind
churn	0	1.35E-25	5.45E-69	0.00255641	0.22911875	1.07E-39	1.75E-217	4.50E-286	7.05E-69	2.40E-241	0.14552755	2.80E-09	1.57E-07
line_	1.35E-25	0	0	2.05E-114	0	0	0	0	0	0	6.86E-13	8.06E-40	0
risk_state	5.45E-69	0	0	0	2.92E-61	0	0	0	0	0	0	0	0
coast_ind	0.00255641	2.05E-114	0	0	2.10E-06	0.13581016	1.28E-35	3.87E-66	0.10242105	0	2.61E-09	0	2.48E-15
risk_complex	0.22911875	0	2.92E-61	2.10E-06	0	0	8.36E-265	0	0	0	3.58E-22	1.12E-17	0.00017772
M_ind	1.07E-39	0	0	0.13581016	0	0	0	0	0.0334989	0	0.8831854	4.87E-22	4.09E-85
seg_strategy	1.75E-217	0	0	1.28E-35	8.36E-265	0	0	0	1.21E-256	0	1.09E-29	4.58E-95	1.44E-78
industry_	4.50E-286	0	0	3.87E-66	0	0	0	0	0	0	1.85E-89	7.15E-220	6.78E-292
price_compl	7.05E-69	0	0	0.10242105	0	0.0334989	1.21E-256	0	0	0	8.09E-17	1.83E-51	3.56E-185
prgm_	2.40E-241	0	0	0	0	0	0	0	0	0	0	0	0
diff_mst_ag	0.14552755	6.86E-13	0	2.61E-09	3.58E-22	0.8831854	1.09E-29	1.85E-89	8.09E-17	0	0	0.98375175	1.04E-07
diff_state	2.80E-09	8.06E-40	0	0	1.12E-17	4.87E-22	4.58E-95	7.15E-220	1.83E-51	0	0.98375175	0	8.90E-56
csf_ind	1.57E-07	0	0	2.48E-15	0.00017772	4.09E-85	1.44E-78	6.78E-292	3.56E-185	0	1.04E-07	8.90E-56	0
PC_cat	1.49E-12	2.59E-187	0	3.49E-30	0	3.10E-218	8.49E-166	0	0	0	0	3.63E-77	1.30E-13
sal_serv_ind	1.37E-05	3.00E-06	0	3.25E-09	5.62E-55	8.75E-06	1.16E-72	1.05E-145	1.41E-46	0	6.04E-25	9.06E-10	6.45E-128
month_nom	4.04E-10	1.32E-40	4.43E-86	0.08775385	5.36E-07	2.75E-14	5.56E-37	1.91E-102	3.06E-07	0	5.06E-07	0.42731354	1.57E-11
term	1.07E-21	9.61E-24	1.28E-08	0.00301083	0.00020276	1.05E-14	3.14E-23	9.27E-41	1.31E-42	0	0.00303227	0.62864307	0.01305841
agnt_rev	6.15E-48	0	0	0	1.64E-204	1.13E-280	0	0	0	0	0	0	0
mstr_rev	2.03E-45	0	0	6.28E-51	9.81E-139	1.13E-161	0	0	0	0	0	0	0
lag_diff_wc_	7.41E-07	0	0	1.38E-10	6.80E-21	4.78E-96	5.37E-103	5.28E-208	2.01E-87	0	0	5.05E-16	0
prem_bin_ac	6.15E-187	0	1.46E-281	2.61E-13	0	0	0	0	0	0	5.40E-11	7.34E-30	1.64E-54
agnt_bin_6	1.82E-17	1.54E-92	0	2.23E-30	3.48E-18	5.98E-25	0	0	1.30E-144	0	0	0	2.99E-105
uw_bin_6	8.27E-32	1.40E-272	0	2.95E-16	1.94E-20	2.99E-26	6.05E-174	0	6.87E-216	0	0	4.41E-07	0
zip_bin_6	0.29038293	2.48E-246	0	0	4.12E-07	1.04E-173	5.66E-210	0	1.09E-14	0	3.36E-39	0	0
county_bin_6	2.34E-34	3.95E-308	0	0	2.89E-14	1.65E-34	7.66E-121	0	2.87E-181	0	9.03E-99	0	0
mstr_bin_6	5.60E-11	1.18E-82	0	1.93E-27	7.64E-09	2.49E-41	6.69E-174	0	2.23E-129	0	0	0	0
se_bin_3	0.0009106	2.11E-259	0	3.22E-08	2.12E-16	7.23E-54	6.08E-30	2.26E-128	9.66E-209	0	1.99E-56	1.79E-12	0
year_feat	1.72E-16	5.76E-12	0.11049796	0.07670164	9.74E-07	1.17E-12	5.45E-15	3.10E-25	3.99E-41	0	0.0210621	0.58641145	0.00638078
quarter_feat	2.94E-06	2.00E-20	2.25E-28	0.02656025	0.0009172	1.41E-14	1.75E-22	3.44E-39	0.03790295	1.77E-181	0.00028621	0.19493696	0.02922741

Figure 19: χ^2 *Matrix 1*

PC_cat	sal_serv_ind	month_nom	term	agnt_rev	mstr_rev	lag_diff_wc_	prem_bin_ac	agnt_bin_6	uw_bin_6	zip_bin_6	county_bin_6	mstr_bin_6	se_bin_3	year_feat	quarter_feat
1.49E-12	1.37E-05	4.04E-10	1.07E-21	6.15E-48	2.03E-45	7.41E-07	6.15E-187	1.82E-17	8.27E-32	0.29038293	2.34E-34	5.60E-11	0.0009106	1.72E-16	2.94E-06
2.59E-187	3.00E-06	1.32E-40	9.61E-24	0	0	0	0	1.54E-92	1.40E-272	2.48E-246	3.95E-308	1.18E-82	2.11E-259	5.76E-12	2.00E-20
0	0	4.43E-86	1.28E-08	0	0	0	1.46E-281	0	0	0	0	0	0	0.11049796	2.25E-28
3.49E-30	3.25E-09	0.08775385	0.00301083	0	6.28E-51	1.38E-10	2.61E-13	2.23E-30	2.95E-16	0	0	1.93E-27	3.22E-08	0.07670164	0.02656025
0	5.62E-55	5.36E-07	0.00020276	1.64E-204	9.81E-139	6.80E-21	0	3.48E-18	1.94E-20	4.12E-07	2.89E-14	7.64E-09	2.12E-16	9.74E-07	0.0009172
3.10E-218	8.75E-06	2.75E-14	1.05E-14	1.13E-280	1.13E-161	4.78E-96	0	5.98E-25	2.99E-26	1.04E-173	1.65E-34	2.49E-41	7.23E-54	1.17E-12	1.41E-14
8.49E-166	1.16E-72	5.56E-37	3.14E-23	0	0	5.37E-103	0	0	6.05E-174	5.66E-210	7.66E-121	6.69E-174	6.08E-30	5.45E-15	1.75E-22
0	1.05E-145	1.91E-102	9.27E-41	0	0	5.28E-208	0	0	0	0	0	0	2.26E-128	3.10E-25	3.44E-39
0	1.41E-46	3.06E-07	1.31E-42	0	0	2.01E-87	0	1.30E-144	6.87E-216	1.09E-14	2.87E-181	2.23E-129	9.66E-209	3.99E-41	0.03790295
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.77E-181
0	6.04E-25	5.06E-07	0.00303227	0	0	0	5.40E-11	0	0	3.36E-39	9.03E-99	0	1.99E-56	0.0210621	0.00028621
3.63E-77	9.06E-10	0.42731354	0.62864307	0	0	5.05E-16	7.34E-30	0	4.41E-07	0	0	0	1.79E-12	0.58641145	0.19493696
1.30E-13	6.45E-128	1.57E-11	0.01305841	0	0	0	1.64E-54	2.99E-105	0	0	0	0	0	0.00638078	0.02922741
0	0	5.78E-06	1.80E-08	0	0	2.02E-95	0	0	0	1.16E-23	8.29E-52	0	0.00E+00	6.63E-11	0.00100245
0	0	0.00013399	0.48054865	0	0	0	2.18E-76	0	0	1.39E-158	8.41E-167	0	0	0.08951579	0.2093324
5.78E-06	0.00013399	0	0	0	0	3.97E-125	6.63E-17	2.61E-120	0	8.94E-90	3.78E-138	1.56E-107	0	0	0
1.80E-08	0.48054865	0	0	0	0	0	8.08E-15	2.33E-124	0	1.85E-115	0	2.65E-167	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2.02E-95	0	3.97E-125	0	0	0	0	2.92E-47	0	0	0	0	0	0	0	7.67E-133
0	2.18E-76	6.63E-17	8.08E-15	0	0	2.92E-47	0	5.11E-96	3.37E-48	3.43E-48	3.03E-32	3.98E-91	3.53E-10	6.15E-11	6.45E-17
0	0	2.61E-120	2.33E-124	0	0	0	5.11E-96	0	0	6.18E-270	0	0	0	8.16E-37	1.98E-122
0	0	0	0	0	0	0	3.37E-48	0	0	1.29E-175	0	0	0	0	0
1.16E-23	1.39E-158	8.94E-90	1.85E-115	0	0	0	3.43E-48	6.18E-270	1.29E-175	0	0	0	0	1.04E-68	2.35E-91
8.29E-52	8.41E-167	3.78E-138	0	0	0	0	3.03E-32	0	0	0	0	0	0	2.31E-259	9.91E-148
0	0	1.56E-107	2.65E-167	0	0	0	3.98E-91	0	0	0	0	0	0	5.21E-34	5.59E-104
0.00E+00	0	0	0	0	0	0	3.53E-10	0	0	0	0	0	0	0	0
6.63E-11	0.08951579	0	0	0	0	0	6.15E-11	8.16E-37	0	1.04E-68	2.31E-259	5.21E-34	0	0	0
0.00100245	0.2093324	0	0	0	0	7.67E-133	6.45E-17	1.98E-122	0	2.35E-91	9.91E-148	5.59E-104	0	0	0

Figure 20: χ^2 *Matrix 2*

	churn	lag_bin_1
churn	0.000000e+00	9.701391e-07
lag_bin_1	9.701391e-07	0.000000e+00
lag_bin_2	1.416166e-07	0.000000e+00
prev_non_bin	5.945621e-03	0.000000e+00
prev_np_bin	1.135942e-02	0.000000e+00
lag_diff_bin	2.816139e-03	0.000000e+00
lag_diff_bin_abs	1.649485e-02	0.000000e+00
seg_strategy	1.746275e-217	0.000000e+00
industry_	4.500605e-286	0.000000e+00

Figure 21: χ^2 *Matrix: Response*

APPENDIX B

Tree-based Algorithms

B.1 Decision Tree Classifier

Decision trees (DT) are non-parametric models that repeatedly partition the decision space according to certain criteria [43, 46], criteria typically involves the minimization of the sum of squared errors (SSE) [31] for regression trees and the Gini[33] index for classifiers. DT use a greedy approach known as recursive binary splitting. The approach makes the optimal split for each subsequent step than minimizes the error function or maximizes some criterion [46] without global consideration [31]. Splitting continues until a stopping criterion is met and the decision space is segmented into regions [43]. For classifiers, the prediction is then calculated by the mode or mean of the region that first satisfied the stopping criteria.

The decision tree divides the feature space X into J different regions $R_1, R_2, ..., R_J$ The classifier model is of the form:

$$f(X) = \sum_{m=1}^{J} c_m I(x \in R_j) \tag{18}$$

with I being an indicator function **s.t.**

$$I(x \in R_j) = \begin{cases} 1 & \text{if } x \in R_j \\ 0 & \text{else} \end{cases}$$

The DT predicts the mode of the classes in R_j. For a regression decision tree, the loss function is typically SSE. For the classifier, the loss function is typically the Gini index. The Gini index is of the form:

$$G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk}) \tag{19}$$

[33]Similar to entropy described in 4.1

s.t. $\hat{p}_{jk}$ = the proportion of observations in R_j from the kth class. Cross entropy described in 4.1 can also be used as a loss function.

B.2 Bagging and Random Forests

Bagging is an established method often used by ensemble tree-based models. Also referred to as bootstrapping aggregating [43], this method creates subsets or bootstrapped training sets of the original dataset using random sampling with replacement. Base models then learn their respective subset and render predictions. Predictions are aggregated into a single prediction [43, 31]. Bagging model is of the form:

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{20}$$

With $\hat{f}^{*b}(x)$ calculated by training the method on the bth set out of B bootstrapped training sets. A class probability is predicted for each method and averaged for the final prediction.

Random forests are an ensemble of decision trees that apply a technique in addition to bootstrap aggregating for feature de-correlation [31], often referred to as *feature* bagging. With feature bagging, not only is a training sample subset used but also a feature subset thus preventing the repeated use of the stronger predictors. The resulting average prediction for the ensemble of trees has less variation and less of a chance of overfitting.

Let p = full set of model features. For a given decision tree within a Random Forest, for each split m features are randomly sampled from p **s.t.** $m \approx \sqrt{p}$

B.3 Boosting, and Feature importance

Similar to bagging, boosting involves an ensemble of decision trees. Boosting is a general error reduction method for weak learners. Algorithmically, it can be understood by analyzing the Adaboost model [18]. Initial weights are assigned to each observation and adjusted after the predictions of each weak learner. Incorrect and correct responses are given more and less weight respectively. Subsequent weak learners are then forced to focus on the previously misclassified observations [18]. The final output is then computed by combining the outputs of the weak learners by weighted majority voting. As a result, the ensemble of models sequentially operate in a complementary manner in order to improve performance [43, 46, 31]. Gradient boosting is the incorporation of gradient descent into the boosting algorithm [46]. In other words, the boosting algorithm outputs predictions minimizing a given loss function using gradient descent.

B.4 SHAP

[42] introduces a framework (SHapley Additive exPlanation or SHAP) that uses game theory and conditional expectation functions for interpreting black box predictions at the sample level and for dataset level feature importance. Given the inconsistencies between current embedded methods discussed in Chapter 4 (Gain vs. Fscore), SHAP feature importance values could provide better results. Figure 17 is one of several possible outputs and functions that SHAP can perform.

B.5 XGBoost

Ensembles are generally sets of models performing the same prediction task in order to arrive at a final solution [55, 43]. The set of models can consist of weak and/or strong base learners, different classes altogether, or even a particular class restricted in a manner than renders all the models weak.

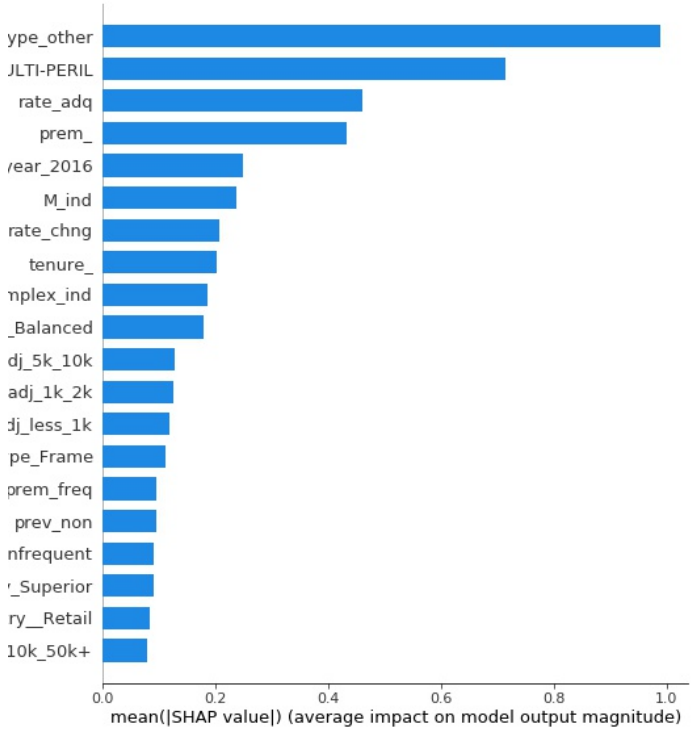


Figure 22: *SHAP: Global*

B.5.1 Gradient Boosted Algorithms

Ensembles increase generalized learning as well as reduce overfitting [43]. Tree-based methods that use bagging[34] and boosting are considered ensemble models. Base learners train on data, subsets of data, and even subsets of features while minimizing their respective loss function.

$$\mathscr{L}(\psi(\mathbf{x}), y) \tag{21}$$

[34] Also called bootstrapping, the method randomly samples subsets of the data for base learners to train on. Results are then averaged to a single prediction

Gradient boosted methods sequentially add homogenous weak learners in order to produce a strong learner. The method assumes an initial weak base learner $F_m(\mathbf{X})$ of M and trains a subsequent additive base learner $h(\mathbf{X})$ with the residual[35] of the previous base learner. The optimal learner $h_m \in M$ to add can be found by minimizing the loss function with the additional learner [18].

$$F_m(\mathbf{X}) = F_{m-1}(\mathbf{X}) + \underset{h_m \in M}{\operatorname{argmin}} L(y, F_{m-1}(\mathbf{X}) + h_m(\mathbf{x})) \tag{22}$$

B.5.2 Gradient Boosted XGB

Starting from equation (16) in Chapter 4.4. References to equation (4.16) are referring to an equation used in [7]. As with gradient boosting shown above, the optimal subsequent learner is selected by including it in the loss function to be minimized. However with XGB, the complexity penalty is also included. The XGB loss function is defined:

$$\mathcal{L}_m = \mathscr{L}(\mathbf{y}, \hat{\mathbf{y}}_{m-1} + F_m(\mathbf{X})) + \Omega(F_m) \tag{23}$$

The computations by Chen and Guestrin optimize by using a 2^{nd} order Taylor approximation on the loss function, not including the regularizer. After removing the constant term and performing the following assignments for the gradient and hessian matrices:

$$\mathbf{G} \leftarrow \sum_i^n g_i \leftarrow \sum_i^n \partial_{\hat{y}_{m-1}} \mathscr{L}((y^{(i)}, \hat{y}_{m-1}))$$

$$\mathbf{H} \leftarrow \sum_i^n h_i \leftarrow \sum_i^n \partial^2_{\hat{y}_{m-1}} \mathscr{L}((y^{(i)}, \hat{y}_{m-1}))$$

[35] This is actually the pseudo residual and is the partial derivation of the loss function using the previous base learner

The approximated loss function becomes:

$$\mathcal{L}_m \approx [\mathbf{G}F_m(\mathbf{X}) + \frac{1}{2}\mathbf{H}F_m^2(\mathbf{X})] + \frac{1}{2}\boldsymbol{\lambda}\mathbf{w}^T\mathbf{w} \tag{24}$$

The set $I_j = \{i|q(\mathbf{x}^{(i)}) = j\}$ is defined for the instance set of a leaf. Recall, the leaves w are a terminal point in the tree index $q(\mathbf{X})$ for a given base learner $F_m(\mathbf{X})$. As such, there will be multiple paths defined by $q(\mathbf{X})$ for data points $1, ...n$ that terminate at a given leaf. Therefore, leafs collect the indices of data points that conclude at their leaf. Each leaf is then designated as sets numbered $1, ...J$.

For example: the leaf j = 5 contains a set of indices s.t. $I_{j=5} = \{2, 5, 3, 8|q(\mathbf{x}^{(2)}) = 5, q(\mathbf{x}^{(5)}) = 5, q(\mathbf{x}^{(3)}) = 5, q(\mathbf{x}^{(8)}) = 5\}$

With the set of data point indices, the score w_j for a given leaf can be optimized. Gradients and Hessians are summed for the identified indices and used to compute an optimal weight for the leaf. Using the same example with leaf set $j = 5$:

$\sum_{i\in I_5} g_i = (g_2 + g_5 + g_3 + g_8) = \mathbf{G}_5$

$\sum_{i\in I_5} h_i = (h_2 + h_5 + h_3 + h_8) = \mathbf{H}_5$

Then $\sum_{i=1}^{n} g^{(i)}$ can be written in terms of j: $\sum_{j=1}^{T} g^{(j)}$ and $g^{(j)}$ can be written back into terms of i: $\sum_{i\in I_j} g^{(i)}$ giving us $\sum_{j=1}^{T}\sum_{i\in I_j} g^{(i)}$. This same conversion is done for $h^{(i)}$. Additionally, since the prediction of each base learner is a score, we represent a single tree $F_m(\mathbf{X}) = \sum_i^n F_m(\mathbf{x}^{(i)})$ as the sum of all its leaf node scores $\sum_{j=1}^{T} w_j$. Equation (4.16) can then be re-written:

$\mathcal{L}_m \approx \sum_{i=1}^{n} g^{(i)}F_m(\mathbf{x}^{(i)}) + \frac{1}{2}\sum_{i=1}^{n} h^{(i)}F_m^2(\mathbf{x}^{(i)}) + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}(w^{(j)})^2$

$\mathcal{L}_m \approx \sum_{j=1}^{T}(\sum_{i\in I_j} g^{(i)})w^{(j)} + \frac{1}{2}\sum_{j=1}^{T}(\sum_{i\in I_j} h^{(i)})(w^{(j)})^2 + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}(w^{(j)})^2$

The $\sum_{j=1}^{T}$ can then be factored out and the loss function optimized to find optimal weight for a given leaf set j.

$$w_*^{(j)} = -\frac{G_j}{H_j + \lambda} \tag{25}$$

$w_*^{(j)}$ is then substituted into equation (4.16) to calculate the score of the tree index q.

B.5.3 Split decisions

To expedite the scoring method, a greedy algorithm adds branches outward from a starting node. Splits are determined by a loss function that using scores before and after the split with I_L and I_R as instance sets of left and right leaves respectively s.t. $I = I_L \cup I_R$

$$\mathcal{L}^{(split)} = \frac{1}{2}\Big(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R^2 + \lambda} - \frac{G^2}{H + \lambda}\Big) - \gamma \tag{26}$$

APPENDIX C

Results: Supplementary

C.1 Bayesian Optimizer

A hyper-parameter is an adjustable parameter that parameterizes the model [8]. Hyper-parameters data types can be continuous, integer, or categorical [13]. If appropriately configured, the settings of the model hyper-parameters can have significant effects on the learning ability and therefore performance of a given model [8]. Therefore, the method used for selecting hyper-parameter settings is important.

Hyper-parameter search is the systematic adjustment of hyper-parameters to best maximize the model performance. There are numerous techniques in practice which typically approach the search process as a non-differentiable, single objective optimization over a mixed-type constrained domain [8]. Given the nuance and importance of correctly configuring hyper-parameters, configuration should be regarded as a necessary procedure, though a procedure that can be automated. Although rules-of-thumb are considered impractical for large amounts of hyper-parameters [8], such as is the case with XGB, they can be used to define the bounds for an automated approach.

Bayesian optimization has been shown to outperform other optimization algorithms [67], to include baseline algorithms such as grid and random search, and in some cases perform on a level comparable with or exceeding domain experts [15]. While Bayesian inference [28] involves the updating of probability distribution uncertainty given new data and requires integration, Bayesian optimization does not require derivatives and uses a Gaussian process.

The following information can generally be found in [19]. In this case, the prior is not a prior distribution $\mathcal{P}_\pi(x_*)$ reflecting existing knowledge x_*, but a prior over functions $\mathcal{P}_\pi(f_*|x_*)$ reflecting existing information. Since the number of possible functions approaches infinity, a finite interval of data $\mathcal{D}$ from $\mathbf{X}^{(train)}$ must be used. Two assumptions about the prior allow for the creation of the posterior:

1. The distribution of functions is gaussian and all f_* defined by the distributions are assumed to be jointly gaussian.

2. The covariance of the prior $\sum_{**}$ is a smoothing kernel $\kappa(x_*^{(i)}, x_*^{(j)})$ that ensures similar function behavior (for example: if $x_i^{(i)} \approx x_j^{(j)}$ then $f(x_i^{(i)}) \approx f(x_i^{(j)})$.

Therefore as new data is introduced, since the functions of new and old data are jointly gaussian, the covariance function for the prior can integrate the new data. Given that the covariance acts as a smoothing function, the addition of new data updates the shape of prior distribution while the μ is also updated. This updated distribution over functions is the posterior distribution (over functions) $\mathcal{P}(f_*|x_*, x, f)$.

Eggensperger et al. [15] reviewed standard hyper-parameter optimization and selection for a classification model involves defining a loss function, acquisition function, and a gaussian process. For model $\mathcal{P}$ with hyper-parameters $\lambda_1, ..., \lambda_n$ with domains $\Lambda_1, ...\Lambda_n$, the hyper-parameter space is defined $\boldsymbol{\Lambda} = \Lambda \times \cdots \times \Lambda$. $\mathcal{P}_{\boldsymbol{\lambda}}$ indicates that model $\mathcal{P}$ is using a particular hyper-parameter setting $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. Using k-fold cross validation as a surrogate function, hyper-parameter optimization is achieved by minimizing the function

$$f(\boldsymbol{\lambda}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(\mathcal{P}_{\boldsymbol{\lambda}}, CV_k, \mathbf{X}^{(train)}, \mathbf{X}^{(valid)}) \tag{27}$$

Bayesian optimization uses a prior distribution over functions to leverage existing knowledge. An acquisition function uses the subsequent posterior distribution of functions set at an initially random $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ to measure knowledge gain for each $\boldsymbol{\lambda}$. K-fold is a surrogate function that slows down Bayes considerably.

C.2 Comparison Models

The `LogisticRegression` (LR) [40] model was implemented with the `sklearn` library.

LR used an $\ell 1$ penalty (Lasso) to allow for additional feature selection given the dimensionality of the dataset. The hyper-parameter `C` is available for adjusting the strength of the regularizer. `class_weight` is also available to automatically balance the dataset through the adjustment of class weight penalties[36]. However, SMOTE was used, precluding the need for algorithm-level solutions for the class imbalance. The `C` parameter was optimized with `BayesSearchCV` [3] for AUC and trained with the reduced dataset.

Table 11: *Logistic Regression:* $\boldsymbol{\lambda}$

$\boldsymbol{\lambda}$	Interval	λ_i	Description
`C`	(0.001, 10)	1.706	Regularization strength
`penalty`	-	$\ell 1$	Frequency-based features

The `RandomForestClassifier` (RFC) [54] model was implemented with the `sklearn` library. Random forests are reviewed in Appendix B. As shown in Table 12, not all hyper-parameters were optimized. However, considerable preliminary analysis was performed with various hyper-parameters for all models (primarily XGB). Typically, intervals were based off the default setting and hyper-parameter documents. Small random samples were then used to observe model behavior and performance.

Table 12: *Random Forest:* $\boldsymbol{\lambda}$

$\boldsymbol{\lambda}$	Interval	λ_i	Description
`n_estimators`	(10, 250)	99	Number of Decision trees
`max_depth`	(1, 30)	14	DT depth
`min_samples_leaf`	(1, 10)	NA	min samples for split
`min_weight_fraction_leaf`	(0, 1e-8)	NA	Required weight for leaf node
`min_impurity_decrease`	(0, 1e-8)	NA	Required impurity decrease for split
`min_impurity_split`	(0, 1e-8)	NA	Impurity-based early stopping threshold
`class_weight`	-	NA	class balancing

RFC has multiple hyper-parameters for individual tree indices (structures) as well as

[36] Misclassifications of the minority class are penalized more

impurity measures for split points. RFC differs from XGB in that RFCs are a ensemble of Decision Tree Classifiers (rather than regression trees). While the XGBoost hyper-parameters focus on gradient boosting, RFC hyper-parameters focus more on the tree index (structure) and impurity measures.

The `XGB` is covered in detail in chapter 4. When setting XGB hyper-parameter intervals in Table 13 and optimizing for AUC, the optimizer (and/or XGB model) was quick to penalize for overly generous intervals, predicting all of either class or severally over-predicting one class. Additionally, using hyper-parameter adjustments for dataset imbalance caused frequent training set over-fitting. Both learning rate adjustment (much lower) and manipulation of the number/depth of trees were used for mitigation. Extensive preliminary testing with XGB hyper-parameters were performed and hyper-parameters are further discussed in recommendations.

Table 13: *XGB:* $\boldsymbol{\lambda}$

$\boldsymbol{\lambda}$	Interval	λ_i	Description
`learning_rate`	(1e-09, 1e-01)	0.0512	Shrinkage
`min_child_weight`	(1, 5)	NA	Controls overfitting
`max_depth`	(0, 30)	7	Regression tree depth
`max_delta_step`	(0, 5)	NA	Imbalance
`subsample`	0.5, 1.0)	NA	Prevent overfitting; subsample of samples
`colsample_bytree`	(0.5, 1.0)	NA	Subsample of features for trees
`colsample_bylevel`	(125, 200	NA	Subsample of features for split
`reg_lambda`	(0.5, 5)	NA	$\ell 2$ penalty
`reg_alpha`	(1e-9, 1e-1)	NA	$\ell 1$ penalty
`gamma`	(1e-9, 1e-1)	NA	Loss reduction required for additional splits
`n_estimators`	(50, 250)	175	Number of Trees
`scale_pos_weight`	(1, 100)	NA	Imbalance
`base_score`	(0.4, 0.6)	NA	Initial prediction of all instances; global bias

C.3 Embedded-selected (Final) Set

The following is a summary of selection and weights using `XGBoost.importances` output [76] and `xgbfir` [75] on the validation set.

Using the validation set: From the SA method feature set, the SFM transformer selected trend features `lag_1`, `lag_diff_ind` and `prev_non`. From the remaining engineered features, the model used 7 of 9 frequency-based features, 1 of 4 time derivative features, 2 of 5 mapped features, and 1 of the 2 added indicator features. All of the base features were used.

From the B1 method feature set, the SFM transformer selected trend features `lag_diff_abs`, `lag_diff` and `prev_non`. From the remaining engineered features, the model used 4 of 10 frequency-based features, 2 of 4 time derivative features, 1 of 3 mapped features, and 1 of the 2 added indicator features. All of the base features were used.

From the full candidate feature set, the SFM transformer selected trend feature `prev_non`. From the remaining engineered features, the model used 9 of 14 frequency-based features, 4 of 6 time derivative features, 3 of 5 mapped features, and 1 of the 3 added indicator features. All of the base features were used.

From the full candidate feature set, the RFE transformer selected trend feature `prev_non`. From the remaining engineered features, the model used 8 of 14 frequency-based features, 4 of 6 time derivative features, 3 of 5 mapped features, and 1 of the 3 added indicator features. All of the base features were used.

Using the test set: The following is a summary of selection and weights using `XGBoost.importances` output and `xgbfir` on the test set.

From the SA method feature set, the SFM transformer selected trend features `lag_diff_ind` and `prev_non`. From the remaining engineered features, the model used 7 of 9 frequency-based features, 1 of 4 time derivative features, 2 of 5 mapped features, and 1 of the 2 added indicator features. All of the base features were used.

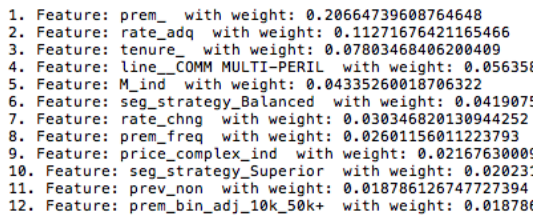
From the B1 method feature set, the SFM transformer selected trend features `lag_diff_ind` and `prev_non`. From the remaining engineered features, the model used 6 of 10 frequency-

based features, 1 of 4 time derivative features, 0 of 3 mapped features, and 1 of the 2 added indicator features. All of the base features were used.

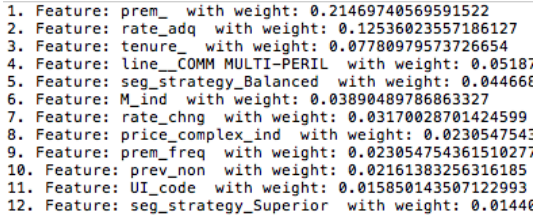
From the Full feature set, the SFM transformer selected trend features `prev_non` and `prev_nb`. From the remaining engineered features, the model used 7 of 14 frequency-based features, 5 of 6 time derivative features, 2 of 5 mapped features, and 0 of the 3 added indicator features. All of the base features were used.

The RFE transformer selected features remained the same.

C.4 Feature Stability Comparison



(a) *Q1*



(b) *Q2*

Figure 23: *Feature Stability*

C.5 Feature Analysis

Key differences in SFS method feature selection included `risk_complex_ind`, `con_type`, and `UI_code`. The mentioned features all appeared in the embedded method selected (final) set and were examples of the SFS methods differing in their selection of certain features. The features demonstrated predictive properties and reasons for rejection were likely related to feature interactions and/or correlation.

Construction type: `con_type` demonstrated an instance where two highly correlated features may have caused the SA model to reject a candidate feature that was selected by the B1 method. `con_type` is a comparatively low-cardinality ($|\mathbf{x}_p| = 7$) categorical original feature that was rejected by the SA method. The addition of `con_type` to the feature sets lessened the Fscore weight of `line_cmp` in Full feature set (and B1) compared

to the SA set (see Figure 24(a) & 24(b)). This may be a result of con_type being specific to line_cmp and therefore highly correlated. Furthermore, the high correlation between con_type and line_cmp may be the reason the B1 method accepted con_type and the SA method rejected it.

Furthermore, if we observe raw Gain scores in Figure 24(c) & 24(d), the presence of con_type increases the Gain for line_cmp. This indicates a strong interaction between the two features. Observing Interaction Gain scores (Figure 25) confirms this interaction.

The SA methods selection log (Figure 26) confirms that line_cmp was added to the base set prior to con_type being assessed. While this may further evince the weakness of the SA method, the performance metrics in Table 8 still indicate the model, using SA selected features, was able to perform on par with the Full model with significantly less features. In the case of con_type, the model easily compensated for the feature using other features, specifically an increased reliance on line_cmp. con_type weight of ≈ 0.10 was adjusted for by an increase use of tenure_ and line_cmp (see Figure 29).

UIC: UI_code was an example of the SA model selecting a candidate feature early on in the selection process that was rejected by the B1 method. UI_code was selected and used by the SA model and appeared in embedded-selected (final) sets for SA, Full, and RFE. However, it was a non-select with B1. If feature UI_code was considered to have predictive properties based off SA method selection and its use in the mentioned final sets, then this indicated a weakness of the B1 method, specifically in the selection of the base features. While there was low correlation between UI_code and the other numerical base features, UI_code could have been correlated with the base categorical features, industry_ and/or seg_strategy. In viewing interaction Gain for the other models (when UI_code was selected), there was a noticeable absence of strong interaction Gain for UI_code and the base categorical features(Figure 27 & 28). Therefore, there may have been an instance of a spurious correlation [72] rather than a weakness in the B1 method.

In which case, UI_code was correlated strongly with either industry and seg_strategy, but not strongly correlated with the response (low predictive value). When industry_ and seg_strategy are not present in the feature set, UI_code appears strongly correlated with the response, as was the case with SA. However, when industry_ and seg_strategy are included, the weak correlation of UI_code is revealed. The absence of interaction in Figure 28 could be caused by the tree-based classifier selecting the more predictive feature out of a group of correlated features.

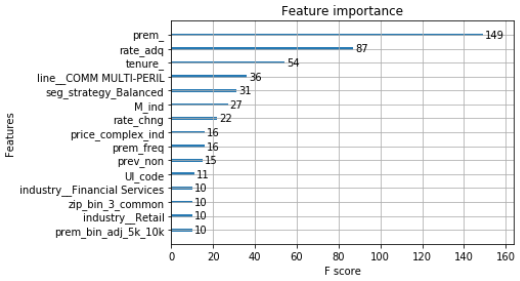
However, the above explanation failed to account for UI_code being selected by embedded methods. It is difficult to speculate as to the precise reason the feature combination performed differently and it is not within the scope of this analysis. However, the fact that the presence of different features in the feature set dictated whether or not the candidate feature UI_code was able to contribute to model performance, was significant in assessing the effectiveness of the SFS methods and contribution of engineered features.

Complexity Indicators: The complexity indicators were pre-engineered[37] features that were designed to capture important policy information related to account complexity. According to domain experts, these indicator features should contribute predictively. Additionally, price_complex was ranked $\in P_{80}$ of feature weights by models using the SA, Full, and RFE sets. However, both complexity indicators were rejected by the B1 method. The same reasoning applied to the omission of UI_code possibly could explain the differences selection between the methods for the complexity indicators.

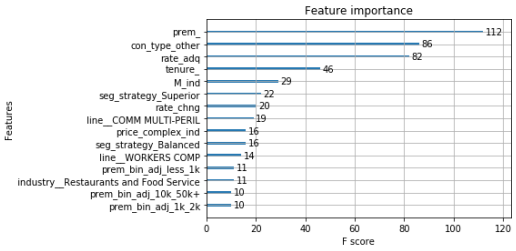
There were numerous other differences between the SFS method selection results. In some instances, the presence of certain base features in the SA method may have caused subsequent candidate feature selection/rejection as a result of an interaction/correlation. This was likely the case with the B1 rejection of PC_cat, which was selected by the SA

[37]Company internally engineered

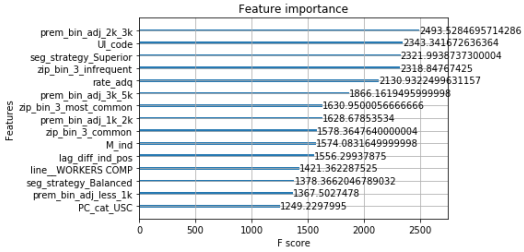
method near the end of the selection process. At which time, the SA base feature set contained all the B1 base features plus over 30 other features. However, often times, the differences occurred early on when the B1 base feature set contained both `industry_` and `seg_strategy` and the SA base set did not. This may be evidence of spurious correlations [72] and the SA model would need to be rerun with a different order to confirm.



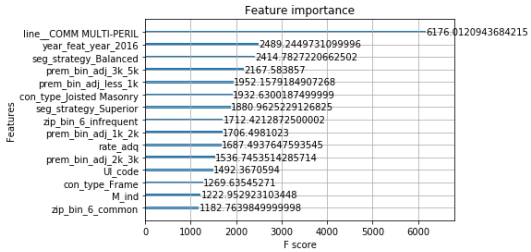
(a) *SA: Fscore*



(b) *Full: Fscore*



(c) *SA: Gain*



(d) *Full: Gain*

Figure 24: *Score Comparison*

Interaction	Gain
rate_adq|rate_adq	228458.9133
con_type_other|line__COMM MULTI-PERIL	190902.8259
con_type_other|prem_	128858.9202
industry__Contractors|line__COMM MULTI-PERIL	95111.41017

Figure 25: *Full Interaction: Gain*

```
no update. Feature sc_bin_3 dropped
line_: auc and f1 updated:
0.7383272110468057, 0.3868973648173754
industry_: auc and f1 updated:
0.7499840211833455, 0.4000826879446927
seg_strategy: auc updated: 0.7502256349384727
year_feat: auc updated: 0.7502256349384727
quarter_feat: auc updated: 0.7502256349384727
PC_cat: f1 updated: 0.40135378063584304
prem_bin_adj: auc and f1 updated:
0.7526964692082936, 0.40394184588012294
agnt_bin_6: auc updated: 0.752857662105973
term: auc updated: 0.752857662105973
abs_Q: auc updated: 0.752857662105973
no update. Feature month_nom dropped
risk_state: auc and f1 updated:
0.7544846919424621, 0.40715729171710846
no update. Feature con_type dropped
prgm : auc updated: 0.7567123321346295
```

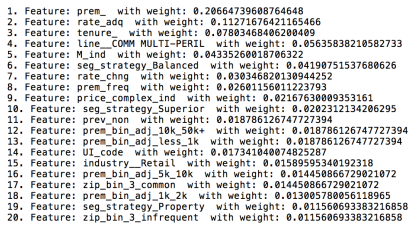
Figure 26: *SA Log*

Interaction	Gain
rate_adq|rate_adq	333691.2583
line__COMM MULTI-PERIL|prem_	84189.56935
prem_|tenure_	53847.25352
tenure_|tenure_	49424.35208
seg_strategy_Balanced|seg_strategy_Balanced	40449.85416
prem_|seg_strategy_Balanced	39566.6952
M_ind|M_ind	35725.8111
M_ind|prem_	25952.98616
UI_code|UI_code	25289.49692
prem_|zip_bin_3_common	22291.79515
prev_non|prev_non	21881.79905
prem_|prem_bin_adj_less_1k	20718.02364
UI_code|prem_	20705.24142
prem_|seg_strategy_Superior	20420.04407
price_complex_ind|rate_chng	19816.04895
zip_bin_3_infrequent|zip_bin_3_infrequent	18550.78139
prem_bin_adj_2k_3k|prem_bin_adj_2k_3k	17175.0785
industry__Tech Services|rate_adq	17118.75641
seg_strategy_Superior|seg_strategy_Superior	17086.1734

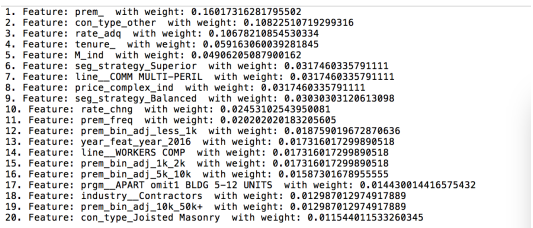
Figure 27: *SA: UIC*

Interaction	Gain
rate_adq|rate_adq	228458.9133
con_type_other|line__COMM MULTI-PERIL	190902.8259
con_type_other|prem_	128858.9202
industry__Contractors|line__COMM MULTI-PERIL	95111.41017
M_ind|line__COMM MULTI-PERIL	61717.20293
prem_|year_feat_year_2016	54155.75585
prem_|seg_strategy_Balanced	51238.18281
seg_strategy_Superior|seg_strategy_Superior	41334.46878
seg_strategy_Balanced|seg_strategy_Balanced	33543.0697
UW_title_Barnett|prem_	30480.34153
prem_|seg_strategy_Superior	28295.71504
M_ind|M_ind	27045.64742
price_complex_ind|rate_chng	26561.75085
prem_|tenure_	26246.99479
con_type_Joisted Masonry|con_type_other	26093.46835
con_type_other|rate_adq	22830.62976
con_type_other|seg_strategy_Balanced	21247.12549
prem_bin_adj_less_1k|prem_bin_adj_less_1k	20472.99321
line__WORKERS COMP|price_complex_ind	19003.40633

Figure 28: *Full: UIC*



(a) *SA:* con_type



(b) *Full:* con_type

Figure 29: con_type *Feature*