IT UNIVERSITY OF COPENHAGEN

Third Semester Project

Modeling Automobile Insurance Claims Risk Using Machine Learning

December 21, 2025

Contributors

Anis Kadem — `anika@itu.dk`
Luca Costa Martins — `lucm@itu.dk`
Etele Kovács — `etko@itu.dk`

GitHub Repository:
`https://github.com/mindenki/2025-ITU-MachineLearning-Project-Claims-Risk-zrt`

Bachelor Degree
Data Science

Contents

Introduction

Modeling insurance claims risk is difficult, as many problems may arise while developing our models. In automobile insurance, the target data is usually highly skewed and dominated by zeros, which makes predicting extremely challenging. Developing an effective model could help insurance companies to assess risk more accurately, price policies more fairly and reduce losses. It could also help in identifying potential insurance fraudsters.

In this project we investigate this problem by predicting the expected number of insurance claims made by customers over a year based on vehicle, driver, and geographic information. Based on this, we define target variables and evaluate several machine learning models. We explored 3 classes of models. First, we developed our own decision tree regressor and feed-forward neural network regressor from scratch. Second, we also ran a decision tree regressor and a feed-forward neural network regressor using Python libraries. Third, we also experimented with three other models: negative binomial model with and without offset, which are standard in insurance claim prediction [1, 2], and random forest classifier, with binary targets (claim/no claim).

The project further included other tasks, such as data cleaning, data exploration, feature engineering and data visualization using principal component analysis (PCA) and clustering.

Dataset, Target and Evaluation Metric

2.1 Dataset Description

The dataset used in the project is a publicly available dataset, containing around 680.000 policies and their associated third-party liability claims observed over the specified amount of time. The dataset given to us is already split into two subsets: a training set, containing around 80% of the data, and a test set, with around 20% of the data.

The entries are represented with 12 columns, each containing information about the policies. These include numerical features, such as `VehAge`, `BonusMalus`, `Density` and categorical features like `Region`, and `VehBrand`. One column contains `Exposure`, which is the length of the policy and one contains `ClaimNb`, which is the number of observed claims in the exposure period.

2.2 Target Variable

Defining our target variable is an essential task which influences the performance of our models. We considered different choices with different benefits.

1. $y_{\texttt{ClaimRate}} = \frac{\texttt{ClaimNb}}{\texttt{Exposure}}$, using `ClaimRate` accounts for the different periods of time observed across policies. This way, we can more easily compare entries with different `Exposure` values. This is also the standard in insurance analytics.

2. $y_{\texttt{log\_ClaimRate}} = \log(1 + \frac{\texttt{ClaimNb}}{\min(\texttt{Exposure}, 0.01)})$, this accounts for the big skew observed in `ClaimNb` and `ClaimRate`, while also capping some extreme values that would distort `ClaimRate`. We use 0.01 as a floor, in order to cap the minimum 1% values.

3. $Y_{binary} = \begin{cases} 1, & \text{if } \texttt{ClaimNb} > 0, \\ 0, & \text{otherwise.} \end{cases}$, we used this target when running a classifier model.

We tried both $y_{\texttt{ClaimRate}}$ and $y_{\texttt{log\_ClaimRate}}$ and observed almost identical results, so using either one is fine.

2.3 Evaluation Metric

To evaluate model performance, we choose to use Root Mean Squared Error (RMSE). RMSE is preferred over Mean Absolute Error (MAE), when we want to penalize large deviations between predicted and observed claims risk more heavily. We do not want to underestimate high risk-policies, so RMSE seems like a great choice for our models.

We also considered Poisson deviance as an evaluation metric. It is often used in insurance related studies, because it accounts for zero-inflation and heavy-tailed claims count. It measures how well a Poisson regression model fits our data, so the lack-of-fit between our predicted and observed rates. This is appropriate for sparse data like ours. We still chose to use RMSE, as it is more straightforward, easier to interpret and simpler to use.

Data Cleaning and Exploratory Data Analysis

3.1 Data Cleaning

Firstly, we systemically cleaned the data to ensure consistency and correctness across all entries. We removed the `IDpol` column, as it is just a unique identifier which we do not need. We identified the numerical and categorical features, and looked for any missing or null values. There were no missing values. We also identified any values that go outside of their expected ranges (e.g., negative ages), but we only found some values exceeding 1 in `Exposure`, which we corrected by capping `Exposure` at 1.

3.2 Exploratory Data Analysis

EDA is one of the most important parts of machine learning projects, as we can gain insight into our data, identify characteristics of our data, spot outliers and skewed data, draw conclusions about future steps, etc. EDA and preprocessing choices were mainly guided by standard practices.[3] We split our Exploratory Data Analysis into three bigger sections: Univariate Analysis, Bivariate Analysis and Multivariate Analysis. In each section we handled numerical and categorical features differently.

3.2.1 Univariate Analysis

In Univariate Analysis, we examine each feature individually to understand its distribution, detect any outliers or any skewness. As Figure 3.1 shows, one of the essential parts of our data, `ClaimNb`'s distribution is highly zero-inflated with a skewness score of 5.14. This is expected from an insurance dataset. Majority of policies report zero claims, with only a small fraction of the policies (4.7%) report one or more claims.

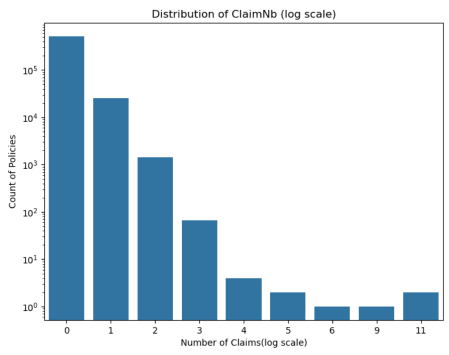
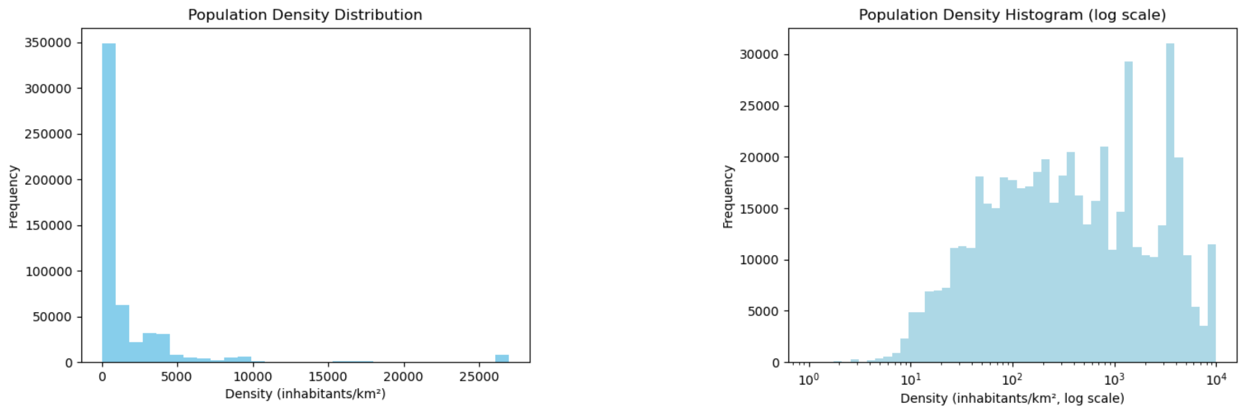


Figure 3.1: Distribution of the target variable `ClaimNb`. The distribution is highly zero-inflated, with the majority of policies reporting no claims and a small fraction reporting one or more claims. The y-axis is shown on a logarithmic scale to improve visibility of non-zero claim counts.

Most numerical variables, such as `VehAge` and `BonusMalus` showed strong right skewness, but we also found more balanced features, like `DrivAge`; and `Exposure`, which exhibits a bimodal pattern around 0 and 1. The skewness could be fixed in some cases, for example log transforming `Density` (Figure 3.2) makes it into roughly symmetric. Skewness also suggests binning in cases.

For categorical variables, we can observe nominal features, such as `VehBrand` and `Region`, and ordinal features, such as `VehPower`. There are more balanced categorical features (`VehGas`) and some which are imbalanced (`Region`).



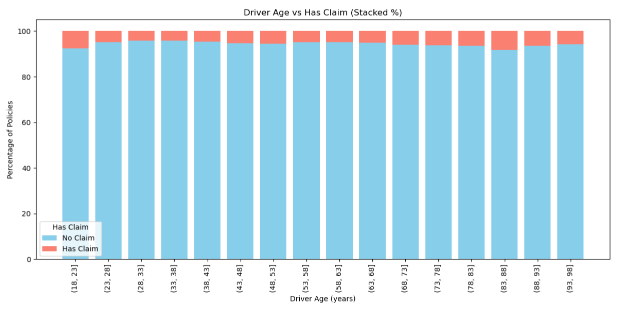
(a) Original distribution of `Density` (b) Log-transformed `Density`

Figure 3.2: Comparison of the `Density` feature before and after log transformation. The log transformation reduces right-skewness and creates a more symmetric distribution, which can improve model performance and interpretability.

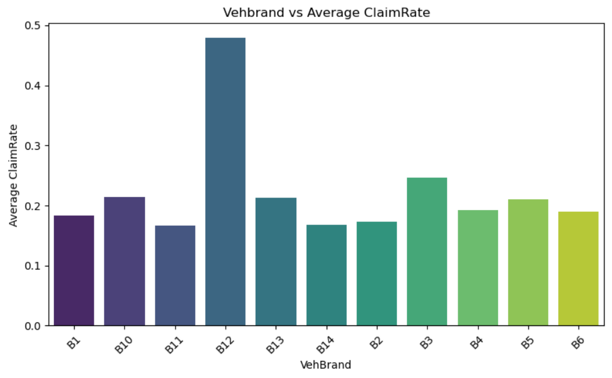
3.2.2 Bivariate Analysis

In this section, we investigated relationships between pairs of variables. Firstly, we compared `ClaimNb` to numerical features and categorical features, after features to features, as numerical to numerical, categorical to categorical and numerical to categorical.

Numerical features showed weak correlation with `ClaimNb`, and numerical features alone hardly tell us anything about `ClaimNb` (Figure 3.3a). Categories show some variation in claim proportions, but these differences are also minor. For example, `DrivAge` reveals slightly higher claim rates among the youngest and oldest drivers, while average claim rates across `VehBrand` mostly lie between 0.1 and 0.5, with `B12` being slightly higher at 0.47.



(a) Stacked bar plot of DrivAge vs claim percentage. Younger and older drivers show slightly higher claim rates, but overall differences are minor.



(b) Average claim rate per VehBrand. Most brands fall between 0.1 and 0.3. B12 shows slightly higher risk at 0.47.

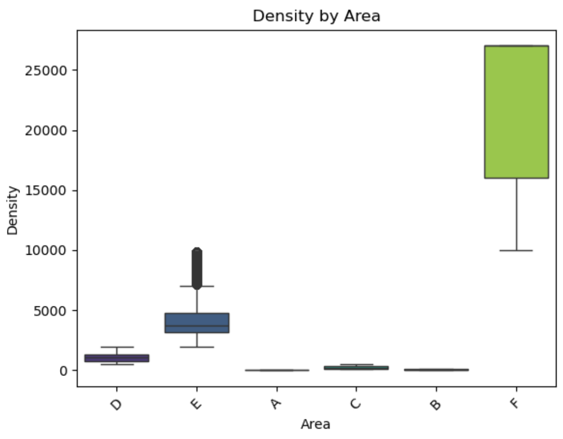
Figure 3.3: Bivariate analysis showing categorical/numerical relationships with claims. Differences across driver age and vehicle brand are relatively small, highlighting the challenge of predicting claims risk.

Examining numerical feature correlations using a heatmap shows us a moderate negative correlation between BonusMalus and DrivAge. Other than these, we found no strong multicollinearity overall.

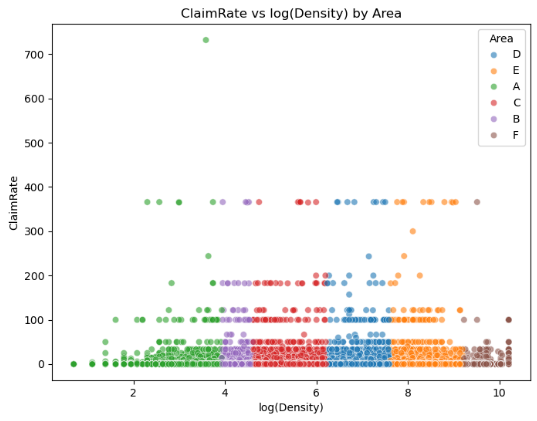
Categorical features have some moderate associations among themselves, but nothing serious.

Looking at the relationships between categorical and numerical features, we can see that most numerical features differ significantly across categories, as confirmed by ANOVA tests. From Figure 3.4a, Density looks as it is affected by Area, which we investigated later in Multivariate Analysis.

Overall, patterns exist between features and target, but they are weak, which means feature engineering is especially important in this project.



(a) Boxplot of Density across different Area categories. The distinction in population density between areas is evident.



(b) Claim rate vs log(Density), grouped by Area. The separation between areas is even clearer, highlighting that Area is strongly related to Density.

Figure 3.4: Analysis of the relationship between Area and Density. Area is not fully independent and is largely derived from population density, suggesting one of the features can be dropped to reduce redundancy and dimensionality.

3.2.3 Multivariate Analysis

In multivariate analysis, we explore interactions between multiple features and the target variable in order to find some combinations of features that could be important or combined. For example, we considered `Area` and `Density` with `ClaimRate` based on our previous deduction, and discovered that `Area` is not an independent feature, and it is derived from the log of `Density`, see Figure 3.4b. As not to introduce redundancy and reduce the number of dimensions, one of the features should be dropped.

Feature Engineering and Selection

Feature engineering is essential to make the features informative and easier to interpret by our models. We might also choose to drop some redundant or not helpful features.

For numerical features, we chose different ways to achieve better performance. As `VehAge` is very right skewed, we chose to make 8 bins out of it (0-1, 2-3, 4-5, 6-10, 11-15, 16-20, 21-30, 30+), so data is better distributed. After that, we applied ordinal encoding to it. We used logarithm transformation on `Density` to lose its skewed distribution. Even though, `VehPower` also is given in numerical values, these are actually categories. We also bin `VehPower` (4 ,5 ,6 ,7 , 8-9,10-11 ,12+), as higher power cars are rare, and then apply ordinal encoding. `DrivAge` and `BonusMalus` were kept as is. We also applied standard scaling, so features have a mean of 0 and a standard deviation of 1.

For categorical features, `VehBrand` and `VehGas` were encoded using one-hot encoding, and as `Region` had a lot of categories with very low counts, we decided to group these rare categories in order to reduce dimensionality, we then applied one-hot encoding here as well. As discussed earlier, `Area` contains redundant data with `Density`, so we decided to drop `Area`.

This part was guided mainly by our exploratory data analysis and domain knowledge. Overall, we wanted to balance interpretability with complexity, that is why we did not reduce the features more aggresively. This way, it was also easier to compare different models.

Visualization via Dimensionality Reduction and Clustering

5.1 PCA Visualization

5.1.1 Raw data

Principal Component Analysis was applied to the raw feature space to analyze the variance structure of the data. Only standard scaling was applied to numerical features, with no additional processing. PCA proved highly effective under this minimal setup, with approximately 90% of the total variance explained by the first 10 principal components. Visualization in the first two components revealed a pronounced striped structure, indicating that variance is largely dominated by one-hot encoded categorical features.

5.1.2 Processed data

PCA was also applied to the processed feature space to assess the effect of processing on the variance structure. Compared to the raw case, variance was more evenly distributed across components, suggesting a reduced dominance of categorical variables. Retaining 90% of the variance required 9 components instead of 10, indicating a modest improvement in feature compactness after processing.

5.2 Clustering

5.2.1 Raw feature space

Clustering was performed on the raw feature space using KMeans. A subsample of the dataset was used to perform a grid search over the number of clusters, guided by inertia and silhouette scores. Applying the selected number of clusters in both the full PCA space and the two-dimensional PCA projection resulted in very similar cluster structures, indicating that clustering outcomes are primarily driven by the dominant principal components.

5.2.2 Processed feature space

The same clustering procedure was applied to the processed feature space. In contrast to the raw case, clustering in the full PCA space (9 components) and in the two-dimensional projection produced visibly different cluster assignments. This indicates that, after processing, information relevant for distance-based clustering is distributed across multiple components rather than being concentrated in the first two.

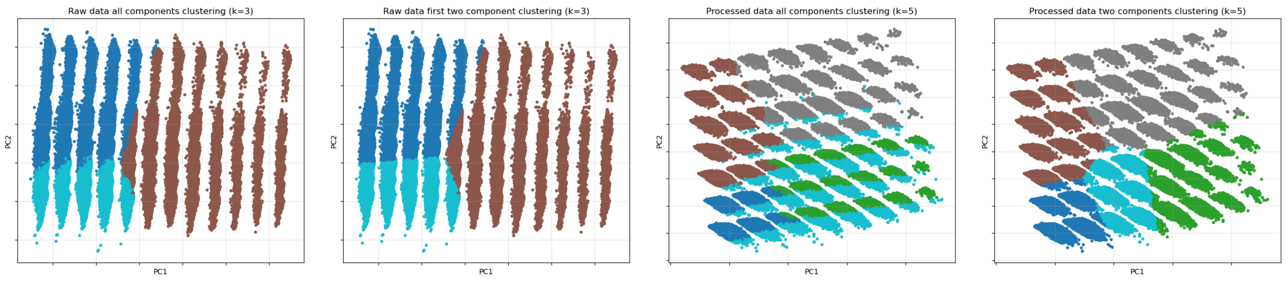


Figure 5.1: PCA visualisations and KMeans clustering results for raw and processed feature spaces.

Methods and Implementations

6.1 Common Modeling Setup

All models use the processed feature representation described earlier in the report, and unless stated no dimension reduction was applied. The input data and the split stayed the same. Targets used by the specific models are stated later. Random seed stayed the same where applicable. The same 5-fold cross-validation was applied to all models. Hyperparameters were selected using a random grid search over predefined parameter ranges. Large language models (ChatGPT) were used to assist with debugging and conceptual clarification during development, but no model code or report text was directly generated [4].

6.2 M1: Decision Tree Regressor

6.2.1 Method application and data used

A decision tree was used to predict continuous numeric values based on thresholds.

Two implementations of the Decision Tree were developed: one from scratch using numerical libraries [5] and one using the `scikit-learn` library.

The scikit-learn implementation was given more hyperparameters. For splitting the data, the `scikit-learn` implementation used Mean Squared Error to make the decision. The scratch implementation used Weighted Gini Impurity. No explicit early stopping criterion was applied.

6.2.2 Model architecture and hyperparameter selection

Both implementations were tested with various hyperparameters selected using a random grid search over predefined parameters. The `scikit-learn` implementation was given more hyperparameters than its scratch counterpart, possibly granting it greater flexibility and variability.

6.2.3 Comparison of implementations

Both implementations showed no sign of having a better predictive performance over the other version. Under identical data, no systematic differences were observed between the two models, indicating that the scratch implementation behaves consistently with the reference implementation.

6.3 M2: Feed-Forward Neural Network

6.3.1 Method application and data used

A feed-forward neural network was used to model the logarithm of the claim rate.

Two implementations of the MLP were developed: one from scratch [6] and one using the PyTorch framework.

Training was performed using mini-batch gradient descent. For the scratch implementation, the loss function was selected via grid search among MSE, MAE, Huber loss, and Log-Cosh loss. For the PyTorch implementation, MSE was used. There were only very subtle differences between the different losses we used in terms of results.

As an experiment we moved the torch model to GPU and used PCA-reduced data. This reduced the average training time per grid-search iteration from approximately 50 minutes to around 20 minutes. We did not use the results from this experiment because all the other models were trained without PCA applied to the dataset. The experiment showed us significant differences in time and little to no differences in results.

6.3.2 Model architecture and hyperparameter selection

The MLP architecture was selected via a grid search over combinations of multiple hidden-layer configurations and node numbers. ReLU activation functions were used in all hidden layers. In the PyTorch version, a Softplus activation was added in the output layer, while the scratch stayed linear.

Several optimizers were evaluated in the scratch implementation, including SGD, SGD with momentum, Adam, and Adagrad, to see if that could improve the performance, but as it turned out optimizers are not the bottleneck. The PyTorch implementation used the Adam optimizer. Learning rates were selected via grid search for the PyTorch model from $\{10^{-3}, 10^{-4}\}$, while a fixed learning rate of 10^{-3} was used for the scratch implementation. No explicit regularization techniques were applied.

6.3.3 Comparison of implementations

The scratch and PyTorch implementations showed very similar trends in terms of training dynamics and predictive performance. Under identical data, there were no systematic differences, proving that the scratch implementation behaves consistently with the reference implementation.

6.4 M3: Additional Models

6.4.1 Method application and data used

In addition to the mandatory models, a Negative Binomial regression model and a Random Forest Classifier were used as alternative approaches for the project. The Negative Binomial model was used for regression on simple claim frequency while the Random Forest was used for binary classification.

For regression, the target variable was the claim rate and claim number with offset. Both offset-based and non-offset-based formulations were considered for the Negative Binomial model. For classification, the target variable was binary, claim or no claim.

6.4.2 Negative Binomial regression

The Negative Binomial model was included due to the presence of overdispersion in the data [1, 2]. The model was implemented using the `statsmodels` library and fitted by maximizing the Negative Binomial log-likelihood, the library default.

Two variants of the model were evaluated: one using exposure through an offset term, and one without an offset, where exposure was part of the target variable. This allowed us to see if modeling exposure enhanced predictive performance.

Model performance was evaluated using both Root Mean Squared Error and Negative Binomial deviance. RMSE was included to maintain comparability with the mandatory regression models, while Negative Binomial deviance provided a model-native measure for overdispersed count data.

6.4.3 Random Forest Classifier

A Random Forest Classifier was used to model the occurrence of claims as a binary outcome. The model was implemented using `scikit-learn`.

The classification model provides a complementary perspective by focusing on the probability of a claim occurring, rather than predicting claim frequency, and is therefore not directly comparable to the regression models.

Results and Evaluation

7.1 Regression Models

Table 7.1 summarizes the performance of the regression models on the test set. We report RMSE, MAE, and Poisson deviance where applicable.

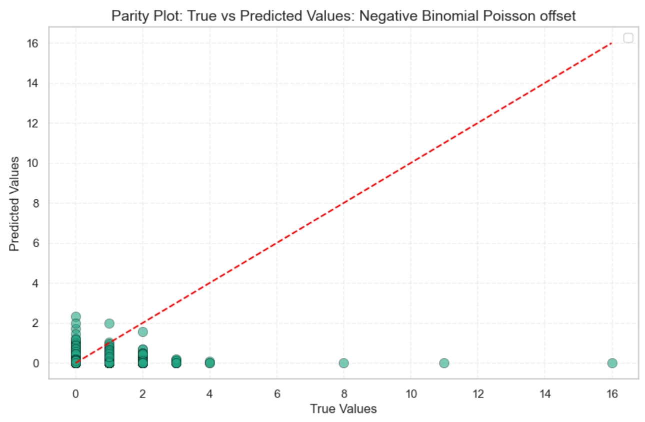
Table 7.1: Regression model performance on test set

Model	RMSE	MAE	Negative Binomial Deviance
DT scratch	3.043	0.241	–
DT sklearn	3.035	0.297	–
MLP pytorch	3.140	1.199	–
NB-non-offset	2.904	–	0.937
NB-offset[*]	0.237	–	0.279

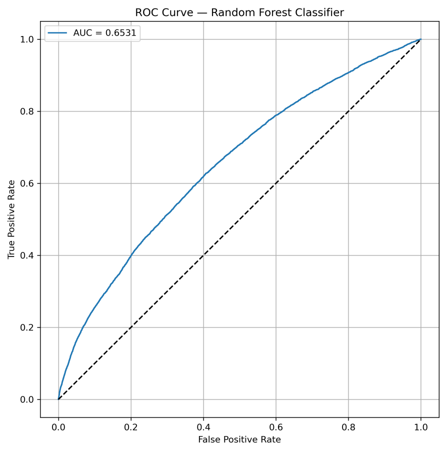
[*] NB-offset is evaluated on claim *counts* rather than claim *rates*, which explains the substantially lower error magnitude.

Figure 7.1a shows the actual vs predicted claim rates for the NB non-offset model, which achieved the best overall performance. The model predicts near-zero values for every policy, while underestimating the rare extreme claims, consistent with the skewed and zero-inflated nature of the data. We have to take into consideration that the offset model did beat it on the same metric. But we cannot compare the RMSE of the offset model with the others, because it models claim number with an offset (exposure), not claim rate.

Note on RMSE interpretation: All RMSE values reported for regression models are computed on the original claim rate scale. Models trained on the log-transformed claim rate had their predictions back-transformed (exponentiated and adjusted) before RMSE computation. This ensures comparability across models, except for the NB-offset model, which predicts claim counts rather than rates. The high RMSE values for the DT and MLP models (around 3) may seem surprisignly large given that most claim rates are below 1. This is primarily due to the extreme skew and zero-inflation in the dataset: a small number of policies have claim rates in the hundreds, while the majority have rates in the tens or below. Consequently, even models that predict near-zero for most policies accumulate large squared errors from the rare extreme claim rates, inflating the RMSE. This highlights that RMSE alone can overemphasize rare outliers in highly skewed datasets.



(a) Predicted vs actual claim numbers for the NB non-offset model. All model's parity plot look similar, usually with less non-zero predictions.



(b) ROC curve for the Random Forest classifier predicting binary claims.

Figure 7.1: Performance of our models.

Commentary: All models performed similarly poorly in terms of RMSE. This suggests limited to non-predictive power, as the zero-inflation was too much for our models. MAE provides more interpretability for overall prediction error.

7.2 Binary Classifier

The Random Forest classifier was trained to predict whether a policy had at least one claim. Table 7.2 shows its test performance in terms of AUC, accuracy, precision, and F1 score.

Table 7.2: Random Forest Classifier performance on test set

Model	AUC	Accuracy	Precision	F1 Score
Random Forest	0.653	0.762	0.093	0.166

Figure 7.1b shows the ROC curve for the classifier. Despite a reasonable AUC, precision and F1 are low due to extreme class imbalance (most policies have zero claims).

Commentary: The classifier demonstrates moderate discriminative ability (AUC 0.65). Low precision indicates many false positives, which is expected due to the highly zero-inflated target. Using weighted loss and cost-sensitive approaches could not help improve the model's performance, which suggested problems with the features.

Discussion

8.1 Overview

In this project, we implemented and evaluated multiple models so we could predict automobile insurance claims. We mostly used regressions to predict claim rates, claim numbers and one random forest classifier with a binary claim/no-claim target. Overall, the results were poor, as all models struggled with the dataset's extreme distribution, which is typical for insurance claims. All models learned to predict values close to zero, which lead to similar performances.

Suprisingly, amongst the regression approaches, simpler models like decision trees slightly outperformed more complex models like neural networks. As the underlying signal was weak for every model, it seems that increasing the model complexity does not help our case, it worsens it. The negative binomial model, while not improving the RMSE loss, better represents the problem at hand, by accounting for zero-inflation and overdispersion by nature. This should be improved, with better feature engineering, to achieve better performance.

The binary classification experiment was included as a complementary analysis rather than a primary modeling approach. While it showed some ability to rank policies by claim risk, performance remained constrained by severe class imbalance. These results reinforce the conclusions drawn from the regression models: the main challenge lies in the data distribution, not the choice of algorithm.

Conclusion: The findings show that predicting automobile insurance claims is an inherently difficult task, and it is fundamentally constrained by its unique data characteristics. The results suggest that model complexity alone cannot overcome the nature of our data and domain specific models such as negative binomial GLMs might be better aligned with the sparse nature. As our models could not learn any huge patterns from the features, we also deduce that feature informativeness is also a larger bottleneck than algorithm selection. Future improvements would include richer features, longer observation timeframes and having severity as a seperate aspect.

[1] M. David and D. Jemna, "Modeling the frequency of auto insurance claims by means of poisson and negative binomial models," *Scientific Annals of Economics and Business*, vol. 62, no. 2, pp. 151–168, 2015. [Online]. Available: https://EconPapers.repec.org/RePEc:vrs:aicuec:v:62:y:2015:i:2:p:151-168:n:2

[2] P. Faroughi, S. Li, and J. Ren, "The applications of generalized poisson regression models to insurance claim data," *Risks*, vol. 11, no. 12, 2023. [Online]. Available: https://www.mdpi.com/2227-9091/11/12/213

[3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.

[4] OpenAI, "Chatgpt (gpt-4) large language model," https://chat.openai.com, 2024, used for debugging assistance and conceptual clarification.

[5] E. Ozeren, "Building a decision tree from scratch," https://medium.com/@enozeren/building-a-decision-tree-from-scratch-324b9a5ed836, 2020, medium blog post, used as help for building the Decision Tree from Scratch.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.