Project 4: Insurance Claim Prediction

(First discussion: Nov 27; Last questions: Dec 11; Deadline: Dec 18)

Responsible: Gabriele Visentin

The goal of this project is to implement and compare different models for insurance frequency claim prediction on real-life data from the French motor third party liability dataset (see file `freMTPL2freq.csv`), which comprises $m = 678{,}007$ car insurance policies.
Below is an overview of the data:

Feature name	**Feature description**	**Feature type**[1] **and range**
`VehPower`	Car power	Discrete, values in $\{4, 5, \dots, 15\}$
`VehAge`	Car age in years	Discrete, values in $\{0, 1, \dots, 100\}$
`DrivAge`	Driver's age in years	Discrete, values in $\{18, 19, \dots, 100\}$
`BonusMalus`	Driver's bonus-malus level	Discrete, starts at 100, decreases if no accidents, increases otherwise
`VehBrand`	Car brand	Categorical, values in {'B1', 'B2', ..., 'B14'}
`VehGas`	Car fuel type	Categorical, values in $\{\text{diesel}, \text{regular}\}$
`Density`	Population density (inhabitants per km^2) at driver's place of residence	Discrete, from 1 to 27,000
`Region`	Driver's region of residence	Categorical, values in $\{\text{R11}, \dots, \text{R94}\}$

Label name	**Label description**	**Label type**
`Exposure`	Duration of insurance policy in years	Continuous, values in $[0, 1]$
`ClaimNb`	Number of insurance claims filed during policy's lifetime	Discrete, values in $\{0, 1, \dots, 5\}$

1. **Poisson GLM.**

   We first fit a Poisson Generalized Linear Model (GLM), which is commonly used for insurance frequency claim prediction.

   Let $\texttt{ClaimNb}_i$ be the number of claims and $\texttt{Exposure}_i$ be the duration in years of the i-th policy. We want to predict the claim frequency $y_i = \texttt{ClaimNb}_i / \texttt{Exposure}_i$ given a training set of insurance policy features $D = \{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}_+,\ 1 \le i \le m\}$.

   Under the Poisson GLM we assume that $y_i \cdot \texttt{Exposure}_i \sim \text{Poisson}(\lambda_i \cdot \texttt{Exposure}_i)$ with mean parameter λ_i of the form

   $$\lambda_i = \exp\left(\langle \theta, x_i \rangle + \theta_0\right), \tag{1}$$

   for some regression coefficients $(\theta, \theta_0) \in \mathbb{R}^d \times \mathbb{R}$ to be estimated.

[1] See the slides from Lecture 2, Section 8 for the definitions of discrete, continuous and categorical features.

The model is fitted by minimizing the following loss (formally known as the *exposure-weighted Poisson deviance*):

$$\mathcal{L}(D, \hat{\theta}) = \frac{1}{\sum_{i=1}^{m} \texttt{Exposure}_i} \sum_{i=1}^{m} \texttt{Exposure}_i \cdot \ell(\hat{\lambda}_i, y_i), \tag{2}$$

where $\hat{\lambda}_i = \exp\left(\langle \hat{\theta}, x_i \rangle + \hat{\theta}_0\right)$ is the estimate of each policy's mean frequency and

$$\ell(\hat{\lambda}, y) = 2\left(\hat{\lambda} - y - y \log \hat{\lambda} + y \log y\right),$$

with the convention that $x \log(x) = 0$ if $x = 0$.

(a) Given the functional dependence in (1), when training a Poisson GLM model it is often necessary to transform the dataset features to guarantee a good fit. This procedure is known as *feature engineering* or *feature pre-processing* and typically requires expert knowledge and extensive data analysis.

Pre-process the following dataset features as indicated:

- `VehPower`: transform to log(`VehPower`),
- `VehAge`: convert to categorical feature with levels $[0, 6), [6, 13), [13, \infty)$.
- `DrivAge`: transform to log(`DrivAge`).
- `BonusMalus`: transform to log(`BonusMalus`).
- `Density`: transform to log(`Density`).

(b) Perform a 90%-10% train-test split, fit a Poisson GLM[2] and report Mean Absolute Error (MAE), Mean Squared Error (MSE) and the loss $\mathcal{L}$ on train and test sets.

Remark: standardize all continuous and discrete features and transform all categorical features using one-hot encoding.

2. **Poisson feedforward neural network model.**

Still working under the Poisson assumption, we can achieve a higher performance by modelling the mean parameter in (1) with a feedforward neural network, which can be understood as performing *automatic feature engineering*.

(a) Implement a feedforward neural network $\hat{\lambda} = F_\theta(x)$ with exponential activation $x \mapsto \exp(x)$ in the output layer. You should experiment with different network architectures and hyperparameters, until you find a network that performs well. You can start with two hidden layers of 20 neurons each with ReLU activation function and train for 100 epochs with batch size 10,000 and learning rate 0.01.

(b) Train the model[3] on the training set of Exercise 1(a) by minimizing the loss $\mathcal{L}$ in Equation (2). Report MAE, MSE and the loss $\mathcal{L}$ on train and test sets and show that the model outperforms the Poisson GLM from Exercise 1(b).

Remark: Try different regularization techniques, such as L^2 regularization. Remember to justify your choice of the regularization hyperparameter(s) by cross-validation.

[2] You may use the implementation available at `sklearn.linear_model.PoissonRegressor`. Remember to instantiate the model with `alpha` equal to zero (i.e. without regularization) and to make sure you minimize the *weighted* Poisson deviance by passing the exposure feature as weight to the argument `sample_weight`.

[3] You can implement the neural network using either `PyTorch` or `TensorFlow`.

3. **Tree-based methods.**

   Implement and compare the following tree-based methods[4] on the training set of Exercise 1(a). Train each model by minimizing the weighted Poisson deviance in Equation (2) and report the MAE, MSE and weighted Poisson deviance on train and test sets.

   (a) Implement a regression tree and optimize the model performance by cross-validating on the minimum impurity decrease.

   (b) Implement a random forest regression and optimize the model performance by cross-validating on the minimum impurity decrease and the number of features to consider when looking for the best split.

   (c) Implement gradient boosted trees and optimize the model performance by cross-validating on the shrinkage parameter and the number of boosting steps.

[4] You may use the implementations in the libraries `sklearn.tree` and `sklearn.ensemble`.