Project 4: Insurance Claim Prediction

(First discussion: Nov 29; Last questions: Dec 13; Deadline: Dec 20)

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 $m = 678{,}013$ car insurance policies.

Below is an overview of the data:

Feature name	**Feature description**
`PolicyID`	Insurance policy ID number
`VehPower`	Car power
`VehAge`	Car age in years
`DrivAge`	Driver's age in years
`BonusMalus`	Driver's bonus-malus level (entrance level is 100, decreases if no accidents)
`VehBrand`	Car brand
`VehGas`	Car fuel type (diesel or regular)
`Density`	Population density per km^2 at driver's place of residence
`Region`	Driver's region of residence (according to pre-2016 French classification)

Label name	**Label description**
`Exposure`	Duration of insurance policy in years
`ClaimNb`	Number of insurance claims filed

1. 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 are interested in predicting 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 $y_i \cdot \texttt{Exposure}_i \sim \text{Poisson}(\lambda_i \cdot \texttt{Exposure}_i)$ for mean parameter λ_i of the form

   $$\lambda_i = \exp\left(\langle \beta, x_i \rangle + \beta_0\right), \tag{1}$$

   for some regression coefficient $\beta \in \mathbb{R}^{d+1}$ to be estimated.

   The model is fitted by maximizing its conditional likelihood on the training set or - which is equivalent - by minimizing its exposures-weighted Poisson deviance, given by:

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

   where $\hat{\lambda}_i = e^{\langle \hat{\beta}, x_i \rangle + \hat{\beta}_0}$ is the estimate of each policy's mean frequency and $\ell(\hat{\lambda}, y) = 2\left(\hat{\lambda} - y - y \log\left(\frac{\hat{\lambda}}{y}\right)\right)$.

   (a) Since the Poisson GLM assumes the specific functional dependence in (1), it is often necessary to pre-process the features to guarantee a good fit. This process is known as *feature engineering* and typically requires expert knowledge and extensive data analysis.

   Pre-process the dataset features as follows:

- `VehPower`: set to 9 for all values greater than or equal to 9 and use VehPower as a categorical feature[1],
- `VehAge`: convert to categorical feature with levels $[0, 6), [6, 13), [13, \infty)$.
- `DrivAge`: transform to log(`DrivAge`).
- `BonusMalus`: set to 150 for all values greater than 150 and transform to log(`BonusMalus`).
- `Density`: transform to log(`Density`).

Transform all categorical features using one-hot encoding[2] and standardize all continuous features. Remember to remove the feature `PolicyID` (because it is not informative) and the features `Exposure` and `ClaimNb` (because they are used as labels).

(b) Perform a 90%-10% train-test split, fit a Poisson GLM[3] and report Mean Absolute Error (MAE), Mean Squared Error (MSE) and weighted Poisson deviance on train and test sets.

(c) Add the following continuous features (after standardization) to the training set:

$$(\texttt{DrivAge},\ \texttt{DrivAge}^2,\ \texttt{BonusMalus} \cdot \texttt{DrivAge},\ \texttt{BonusMalus} \cdot \texttt{DrivAge}^2),$$

and show that the Poisson GLM achieves a better performance on this dataset.
Explain why.
(Hint: how does the estimated frequency $\overline{\lambda}_B = \left(\sum_{i \in B} \texttt{ClaimNb}_i\right) / \left(\sum_{i \in B} \texttt{Exposure}_i\right)$ vary for different `DrivAge` buckets B? How does claim frequency depend jointly on `BonusMalus` and `DrivAge`?)

2. 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 used to perform feature engineering in an automatic way.

(a) Implement a feedforward neural network $\hat{\lambda} = F_\theta(x)$ with tanh activation function in the hidden layers and exponential activation $x \mapsto \exp(x)$ in the output layer. You can start by implementing a network with two hidden layers with 10 neurons each, but you should optimize model performance by trying different architectures and regularization techniques.

(b) Train the model on the training set of Exercise 1(a) by minimizing the weighted Poisson deviance in Equation (2). Report MAE, MSE and weighted Poisson deviance on train and test sets and show that the model outperforms the Poisson GLM from Exercise 1(c).

3. 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.

[1] A *categorical feature* is a feature that takes finitely many values $\{v_1, \dots, v_k\}$, called *levels*, which are not assumed to be in an ordered relationship.

[2] If $x \in \{v_1, \dots, v_k\}$ is a categorical feature, its one-hot encoding $x \mapsto \phi(x) \in \{0, 1\}^k$ is obtained by setting $\phi(v_i) = e_i$, where $\{e_1 = (1, 0, \dots, 0), e_2 = (0, 1, \dots, 0), \dots, e_k = (0, 0, \dots, 1)\}$ is the canonical basis on $\mathbb{R}^k$.

[3] You may use the implementation available at `sklearn.linear_model.PoissonRegressor`.

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