Predictive Analytics Manual

October 15, 2019

Contents

1 Introduction

Predictive analytics is the branch of the advanced analytics which is used to predict about unknown future using many techniques from data mining, statistics, predictive modeling and artificial intelligence to analyze the current or historical data. The patterns found in the historical data can identify the risks for future. In this manual, we mainly focus on predictive models including regression models and machine learning models.
Machine learning is one popular field of artificial intelligence that usually uses statistical techniques and skills to train the computers with data. Machine learning are usually classified into two categories, supervised learning and unsupervised learning, depending on whether there is a "label" in the learning system.

Table 1 displays the main models implemented in different model engines, i.e. , R Model, Python Model, XGBoost Model. In Section 2, we will give the mathematical expression and the algorithm of each model, the strength and weakness due to the structure of the model, meanwhile we will also talk about what kind of data the model is good for.

In section 3 we talk about the data preprocessing for missing values in categorical variables and continuous variables, respectively.

2 Predictive Models

2.1 Nystroem Kernel SVM Regressor

The Kernel method only uses the observations close to the target point x_0 to fit the simple model via assigning the kernel $K(x_i, x_0)$ to x_i based on the distance between x_i and x_0. The Kernel function is used to calculate the inner product of two vectors which is also the fundamental calculation for kernel-based predictors (such as Support Vector Machines). SVMs are supervised learning models which are used for classification and regression. It efficiently uses the kernel trick which can map their inputs into high-dimensional feature spaces for regularizing non-linear classification and regression.

2.1.1 Computing SVM for Classification

As shown in Figure 1 (from Hastie et al. (2009)), the margins separate the data into two regions: $\{-1, 1\}$. We can choose x^- s.t. $f(x^-) = -1$ and x^+ to be the closest point s.t

Table 1: Model Classification

Python Model	Nystroem Kernel SVM Regressor
	Gradient Boosted Trees Regressor
	Auto-tuned K-nearest neighbors regressor (Minkowski distance)
	ExtraTrees Regressor
	Random Forest Regressor
	Decision Tree Regressor
Blend Model	AVG Blender
	ENET Blender
	Advanced AVG Blender
XGBoost Model	Extreme gredient boosted trees regressor with early stopping
	Frequency-Severity extreme Gradient Boosted Trees
Data Robot Model	Generalized Additive2 Model
	Elastic-Net regressor
	Generalized Additive Model
	Elastic-Net Regressor
	Frequency-Severity ElasticNet
	Two-Stage(Elastic net classifier and elastic net regressor)
	Linear regression
	Ridge Regressor
	RuleFit regressor
LightGBM Model	Light Gradient Boosting on ElasticNet Predictions
	Light Gradient Boosted Trees Regressor with early stopping
Vowpal Wabbit Model	Vowal Wabbit Regressor
Eureqa Model	Eureqa Generalized Additive Model
R Model	Breiman and Cutler Random Forest Regressor
	Gradient Boosted Trees Regressor
TensorFlow Model	TensorFlow Neural Network Regressor

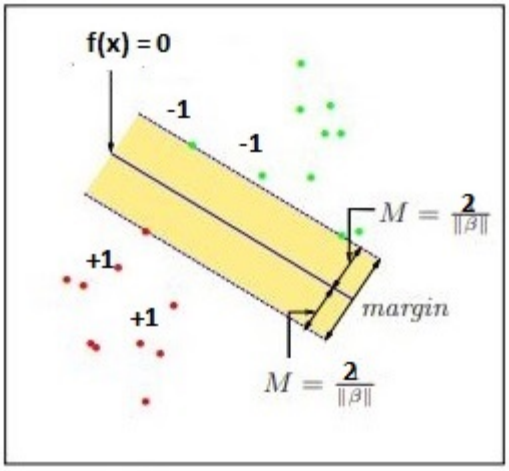

Figure 1: SVM classifiers

$f(x^+) = +1$, thus $x^+ = x^- + c \cdot \beta$, where β is perpendicular to the boundaries. The points on the boundaries satisfies:

$$
\begin{aligned}
& x^-\beta + b = -1 \quad x^+\beta + b = +1 \\
& \implies \beta(x^- + c \cdot \beta) + b = +1 \\
& \implies c\|\beta\|^2 + \beta x^- + b = +1 \\
& \implies c\|\beta\|^2 - 1 = +1 \\
& c = \frac{2}{\|\beta\|^2} \\
& \text{Margin length} : M = \|x^+ - x^-\| = \|c\beta\| = \frac{2}{\|\beta\|}
\end{aligned}
$$

We need to minimize $\|\beta\|$ if we want to maximize M, the margin. Then the primal problem is in the form of:

$$\min\|\beta\|^2 \quad \text{s.t.} \quad y_i(x_i\beta + b) \geq +1$$

The Lagrange (primal) function is

$$L_P = \|\beta\|^2 + \sum_i \alpha_i(1 - y_i(x_i\beta + b)) \tag{1}$$

which we minimize w.r.t β and b. If $y_i(x_i\beta + b) \geq +1$ then β satisfies the subjection in the primal, thus we get $\alpha_i = 0$ since the penalty term is useless. On the other hand, if $y_i(x_i\beta + b) < +1$, then $\alpha \to \infty$. So $\alpha_i > 0$ indicates that the constraint on $y_i(x_i\beta + b)$ is tight, that is, $y_i(x_i\beta + b) = 0$. This also means that the non-zero α_i's are only for those observations on the boundaries, which are known as *support vectors*. Setting the respective

derivatives to 0, then we get

$$\beta = \sum_i \alpha_i y_i x_i \quad b = \frac{1}{N} \sum_i (y_i - x_i \beta)$$

The solution for β depends on those *support vectors*. Plug in to equation 1, we obtain the Lagrangian Dual function:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{2}$$

We maximize L_D subject to $\sum_i \alpha_i y_i = 0$.

The equation 2 and its solution can be represented by choosing different inner product (kernel function) with the input features,

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{3}$$

The kernel function $K(x_i, x_j)$ computes the inner products in the transformed space in order to solve non-linear problems.

2.1.2 Nystroem Method for estimating the Kernel

While a major problem for SVMs is that the amount of computation for the kernel is required to find the solution scales as $\mathcal{O}(n^3)$. To solve this problem, we can use Nystroem method to approximate the eigendecomposition of the Gram matrix. The Gram matrix K in an inner product space is the Hermitian matrix of inner products of a set of vectors $v_1, \ldots, v_n$ and each entry is given by $G_{ij} = \langle v_i, v_j \rangle$. Using Nystroem method (Williams and Seeger (2001)), it can be proved that the Gram matrix can be approximated by $\tilde{K} = K_{n,m} K_{m,m}^{-1} K_{m,n}$ where $K_{n,m}$ is the $n \times m$ block of the original matrix K. So large n_component will lead to higher accuracy but with high memory usage. Therefore, there is a small tradeoff in accuracy.

Summary: In the datarobot platform, support vector machines are very efficient in high-dimensional spaces (especially for the number of dimensions >> number of observations) which use Nystroem as the kernel approximation method and Radial Basis Function Kernel as default. SVMs can handle both continuous and discrete variables with some appropriate data preprocessing: discrete variables should be rescaled and continuous variables should be normalized.

2.1.3 Parameters in Model

- *approx*: The kernel approximation method to use ('nystroem', 'fourier'). The default method is Nystroem.
- *n_components*: the dimension of the submatrix chosen to approximate the Gram matrix.
- γ: The default kernel is RBF kernel: $\exp(-\gamma\|x - x'\|^2)$ with a parameter being considered: γ. γ determines the influence a single training example has. With larger γ, the closer other examples must be affected.
- α: Small positive number to improve the conditioning of the problem.

Key words: Large data set, high-dimension, low cost, small tradeoff in accuracy.

2.2 K-nearest neighbors Regressor (Minkowski Distance)

Neighbors-based regression is supervised and non-parametric method of learning: it does not try to build a general model, but simply stores instances of the training data (Hastie et al. (2009)). The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set. Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based the mean of the labels of its nearest neighbors.

The k-nearest neighbor fit for $\hat{Y}$ is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{4}$$

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample (Hastie et al. (2009)). The neighborhood means the distance metric which is Minkowski Distance, and with p=2 is equivalent to the standard Euclidean metric. From the equation 4, the boundary will become smoother as the value of k increasing. The error obtained from the training set and validation are the two parameters to select k.

Key words: easy to interpret the result, short calculation time

2.3 Decision Trees

Decision Trees is a non-parametric supervised method for classification and regression. Decision trees are effective for learning non-linear functions, and are particular adept at finding

threshold effects. However, decision trees tend to exhibit very high variance, which can be corrected through the use of ensemble methods such as bootstrap-aggregation (bagging), random forest, or gradient boosting machines.
A terminal node (or external node) is any node that doesn't have child nodes. An internal node is any node of a tree that has child node. The root is the top node in a tree. The degree is the number of children for a given node.

2.3.1 Parameters for Tuning in Model

- *criterion*: The function to measure the quality of a split, and the default criterion is mean squared error.
- *max_features*: The number of features to consider when looking for the best split.
- *max_depth*: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- *min_samples_split*: The minimum number of samples required to split an internal node.

2.3.2 Regression & Classification Trees

Regression Trees
In a regression tree, suppose we have M regions $R_1, \ldots, R_M$ and $f(x) = \sum_{i=1}^{M} c_m I(x \in R_m)$ (Hastie et al. (2009)). Then define

$$N_m = \#\{x_i \in R_m\} \hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i \tag{5}$$

Classification Trees
If the response variable belongs to a class taking values from 1 to K. In the region R_m with N_m observations. Define

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

which measures the impurity in node m. The observations in node m are classified to class $k(m) = \text{argmax}_k \hat{p}_{mk}$, the majority class in node m. The node impurity is defined as Gini Index:

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = \sum_{k=1}^{K} \hat{p}_{mk} - \sum_{k=1}^{K} \hat{p}_{mk}^2 = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2$$

which can be differentiable and more amenable to numerical optimization.

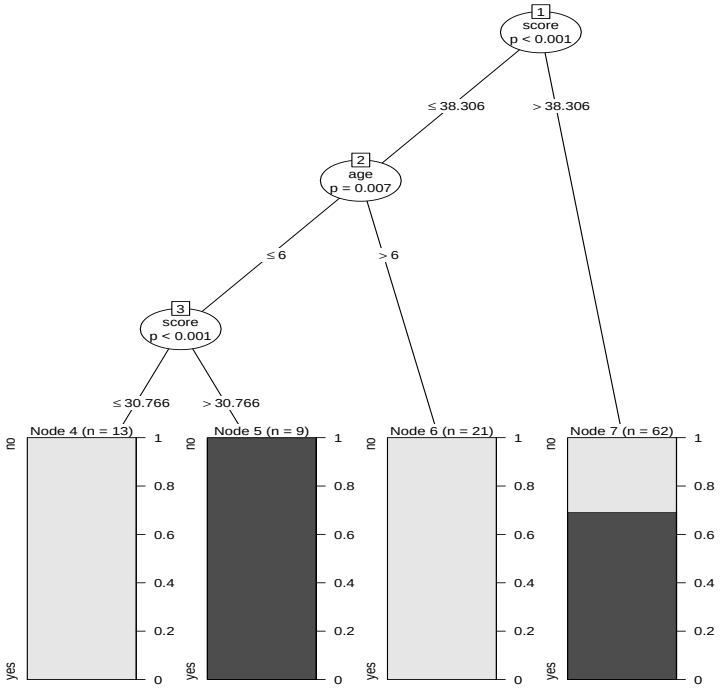

Figure 2: Illustration of a Decision Tree

2.3.3 Measure Metric

Gini impurity

Details in classification section.

Information gain

One can use Information Gain to decide which feature to split in building the tree. Information gain is difference $I(Y, X_i) = H(Y) - H(Y|X_i)$ where $H(\cdot)$ is the entropy loss. The entropy of a random variable Y is $H(Y) = -\sum_y P(Y = y) \log_2 P(Y = y)$. In fact, $H(Y)$ is the expected number of bits needed to encode a randomly drawn value of Y. According to information theory, most efficient code assigns $-\log_2 \Pr(Y = y)$ bits to encode the message $Y = y$. So the expected number of bits is $\sum_y P(Y = y) \log_2 P(Y = y)$.

The conditional entropy is calculated by $H_L \times P_L + H_R \times P_R$ where H_L, H_R are the entropy on the left and right child nodes after splitting, P_L, P_R are the fraction of the data goes to the left node or right node after splitting. The tree splits each node at the most informative feature (with the largest information gain). When the information gain becomes 0, the feature doesn't divide the working net at all.

2.3.4 Notable Decision Tree Algorithms

CART: Classification and Regression Trees.
It construct binary trees which means that each split has exactly two outgoing edges. It can handle both categorical and numerical features. The split is based on Gini impurity, by minimizing the Gini impurity the decision tree finds the best split.

For numerical features, the candidate splits are simply the midpoint between pairs of consecutive values for the feature. For example, a numerical independent variable X_1 taking values of $\{3, 5, 10, 14, 24\}$, then the possible split points for X_1 are $\{4, 7.5, 12, 19\}$. These splits are ranked by how much they reduce impurity. The reduction is the difference between the impurity before and after the split.

CART procedure can be modified to handle missing data without ignoring the observations with missing values. Modification: Let S and D be the splitting criterion and dataset, and let D_0 be the data set with missing values. Under S, the information gain $\Delta I(D) = I(D) - \sum_k \Pr(X = k) I(D_k | X = k)$. For $D_0 \neq \emptyset$, $I(D - D_0) = \frac{|D - D_0|}{|D|} \Delta I(D - D_0)$ and also assign the weight $\omega = \frac{|D - D_0|}{|D|}$ to the sample with missing value. Therefore, the information gain is calculated using sum of weights instead of counts.

CART is also robust to outliers since the splitting happens based on proportion of samples within the split ranges and not on absolute values. The pruning strategy for CART uses cost-complexity pruning method (we will discuss in the following section).

C4.5
The learning algorithm C4.5 is similar to CART except that the splitting criteria is Gain Ratio instead of Gini impurity. C4.5 can also handle both categorical and numerical variables. The missing attribute values are simply not used in gain and entropy calculation hence C4.5 sometimes constructs empty branches with zero values. This is a disadvantage of C4.5.
For continuous data C4.5 uses a threshold value In algorithm C4.5, error based pruning strategy is used.

2.3.5 Pruning (Avoid Overfitting)

Pruning in decision trees is used to reduce the size of the tree by removing sections of the tree with little power to split and hence to reduce the complexity and overfitting.
Overfitting is a significant problem in decision tree and many other predictive models. Overfitting happens when the learning algorithm continues to develop hypothesis that reduces

training set error at the cost of increasing test set error.

Pre-Pruning (Early Stopping Rule) & Post-Pruning
Pre-pruning stops growing the tree earlier before it perfectly trains the data set. The algorithm stops
1. when the error estimate is above some threshold,
2. when the distribution of instances become independent from available features, using Chi-square Test of independence,
3. if expanding the current node doesn't improve impurity.

Post-pruning prunes the tree after the tree perfectly classifies the training data set. Post-pruning is commonly used because it is not easy to precisely implement early stopping in pre-pruning. After the tree is grown to its entirety, trim the nodes from bottom to top, replace the sub-tree by a leaf node is the pre-defined error measure increases after trimming. Split the data set into training and validation set as shown in Figure 3. Build the tree using training set and evaluate the effect of post-pruning nodes using the validation set. The commonly used method in pruning trees is cost-complexity pruning.
What is cost complexity pruning? Based on Equation 5, define $Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$, and the cost complexity criterion is

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

For each α, by minimizing $C_\alpha(T)$, one can show that there is a unique smallest subtree T_α. We can estimate α by five- or tenfold cross-validation: choose the estimation $\hat{\alpha}$ which minimizes the cross-validated sum of squares.

2.3.6 Advantages

Simple and easy to understand and interpret the result, handle both numerical and categorical variable, little data preprocessing and good at handling large datasets.

Key words: high variance, very non-robust.

2.4 Ensemble Methods

Ensemble learning methods have become the most powerful methods which aim at combining the predictions of several base estimators to improve the robustness rather than a single

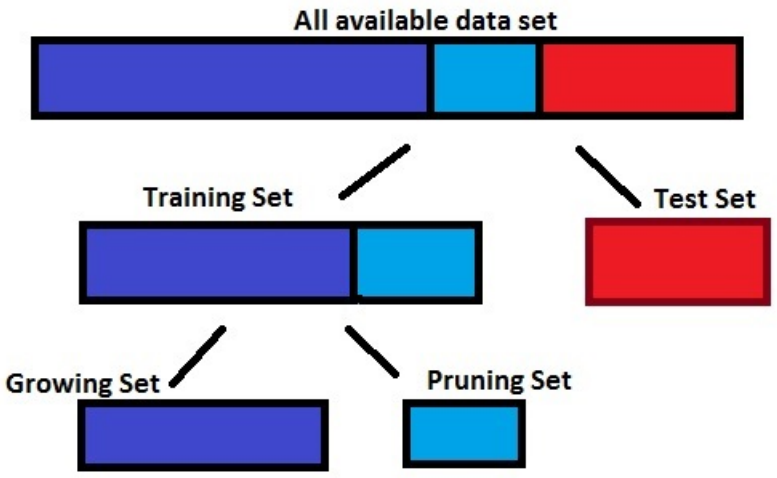

Figure 3: Partitioning Data in Decision Tree

estimator. The ensemble learner has the following structure:

$$F(\mathbf{x}) = f_0 + \sum_i v_i f_i(\mathbf{x})$$

Given a family of function, the individual members of the ensemble are generated using the algorithm shown below by Friedman et al. (2008):

Algorithm 1 Algorithm for Ensemble

Data: $\{x_i, y_i\}_i^N$

Result: Ensemble $\{f_m(x)\}_1^M$

initialization: $F_0(x) = \text{argmin}_\rho \sum_i L(y_i, \rho)$

for m =1, ..., M **do**

$\text{P}_m = \text{argmin}_P \sum_{i \in \eta} L(y_i, F_{m-1}(x_i) + f(x_i; P))$

$\text{f}_m(x) = f(x; P_m)$, fit the base learner$f_m(x)$with the training data set$\{x, P_m\}$

$\text{F}_m(x) = F_{m-1}(x) + v \cdot f_m(x)$

end

$\eta < N$ represents a subsample of index randomly drawn without replacement from the original training data index. v is the shrinkage parameter with $0 \leq v \leq 1$ to control the learner rate of each ensemble member.

There are two popular ensemble methods which are commonly used, averaging method which simply averages the estimator generated from different independent models thus reduces the variance, i.e. Bagging, Random Forest. Another method is boosting method which builds sequentially with weak models to produce powerful models (Pedregosa et al. (2011)).

2.4.1 Bagging Method

A bagged ensemble is obtained by drawing the samples randomly with replacement and using the loss function to be squared-error loss, $L(y, f(x)) = (y - f(x))^2$. The bagging methods usually work best with strong models by adding regulation into the algorithm then reduce the variance of the base estimator. While the boosting methods, i.e, GBoosted model, XGboosted models, usually work best with weak models.

An example of a bagging ensemble is K-nearest neighbors, we will discuss K-nearest neighbors in section 2.2.

2.4.2 Forest of Random Trees and the Extra Trees

We discuss this in section 2.9.

2.4.3 AdaBoost

AdaBoost is obtained by setting the loss function to be exponential loss $L(y, \hat{y}) = \exp(-y\hat{y})$ for $y \in \{-1, 1\}$. And predictions are the sign of the final function $F_M(x)$. The pseudo code for AdaBoost algorithm is

Algorithm 2 Algorithm for AdaBoost

Data: Data

Result: $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$

initialization the weights $w_i = \frac{1}{N}, i = 1, \ldots N$

for m =1, ..., M **do**

  Fit a classifier $G_m(x)$ to the training data using weight w_i

  Compute $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$

  Compute $\alpha_m = \log \frac{1-err_m}{err_m}$

  Set $w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(x_i))]$, for $i = 1, \ldots, N$.

end

The power of Ada boosting is that, the observations that are hard to be classified correctly will be given increasing weight as the iteration processed.

2.4.4 Gradient Boosting Trees

(We will explain in the section 2.5.)

2.5 Gradient Boosted Trees Regressor

Gradient Boosting is a supervised learning algorithm which is used for classification and regression problems and ensemble sequentially weak models, i.e., decision trees, to increase the power of prediction. GBMs are a generalization of Adaboost algorithm (Freund et al. (1999)) to handle arbitrary loss functions. The basic step is to use a training sample $\{y_i, \mathbf{x}_i\}_n^N$ to obtain the estimation $\hat{F}(\mathbf{x})$ the function $F^*(\mathbf{x})$ mapping $\mathbf{x}$ to y:

$$F^*(\mathbf{x}) = \text{argmin}_F \mathbf{E}_{y,x} L((y, F(\mathbf{x}))) \tag{6}$$

where $L(\cdot, \cdot)$ is the targeted loss function. The frequently employed loss functions $L(y, F)$ include least squared-error $(y - F)^2$, absolute error $|y - F|$ for $y \in \mathcal{R}^1$ (regression), exponential loss $\exp(-yF)$ and negative binomial log-likelihood $\log\left(1 + e^{-2yF}\right)$ for $y \in \{-1, 1\}$ (classification), Poisson loss or Tweedie loss if y is a count or zero-inflated variable, respectively.

From iteration $1 \leq m \leq M$, let $f_{m+1}(x) = f_m(x) + h(x)$ where $h(x) = y - f_m(x)$, that is, the residual $y - f_m(x)$ is fitted by $h(x)$. Then $\hat{f}(x) = \sum_{m=1}^{M} \gamma_m h_m(x) + \text{constant}$. Start from constant prediction, add a new function each time:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Therefore, the pseudo-code for generic gradient boosting method is For the iteration m, we

Algorithm 3 Pseudo Algorithm for Gradient Boosting

Data: The training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$

Result: $\text{f}_M(x)$

for *m from 1 to M* **do**

$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$

Fit the base learner $h_m(x)$ using the training data set $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$

Compute $\gamma_m = \text{argmin}_\gamma \sum_{i=1}^n L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$

$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{n_m} \gamma_{jm} I(x \in R_{jm})$

end

fit a decision tree $h_m(x)$ to the data set $\{x_i, r_{im}\}$, suppose that h_m divides the input space into $R_{jm}, j = 1, 2, \ldots, n_m$ different regions and let γ_{jm} be the predicted value in each region, then

$$h_m(x) = \sum_{j=1}^{n_m} \gamma_{jm} I(x \in R_{jm})$$

The learning rate in each iteration can be controlled by the parameter v, then the last step in Algorithm 3 can be replaced by

$$f_m(x) = f_{m-1}(x) + v \cdot \sum_{j=1}^{n_m} \gamma_{jm} I(x \in R_{jm})$$

There is a tradeoff between the parameters v and M because smaller v usually lead to larger M and favor better test error. In practice, we usually set $v < 0.1$ and choose M by early stopping (Details in section 2.6.4).

2.5.1 Application to Tweedie Response Variable

The Tweedie response Y can be expressed by the compound Poisson model:

$$Y = \sum_{i=1}^{N} X_i$$

where $N \sim \text{Poisson}(\lambda)$ and $X_i \sim \text{Gamma}(\alpha, \beta)$. When $N = 0$, the distribution of Y has a probability mass $\Pr(Y = 0) = \Pr(N = 0) = \exp(-\lambda)$. Then the probability density of Y can be written as two parts given some parameters:

$$f_Y(y|\lambda, \alpha, \beta) = \Pr(N = 0) f_0(y) + \sum_{i=1}^{\infty} \Pr(N = i) f_Y(y|N = i) \tag{7}$$

$$= \exp(\lambda) f_0(y) + \sum_{i=1}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!} \frac{y^{i\alpha-1} e^{-y/\beta}}{\beta^{i\alpha} \Gamma(i\alpha)} \tag{8}$$

where f_0 is the Dirac delta function[1] at zero. The compound Poisson distribution is proved to belong to the tweedie model (Tweedie (1984)):

$$f(y|\theta, \phi) = a(y, \phi) \exp\left(\frac{y\theta - \kappa(\theta)}{\phi} \right)$$

where $a(\cdot)$ is the given normalizing function and $(\cdot)$ is the given cumulant function, $\theta \in \mathcal{R}$ and $\phi \in \mathcal{R}_+$ is the dispersion parameter. For the exponential family model, $\text{E}(Y) \equiv \mu = \kappa'(\theta)$

[1]The Dirac delta function is equal to zero everywhere except for zero and its integral over the entire real line is one.

and $\text{Var}(Y) = \phi\kappa''(\theta)$. While based on the mean-variance relationship of tweedie variable, $\text{Var}(Y) = \phi\mu^\rho$, then (Yang et al. (2017))

$$\theta = \begin{cases} \frac{\mu^{1-\rho}}{1-\rho}, & \rho \neq 1 \\ \log\mu, & \rho = 1 \end{cases}, \quad \kappa(\theta) = \begin{cases} \frac{\mu^{2-\rho}}{2-\rho}, & \rho \neq 2 \\ \log\mu, & \rho = 2 \end{cases} \tag{9}$$

If we set

$$\lambda = \frac{1}{\phi}\frac{\mu^{2-\rho}}{2-\rho}, \quad \alpha = \frac{2-\rho}{\rho-1}, \quad \gamma = \phi(\rho-1)\mu^{\rho-1}$$

The Tweedie loss function can be expressed as

$$L(y,\mu) = 2\left[\frac{y^{2-\rho}}{(1-\rho)(2-\rho)} - \frac{y\mu^{1-\rho}}{1-\rho} + \frac{\mu^{2-\rho}}{2-\rho}\right]$$

where $\rho = \frac{\alpha+2}{\alpha+1}$, $\mu = \lambda\cdot\alpha\cdot\theta$.

If we extend this model to insurance policies, let N_i be the number of claims in the policy i, then $Y_i = \sum_{k=1}^{N_j} X_k$ is the claim size for policy i, where $\text{E}(Y_i) = \mu_i$ We have $\log(\mu_i) = F(x_i)$, therefore, the log-likelihood function can be expressed as (Yang et al. (2017)):

$$l(F(\cdot),\phi,\rho|\{y_i,x_i,\omega_i\}_{i=1}^n) = \sum_{n=1}^{n}\log f_Y(y_i|\mu_i,\phi/\omega_i,\rho) \tag{10}$$

$$= \sum_{i=1}^{n}\frac{\omega_i}{\phi}\left(y_i\frac{\mu_i^{1-\rho}}{1-\rho} - \frac{\mu_i^{2-\rho}}{2-\rho}\right) + \log a(y_i,\phi/\omega_i,\rho) \tag{11}$$

If we plug the above negative log-likelihood function in the equation 6, then

$$F^*(\mathbf{x}) = \text{argmin}_{F\in\mathcal{F}}\{-l(F(\cdot),\phi,\rho|\{y_i,x_i,\omega_i\}_{i=1}^n)\} = \text{argmin}_{F\in\mathcal{F}}\sum_{i=1}^{n}\Psi(y_i,F(x_i))|\rho) \tag{12}$$

where

$$\Psi(y_i,F(x_i))|\rho) = \omega_i\left\{-\frac{y_i\exp[(1-\rho)F(x_i)]}{1-\rho} + \frac{\exp[(2-\rho)F(x_i)]}{2-\rho}\right\} \tag{13}$$

Once we derive the loss function for tweedie model, the algorithm can be implemented as in Algorithm 3.

2.5.2 Feature Importance

Suppose that $X = \{X_1, X_2, \ldots, X_p\}$ is the input features and let $\mathcal{I}_{X_j}(T_m)$ (introduced by Breiman et al. (1984)) denote the variable importance measure in a single tree T_m that has L terminal nodes. $\mathcal{I}_{X_j}(T_m)$ is defined as the heterogeneity reduction of Y produced by X_j.

$$\mathcal{I}_{X_j}(T_m) = \sum_{=1}^{L-1}\hat{\delta}_l^2 I(\upsilon(X_j) = l))$$

The sum is over the $L-1$ nodes or leaves and $\hat{\delta}_l^2$ is the estimated squared error different between the constant fit over the entire region and the binary fits. $v(X_j) = l$ is the event that X_j is selected in the node l. Then for entire iterations, the importance measure of X_j is simply averaged over the total M trees

$$\mathcal{I}_{X_j} = \frac{1}{M}\mathcal{I}_{X_j}(T_m)$$

2.5.3 Partial Dependence Plots

Partial dependence plot displays the dependence between the target function $f(X)$ and some certain features. Therefore, after we identify the most relevant features, we need to understand in depth of the dependence.

Suppose that we have p features $x = \{x_1, \ldots, x_p\}$ and let $A \subset \{1, 2, \ldots, p\}$. By Hastie et al. (2009), we can write $\hat{y} = f(x)$ and $f(x) = f(x_A, x_{A^c})$ and the average partial dependence is estimated as

$$\bar{f}(x_A) = \frac{1}{N}\sum_{i=1}^{N} f(x_A, x_{i,A^c}) \tag{14}$$

Visualize the partial dependence plot on selected features can help to understand the property of these features. The partial dependence function tells us how the value of x_A influences the model predictions $\hat{y}$ after we have "averaged out" the influence of all other variables.

2.5.4 Parameters in Model

- *loss*: loss function to be minimized. 'ls' is least squares regression, 'lad' refers to least absolute deviation which is highly robust based on order information of the input variables.
- *learning_rate*: shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
- *n_estimators*: the number of boosting stages to perform. So GBM is robust to overfitting, thus with larger number may lead to better fitting.
- *max_depth*: Max number of nodes for a single tree.

2.6 Extreme Gradient Boosted Trees Regressor with Early Stopping

The Extreme GBM is an efficient and parallel version of GBM. The difference between XGBM and GBM is that the XGBM introduces a regulation term of penalty of the com-

plexity, which makes the model better avoid overfitting problem.

Loss functions: Similar with GBMs, The xgboost regressor uses least squares loss by default, but can also use poisson loss for count problems, tweedie loss for zero-inflated count problems, and gamma loss for right skewed, positive distributions. **The loss function in xgboost must be twice differentiable**.

2.6.1 Algorithm of Extreme Gradient Boosting

Start from constant prediction, add a fitted function eachiteration:

$$\begin{aligned}
\hat{y}_i^{(0)} &= constant \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

f can be identified via minimizing the loss function $L^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{t} \Omega(f_k)$ where $\Omega(f_k)$ is the penalty term for the tree in the iteration k, $f_k \in \mathcal{F}$. We have

$$L^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k} \Omega(f_k) = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

Recall the Taylor expansion:

$$f(x + \Delta x) \approx f(x) + f'(x)x + \frac{1}{2} f(x)\Delta x^2$$

and define $g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$. Then

$$L^{(t)} \approx \sum_{i=1}^{n} [L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

then the loss function (without constant) becomes

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Next we need to find out the explicit form of the complexity of the trees. First we define a mapping $q : \mathcal{R}^d \to \{1, 2, \cdots, T\}$ that maps the input to the index of the region, and a vector ω of scores in each region. The function is defined by

$$f_t(\mathbf{x}) = \omega_{q(\mathbf{x})}$$

Therefore, the penalty term for the complexity is in the form of

$$\Omega(f_t) = \underbrace{\gamma T}_{\text{Number of leaves}} + \overbrace{\frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2}^{L_2 \text{ norm of leaf scores}}$$

And it penalizes on the number of regions and the sum of squared scores of each region in order to fix overfitting problem.

The last step is to define the information gain of the trees Let $I_j = \{i|q(\mathbf{x}_i) = j\}$ denote the instance set of region j. Then

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)\omega_j^2] + \gamma T \tag{15}$$

Hence we can identify the optimal ω_j^* for region j if we set $q(x)$ as fixed,

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

Plug ω_j^* into equation 15:

$$\tilde{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{16}$$

This objective is a function of q because it depends on the structure of the mapping. By using this objective function we can apply a greedy algorithm to the tree growing. Let the instance set $I = I_R \bigcup I_L$ where I_R and I_L denote the right and left split, respectively. Therefore the gain can be calculated by

$$gain = \frac{1}{2}[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}] - \gamma \tag{17}$$

$$= \frac{1}{2}\Bigg[\overbrace{\frac{G_L^2}{H_L + \lambda}}^{\text{the score from left split}} + \underbrace{\frac{G_R^2}{H_R + \lambda}}_{\text{the score from right split}} - \overbrace{\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}}^{\text{the score without split}} \Bigg] - \gamma \tag{18}$$

where γ is the the complexity cost if we add a new leaf. We can handle missing values using the information gain obtained in equation 17, and the pseudo-code for the extreme gradient boosting algorithm will be implemented in the following way:

Algorithm 4 Pseudo-code of the tree split finding with missing value.

Input: A dataset X

Input: I, instance set of current node

Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$

Input: $d \rightarrow$ feature dimension

Output: Split and default direction with max gain

gain $\leftarrow 0$;

$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$

for $k = 1 \rightarrow m$ **do**

  enumerate missing value goto right

  $G_L \leftarrow 0 \quad H_L \leftarrow 0$

  for j *in sorted(I_k, ascent order by x_{jk})* **do**

    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$

    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$

    gain $\leftarrow \max(\text{gain}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$

  end

  enumerate missing values goto left

  $G_R \leftarrow 0 \quad H_R \leftarrow 0$

  for j *in sorted(I_k, descent order by x_{jk})* **do**

    $G_R \leftarrow G_R + g_j$, $H_L \leftarrow H_R + h_j$

    $G_L \leftarrow G - G_R$, $H_L \leftarrow H - H_R$

    gain $\leftarrow \max(\text{gain}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$

  end

end

Output: Split and default direction with max gain

2.6.2 Difference Between XGBoosted and GBoosted

- The growths of the tree in each iteration are different. In XGBoosted model, the information gain defined at each node is different from the one in decision tree,

$$\text{gain (in xgboost)} = \frac{1}{2}\left[\frac{G_L}{H_L+\lambda} + \frac{G_R}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

  Using the information gain derived from the Taylor expansion, we determine and stop the split if the gain becomes negative. There is a trade-off between simplicity and predictiveness, since the a split may benefit future splits.

  In the GBoosted model, the split at each node is decided by the gradient of the Loss

function at the previous fitted tree, which can be expressed as

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

The targets r_{im} determines the terminal regions R_{jm} which will be used to compute the current tree.

- There is no regulation term in the GBoosted model, while the penalty for the complexity of the model is included in the XGBoosted model. The complexity is defined by $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$. This regularization term can efficiently avoid overfitting problem.

2.6.3 XGB models on ElasticNet predictions

In datarobot engine, the XGB models often use the raw predictions obtained from previous model, i.e. ElasticNet model, as offset. For different distributions, the non-zero offsets are treated differently which will give us the corresponding initial values. For tweedie response variable:

$$\Psi(y_i, F(x_i))|\rho) = \omega_i \left\{ -\frac{y_i \exp[(1-\rho)F(x_i)]}{1-\rho} + \frac{\exp[(2-\rho)F(x_i)]}{2-\rho} \right\}$$

$$\text{Initial Value} : f_0(x) = \log\left(\frac{\sum \omega_i y_i}{\sum \omega_i \ \ \text{offset}_i}\right)$$

2.6.4 Early Stopping

Early stopping training can determine the number of iterations in boosting. The training data is also split into a training set and a test set, and at each iteration a specified metric is computed to score the model on the test set. If the metric of test set doesn't improve for 200 iterations (tunable in advanced tuning), the training procedure stops, and the model returns the fit at the best tree so far. The approach can save a lot of time by not continuing for thousands more trees thus to avoid overfitting and further trees will not result in more accuracy. We can specify the stopping metric used for early stopping, *logloss* is for classification, *deviance* is for regression and so on.

2.7 Light GBM

Light GBM uses gradient boosting method based on tree learning algorithm, however, Light GBM grows tree vertically compared with other algorithm, i.e, GBM or XGBM, which grows

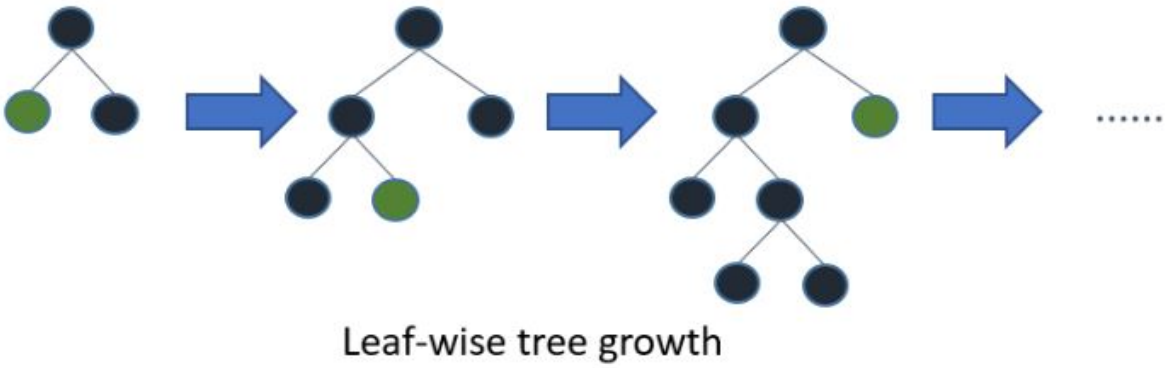

Figure 4: Illustration for Leaf-wise tree grwoth

tree horizontally (Figure 4). This means that the Light GBM grows tree leaf-wise instead of level-wise. It chooses the leaf with maximum delta loss. For a single tree, the leaf-wise algorithm will reduce more loss than level-wise algorithm. Light GBM can deal with large dataset and takes less memory to run. Instead of using pre-sorted algorithm which is often used in XGB, Light GBM uses Histogram-based algorithm which bucket continuous feature into discrete bins so that speeding up training. Light GBM also uses two new sampling methods, GOSS (Gradient-based One-side Sampling) and EFB(Exclusive Feature Bundling) which are done by Ke et al. (2017).

What is *data gradient*? Gradient is the slope of the tangent of loss function. so the points with large gradients in some sense are important for finding the optimal split as they have higher error. In Adaboost, the sample weight can be a good indicator for the importance of the data, however, in GBM, there is no such weight, thus the training error for the data instance with small gradient will be small.

We cannot directly discard these data instances with small gradient which will change the distribution and hurt the accuracy of the learning. So the GOSS can keep a good balance between the number of data reducing and the accuracy of the learning. Instead of using histogram-based algorithm dealing with the entire dataset, GOSS can reasonably reduce the computational cost on the large-scale dataset. And GOSS also introduce a constant multiplier for the data points with small gradient to keep the same data distribution when computing the information gain. Therefore, with appropriate choice of the multiplier, GOSS can keep a good balance between data reduction and the accuracy of the learners.

2.7.1 Gradient-based One-side Sampling

The GOSS keeps all the data instances with large gradient and performs random sampling on the instances with small gradient. First, GOSS picks up the top $a \times 100\%$ instances after sorting the data according to the absolute value of their gradients. Then $b \times 100\%$ instances are randomly sampled from the rest of the data. Finally, GOSS applies a constant multiplier $\frac{1-b}{a}$ to the information gain calculation. The LightGBM with GOSS (Ke et al. (2017)) is implemented as following:

Algorithm 5 Pseudo Code of GOSS

Input: I: Training Data ; d: iterations ;
Input: a: sampling ratio of large gradient data ;
Input: b: sampling ratio of small gradient data ;
Input: loss: loss function; L: weak learner ;

models $\leftarrow \{\}$, proportion $\leftarrow \frac{1-a}{b}$;

topN $\leftarrow a\times$ nrow(I), randN $\leftarrow b\times$ nrow(I) ;
for *i=1 to d* **do**
  prediction $\leftarrow$ models.predict(I) ;
  g $\leftarrow$ loss(I, prediction), weight $\leftarrow \{1, 1, \dots\}$;
  SortedLossInd $\leftarrow$ GetSortedIndices(abs(g)) % Sorted the data gradients ;
  topSet $\leftarrow$ SortedLossInd[1:topN] % Selected the top N data instances ;
  randSet $\leftarrow$ RandomSelect(SortedLossInd[topN:nrow(I)], randN) ;
  usedSet $\leftarrow$ topSet + randSet ;
  weight[randSet] $\times =$ proportion $\triangleright$ Assign weight proportion to the small gradient data.
  new Model $\leftarrow$ L(I[usedSet], -g(usedSet), weight[usedSet]) ;
  models.append(newModel)
end

2.7.2 Information Gain for GOSS

Recall that in decision tree, the model splits each node at the most informative feature. In Gradient Boosting Trees, suppose that we have n i.i.d data instances and each $x_i, i = 1, 2, \dots, n$, is in space $\mathcal{X}^s$. In each iterations, $g_1, \dots, g_n$ represent the negative gradient of the loss function w.r.t the previous learner function. Let O be the training dataset on a fixed node of the decision tree, the variance gain of the splitting feature j at point d for this node is defined as

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{(\sum_{\{x_i \in O: x_{ij} \leq d\}} g_i)^2}{n^j_{l|O}(d)} + \frac{(\sum_{\{x_i \in O: x_{ij} > d\}} g_i)^2}{n^j_{r|O}(d)}\right) \tag{19}$$

where $n_O = \sum I[x_i \in O], n^j_{l|O} = \sum I[x_i \in O : x_{ij} \leq d], n^j_{r|O} = \sum I[x_i \in O : x_{ij} > d]$. For feature j, the model selects $d^* = \text{argmax}_d V_j(d)$ and calculate the largest $V_j(d^*)$. According to this calculation, the decision tree will split the feature j^* at point d^* into left and right child nodes.

If we apply this calculation to GOSS, and let A denote the instance subset of top $a \times 100\%$ instances with larger gradient , and for the remaining set (complement of set A) $A^c = (1-a) \times 100\%$, let B denote the randomly selected subset with size $b\times$ length(A^c). The the variance gain of splitting feature j at point d for this node is (Ke et al. (2017)):

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n^j_l(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n^j_r(d)}\right) \tag{20}$$

where $A_l = \{x_i \in A : x_{ij} \leq d\}, A_r = \{x + i \in A : x_{ij} > d\}, B_l = \{x_i \in B : x_{ij} \leq d\}, B_r = \{x_i \in B : x_{ij} > d\}$. We can prove that GOSS will only lose only a little accuracy and outperform random sampling. The gap $\epsilon(d)$ between the variance gain calculated by GOSS and the variance gain for the entire training data is bounded by the following term

Theorem 2.6.1 If we have a training dataset O, let $\epsilon(d) = |\tilde{V}_j(d) - V_j(d)|$, and with probability at least $1-\delta$,

$$\epsilon(d) \leq C^2_{a,b} \ln 1/\delta \cdot \max\left\{\frac{1}{n^j_l(d)}, \frac{1}{n^j_r(d)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{n}} < \infty, \tag{21}$$

where $C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in A^c} |g_i|$ and $D = \max\left(\frac{\sum_{x_i \in l|O} |g_i|}{n^j_l(d)}, \frac{\sum_{x_i \in r|O} |g_i|}{n^j_r(d)}\right)$

2.7.3 Exclusive Feature Bundling

A sparse feature space usually happens in high-dimensional data space. The sparsity can provide us a safe way to reduce the number of features, especially the some features are mutually exclusive, i.e., they never take non zero values simultaneously. The method EFB (Exclusive Feature Bundling) is a safe algorithm to bundle the exclusive features into a single feature without losing much information. There are tow problem we need to solve before we introduce the EFB algorithm. 1. Which features should be bundled together. 2. How to construct the bundle.

Theorem 2.6.2 The problem of partitioning features into a smallest number of exclusive bundles is NP-hard.

The first problem can be reduced from the graph coloring problem which is NP-hard. Therefore, it's impossible to find an exact solution in polynomial time, however we can find a good approximation algorithm to identify the features to be bundled. Next, the histogram-based

algorithm can help to store the exclusive features into different bins. So in this way, we can reasonably construct the feature bundles.

EFB algorithm can bundle many sparse features into fewer dense features which is a nearly lossless approach and can avoid unnecessary computation for zero feature values.

2.8 CatBoost

CatBoost standing for "Category" and "Boosting", is an open-sourced machine learning algorithm created by Yandex. There are two critical algorithm advances in CatBoost (Prokhorenkova et al. (2017)):
1. the implementation of ordered boosting, a permutation-driven alternative to the classic algorithm.
2. Innovative algorithm for processing categorical features.
Both techniques are used to solve prediction shift caused by target leakage. Target leakage is the creation by information in the training set while not in the prediction period, hence it allows the model to make unrealistic good predictions.
Suppose that F is the prediction model obtained by boosting algorithm which relies on the targets of all training examples. It can be proved that there is a shift between the distribution of $F(x_k)|x_k$ for a training example x_k and that of $F(x)|x$ for a test example x. A similar issue also occurs in the standard algorithms when deal with categorical features. The CatBoost uses a modified boosting algorithm called ordered boosting which can avoid target leakage. After the categorical features encoding, CatBoost with the same process as all standard gradient boosting implementations, builds each new tree to approximate the gradients of the current model.

2.8.1 Categorical Features Encoding

Categorical feature is a feature which has discrete levels and these levels are usually not comparable with each other. For the multi-level features, they cannot be used in binary decision trees directly. In the data preprocessing step, a common practice is to convert categorical features to numerical values. The one-hot encoding is to create binary variables identifying categories, but it requires large memory and computational cost. Therefore, the work done by Prokhorenkova et al. (2017) proposed a more efficient way *target-based statistics* that maps a categorical feature to one numerical feature.

Target-based statistics (TBS)
Suppose that given the ith categorical feature x^i, for the kth training sample, we use the re-

sponse variable y_k to calculate some numerical value $\hat{x}_k^i$ to replace x_k^i. Usually $\hat{x}_k^i \approx \mathbf{E}(y|x^i = x_k^i)$. which is an estimate of the conditional expectation of y given the category. We desire two propositions:
1. $\mathbf{E}(\hat{x}^i|y) = \mathbf{E}(\hat{x}_k^i|y_k)$, 2. It is desirable for $\hat{x}_k^i$ to have a low variance.

There are several ways to calculate TBS:
1. Greedy TBS: The straightforward approach to estimate the target value.

$$\hat{x}_k^i = \frac{\sum_{x_j \in \mathcal{D}_k} 1_{\{x_j^i = x_k^i\}} y_j}{\sum_{x_j \in \mathcal{D}_k} 1_{\{x_j^i = x_k^i\}}}$$

where $\mathcal{D}_k \in \{x_1, \ldots, x_n\}$ is the set containing the samples with level equivalent to x_k^i. However, there is a conditional shift, that is, the distribution of $\hat{x}^i|y$ differs for training and test sets.

2. Greedy TBS with Prior

$$\hat{x}_k^i = \frac{\sum_{x_j \in \mathcal{D}_k} 1_{\{x_j^i = x_k^i\}} y_j + aP}{\sum_{x_j \in \mathcal{D}_k} 1_{\{x_j^i = x_k^i\}} + a}$$

$a > 0$ is a parameter. The prior added is to reduce the noise from the low-frequency categories. P can be calculated as the average of the label values in $\mathcal{D}_k$.

3. Holdout TBS
Randomly partition the dataset $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_\infty$, use $\mathcal{D}_0$ to calculate TBS and $\mathcal{D}_1$ to perform training. This holdout TBS can hold proposition 1, however this TBS may violates proposition 2 due to the significantly reduction in data set for training.

4. Leave-one-out TBS
The leave-one-out data set $\mathcal{D}_k = \mathcal{D} \setminus x_k$ is used to train the model and $\mathcal{D}_k = \mathcal{D}$ is the test set.

$$\hat{x}_k^i = \frac{\sum_{x_j \in \mathcal{D} \setminus x_k} 1_{\{x_j^i = x_k^i\}} y_j + aP}{\sum_{x_j \in \mathcal{D} \setminus x_k} 1_{\{x_j^i = x_k^i\}} + a}$$

5. Ordered TBS
First perform a permutation σ of the dataset, then

$$\hat{x}_k^i = \frac{\sum_{x_j : \sigma(j) < \sigma(k)} 1_{\{x_j^i = x_k^i\}} \cdot y_j + aP}{\sum_{x_j : \sigma(j) < \sigma(k)} 1_{\{x_j^i = x_k^i\}} + a} \tag{22}$$

The ordered TBS satisfies the proposition 1.

2.8.2 Prediction Shift in Boosting Algorithms

The target function in GBM: $F_m = F_{m-1} + vh_m$, the gradient step h_m is chosen to be $\text{argmin}_{h\in H}\mathbf{E}[L(-g_m(x,y),h(x))]$ where $g_m(x,y) := \frac{\partial L(y,F)}{\partial F}\Big|_{F=F_{m-1}}$

1. The conditional distribution of $g_m(x_k,y_k)|x_k$ is shifted from the distribution of $g(x,y)|x$ on test set;
2. The base predictor $h_m = \text{argmin}_{h\in H}\frac{1}{n}\sum_{k=1}^{n} L(-g_m(x_k,y_k),h(x_k))$ is biased with respect to $\text{argmin}_{h\in H}\mathbf{E}[L(-g_m(x,y),h(x))]$;
3. This bias affects the ability of F_m to predict.

2.8.3 Ordered Boosting Implementation

The CatBoost (Prokhorenkova et al. (2017)) propose a boosting algorithm which does not suffer from the prediction shift problem. The base predictor $h_m(x)$ is to approximate the residual function $r_{m-1}(x,y)$.

Algorithm 6 Algorithm for Ordered Boosting

Data: $\{\mathrm{x}_k, y_k\}_{k=1}^{n}, I$
Result: M_n
Initialization, $M_i \leftarrow 0$ for $i = 1,\ldots,n$
for *j =1 to I* **do**
  for *i=1 to n* **do**
    $\mathrm{r}_i = y_i - M_{\sigma(i)-1}(i)$
    for *i =1 to n* **do**
      $M \leftarrow LearnModel\{(x_j,r_j) : \sigma(j) \le i\} \quad M_i \leftarrow M_i + M$
    end
  end
end

2.8.4 Similarities of Parameters

The XGBoost cannot handle categorical features while only accepts numerical features like random forest. Therefore, the preprocessing step, i.e. one-hot encoding, is necessary for XGBoost.

Table 2: Parameters in Boosting Models

Characteristics	***XGBoost***	***CatBoost***	***Light GBM***
Parameters that control overfitting	1. learning rate, 2. max depth of tree, 3. weight of child leaf	1. learning rate, 2. depth of tree, 3. L2 regulation for leaf value calculation	1. learning rate, 2. max depth of tree (number of leaves–since Light GBM is leaf-wise), 3. L2 regulation for leaf value calculation
Parameters for categorical features	Not available	1. cat_feature: specify the index of categorical features, 2. one_hot_max_size: use one-hot encoding for categorical features with levels less than 255.	categorical_feature: specify the categorical features
Parameters that control speed	1. colsample_bytree: subsample ratio of columns, 2. sample: subsample ratio of training set, 3. n_estimators: max number of trees, large number leads to overfitting	1. rsm: random subspace method. The percentage of featuers used to split, 2. iterations: max number of trees, large number leads to overfitting	1. feature_fraction: percentage of features used in each iteration, 2. bagging_fraction: data to be used in each iteration and speed up the training and avoid overfitting, 3. num_iterations: boosting iterations.

2.9 Random Forest Regressor & Extra Trees Regressor

2.9.1 Random Forest Algorithm

Random forest is an ensemble method which fits decision trees based on the samples randomly chosen with replacement (i.e. bootstrap sampling) from of the training data. The difference between a decision tree and a random forest is that: The features chosen to be split is no longer the best split among all the features, the split in the random forest is chosen among a random subset of the features. An underlying assumption of the ensemble process is that the base learners are independent. The generalization error tends to increase as the trees become more correlated. Due to the randomness (which reduces the correlation), the variance of the estimator decreases even though the bias increases (Hastie et al. (2009)).

Algorithm 7 Pseudo Code for Random Forest

Output: The ensemble trees $\{T_b\}_1^B$

for *b = 1 to B* **do**

1. draw a bootstrap sample Z^* from training dataset with size N;
2. Fit a tree T_b with Z^* and repeat the following steps until the minimum node size is reached:

i). Select m features from the entire p features;

ii). Pick the best split among the m features.

iii). Split the node into two child nodes.

end

If a new data x is given, the prediction is evaluated:
regression: $\hat{f}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$;
classification: $\hat{C}^B(x)$ = majority vote in $\{\hat{C}_b(x)\}_1^B$ where $\hat{C}_b(x)$ is the classification result from the bth tree.

2.9.2 Analysis of Random Forest

Random Forest algorithm is very easy to measure the importance of each feature. By calculating how much a certain feature used in the tree nodes reduces impurity across all trees in the forest we can measure and rank the feature importance, furthermore, we can decide to drop some features with less importance.

Due to randomness, Random Forests are more robust to outliers and noise than AdaBoost. Random Forest also prevents overfitting which usually happens in decision tree, by creating random subsets of the features and ensemble small trees. However, this usually makes computation slower with large number of trees built in the random forest.

Out-of-sample: For each sample $(x_i, y_i), i = 1, \ldots, N$, construct the predictor by averaging only those trees built from the bootstrap samples in which (x_i, y_i) didn't appear. A very important feature of Random Trees is that it uses out-of-bag samples. An OOB error estimate is almost the same with that obtained by N-fold cross validation. And once the OOB error stabilizes, the training can be terminated.

2.9.3 Extremely Randomized Trees

The Extra Trees (Geurts et al. (2006)) (built in context of numerical features) is similar to random Forest except that the cut-point of splits for each variable is also fully at random

and that it learns to grow trees based on the whole samples (instead of a bootstrap sampling). This decreases the variance of the model but potentially increases its bias. Another advantage of the Extra Trees model is that it is computationally very efficient because the splits are random thus we don't have to sort the data to find the split.

In the Extra Trees algorithm (Geurts et al. (2006)), there are two parameters: K, the number of attributes randomly selected at each node and s_{min}, the minimum sample size for splitting a node. The score measure for classification and regression problem are different. Similar

Algorithm 8 The Pseudo Code for Extra Tress Splitting (for a numerical feature)

Data: A subset S and K features $\{x_1, \ldots, x_K\}$

if $|S| < s_{min}$ **then**
| Stop, return nothing

end

if *All attributes or the outputs in S are constant* **then**
| Stop, return nothing

end

Pick_a_random_split (S, x):
Inputs:A subset S and an attribute x
Let x^S_{max} and x^S_{min} be the maximum and minimum values of x in S
Pick a random cut-point x_c uniformly in $[x^S_{max}, x^S_{min}]$
Return: the split $[x < x_c]$

Split_a_node(S):
Select K attributes $\{x_1, \ldots, x_K\}$ among all non constant attributes in S
Draw K splits $\{s_1, \ldots, s_K\}$ where $s_k =$ **Pick_a_random_split**$(S, x_k), k = 1, \ldots, K$
Return: A split s^* s.t. $Score(s^*, S) = max_{k=1,\ldots,K} Score(s_k, S)$.

with Random Forest, we denote M to be the number of trees in the Extra-Trees ensemble process. The parameters K, s_{min}, M can be estimated in a manual or an automatic way (cross validation) and they have different effect in the ensemble process:

- K determines the strength of the feature selection process
- s_{min} determines the strength of averaging output noise
- M determines the strength of variance reduction of the ensemble model aggregation.

2.9.4 Parameters in Model

- *estimator*: 'Random Forest Regressor' or 'ExtraTrees Regressor'

- *n_estimator*: The number of trees in the forest.
- *criterion*: Tree-specific criterion: MSE.
- *max_depth*: The maximum depth of the tree.
- *bootstrap*: Whether bootstrap samples are used when building trees.

Key words: Less variance but bias increases, better for larger dataset compare to Gradient Boosted model.

2.10 RuleFit Regressor

The standard linear model usually doesn't consider the interaction between the features. The RuleFit model fits a sparse linear model with original features and with the decision rules as the new features. The RuleFit models are built with two components: the linear combinations of simple rules derived from the decision tree and a sparse linear model (Friedman et al. (2008)). Any algorithm that can create a lot of trees can be used in RuleFit.

2.10.1 RuleFit Ensembles

Decision Rule Generaion
Let S_j be the set of all possible values of x_j, and s_{jm} be a specific subset of S_j. The base learner is defined as

$$r_m(x) = \prod_{j=1}^{N} I(x_j \in s_{jm})$$

where $I(\cdot)$ is an indicator function thus r_m takes value of $\{0, 1\}$. The Ensemble Algorithm 2.4 produces the trees $\{f_m(x)\}_1^M$ and the rule ensemble $\{r_k(x)\}_1^K$ is constituted by these trees. The total number of rules is

$$K = \sum_{m=1}^{M} 2(t_m - 1)$$

where t_m is the number of terminal nodes for the mth tree, and $t_m - 1$ is the number of internal nodes, so the number of rules $K = \sum_{m=1}^{M} 2(t_m - 1)$ for M trees. The predictive model can be written as

$$F(\mathbf{x}) = \hat{c}_0 + \sum_{k=1}^{K} \hat{c}_k r_k(\mathbf{x})$$

with

$$\{\hat{c}_k\}_0^K = \text{argmin}_{\{c_k\}_0^K} \sum_{i=1}^{N} L\left(y_i, c_0 + \sum_{k=1}^{K} c_k r_k(x_i)\right) + \lambda \sum_{k=1}^{K} |c_k|.$$

Sparse Linear Model

First we do the data preprocessing by winsorising the original features in order to improve the robustness against outliers

$$l_j^*(x_j) = \min(\delta_j^+, \max(\delta_j^-, x_j))$$

where δ_j^+ and δ_j^- are the $1-\beta$ and β quantiles of the data distribution $\{x_{ij}\}_{i=1}^N$ for each variable x_j. A rule of thumb choice for β is 0.025. The normalization is necessary to make all the features have the same prior influence

$$l_j(x_j) = \frac{0.4 \cdot l_j^*(x_j)}{std(l_j^*(x_j))}$$

The final predictive model combining the two components is

$$F(\mathbf{x}) = \hat{c}_0 + \sum_{k=1}^{K} \hat{c}_k r_k(x) + \sum_{j=1}^{p} \hat{b}_j l_j(x_j) \tag{23}$$

with

$$(\{\hat{c}_k,\}_0^K, \{\hat{b}_j\}_j^p) = \text{argmin}_{\{c_k\}_0^K, \{\hat{b}_j\}_j^p} \sum_{i=1}^{N} L\left(y_i, c_0 + \sum_{k=1}^{K} c_k r_k(x_i) + \sum_{j=1}^{p} b_j l_j(x_j)\right) + \lambda(\sum_{k=1}^{K} |c_k| + \sum_{j=1}^{p} |b_j|). \tag{24}$$

2.10.2 Feature Importance of RuleFit

The feature importance measurements in RuleFit includes all the original features and all the decision rules, however, due to the Lasso penalty used in estimating the coefficients, not all the features will have non-zero weights.

2.10.3 Advantages and Disadvantages

Advantages:

- The RuleFit adds features interactions automatically to the linear model to solve the problem of adding the interactions manually in the linear models.
- The RuleFit can be used in both regression and classification.
- It's easy to interpret the results obtained in the RuleFit models.

- The model can calculate the feature importance for each feature which improves the interpretability.

Disadvantages:

- The RuleFit model ultimately is a linear model, the weght interpretation sometimes is unintuitive. It gets tricky if we have overlapping rules.

2.11 Generalized Additive 2 Model

2.11.1 Standard Additive Model

The additive model can be written as

$$Y = \alpha + \sum_j f_j(X_j) + \epsilon$$

By Hastie et al. (2009), the penalized sum of squares has the form

$$\text{PRSS}(\alpha, f_1, f_2, \dots) = \sum_{i=1}^{N} \left(y_i - \alpha - \sum_j f_j(x_{ij}) \right)^2 + \sum_j \lambda_j \int f_j''(t_j)^2 dt_j \tag{25}$$

An iterative procedure can help find the solution to equation 25 which is known as "Backfitting" and is also one of the two popular methods (another one is Boosting method) to fit GAMs: We assign the smoother, $\mathcal{S}_j$, to $y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_i^N$ in each iteration. A smoother

Algorithm 9 The Backfitting Algorithm for Additive Models.

1. Initialize: $\hat{\alpha} = \frac{1}{N} \sum_i^N y_i$, $\hat{f}_j \equiv 0, \forall i, j$.

2. Cycle: $j = 1, 2, \dots, p, \dots, 1, 2, \dots, p, \dots,$
$\hat{f}_j \leftarrow \mathcal{S}_j \left[\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_i^N \right],$
until the functions $\hat{f}_j$ change less than a prespecified threshold.

is used to summarize the trend of response variable and the nature of it is non-parametric. So it doesn't assume the rigid form of Y regressing on X and it allows a summation of some different functions instead of using only one unknown function. The minimizer of equation 25 is a cubic spline smoother, which can optimize the lease square while has the second continuous derivative. The Generalized Additive Model is of the following form

$$g(\mathbf{E}(y|X_1, \dots, X_p)) = \alpha + \sum_j^p f_j(x_j)$$

where g is the link function and $f_j's$ are unspecified smooth functions. For example, the additive logistic regression model is

$$\log\left(\frac{\pi(X)}{1-\pi(X)}\right) = \alpha + \sum_{j}^{p} f_j(X_j).$$

Once splines are used to fit the additive model, the GAMs reduces to fit GLMs with different basis functions to maximizing log-likelihood via IRLS (iteratively reweighted least squares) algorithm (Hastie et al. (2009)).

2.11.2 GAMs with Pairwise Interaction (Intelligible Models)

The standard GAMs is significantly less accurate in prediction compared to other models with interactions of predictors. So the work done by Lou et al. (2012) proposes the GA^2M -model implemented by the *FAST* algorithm, which includes the univariate terms and some pairwise interaction terms. Meanwhile, this paper also shows that this intelligible models have almost the same performance as the best full-complexity models for classification and regression. The *FAST* algorithm efficiently searches for all possible (c_i, c_j) which are the cut for predictor variables x_i, x_j, respectively. The two lines $x_i = c_i, x_j = c_j$ parallel to axis x_i and x_j thus divide the plane into four quadrants. We can construct the interaction predictor T_{ij} by taking the mean of the all points in each quadrant. The "FAST" algorithm then picks the best T_{ij} with lowest RSS. To introduce this algorithm, we need to define some notations first.

Let $d(x_i) = \{v_i^1, \ldots, v_i^{d_i}\}$ be a sorted set of possible values (or domain) of x_i and $d_i = |d(x_i)|$ which is the length (or count) of $d(x_i)$. Define $H_i^t(v)$ as the sum of targets when $x_i = v$ and $H_i^w(v)$ as the sum of weights (or count) when $x_i = v$. $H(\cdot)$ can be any predictive function. Also we define $CH_i^t(v) = \sum_{u \le v} H_i^t(u)$ and $CH_i^w(v) = \sum_{u \le v} H_i^w(u)$ as the cumulative histogram for the sum of targets and weights. Accordingly, $\overline{CH}_i^t(v) = \sum_{u > v} H_i^t(u) = CH_i^t(v^{d_i}) - CH_i^t(v)$ and $\overline{CH}_i^w(v) = \sum_{u > v} H_i^w(u) = CH_i^w(v^{d_i}) - CH_i^w(v)$, and $H_{ij}^t(u, v)$, $H_{ij}^w(u, v)$ are defined as the sum of targets and weights, respectively, when $(x_i, x_j) = (u, v)$. Figure 5 is an example showing how to compute sum of the targets in each quadrant. So a, b, c and d represent the cumulative sum of targets in four quadrants formed by c_i and c_j where a given by any cut (c_i, c_j) can be evaluated efficiently using dynamic programming. Then we store the values (a, b, c, d) into the lookup tables $T^t(c_i, c_j)$ and $T^w(c_i, c_j)$. The following algorithm shows how to construct $T^t(c_i, c_j)$.
Based on the algorithm, we can construct T_{ij} once we obtain T^t, T^w, i.e. for the leftmost region a, the values in $T_{ij}.a = \frac{T^t(c_i,c_j).a}{T^w(c_i,c_j).a}$. The calculation for RSS based on T_{ij} is very efficient.

Algorithm 10 Algorithm for Lookup Table Construction

Input: $d(x_i), d(x_j)$

Initial Value: $\Sigma \leftarrow 0$

for *k=1 to* d_j **do**

$\Sigma \leftarrow \Sigma + H_{ij}^t(v_i^1, v_j^k)$

a[1, k] $\leftarrow \Sigma$

$\mathrm{T}^t(v_i^1, v_j^k) \leftarrow ComputeValues(CH_i^t(v_i^1), CH_j^t(v_j^k), a[1,k])$

end

for *l=2 to* d_i **do**

$\Sigma \leftarrow 0$

for *k=1 to* d_j **do**

$\Sigma \leftarrow \Sigma + H_{ij}^t(v_i^l, v_j^k)$

$a[l,k] = \Sigma + a[l-1,k]$

$T^t(v_i^l, v_j^k) \leftarrow ComputeValues(CH_i^t(v_i^l), CH_j^t(v_j^k), a[l,k])$

end

end

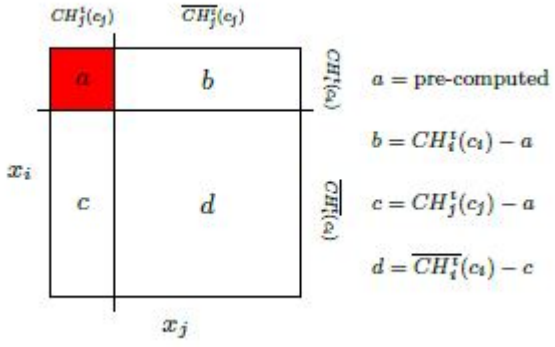

Figure 5: Illustration for computing the targets in each quadrant for given cut values.

Let $T_{ij}(x_k).l$ denote the prediction value on region l where $l \in \{a, b, c, d\}$.

$$\begin{aligned} RSS &= \sum_{k=1}^{N} (y_i - T_{ij}(x_k))^2 \\ &= \sum_{k=1}^{N} y_k^2 - 2 \sum_{l} T_{ij}.l \cdot (T_{ij}.l \; T^w.l) + \sum_{l} (T_{ij}.l)^2 T^w.l \\ &= \sum_{k=1}^{N} y_k^2 - 2 \sum_{l} T_{ij}.l \cdot T^t.l + \sum_{l} (T_{ij}.l)^2 T^w.l \end{aligned}$$

We are only interested in $\sum_l (T_{ij}.l)^2 T^w.l - 2 \sum_l T_{ij}.l \cdot T^t.l$ for each pair of (x_i, x_j) to rank RSS then to pick up the best T_{ij}.

2.11.3 Two-Stage Construction

- In stage 1, construct the best one dimensional additive model (backfitting or gradient boosting).
- In stage 2, fix the one-dimensional functions, construct the pairwise interaction terms based on residuals.

In stage 2, if we use regression tree to shape pairwise function would be problematic

2.11.4 Parameters in Model

- *pairwise_interactions*: indicates with or without interaction.
- *topk_interactions*: how many pairwise interactions.

Key words: mixed type of data and pairwise interactions.

2.12 Elastic Net Regressor with predefined α

The elastic net aims to minimize the following function (usually under squared loss function):

$$\min_{\beta_0, \beta \in \mathcal{R}^{p+1}} \left[\frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 + \lambda P_\alpha(\beta) \right]$$

where

$$P_\alpha(\beta) = (1 - \alpha) \frac{1}{2} \|\beta\|_{l_2}^2 + \alpha \|\beta\|_{l_1} = \sum_{j=1}^{p} \left[\frac{1}{2} (1 - \alpha) \beta_j^2 + \alpha |\beta_j| \right]$$

is the elastic-net penalty (Zou and Hastie (2005)). P_α is a compromise between the ridge regression penalty ($\alpha = 0$) and the lasso penalty ($\alpha = 1$).

2.12.1 Linear Regression

The form of linear regression is:

$$\hat{y} = \mathbf{X}\beta$$

By using least square method, the estimation $\hat{\beta}$ for β is: $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$.

The assumptions for linear regression:

- Linear: The model expression is linear.
- Independence: Predictor variables are independent, otherwise the multicollinearity will cause the estimates for coefficient inaccurate and infinitely large.
- Normality: Response variable is normally distributed. Errors: The errors are identically normally distributed.

Sometimes the assumptions are difficult to be satisfied, and the violations of the assumptions will lead to very bad results. For example, highly correlated predictor variables can make coefficients estimates strangely large and bias.

2.12.2 Ridge Regression

The least square estimate of β is obtained by minimizing

$$RSS(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta$$

, then

$$\hat{\beta}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

The ridge regression will shrink the coefficients of the correlated predictors towards each other (Friedman et al. (2010)). With suitable prior distribution, the ridge regression can also be derived as the mean or mode of a posterior distribution.

2.12.3 Lasso Regression

The Lagrangian form of lasso problem is

$$\hat{\beta}^{lasso} = \text{argmin}_\beta \left[(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda \sum_{j=1}^{p} |\beta_j| \right]$$

The lasso (Tibshirani (1996)) is a popular method in regression which uses an L_1 penalty to achieve a sparse solution. And lasso tends to pick one predictor and ignore the rest if the variables are high correlated.

2.12.4 Extension to Tweedie Loss

If we apply tweedie loss function derived in equation (12) to elastic net regressor, then our goal is to minimize

$$\text{argmin}_\beta \sum_{i=1}^{n} \Psi(y_i, \mathbf{x}_i'\beta)|\rho) + \lambda P_\alpha(\beta)$$

where

$$\Psi(y_i, \mathbf{x}_i'\beta))|\rho) = \omega_i \Bigg\{ - \frac{y_i \exp[(1-\rho)\mathbf{x}_i'\beta)]}{1-\rho} + \frac{\exp[(2-\rho)\mathbf{x}_i'\beta)]}{2-\rho} \Bigg\}$$

We fit the model with Lasso regression if $\alpha = 1$ and Ridge regression if $\alpha = 0$. In the Elastic Net Regression used in DataRobot, α is set to be 0.5 which provides a compromise between lasso and ridge methods. The weight $\alpha = 0.5$ gives average values of coefficients based on the algorithm of Ridge regression and eliminate some coefficients of the correlated predictor variables. This penalty is useful in the $p >> N$ situation and other situation with multicollinearity problem.

2.12.5 R and Python

For R, the package "elasticnet" provides functions for fitting the entire solution path of the Elastic-Net. For Python, please check `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html`

Key words: Solve multicollinearity problem, and $p >> n$ (the number of features much greater than the number of observations).

2.13 Frequency-Severity ElasticNet

Frequency-Severity Model is a two-stage modelling which is based on Elastic Net regression. A Poisson regression is first to trained to predict the event count and then a gamma regression is trained to predict the severity of the event and using the event count as weight. The Elastic nets are created to create frequency and severity, separately. The final predictions are the product of the 2 regressions predictions.

2.13.1 Parameters in Model

- *FREQ_enet_alpha*: For the frequency part, $0 \leq \alpha \leq 1$, $\alpha = 1 \Rightarrow L_1$ penalty, $\alpha = 0 \Rightarrow L_2$ penalty.
- *FREQ_enet_lambda*: The weight fo the penalty term.
- *SEV_enet_alpha*: For the severity part, same with *FREQ_enet_alpha*.

Key words Frequency + Severity two-stage model. Poisson, Gamma, Tweedie, Multi-collinearity

2.14 Generalized Linear Models

The GLMs permits more general distributions than Normal for the response Y and a link function is used to relate the linear model to the mean of the response variable through a suitable scale. The common distributions includes Bernoulli, Binomial, Poisson, Negative Binomial, Multinomial, Gamma, Beta, etc, which all belong to exponential family.

If the distribution of Y is in the exponential family then its pdf (pmf) is of the form

$$f_Y(y|\theta,\phi) = \exp\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\} \tag{26}$$

where both θ and $\phi > 0$ are scalar parameters and a, b, c are know functions. ϕ is called the dispersion parameter. The functions a, b, c determine a particular family in the class, such as Bernuoulli, Binomial, Poisson.

We know that

$$\int f(y;\theta)dy = 1$$

Differentiating both sides w.r.t θ, and by Fubini's Theorem, we can switch the order of the integration and differentiation, then

$$\begin{aligned}
\frac{d}{d\theta}\int f(y;\theta)dy &= 0 \\
\int \frac{d}{d\theta} f(y;\theta)dy &= 0 \\
\int \frac{d^2}{d\theta^2} f(y;\theta)dy &= 0
\end{aligned}$$

For the exponential family,

$$\begin{aligned}
\frac{d}{d\theta}f(y;\theta) &= \frac{y - b'(\theta)}{a} f(y;\theta) \\
\int \frac{d}{d\theta}f(y;\theta) &= \int \frac{y - b'(\theta)}{a} f(y;\theta) \\
0 &= \int y f(y;\theta)dy - b'(\theta)\int f(y;\theta) \\
\mu = \mathbf{E}[Y] &= b'(\theta)
\end{aligned}$$

Similarly, $\mathbf{Var}(Y) = b''(\theta)a(\phi)$.

The mean of response variable $\mu = \mathbf{E}(Y)$ is related to a linear function of $X_1, \ldots, X_k$ by a link function g, that is

$$g(\mu) = g(\mathbf{E}(Y)) = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k = \mathbf{X}'\beta$$

The most commonly used link function is the canonical link, that is, $\theta = \mathbf{X}'\beta$.

Parameter Estimate

There are two commonly used methods: Maximum Likelihood and Bayesian method.
The MLE can be obtained by Newton-Raphson method or Fisher Scoring method by using the score function and information matrix derived from the log likelihood functions.
For Bayesian method, usually the closed form of posterior distribution cannot be explicitly found and so must be approximated, often using Laplace approximations or MCMC such as Gibbs sampling.

Residuals and Model Checking

A general approach to assess the adequacy of a model is to compare the observed response to those predicted by the model. There are two ways to estimate the discrepancy between them. First one is Pearson chi-squared residual which measure discrepancy on a scale proportional to the standard deviation. Another measure is Deviance residuals which is based on the likelihood function under the assumed model and a saturated model.

The ith Pearson residual is defined as

$$r_i = \frac{y_i - \mathbf{E}(Y_i)}{\sqrt{\mathbf{Var}(Y_i)}}$$

Then the Pearson chi-squared statistic is given by

$$X^2 = \sum_{i=1}^{n} r_i^2 \overset{asympt}{\sim} \chi^2_{n-k-1}$$

where k is the number of covariates.

Let $\hat{\mu}_i$ denote the ith prediction from the model. Then the ith Deviance can be written as

$$D(y_i, \hat{\mu}_i) = 2\phi[l(y_i, \phi; y_i) - l(\hat{\mu}_i, \phi; y_i)]$$

where $l(y_i, \phi; y_i)$ is the log-likelihood function. A likelihood function is a function of the parameters of a statistical model given observed data.
l_i for Poisson distribution, $y_i \sim Poisson(\lambda_i), \mu_i = \lambda_i$,

$$l_i = y_i \log \lambda_i - \lambda_i + (-\log y_i!) = y_i \log \mu_i - \mu_i + (-\log y_i!)$$

l_i for Binary/Binomial distribution, $z_i \sim Bin(m_i, p_i), y_i = z_i/m_i, \mu_i = p_i$,

$$\begin{aligned} l_i &= m_i y_i \log\left(\frac{p_i}{1-p_i}\right) - [-m_i \log(1-p_i)] + \log\binom{m_i}{m_i y_i} \\ &= m_i y_i \log\left(\frac{\mu_i}{1-\mu_i}\right) - [-m_i \log(1-\mu_i)] + \log\binom{m_i}{m_i y_i} \end{aligned}$$

Hypothesis Testing for Parameters and Nested Models
Wald z-statistic for testing $\beta_j = 0$:

$$z_j = \frac{\hat{\beta}_j}{se(\hat{\beta}_j)} \overset{asympt}{\sim} N(0,1)$$

Equivalently, $\chi_j^2 = (\frac{\hat{\beta}_j}{se(\hat{\beta}_j)})^2 \overset{asympt}{\sim} \chi_1^2$.

Suppose that we want to compare two nested models, where model M_0 is nested with M_1, and we wish to test:

$$\begin{aligned} H_0 &: M_0(\text{simple model}) \text{ is true} \\ H_1 &: M_1(\text{more complicated model}) \text{ is true} \end{aligned}$$

The likelihood ratio test statistic is $G^2 = -2\log\frac{L(H_0)}{L(H_1)}$ where $L(\cdot)$ is the maximized likelihood function under H_j for $j = 0, 1$.
Alternatively, we can use deviance obtained from M_0 and M_1 to calculate $G^2 = \frac{D_0 - D_1}{\phi}$.
The degrees of freedom for this test are the difference between the number of parameters in the two models: $q = df_0 - df_1$. Under H_0, $G^2 \sim \chi_q^2$.

2.14.1 Logisitic Model

If $Y \sim Bernoulli(p)$ where $Prob(Y = 1) = p$, then

$$\begin{aligned} f_Y(y|p) &= p^y(1-p)^{1-y}, y = 0, 1 \\ &= \exp\{y\log\left(\frac{p}{1-p}\right) - [-\log(1-p)] \end{aligned}$$

Thus, $\theta = \log\frac{p}{1-p}$, $b(\theta) = -\log(1-p) = \log(1+\exp(\theta))$, $a(\phi) = 1$ and $c(y,\phi) = 0$.

$$\begin{aligned} \mu = \mathbf{E}(Y) = b'(\theta) &= \frac{\exp(\theta)}{1+\exp(\theta)} = p \\ \mathbf{Var}(Y) = b''(\theta)a(\phi) &= \frac{\exp(\theta)}{(1+\exp(\theta))^2} = p(1-p) \end{aligned}$$

As we derived above,

$$\theta = \log \frac{p}{1-p} \overset{\text{canonical link}}{=} g = \mathbf{X}'\beta \Rightarrow p = \mathbf{E}(Y) = \frac{\exp(\mathbf{X}'\beta)}{1+\exp(\mathbf{X}'\beta)} \tag{27}$$

If the response variable $Z \sim Binomial(m, p)$, then let $Y = Z/m$ be the proportion of successes. The canonical link function is the same with equation 27.

The canonical link under binary and binomial distribution is called logit link. So the binary regression model is a GLM with Bernoulli random component and a logit link.

Binary and Binomial under Different Links

The general regression model for a binary or binomial proportion assumes $F^{-1}(p) = \mathbf{X}'\beta$, F is a cumulative distribution function. In logit link, $F(\mathbf{X}'\beta) = \frac{\exp(\mathbf{X}'\beta)}{1+\exp(\mathbf{X}'\beta)}$. There are three other popular link functions

- probit link: $F = \Phi$ which is the standard normal cdf, $F^{-1}(p) = \Phi^{-1}(p)$.
- C log-log link: $F^{-1}(p) = \log(-\log(1-p))$.
- t_v link: F is the t-distribution with degree of freedom v.

Likelihood Function and MLE for Parameters

If for n observations, $Z_i \sim Binomial(m_i, p_i)$, and $Y_i = Z_i/m_i$, where m_i is known, $i = 1, \ldots, n$. Z_i takes value from $\{0, 1, \ldots, m_i\}$. The likelihood function and log likelihood function are

$$L(\beta) = \prod_{i=1}^{n} \binom{m_i}{m_i y_i} p_i^{m_i y_i} (1-p_i)^{m_i - m_i y_i} = \prod_{i=1}^{n} \binom{m_i}{m_i y_i} \frac{\exp(m_i y_i x_i'\beta)}{[1+\exp(x_i'\beta)]^{m_i}}$$

$$l(\beta) = \sum_{i=1}^{n} \{m_i\{y_i x_i'\beta - \log[1+\exp(x_i'\beta)]\} + \log \binom{m_i}{m_i y_i}\}$$

The Newton-Raphson algorithm and Fisher-scoring algorithm with $\frac{\partial l(\beta)}{\partial \beta}$ and $\frac{\partial^2 l(\beta)}{\partial \beta^2}$ we can iteratively approximate the parameter β. Until the iterative estimator converges, i.e. β^{t+1} is sufficiently close to β^t, we can use this estimates as MLE of true parameter β.

Predict New Observations

Let $\hat{p}_h$ be the fitted probability given by the new data x_h and

$$\hat{p}_h = \frac{\exp\left(x_h'\hat{\beta}\right)}{1+\exp\left(x_h'\hat{\beta}\right)}$$

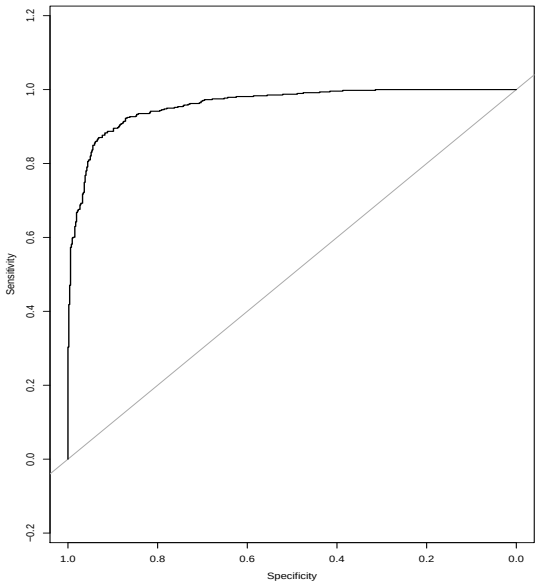

Figure 6: ROC Curve of Model

Using the value of $\hat{p}_h$, we need to determine whether $\hat{y}_h$ is predicted to be 1 (for some large values of $\hat{p}_h > p_{cutoff}$) or to be 0 ($\hat{p}_h \leq p_{cutoff}$). There are three ways to reasonably pick up cutoff points,

- Use 0.5 as cutoff: use when event and non-event are equally likely in the population.
- Select the cutoff which gives the smallest proportion of incorrect predictions.
- Use prior information.

ROC Curve

We define sensitivity as $P(\hat{Y} = 1|Y = 1)$ and specificity as $P(\hat{Y} = 1|Y = 1)$. The ROC curve plots sensitivity versus 1-specificity for several possible cutoff points between 0 and 1 (Figure 6). A popular way of summarizing the discrimination ability of a logistic model is to report the area under the ROC curve (AUC). The larger the area under the ROC, the more powerful the model is.

Logistic Model Checking

The ith Pearson residual in Logistic model is defined as

$$r_i = \frac{y_i - \mathbf{E}(Y_i)}{\sqrt{\mathbf{Var}(Y_i)}} = \frac{y_i - \hat{p}_i}{\sqrt{\hat{p}_i(1-\hat{p}_i)/m}}$$

Then the Pearson chi-squared statistic is given by

$$X^2 = \sum_{i=1}^{n} r_i^2 \overset{asympt}{\sim} \chi^2_{n-k-1}$$

where k is the number of covariates.

Let $\hat{\mu}_i$ denote the ith prediction from the model. Then the ith Deviance can be written as

$$D(y_i, \hat{\mu}_i) = 2\phi[l(y_i, \phi; y_i) - l(\hat{mu}_i, \phi; y_i)]$$

And in Logistic model, $\phi = 1$, then

$$D(y; \hat{\mu}) = 2\sum_{i=1}^{n}\{m_i[y_i \log \frac{y_i}{\hat{\mu}_i} + (1 - y_i) \log \frac{1 - y_i}{1 - \hat{\mu}_i}]\}$$

where $\hat{\mu}_i = \hat{p}_i$.
Therefore, if given two nested models, where model M_0 is nested with M_1, and we wish to test:

$$H_0 : M_0(\text{simple model}) \text{ is true}$$
$$H_1 : M_1(\text{more complicated model}) \text{ is true}$$

We have $G^2 = (D_0 - D_1)/\phi \overset{approx}{\sim} \chi^2_q$. We reject H_0 if $G^2 > \chi^2_{1-\alpha,q}$ where $q = df_0 - df_1$ at a significance level α.

2.14.2 Poisson Regression Model

Suppose that $Y \sim Poisson(\lambda)$,

$$\begin{aligned} f_Y(y|\lambda) &= \frac{\lambda^y}{y!} \exp(-\lambda) \\ &= \exp\{y \log \lambda - \lambda + (-\log y!)\} \end{aligned}$$

Thus, $\theta = \log \lambda, b(\theta) = \lambda = \exp(\theta), a(\phi) = 1, c(y, \phi) = -\log y!$. Also we have $\mu = \mathbf{E}(Y) = b'(\theta) = \lambda$ and $\mathbf{Var}(Y) = b''(\theta)a(\phi) = \exp(\theta) = \lambda$.
For Poisson rate, if $Z \sim Poisson(m\lambda)$, where λ is the rate of events per unit length. Let $Y = Z/m$ be the observed count per unit length, then

$$\begin{aligned} f_Y(y|\lambda) &= \frac{(m\lambda)^{my}}{(my)!} \exp(-m\lambda) \\ &= \exp\{m[y \log \lambda - \lambda] + \log \frac{m^{my}}{(my)!}\} \end{aligned}$$

Then $\theta = \log\lambda, a(\phi) = 1/m, \mu = \mathbf{E}(Y) = \lambda, \mathbf{Var}(Y) = \lambda/m$.

Similarly, the poisson regression with canonical link is $\theta = \log\lambda \overset{canonical}{=} \mathbf{x}'\beta \Rightarrow \lambda = \mathbf{E}(Y) = \exp(\mathbf{x}'\beta)$. The log link is the canonical link under Poisson regression.

MLE using NR / FS method

The likelihood and log-likelihood functions using the canonical link function are

$$L(\beta) = \prod_{i=1}^{n} \frac{\exp(y_i x_i'\beta)}{y_i!} \exp(-\exp(x_i'\beta))$$

$$l(\beta) = \sum_{i=1}^{n} \{y_i x_i'\beta - \exp(x_i'\beta) - \log(y_i!)$$

Using the score function derived from $\frac{\partial l(\beta)}{\partial \beta}$ and the Fisher information $\frac{\partial^2 l(\beta)}{\partial \beta^2}$, the MLE of β can be obtained using NR or FS iterative algorithm until the estimators are convergent.

Same with logistic regression model, we can use Pearson chi-squared residuals or Deviance to perform the tests for parameters and nested models. For a correctly specified Poisson regression model, the Pearson chi-squared statistic and the deviance statistic divided by their degrees of freedom, should be approximately equal to 1. When these ratios are much larger than 1, the assumption of nominal Poisson variation is violated and the data shows overdispersion.

2.14.3 Overdispersed Poisson and Negative Binomial Model

Overdispersion implies that the data shows the variation of the response is greater than it should be under the given model. The two most common solutions are: changing a poisson model to overdispersed model, or changing a poisson to negative binomial model.

Overdispersed Poisson

For Poisson regression, the dispersion parameter ϕ should be 1. However, for the overdispersion problem, we allow $\phi \neq 1$ and have $\mathbf{Var}(\mu_i) = \phi\mu_i$ for an overdispersed Poisson model.

The parameter ϕ can be estimated by either Pearson chi-squared statistic

$$\hat{\phi} = \frac{X^2(y, \hat{\mu})}{df}$$

or the Deviance chi-squared statistic

$$\hat{\phi} = \frac{D(y, \hat{\mu})}{df}$$

where $df = n - k - 1$, k is the number of parameters. The case $\phi > 1$ represents overdispersion, while $\phi < 1$ indicates underdispersion. Since the overdispersed Poisson does not truly belong to exponential family pmf, we **only use deviance test** $(D_0 - D_1)/\phi$ when testing nested models.

Negative Binomial Model

Negative binomial regression can be considered as a generalization of Poisson regression since it has the same mean structure with Poisson and it has an extra parameter to model the overdispersion. The negative binomial pmf is given by

$$f_Y(y|k,\mu) = \frac{\Gamma(y+k)}{\Gamma(k)\Gamma(y+1)}\left(\frac{k}{\mu+k}\right)^k\left(1-\frac{k}{\mu+k}\right)^y, y = 0, 1, \dots$$

where k, μ are parameter, and only when k is **known**, this pmf belongs to an exponential family. Usually k is unknown and can be estimated by maximum likelihood. The variance of negative binomial distribution is a function of mean and the dispersion parameter k.

$$\mathbf{Var}(Y) = \mu + \mu^2/k$$

As the dispersion parameter turns larger, the variance converges to the same value with the mean, and the negative binomial turns to poisson distribution.

The parameter μ is the mean incidence rate of y per unit of exposure. For each observation y_i, the mean $\mu_i = \exp(\beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi})$. Using these notations, the negative binomial regression model for the observation i can be written as:

$$\Pr(Y = y_i|\mu_i, k) = \frac{\Gamma(y_i+k)}{\Gamma(k)\Gamma(y_i+1)}\left(\frac{k}{\mu_i+k}\right)^k\left(1-\frac{k}{\mu_i+k}\right)^{y_i}$$

This equation gives the log-likelihood function as

$$\mathcal{L} = \sum_i \log \Pr(Y = y_i|\mu_i, k) \tag{28}$$

$$= \sum_i \left\{ (\sum_{j=0}^{y_i-1} \log(j+k)) - \log(\Gamma(y_i+1)) - (y_i+k)\log(1+\mu_i/k) + y_i \log\left(\frac{\mu_i}{k}\right) \right\} \tag{29}$$

Since the calculation of G^2 using deviance requires k to be fixed for both reduced and full models while this is not typically done in software, so only G^2 computed by **log-likelihood ratio** is recommended in Negative binomial, thus we have

$$\begin{aligned} G^2 &= 2\sum_i [\mathcal{L}(y_i) - \mathcal{L}(\mu_i)] \\ &= 2\sum_i \left\{ y_i \log\left(\frac{y_i}{\mu_i}\right) - (y_i+k)\log\left(\frac{1+y_i/k}{1+\mu_i/k}\right) \right\} \sim \chi^2_{n-p-1} \end{aligned}$$

where p is the number of parameters. The statistic D can be used to compare nested models.

2.15 Two-Stage(Logistic Regressor and Ridge Regressor)

The insurance data, especially life insurance, will contain most "0"s. So the first stage is to use logistic model to predict "0" and non "0". Then it uses Ridge Regressor to fit the non-zero part of the data. The advanced zero-inflated model is the Tweedie model.

2.15.1 Logistic Model

We explain in section 2.14.1

2.16 TensorFlow Neural Network Regressor

Neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

The Neural Network regressor uses least squares loss , poisson loss for count problems, tweedie loss for zero-inflated, positive distributions, and gamma loss for right skewed, positive distributions. In addition, it uses early stopping to determine the best number of iterations for the stochastic gradient descent algorithm. The training data is also split into training and test parts, and the model is scored in the test set for each iteration. The training procedure will stop until the test score begins to decrease. The default learning algorithm is Adam (adaptive moment estimation) (Le et al. (2011)) which uses stochastic gradient method.

Based on Kingma and Ba (2014), the algorithm 11 describes how the *Adam* using stochastic gradient optimization to obtain the parameters in each layer.
For example, the cost function for the neural network (with regulation) for multiclassification problem is

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}\log\big((h_\theta(x^{(i)}))_k\big) - (1-y_k^{(i)})\log\big(1-(h_\theta(x^{(i)}))_k\big)] + \frac{\lambda}{2m}[\Theta_1^2+\Theta_2^2+\cdots] \tag{30}$$

And it is important to randomly choose the initial parameters when training the models. Usually we select Θ uniformly distributed in $[-\epsilon,\epsilon]$ where $\epsilon = \frac{\sqrt{6}}{L_{in}+L_{out}}$, L_{in}, L_{out} = number of layers adjacent to Θ. The initial values with large weights usually cause poor solutions (Hastie et al. (2009)).

Algorithm 11 pseudo-algorithm for Adam

Require:

α: Stepsize

$\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

θ_0: Initial parameter vector

$m_0 \leftarrow 0$, initial 1st moment vector

$v_0 \leftarrow 0$, initial 2nd moment vector

$t \leftarrow 0$, initial time step

while θ_t *not converged* **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Stochastic gradient at timestep t)

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update first moment based on g_t)

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update second raw moment based on g_t)

$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute the bias-corrected first moment estimate)

$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute the bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ (Update the parameter)

end

return θ_t

2.16.1 Parameters in Model

- *n_hidden_units*: Number of units present in each layer.
- *W_regularizer*: Value of the L_2 penalty applied to weights.
- *activation*: Non-linearity to use after each layer.

  sigmoid: logistic sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$.

  tanh: the hyperbolic tan function $f(x) = \tanh(x)$.

  relu: $f(x) = \max(0, x)$ which is also the default select.
- *n_layers*: Number of layers in the model.

In the R package, the models include:

- keras: A high-level interface for neural network;
- tfestimators: implementations of common model types such as regressors and classifiers
- tensorflow: low-level interface to tensorflow computational graph
- tfdatasets: scalable input pipelines for tensorflow models.

Key words:

2.17 Vowpal Wabbit Regressor

The default learning algorithm is a variant of online gradient descent. The main difference from vanilla online gradient descent is fast and correct handling of large importance weights. Various extensions, such as conjugate gradient (CG), mini-batch, and data-dependent learning rates, are included (Agarwal et al. (2014)).

The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).

2.18 Eureqa Generalized Additive Model

Eureqa GAM is a surrogate model that approxmates Gradient Boosted Model predictions while using Eureqa modeling engine.

3 Data Preprocessing

3.1 Missing Values

- Smooth Radit Transformation:
  The ridit for each level of the one categorical variable is the percentile rank of this level in the whole population. For example, $x_{i1}, \ldots, x_{iJ}$ denote the ordered category x_i, then $p_j = \text{Prob}(x_j)$ and the ridit score is $w_j = 0.5p_j + \sum_{k<j} p_k$.
- Search For Differences:

3.2 Categorical Variable

Categorical variables must be treated appropriately instead of being fit into a regression model in the raw form. The methods to deal with categorical variables include convert to number, combine levels and one-hot encoding.
Convert to Number

- create a new feature using the mean or the mode of each categorical bin.
- create two new features one for the lower bound of the bin and one for upper bound.

Combine levels:

- Using frequency to combine rare levels or redundant levels.
- Group the levels based on business logic, i.e, combine the states by regions.

One-hot encoding: Convert a categorical input into continuous variable. Each number represents a level of this variable.

4 Summary

5 Model Selection Criteria

1. The Gamma, Poisson, Tweedie deviance is suitable for the models with Gamma continuous, count and zero-inflated response variables.

2. Gini impurity, in the trees models, is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k} p_k = \sum_{i=1}^{J} p_i(1-p_i) = \sum_{i=1}^{J}(p_i - p_i^2) = 1 - \sum_{i=1}^{J} p_i^2$$

3. R Squared: $R^2 = \frac{SSR}{SST} = \frac{\text{sum of squared regression}}{\text{sum of squared total}}$

6 Model Assessment and Selection

Assessment of the performance of a learning method which aims at predictions is very important in selecting the learning method on model, and measures the quality of our choice of the final model.
Model Assessment: is to estimate the prediction error based on the test data (or new data) using the ultimate model.
Model Selection: is to estimate the performance of the model candidates in order to choose the best one.

The validation step can be analytically, i.e. AIC, BIC and Bayesian method, or by sample reuse, i.e. cross validation and the bootstrap. The analytical methods such as AIC, BIC are working with a special class of estimates that are linear in their parameters (Hastie et al. (2009)). While cross validation and the bootstrap can handle the non linear estimates and any loss function.

Table 3: Some Characteristics of Ensemble Models

Ensemble Models	**Ensemble Method**	**Characteristics**
Random Forest	1. Parallel ensemble of deep independent trees. 2. Randomly choose a subset of features to be split. 3. Take the average of the trees as prediction	1. Good at handling mixed type of data, missing values and large dataset, 2. Robust to outliers, 3. Good at dealing with irrelevant inputs, 4. Easy to interpret the results, 5. Poor predictive power.
Extra Trees	1. Parallel ensemble of deep independent trees. 2. Randomly choose a subset of features to be split. 3. Take the average of the trees as prediction 4. Randomly choose split nodes	1. Good at handling mixed-type of data, missing values and large dataset, 2. Robust to outliers, 3. Good at dealing with irrelevant inputs, 4. Easy to interpret the results, 5. Poor predictive power.
Gradient Boosted Model	1. Add various classes of weak learns to the ensemble sequentially, 2. Sequantial training with repsect to errors obtained from specific loss funtion, so each tree is grown using information from previous trees.	1.Good at handling mixed -type of data and missing values , 2. Lots of flexilibity on different loss functions 3. Usually has overfitting problem 4. Computationally expensive 5. Less interpretable power
Extreme Gradient Boosted Model	1. Add various classes of weak learns to the ensemble sequentially, 2. Sequantial training with repsect to errors obtained from specific loss funtion + penalty of the model complexity, so each tree is grown using information from previous trees.	1.Good at handling mixed -type of data and missing values , 2. Lots of flexilibity on different loss functions 3. Good at handling overfitting problem since XGBM has added a model complexity penalty term. 4. Computationally expensive 5. Less interpretable power
Light Gradient Boosted Model	1. Leaf-wise decision tree growth, the trees are grown Vertically, 2. Using GOSS ane EFB to process large data instances	1. Faster training speeding and higher efficiency, 2. Lower memory usage, 3. Good at handling large datasets, 4. Parallel learning supported, 5. May lead to overfitting problem when data is small
RuleFit Model	Original features from linear models and with decision rules as new features	1. Add feature interactions then to improve the accuracy and easy to interpret. 2. The weight interpretation sometimes is unintuitive.
Generalized Additive Model	1. Backfitting 2. Sequential (Gradient Boosting) Model components: One dimensional feature + significant interactions of two features	1. Easy to fit and the resutls can be interpreted by users 2. The accuracy is significantly less than the more complex models that allow interactions
Generalized Additive 2 Model	1 Same fitting method with GAM with one dimensional covariate 2. Using FAST algorithm to find significant interations	1. In large scale dataset, FAST algorithm is effective in ranking interaction of pairs of features, 2. More accurate than standard GAM and almost the same performance with complex models.

Table 4: Some Characteristics of Kernel and Regression Models

Kernel Method Models	**Model Computation**	**Characteristics**
SVM	Optimize primal and dual function which contains kernels of two features	1. Able to express linear combination of features, 2. Strong predictive power, 3. Poor at dealing with mixed-type of data, missing value and robust to outliers 4. Poor at interpreting results.
K-Nearest	Given a target point and some norm function on the feature space, rank the nearest K points.	1. Good at handling missing value, robust to outliers, 2. Strong predict power, 3. Poor at mixed-type data and large dataset 4. Poor at dealing with irrelevant inputs and interpreting results.
Regression Models	**Model Component**	**Characteristics**
Generalized Linear Model	A probability distribution from exponential family, a linear predictor and a link function.	Allow response variables to have different distributions including linear regression, logistic regression and Poisson regression.
Elastic Net Regression	The elastic net regularizes the regression method combining L_1 and L_2 penalties of the lasso and ridge methods.	1. Good at handling multicollinearity problem , 2. May lose information from the feature it "ignores" in the lasso penalty, 3. Can be applied to Principle Component Anlaysis and SVMs.
Frequency Severtiy	A Poisson regression is first to fit the event count and a Gamma regression is to fit severity.	The final predictions are the product of 2 regression predictions.

Table 5: Different Metrics

FVE Gamma

FVE Poisson

FVE Tweedie

Gamma Deviance

Gini Norm

MAE (Mean Absolute Error)

Poisson Deviance

R Squared

RMSE

RMSLE

Tweedie Deviance

6.1 Bias-Variance Decomposition and Tradeoff

Bias is the difference between the expected (or average) prediction of fitted model and the correct value which we try to predict, that is $\mathbf{E}(\hat{f}(x) - f(x))$, where $\hat{f}(x)$ is a prediction model that is estimated from a training set.
variance: is the variability of a model prediction for a given data point, that is, the amount by which $\hat{f}$ would change if we fit with a different training data set.
The bias-variance tradeoff is a central problem in supervised learning. Usually the models have lower bias in parameter estimation will have a higher variance of this estimation across samples and vice versa. The models with lower bias are usually more complex, trying to represent the training set more accurately, and the high bias models tend to be relatively simple. The application of bias-variance decomposition to regression and classification can help to regularize the model such as Lasso and Ridge regression.

The bias-variance decomposition is a way to analyze a learning algorithm's expected generalization error in the view of sum of three erro sources: irreducible error + variance + bias. The derivation of the bias-variance decomposition for squared error proceeds as follows:

$$\begin{aligned}
Error(x) &= \mathbf{E}[(y - \hat{f}(x))^2] \\
&= \mathbf{E}[y^2 + \hat{f}^2(x) - 2y\hat{f}(x)] \\
&= \mathbf{Var}(y) + \mathbf{E}[\hat{f}(x)^2] - 2\mathbf{E}[y\hat{f}(x)] \\
&= \mathbf{Var}(y) + \mathbf{E}[y]^2 + \mathbf{Var}(\hat{f}(x)) + (\mathbf{E}(\hat{f}(x)))^2 - 2f(x)\mathbf{E}(\hat{f}) \\
&= \mathbf{Var}(y) + \mathbf{Var}(\hat{f}(x)) + (f^2(x) - 2f(x)\mathbf{E}(\hat{f}(x)) + \mathbf{E}(\hat{f}(x))^2) \\
&= \sigma^2 + \mathbf{Var}(\hat{f}(x)) + [\mathbf{E}\hat{f}(x)) - f(x)]^2 \\
&= \sigma^2 + \mathbf{Var}(\hat{f}(x)) + Bias^2(\hat{f}(x))
\end{aligned}$$

where $Bias(\hat{f}(x)) = \hat{f}(x)) - f(x), \mathbf{E}[\hat{f}(x)] = f(x)$. If the data is from test dataset, then $\mathbf{E}[y_0 - \hat{f}(x_0)]$ becomes the expected test MSE, refers to the average MSE calculated from test dataset. σ^2 is the irreducible error.

6.2 AIC, Bayesian Approach and BIC

The AIC (Akaike Information Criterion) is defined as AIC$= -2 \cdot \text{loglik} + 2 \cdot d$, d is the dimension of features, and loglik is the maximized log-likelihood: loglik $= \sum_i \log \Pr_{\hat{\theta}}(y_i)$ where $\hat{\theta}$ is the maximum-likelihood estimate of θ. AIC is used in model selection, and we can simply choose the model with smallest AIC.

The BIC (Bayesian Information Criterion) is defined as BIC$= -2 \cdot \text{loglik} + (\log N) \cdot d$. BIC arises in the Bayesian approach to model selection.

What is Bayesian approach? Suppose we have a set of candidate models $\mathcal{M}_i, i = 1, \ldots, M$ with parameter θ_i. If the prior distribution of θ_i is $\Pr(\theta_i|\mathcal{M}_i)$ in model $\mathcal{M}_i$, then the posterior distribution of θ_i given data is (Hastie et al. (2009)):

$$\begin{aligned}\Pr(\mathcal{M}_i|T) &\propto \Pr(\mathcal{M}_i) \cdot \Pr(T|\mathcal{M}_i) \\ &\propto \Pr(\mathcal{M}_i) \cdot \int \Pr(T|\theta_i, \mathcal{M}_i) \Pr(\theta_i|\mathcal{M}_i) d\theta_i\end{aligned}$$

where T is the training data set. And the posterior odds is defined by:

$$\frac{\Pr(\mathcal{M}_i|T)}{\Pr(\mathcal{M}_j|T)} = \frac{\Pr(\mathcal{M}_i)}{\Pr(\mathcal{M}_j)} \cdot \frac{\Pr(T|\mathcal{M}_i)}{\Pr(T|\mathcal{M}_j)}$$

If the odds are greater than 1, which means that the posterior probability of model i is greater than model j, then we choose model i, otherwise we choose j. $BF(T) = \frac{\Pr(T|\mathcal{M}_i)}{\Pr(T|\mathcal{M}_j)}$ is called Bayes factor. If the prior distribution is uniform, then $\Pr(M_i)$ is constant. We approximate the integral $\int \Pr(T|\theta_i, \mathcal{M}_i) \Pr(\theta_i|\mathcal{M}_i) d\theta_i$ by Laplace approximation which gives us:

$$\log \Pr(T|\mathcal{M}_i) = \log \Pr\Big(T|\hat{\theta}_i, \mathcal{M}_i\Big) - \frac{d_i}{2} \cdot \log N + \mathcal{O}(1)$$

If we define the loss function to be $-2 \log \Pr\Big(T|\hat{\theta}_i, \mathcal{M}_i\Big)$, which exactly gives the BIC.

$$\begin{aligned}-2 \log \Pr(T|\mathcal{M}_i) &\propto -2 \log \Pr\Big(T|\hat{\theta}_i, \mathcal{M}_i\Big) + d_i \log N + \log \Pr(M_i) \\ &\propto -2 \log \Pr\Big(T|\hat{\theta}_i, \mathcal{M}_i\Big) + d_i \log N + \text{constant}\end{aligned}$$

So choosing the model with smallest BIC is equivalent choosing the one with largest posterior probability.

As the sample size $N \to \infty$, BIC is asymptotically consistent as a selection criterion since the probability that BIC will choose the correct model given a family of models approaches to 1 as $N \to \infty$, however, this is not the case for AIC. AIC tends to choose more complex model as $N \to \infty$ since AIC doesn't penalize the sample size N in the process. On the other hand, BIC usually choose simple models because it has a heavy penalty on complexity.

6.3 K-Fold Cross Validation

The most commonly used method to estimate the prediction error is cross validation. The $K-$fold cross validation uses part of the available data to fit the model and a different part to test it. We split the data into K roughly equal-sized parts, for the kth part ($1 \leq k \leq K$), we fit the model with the other $K-1$ parts and calculate the prediction error using the kth

part of data.

First we define a mapping $\phi : \{1, 2, \ldots, N\} \mapsto \{1, 2, \ldots, K\}$ to indicate the observation i randomly allocated into the $\phi(i)$th part. Denote $\hat{f}^{-\phi}$ be the fitted model with the kth part removed. Then the cross-validation estimate of prediction error is

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\phi(i)}(x_i))$$

We can also use cross validation for parameter tuning. For example, in Lasso regression, λ can be estimated using the $K-$fold cross validation. We consider

$$\hat{\beta}^{lasso} = \text{argmin}_{\beta} \|y - X'\beta\|^2 + \lambda|\beta|$$

Given appropriate range of λ and a loss function, λ is chosen to minimize the error $CV(\hat{f}_\lambda) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\phi(i)}\lambda(x_i))$, which is $\hat{\lambda} = \text{argmin}_{\in\{\lambda_1,\ldots\}} CV(\hat{f}_\lambda)$.

How to choose K? Usually the larger K we choose, the more unbiased the CV estimator is for the true (expected) prediction error, however, the higher variance. The cost of computation is also a problem. Usually we choose $K = 5$ or 10, more accurately, if the slope of the learning curve is flat enough at 90% of the entire dataset, then the bias can be ignored using 10 fold CV. The learning curve on a given task is a plot of 1-error versus the size of training set (as shown in Figure 7). And if the dataset is large enough, 5 fold or 10 fold will be reasonable to use.

6.4 Bootstrap Method

The bootstrap is a general tool to assess the model accuracy using extra-sample method. Suppose we have training data set $Z = \{x_i, y_i\}_{i=1}^{N}$ and randomly draw datasets with replacement from Z, each sample the same size as Z. This is done by B times which gives B bootstrap datasets, $\tilde{Z}_1, \ldots, \tilde{Z}_B$. We fit the model to each of the bootstrap datasets and check the behavior of these models for B replicates.

Let $Q(Z)$ be any quantity calculated from Z, i.e. the prediction error. The unbiased estimate of variance of $Q(Z)$ is defined as

$$\widehat{\mathbf{Var}}(Q(Z)) = \frac{1}{B-1} \sum_{b=1}^{B} (Q(\tilde{Z}_b) - \overline{Q})^2$$

where $\overline{Q}$ is the average value of $Q(Z)$ over B replicates, $\overline{Q} = \frac{1}{B} \sum_{b=1}^{B} Q(\tilde{Z}_b)$.

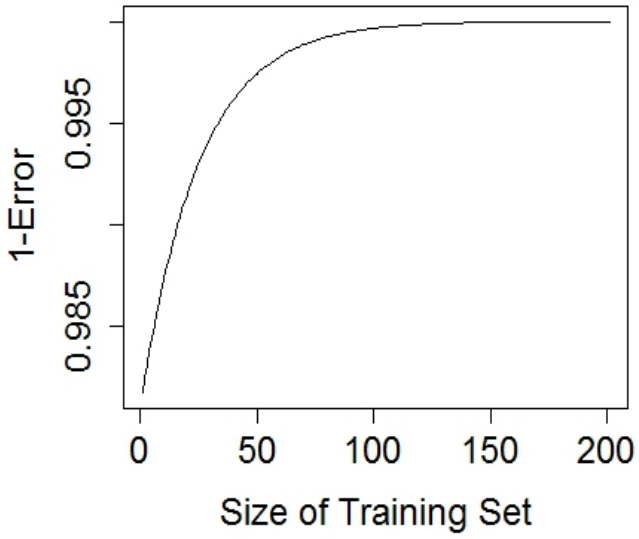

Figure 7: Learning Curve: 1-Error versus size of training set

For estimating the prediction error, to avoid overfitting (because of the overlap between the bootstrap samples and original data set), similar with cross validation, we use leave-one-out bootstrap estimates.

$$\widehat{Error} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|S_{-i}|}\sum_{b\in S_{-i}} L(y_i, \hat{f}_b(x_i))$$

where S_{-i} is the indices set such that the ith sample is not included in this set.

References

Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. (2014). A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.

Friedman, J. H., Popescu, B. E., et al. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. (2011). On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress.

Lou, Y., Caruana, R., and Gehrke, J. (2012). Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 150–158, New York, NY, USA. ACM.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2017). Catboost: unbiased boosting with categorical features. *arXiv preprint arXiv:1706.09516*.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Tweedie, M. C. K. (1984). An index which distinguishes between some important exponential families. In Ghosh, J. K. and Roy, J., editors, *Statistics: Applications and New Directions*. Proceedings of the Indian Statistical Institute Golden Jubilee International Conference, Calcutta: Indian Statistical Institute.

Williams, C. K. I. and Seeger, M. (2001). Using the nyström method to speed up kernel machines. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press.

Yang, Y., Qian, W., and Zou, H. (2017). Insurance premium prediction via gradient tree-boosted tweedie compound poisson models. *Journal of Business & Economic Statistics*, pages 1–15.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320.