1 Classification

1.1 Introduction

Classification is a supervised machine learning approach to categorizing data into **distinct number of classes** where we can assign label to each class. Given a set of data $\{x^{(i)}, y^{(i)}\}$ where x is the feature space in $m \times (n+1)$ dimension, y is the classfication output such $y \in \{0, 1\}$ for binary output or $\{1, 2, ...n\}$ for multiclass output. Classification algorithms are most used for Spam detection, Voice and image recognition, sentiment analysis, fraud detection and many more.

Machine learning classification algorithms come in different types and they include:

- Logistic regression
- Perceptron
- Naive Bayes
- K-Nearest Neighbor
- Support vector Classifier
- Decision trees
- Boosted trees (Adaboost)
- Random Forest

1.2 Logistic Regession

Logistic regression is a discriminative model since it focuses only on the posterior probability of each class $Pr(Y|x;\beta)$. It is also a generalized linear model, mapping output of linear multiple regression to posterior probability of each class $Pr(Y|x;\beta) \in \{0, 1\}$. probabilty a data-sample belongs to class 1 is given by

$$Pr(Y = 1|X = x;\beta) = \sigma(z), \textit{where } z = \beta^T x \tag{1}$$

$$P(Y = 1|X = x;\beta) = \sigma(\beta^T x) \tag{2}$$

where

$$Pr(Y = 1|X = x;\beta) + Pr(Y = 0|X = x;\beta) = 1 \tag{3}$$

$$Pr(Y = 0|X = x;\beta) = 1 - Pr(Y = 1|X = x;\beta) \tag{4}$$

Hence, probability that a data-sample belongs to class 0 is give by:

$$Pr(Y = 0|X = x;\beta) = 1 - \sigma(z) \tag{5}$$

$\sigma(z)$ is called the **logistic sigmoid function** and is give by

$$\sigma(z) = \frac{1}{1 + \exp^{-z}} \tag{6}$$

The uniqueness of this function is that it maps all real numbers $\mathbb{R}$ to range $\{ 0, 1\}$.

Again, we know

$$log(odds(Pr(Y=1|X=x;\beta))) = \frac{Pr(Y=1|X=x;\beta)}{Pr(Y=0|X=x;\beta)} = \frac{Pr(Y=1|X=x;\beta)}{1-Pr(Y=1|X=x;\beta)} \quad (7)$$

Assuming $P(Y=1|X=x;\beta) = p(x)$, the next most obvious idea is to let $logp(x)$ be a linear function of x, so that changing an input variable multiplies the probability by a fixed amount. This is done by taking a *log* transformation of $p(x)$.

Formally, $logit(p(x)) = \beta_0 + \beta^T x$ making

$$logit(p(x)) = log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta^T x \quad (8)$$

Simplifying for $p(x)$ and $1-p(x)$ we have

$$\frac{p(x)}{1-p(x)} = \exp(\beta_0 + \beta^T x) \quad (9)$$

$$p(x) = (1-p(x))\exp(\beta_0 + \beta^T x) \quad (10)$$

$$p(x) = \exp(\beta_0 + \beta^T x) - p(x)\cdot\exp(\beta_0 + \beta^T x) \quad (11)$$

$$p(x) + p(x)\cdot\exp(\beta_0 + \beta^T x) = \exp(\beta_0 + \beta^T x) \quad (12)$$

$$p(x)(1+\exp(\beta_0 + \beta^T x)) = \exp(\beta_0 + \beta^T x) \quad (13)$$

$$p(x) = \frac{\exp(\beta_0 + \beta^T x)}{1+\exp(\beta_0 + \beta^T x)} = \frac{\frac{1}{exp(\beta_0+\beta^T x)}\cdot\exp(\beta_0 + \beta^T x)}{\frac{1}{exp(\beta_0+\beta^T x)} + \frac{exp(\beta_0+\beta^T x)}{exp(\beta_0+\beta^T x)}} = \frac{1}{1+\exp -(\beta_0 + \beta^T x)} \quad (14)$$

$$1-p(x) = \frac{1}{1+\exp(\beta_0 + \beta^T x)} \quad (15)$$

1.2.1 Learning Logistic regression by maximum likelihood

Assume that $P(Y=1|X=x;\beta) = P(x;\beta)$, for some function p parameterized by β. Further assume that observations are independent of each other. The conditional likelihood function is given by Bernoulli sequence:

$$\Pi_{i=1}^n Pr(Y=y_i|X=x_i;\beta) = \Pi_{i=1}^n p(x_i;\beta)^y(1-p(x_i;\beta)^{(1-y_i)}) \quad (16)$$

The probability of a class is p, if $y_i = 1$, or $1-p$, if $y_i = 0$. The likelihood is then

$$L(\beta_0,\beta) = \Pi_{i=1}^n p(x_i)^y(1-p(x_i)^{(1-y_i)}) \quad (17)$$

taking the log of this likelhood we have

$$l(\beta_0,\beta) = \sum_{i=1}^n y_i logp(x_i) + (1-y_i)log(1-p(x_i)) \quad (18)$$

$$= \sum_{i=1}^n y_i logp(x_i) - y_i log(1-p(x_i)) + log(1-p(x_i)) \quad (19)$$

$$= \sum_{i=1}^n log(1-p(x_i)) + y_i log\left(\frac{p(x_i)}{1-p(x_i)}\right) \quad (20)$$

replace $log\left(\frac{p(x)}{1-p(x)}\right)$ with $\beta_0+x\cdot\beta$ as seen in equation (8) and $(1-p(x))$ with $\frac{1}{1+\exp(\beta_0+x\cdot\beta)}$. Hence,

$$l(\beta_0,\beta)=\sum_{i=i}^{n} log\left(\frac{1}{\exp(\beta_0+x\cdot\beta)}\right)+y(\beta_0+x\cdot\beta) \tag{21}$$

$$=\sum_{i=i}^{n} -log(1+\exp(\beta_0+x\cdot\beta))+y(\beta_0+x\cdot\beta) \tag{22}$$

$$\bigtriangledown\, l(\beta_0,\beta)=-\sum_{i=1}^{n}\frac{1}{1+\exp(\beta_0+x\cdot\beta)}x_{ij}\exp(\beta_0+x\cdot\beta)+\sum_{i=1}^{n} y_i x_i \tag{23}$$

$$=\sum_{i=1}^{n}(y_i-p(x_i;\beta_0,\beta))x_{ij} \tag{24}$$

Since this is a transcendental equation and there is no closed-form solution, we would apply the gradient ascent optimization algorithm.

1.2.2 Gradient ascent

Algorithm:
Using the cost function equal to the negative average of log-likelihood. eq(18)

$$\max_{\beta} J(\beta)=-\frac{1}{m}\left[\sum_{i=1}^{m} y_i log p(x_i)+(1-y_i)log(1-p(x_i))\right] \tag{25}$$

Repeat {

$$\beta_{j+1}=\beta_j+\alpha\cdot\bigtriangledown J(\beta) \tag{26}$$

}simultaneously updating β_j

Where $\bigtriangledown J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(y_i-p(x_i;\beta_0,\beta))x_{ij}$ and $p(x)$ remains logistic function given by $\frac{1}{1+\exp-(x\cdot\beta)}$. Newtons numerical optimization method is another approach to solving this problem- a case where the second order differential in required.

1.2.3 Learning Logistic regression by least-squares

We define a least-square optimization problem

$$\min_{\beta} J(\beta)=\frac{1}{2}\sum_{i=1}^{N}(y_i-p(x_i;\beta_0,\beta))^2 \tag{27}$$

- $y_n\in[0,\,1]$ is the desired posterior probability for the points x_i.
- $p(x_i;\beta_0,\beta)$ is the parametric function to be learned and it defined by equation (14).
- learning the classifier by regression equally means estimating the desired posterior probabilities with the function $p(x_i;\beta_0,\beta)$.

We can rewrite the objective function as

$$\min_{\beta} J(\beta)=\frac{1}{2}||(y_n-p(x_n;\beta_0,\beta))||_2^2 \tag{28}$$

$$= \frac{1}{2}(y_n - p(x_n;\beta_0,\beta))^T(y_n - p(x_n;\beta_0,\beta)) \tag{29}$$

$$= \frac{1}{2}(y^T y - y^T p(x_n;\beta_0,\beta) - y_n p(x_n;\beta_0,\beta)^T - p(x_n;\beta_0,\beta)^2) \tag{30}$$

Using the trace property from linear algebra $tr(A^T B) = tr(AB^T)$, we can deduce that $y_n p(x_n;\beta_0,\beta)^T = y_n^T p(x_n;\beta_0,\beta)$

$$= \frac{1}{2}(y^T y - 2y^T p(x_n;\beta_0,\beta) + p(x_n;\beta_0,\beta)^2) \tag{31}$$

$$\bigtriangledown J(\beta) = \frac{1}{2}\left[\frac{\partial y^T y}{\partial\beta} - 2y^T\frac{\partial p(x_n;\beta_0,\beta)}{\partial\beta} + \frac{\partial p(x_n;\beta_0,\beta)^2}{\partial\beta}\right] = 0 \tag{32}$$

Where $\frac{\partial p(x_n;\beta_0,\beta)}{\partial\beta} = \frac{-xexp(-x\beta)}{(1+exp(-x\beta))^2}$ and $\frac{\partial p(x_n;\beta_0,\beta)^2}{\partial\beta} = \frac{-2xexp(-x\beta)}{(1+exp(-x\beta))^3}$

$$\bigtriangledown J(\beta) = \frac{1}{2}\left[\frac{2yxexp(-x\beta)}{(1+exp(-x\beta))^2} - \frac{2xexp(-x\beta)}{(1+exp(-x\beta))^3}\right] \tag{33}$$

$$= \left[\frac{yxexp(-x\beta)}{(1+exp(-x\beta))^2} - \frac{xexp(-x\beta)}{(1+exp(-x\beta))^3}\right] \tag{34}$$

$$= \frac{xexp(-x\beta)}{1+exp(-x\beta)}\left[y - \frac{1}{1+exp(-x\beta)}\right] \tag{35}$$

$$= [y - p(x_n;\beta_0,\beta)]\,(1 - p(x_n;\beta_0,\beta))x \tag{36}$$

The β parameters can be evaluated using the **Gradient descent algorithm**.

1.2.4 Regularized Logistic regression

Regularized logistic regression is much like regularized linear regression (Ridge regression) execept with the addition of a division term $2m$. The cost function for logistic regression can be altered with the addition of a regularization term $\frac{\lambda}{2m}$. The cost function for the optimization problem is described using the equation below.

$$\max_{\beta} J(\beta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i log p(x_i) + (1-y_i)log(1-p(x_i)) + \frac{\lambda}{2}\sum_{i=1}^{n}\beta_j^2\right] \tag{37}$$

which can be solved using the gradient ascent algorithm above with an addition of $(\lambda/m)\beta$
Repeat {

$$\beta_{j+1} = \beta_j + \alpha\cdot\bigtriangledown J(\beta) \tag{38}$$

}simultaneously updating β_j

$$where \bigtriangledown J(\beta) = -\frac{1}{m}\left[\sum_{i=1}^{n}(y_i - p(x_i;\beta_0,\beta))x_{ij} + \lambda\beta_j\right] \tag{39}$$

1.3 Perceptron

Perceptrons, the simplest form of neural network, are parametric nonlinear function approximators $f(x;\beta_0,\beta)$ used for classification and regression purpose. The algorithm is originally inspired by neuronal circuits in the brain. Neuron activities in the brain is responsible for reflexes and problem solving intelligence, including but not limited to navigation, planning, object recognition, visual and speech perception.

1.3.1 The perceptron

Linear perceptron computes a linear function $y = \sum_{i=1}^{N} w_0 + w_i x_i \in \mathbb{R}$ (or $w^T x \in \mathbb{R}^{N+1}$) of an input vector $\mathbf{x} \in \mathbb{R}^N$. for K outputs $y = \mathbf{W}x$ with $\mathbf{W} \in \mathbb{R}^{k\times(N+1)}$

- Two class classification, k = 2

$$y = \sigma(\mathbf{w}^T x) = \frac{1}{1 + exp(-\mathbf{w}^T x)} \in (0,1)(logistic) \tag{40}$$

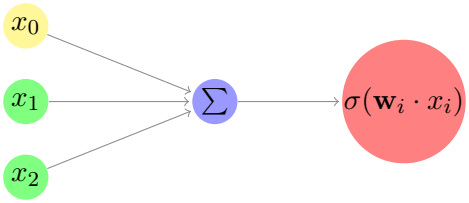
- For k > 2 classes

$$y_k = \frac{exp(\mathbf{w}_k^T x)}{\sum_{j=1}^{K} exp(\mathbf{w}_k^T x)} \in (0,1), k = 1,\ldots,K \quad with \quad \sum_{j=1}^{K} y_k = 1 \tag{41}$$

This activation function is known as the **softmax** activation function.

1.3.2 Training a perceptron

- Single Layer Perceptron for classification



The edges of the networks are the respective parametric weights. Using gradient descent algorithm, we estimate the weights thats minimizes number of misclassified errors.

$$\sigma(\mathbf{w}_i \cdot x_i) = \begin{cases} +1 & if \mathbf{w}_i \cdot x_i > 0 \\ -1, & otherwise \end{cases}$$

Since perceptron is based on nonlinear neuron,

$$y = sign\left(\sum_{i=1}^{N} w_i x_i\right) \tag{42}$$

Let C_1 & C_2 respectively denote class $+1$ and -1. the weights are adjusted using an adaptive learning rule **perceptron convergence algorithm**. If the two classes C_1 and C_2 are linearly seperable (meaning they lie on opposite sides of the line for 2-D, or in general, hyperplane), the perceptron learns the decision boundary by minimizing the error function (**perceptron criterion**)

$$E(\mathbf{w}) = -\eta \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}) \tag{43}$$

η is the class label here.

$$\mathbf{w}^T \mathbf{x} \geq 0 \;\forall\; \mathbf{x} \in C_1$$
$$\mathbf{w}^T \mathbf{x} < 0 \;\forall\; \mathbf{x} \in C_2$$

perceptron convergence algorithm

Perceptron minimizes error function by Stochastic gradient descent

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \nabla E(\mathbf{w}) = \mathbf{w}_i + \eta_i \mathbf{x}_i \tag{44}$$

*If the i^{th} member of a training set x_i is correctly classified by the weight vector $\mathbf{w}$ computed at i^th iteration of the algorithm, $\mathbf{w}$ remains unchanged. Hence

$$w_{i+1} = \begin{cases} w_i & if \mathbf{w}^T\mathbf{x} \geq 0 \rightarrow C_1 \\ w_i & if \mathbf{w}^T\mathbf{x} < 0 \rightarrow C_2 \end{cases} \quad (45)$$

*Otherwise the weights of the perceptron is updated according to the rule

$$w_{i+1} = \begin{cases} w_i - \eta_i x_i & if \mathbf{w}^T\mathbf{x} \geq 0 \rightarrow C_2 \\ w_i + \eta_i x_i & if \mathbf{w}^T\mathbf{x} < 0 \rightarrow C_1 \end{cases} \quad (46)$$

perceptron learning by least-squares

Stochastic gradient descent estimates the weights by minimizes the least-squares error or cost function $E(\mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2}(y_n - \mathbf{w}^T\mathbf{x}_n)^2$. weights are repeatedly updated using $\mathbf{w} \leftarrow \mathbf{w} + \triangle\mathbf{w}$ where $\triangle\mathbf{w} = -\eta\nabla E(\mathbf{w}) = \eta(y_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n$. The learning rate parameter η is a positive constant within range $0 < \eta \leq 1$. This approach converges faster than the perceptron convergence algorithm (which is just a variant of SGD). Note that it is important to scale the train data before feeding into the perceptron algorithm.