Regression for Classification

Linear and logistic regression for binary and multi-class classification.

Victor Ivamoto

April, 2020

Contents

Chapter 1

Introduction

This is the report for the machine learning course at Universidade de Sao Paulo. The goal is the code implementation of a linear classification model for binary and multiple classes.

Before designing the linear classification, the data shall be pre-processed using codification, handling of missing data and normalization. The dataset shall be partitioned in training and test sets, with 2/3 of the dataset used for training and 1/3 for test. The training set shall be the initial portion of the dataset.

The report describes the theoretical fundamentals of regression models used in classification, including regularization and logistic regression, the methodology used during data preparation and the results from applying the models in the datasets.

The Python code to create the models and R Markdown code of this report are available in GitHub.

Chapter 2

Regression

Regression is a popular method used in machine learning for predicting the outcome of classification and continuous problems. It's often used as a baseline for other ML models, i.e. complex models are expected to perform better than regression.

2.1 Linear Classification

Binary classification is the simplest form of regression for classification. Given an independent variable Y, one or more dependent variables $X_1, ..., X_p$, and a set of observations X_i and Y. We can find a function $f(\vec{x})$ that approximates a linear combination of X_i and Y, with $Y = y \in \{-1, 1\}$ as:

$$f(\vec{x}) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_d x_d$$

The coefficients, or weights, ω are unknown and are applied to all values of x_i. If the number of variables X_i is two, i.e. $p = 2$, then the function $f(\vec{x})$ is a line, and if $p > 2$ then $f(\vec{x})$ is a hyper-plane. Note that $f(\vec{x}) \to \mathbb{R}$ can return any real number, which is not helpful for a classification problem with two possible outcomes. The solution is to apply a function $h(\vec{s})$ that converts the output of $f(\vec{x})$ to the desired values:

$$h(\vec{x}) = sign(f(\vec{x})) = \begin{cases} c & \text{if } f(\vec{x}) \geqslant 0 \\ \bar{c} & \text{if } f(\vec{x}) < 0 \end{cases}$$

For simplification, $h(\vec{x})$ can be written as:

$$h(\vec{x}) = sign\left(\sum_{i=1}^{d} \omega_i x_i + \omega_0\right)$$

If we set $x_0 = 1$, then we can rewrite this equation to:

$$h(\vec{x}) = sign\left(\sum_{i=0}^{d} \omega_i x_i\right)$$

In order to make $h(\vec{x})$ as close as possible to $f(\vec{x})$, we define the quadratic error function $E(\vec{x})$.

$$E(h) = (h(\vec{x}) - f(\vec{x}))^2 = \frac{1}{N}\sum_{i=1}^{N}(h(\vec{x}_i) - y_i)^2$$

Rewriting this expression as a function of $\vec{\omega}$, we get:

$$E(\vec{\omega}) = \frac{1}{N}\sum_{i=1}^{N}(\vec{\omega}.\vec{x}_i - y_i)^2$$

This is also called *Mean Squared Error*, or MSE, and is widely used in regression. The MSE penalizes large deviations from the mean.

Now, we want to minimize this error, so the predicted values approximate the true values. Calculating the gradient of MSE, equating to zero and solving for $\vec{\omega}$, we have the coefficient values that minimizes the error.

$$\vec{\omega} = (X^\top X)^{-1} X^\top \vec{y}$$

2.2 Regularized Least Squares

A regularization term can be added to the error function to control over-fitting.

$$E(\vec{\omega}) + \lambda E_{\vec{\omega}}(\vec{\omega})$$

Where λ is the regularization coefficient that controls the relative importance of the error $E(\vec{\omega})$ and the regularization term $E_{\vec{\omega}}(\vec{\omega})$. The regularization term is a generic function of the form:

$$E_{\vec{\omega}}(\vec{\omega}) = \sum_{j=1}^{M} |\omega_j|^q$$

If $q = 1$ the term is called *lasso*, for $q = 2$ the term is called *ridge regression*. Other values of q can be used, although that is not common.

λ is a tuning parameter and there are many methods to calculate. One simple way is to choose a grid of values, use cross-validation and select the tuning parameter for which the error is smallest.

2.3 Multi-class Classification

In the linear regression model discussed previously, the function $f(\vec{x}) \rightarrow \mathbb{R}$ returns a real number, which can be mapped to just two classes, $y_i \in \{-1, 1\}$. In the multi-class classification, the outcome variable Y belongs to two or more classes $C_1, C_2, ..., C_k$. In other words, $Y = y \in \{C_1, ..., C_k\}$

Mapping each class to a real number adds an error in the model because the distance between two classes may differ from the distance between the other two. For example, if $k = 3$ and $Y = y_i \in [-1, 0, 1]$, the distance between -1 and 0 is smaller than the distance between -1 and 1.

In this case, one approach is to map each class C_p into an independent outcome $Y_p = y_p \in \{-1, 1\}$. This method converts the multi-class into binary regression, and allows the use of algorithm described earlier.

2.4 Logistic Regression

The linear regression function returns a real number that may be any value, which is then converted to two values in classification problems. The logistic regression function calculates the probability of the outcome.

For a given data set $(\vec{x}, y)$, where $y \in \{-1, 1\}$, the conditional probability of y given $\vec{x}$ is defined as:

$$Pr(y|\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } y = 1 \\ 1 - f(\vec{x}) & \text{if } y = -1 \end{cases}$$

This is easy to understand, since $Pr(y = 1) = 1 - Pr(y \neq 1)$. The function $f(\vec{x})$ used in binary logistic regression is the sigmoid $\theta(s)$:

$$\theta(s) = \frac{e^s}{1 + e^s}$$

The sigmoid has the nice property $\theta(-s) = 1 - \theta(s)$. Applying the sigmoid in the probability equation and assuming $s = \vec{\omega}.\vec{x}$:

$$Pr(y|\vec{x}) = \begin{cases} \theta(s) & \text{if } y = 1 \\ 1 - \theta(s) & \text{if } y = -1 \end{cases}$$

Which is the same as $Pr(y|\vec{x}) = \theta(y\vec{\omega}.\vec{x})$, since $y \in \{-1, 1\}$.

The error function is defined as:

$$E(\vec{\omega}) = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n \vec{\omega}.\vec{x}_n})$$

Notice that the error is a function of $\vec{\omega}$, since we can't modify the observed values $(\vec{x}, y)$. Now, in order to improve the accuracy we need to minimize the error, i.e. find the values of $\vec{\omega}$ that results in the smallest error. In calculus, the gradient of a function points to the direction of the maximum value, so its negative points to the minimum position. The gradient of the error is:

$$\bigtriangledown E(\vec{\omega}) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1 + e^{y_n \vec{\omega}.\vec{x}_n}}$$

2.4.1 Multi-class Logistic

Similar to binary logistic regression, the multi-class probability of each class can be calculated using the softmax function:

$$\text{Pr}(Y = k|X = x) = \text{softmax}(\text{W}^\top \text{x}), \quad k = 1, ..., c$$

Where the softmax function is defined as:

$$\text{softmax}(\eta) = \frac{e^\eta}{\sum e^\eta}$$

The gradient of the error function is given by:

$$\bigtriangledown E(\vec{\omega}) = x_i(p_i - y_i)^\top$$

Where $p_i = \text{softmax}(W^\top x_i)$.

Chapter 3

Data Preparation

3.1 Dataset Description

Understanding how the data was obtained is essential before creating the model, as this step is helpful during the pre-processing phase. Some useful information are how the data was collected, the motivation for collecting the data, the limitations during the collection phase and the methodology or process of collecting and recording the information. This information may have significant impact in the data analysis result.

The main purpose of this report is the study and implementation of regression algorithms, for later comparison of the models performance. Therefore, a deep understanding of the data origin is not necessary. All information needed to build the models are available in the datasets.

This project uses the datasets iris, hepatitis and diabetes available in UCI machine learning repository and Kaggle. The dataset descriptions provided by the donors are available in their respective websites, as listed below.

Hepatitis: https://archive.ics.uci.edu/ml/datasets/Hepatitis

Iris: https://archive.ics.uci.edu/ml/datasets/Iris

Diabetes https://www.kaggle.com/uciml/pima-indians-diabetes-database

3.1.1 Hepatitis Dataset

The dataset contains 19 predictors and the outcome variable ‘Class’. The classes of all categorical variables are 1 and 2. There are many missing values in this dataset, which are represented by a question mark ‘?’.

There are 155 samples.

Attribute Information:

1. Class: 1, 2
2. AGE: 10, 20, 30, 40, 50, 60, 70, 80
3. SEX: 1, 2
4. STEROID: 1, 2
5. ANTIVIRALS: 1, 2
6. FATIGUE: 1, 2
7. MALAISE: 1, 2
8. ANOREXIA: 1, 2
9. LIVER BIG: 1, 2
10. LIVER FIRM: 1, 2
11. SPLEEN PALPABLE: 1, 2
12. SPIDERS: 1, 2
13. ASCITES: 1, 2
14. VARICES: 1, 2
15. BILIRUBIN: 0.39, 0.80, 1.20, 2.00, 3.00, 4.00
16. ALK PHOSPHATE: 33, 80, 120, 160, 200, 250
17. SGOT: 13, 100, 200, 300, 400, 500,
18. ALBUMIN: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
19. PROTIME: 10, 20, 30, 40, 50, 60, 70, 80, 90
20. HISTOLOGY: 1, 2

3.1.2 Iris Dataset

The iris dataset contains four predictors, all of them are continuous variables. The outcome 'class' is categorical with three classes.

Iris comes with 150 observations equally distributed among the three classes.

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   – Iris Setosa
   – Iris Versicolour
   – Iris Virginica

3.1.3 Diabetes Dataset

The diabetes dataset contains eight predictors, all of them are continuous variables. The outcome variable 'class' is categorical with two classes. This dataset was provided with the train and test partitions, and the variables were renamed, as outlined below.

Diabetes comes with 538 and 230 observations for the train and test sets respectively.

Attribute Information:

1. atr1: continuous
2. atr2: continuous
3. atr3: continuous
4. atr4: continuous
5. atr5: continuous
6. atr6: continuous
7. atr7: continuous
8. atr8: continuous
9. class: 0, 1

3.2 Outcome: Multi-class Classification (1 of n)

We create a column for each class and use the 1 of n approach:

$$X.\vec{\omega} = Y$$

Where:

- X is the matrix of features,
- $\vec{\omega}$ is the weight vector and
- Y is the outcome matrix.

In this activity, only the iris dataset is multi-class, so we add a column for each flower class in the format 1 of n. For example, the **setosa** column contains 1 for setosa observations and -1 otherwise.

sepal_width	petal_length	petal_width	class	setosa	versicolor	virginica
0.625000	0.067797	0.041667	Iris-setosa	1	-1	-1
0.416667	0.067797	0.041667	Iris-setosa	1	-1	-1
0.458333	0.084746	0.041667	Iris-setosa	1	-1	-1
0.666667	0.067797	0.041667	Iris-setosa	1	-1	-1
0.583333	0.067797	0.083333	Iris-setosa	1	-1	-1

3.3 Missing Values

Missing values is the result of many different reasons, so we need to examine them carefully. The first step is to understand how the data was collected and the reason for the

missing values. Contacting the person that created the dataset is a good starting point. However, this is not always possible, as the data may be old or the person is not available.

In the case of hepatitis dataset which contains many missing values, suppose the treatment consists of a drug that causes significant side effects. The patient may be likely to drop the treatment, or may have died during the study if the disease is severe.

In this situation, the pattern of missing data is related to the outcome and is called “informative missingness”, and can induce significant bias in the model.

For missing data concentrated in a sample, we can simply remove that sample, as this may be a strong indication that the treatment was halted for some reason.

Similarly, predictors with high percentage of missing data may be discarded.

Another approach is data imputation, which consists in estimating the predictor variables based on other predictor variables. We can use machine learning algorithms to accomplish this, such as decision trees and regression.

Note that both methods, removing missing data and predicting new values, add uncertainty in the model and therefore in the outcome prediction.

The hepatitis dataset contains missing values represented by ‘?’. For simplicity, in this exercise we’ll adopt the following criteria for this dataset:

1. Remove samples and predictors with more than 20% of missing values.
2. For continuous variables, replace missing values with the mean of existing values.
3. Remove the remaining missing values for categorical variables.

After applying this procedure, the number of observations reduced from 155 to 141, a 9% reduction, and one predictor was removed.

The diabetes dataset contains invalid values, such as blood insulin level of zero. These values won’t be removed nor replaced.

The iris dataset is complete.

3.4 Categorical Variables

Categorical variables may come with different values in the dataset and need to be standardized before processing. Categorical predictors are binary for all datasets.

The regression formula assumes $y \in \{-1, 1\}$, so for this reason all categorical values for both predictors and outcomes are converted to $\{-1, 1\}$.

The multi-class logistic regression formula uses $y \in \{0, 1\}$, so only y in the training and test sets is converted.

3.5 Training and Test Sets

The diabetes dataset was provided with training and test sets, so there's no need for creating them.

The iris dataset contains 150 samples ordered by class. So, evenly splitting 2/3 for train and 1/3 for test results that all samples for the first two classes go to the train set and all samples of the third class go to the test set. As we can imagine, training the model and predicting new values with these sets will fail.

The approach used was to pick every three samples for test and the other two for train. For example, the training samples are `{0, 1, 3, 4, 6, 7, ...}` and the test samples are `{2, 5, 8, 11, ...}`.

The hepatitis dataset was split 2/3 for train and 1/3 for test, according to the instructions for this activity.

3.6 Data Normalization

Each dependent variable is normalized using min-max or z-score.

Min-max fits the values in the range $[0, 1]$ using the formula

$$x = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

where x_{max} and x_{min} are the maximum and minimum values respectively.

Z-score assumes the values are normally distributed and transforms to a distribution with $\mu = 0$ and $\sigma = 1$. The formula is:

$$z = \frac{x - \mu}{\sigma}$$

Where z is the new value of x.

All continuous dependent variables are normalized with the z-score method. For each continuous predictor in the train set, the z-score transformation is applied and the mean and standard deviations are used in the z-score transformation in the test set.

Chapter 4

Train and Predict

This section describes the process of training a model for each regression model and predicting new values.

The general process is the same for all datasets and regression models, and is outlined below:

1. Define `X` and `Y` variables for train and test sets.
2. Compute the weights `w` using `X` and `Y` from the train set as input.
3. Predict new values, `y_hat`, with `w` and `X` from the test set.
4. Compare `y_hat` with `Y` from the test set.

We add a column filled with 1 in the `X` arrays for both the train and test sets, which corresponds to $x_0 = 1$ in the formula below.

$$f(\vec{x}) = \omega_0 + \omega_1 x_1 + ... + \omega_n x_n = \omega_0 + \sum_{n=1}^{N} \omega_n x_n = \sum_{n=0}^{N} \omega_n x_n \text{ , for } x_0 = 1$$

Each model uses different formulas for computing the weights `w` and for estimating the outcome `Y`. These formulas are discussed in the next sections.

4.1 Linear Classification

The formula used to calculate the weights for binary and multi-class is:

$$\vec{\omega} = (X_{train}^{\top} X_{train})^{-1} X_{train}^{\top} \vec{y}_{train}$$

For binary classification, we apply this formula to predict new values:

$$\hat{y} = sign(X_{test}.\vec{w})$$

The prediction of the multi-class regression in iris dataset is the matrix $\hat{Y} = X.\vec{\omega}$ with one column for each class. The predicted class has the highest value and is converted to 1, while the other classes are converted to -1. The pseudo code of the conversion is:

```
1 y_hat <- X_test * w
2 for i = 0 to Number of rows in y_hat
3    for j = 0 to Number of Columns in y_hat
3       if y_hat(i, j) = max(y_hat(i))
3           y_hat(i,j) = 1
4       else
5           y_hat(i,j) = -1
```

4.2 Regularization

Regularization adds the parameter λ in the weight and error formulas. In order to estimate the best λ, we use *k*-fold cross-validation, explained later, and the simple process outlined below.

1. Split the train set into *k* folds, pick 1 fold for train and *k-1* folds for validation.
2. Calculate ω for several values of λ.
3. Select the λ that results in the lowest error $E(\vec{\omega})$

The formula used to calculate $\vec{\omega}$ is:

$$\vec{\omega} = \left(\frac{\lambda}{N} I + X_{train}^{\top} X_{train}\right)^{-1} X_{train}^{\top} \vec{y}$$

The formula used to calculate the error $E(\vec{\omega})$ is:

$$E(\vec{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=0}^{d} (x_{i,j} \omega_j) - y_i \right)^2 + \gamma \sum_{j=1}^{M} |\omega_j|^q$$

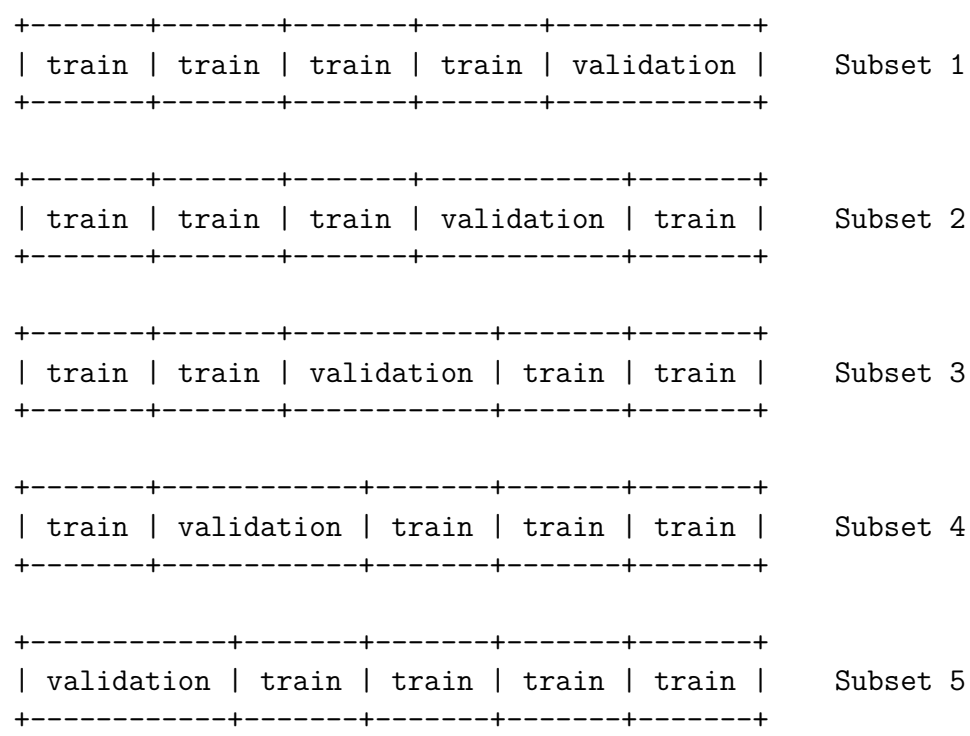
Where $x_{i,j}$ and y_i are from the validation set.

In this activity, we use the *lasso* model, with $q = 1$ in the formula above.

4.2.1 *k*-fold Cross-Validation

In this method we split the training set into *k* folds of equal size, and then use *k-1* folds for training and 1 fold for validating the model. This approach keeps the test set untouched, i.e. it's not used to train or test the model until the model is ready.

An example of $k = 5$ is shown in the figures below. The train set is split into 5 folds of equal size, with one fold used for validation and 4 folds for train. The process is repeated 5 times, and in each subset the validation uses a different portion of the original training set.



The most common values of *k* are 5 and 10, and in this activity we use $k = 10$.

4.3 Logistic Regression

Recall that we want to find the weights that minimize the error function using derivative. Calculating the weights for linear classification is straight forward, we simply isolate $\vec{\omega}$ in the error derivative equation. The derivative of the error function in logistic regression is non-linear, requiring the use of iterative computation.

The pseudo code for calculating the gradient is the same for both the binary and multi-class cases, the only difference is the gradient formula.

```
1 Initialize weight with random numbers
2 Define minimum gradient, learning rate
3 Calculate initial gradient
4 For t = 1, 2, 3, ... do
5     calculate new gradient
6     update learning rate if variable gradient descent is used
7     update weight(t+1) = weight(t) - learning rate x gradient
8     stop if norm(gradient) < norm(minimum gradient)
9 Return weight
```

4.3.1 Binary Logistic Regression

The gradient formula used to compute the weights is presented below.

$$\bigtriangledown E(\vec{\omega}) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n\vec{x}_n}{1+e^{y_n\vec{w}.\vec{x}_n}}$$

After computing the weights, we predict new values using this formula:

$$\text{Pr}(y|\vec{x}) = \theta(X.\vec{\omega}) = \text{sigmoid}(X.\vec{\omega}) = \frac{e^{X.\vec{\omega}}}{1+e^{X.\vec{\omega}}}$$

Since $\text{Pr}(y|\vec{x}) \in [0,1]$ and $y \in \{-1,1\}$, $\widehat{y}$ is given by this formula:

$$\widehat{y} = \begin{cases} 1 & \text{if} \quad \text{Pr}(y|\vec{x}) \geqslant 0.5 \\ -1 & \text{if} \quad \text{Pr}(y|\vec{x}) < 0.5 \end{cases}$$

4.3.2 Multi-class Logistic Regression

The gradient formula for multi-class is this:

$$\bigtriangledown l_i(W) = \sum\left(x_i(\hat{\text{p}}_i - y_i)^\top\right)$$

Where:

$$\hat{\text{p}}_i = \text{softmax}(W^\top x)$$

And the formula to predict new values:

$$\widehat{Y} = \text{Pr}(Y = k|X = x) = \text{softmax}(W^\top X)$$

$$\widehat{y}_i = \begin{cases} 1 & \text{if} \quad \Pr(Y = k) > \Pr(Y \neq k) \\ 0 & \text{if} \quad \Pr(Y = k) < \Pr(Y \neq k) \end{cases}$$

Chapter 5

Results

The results of this experiment may vary across implementations with different methodologies, such as creating the train and test sets, regularization parameter, start and stop parameters in gradient descent, among other reasons.

5.1 Diabetes

Surprisingly, the linear regression model was slightly better than regression with regularization, even though the best performing regularization parameter was $\lambda = 0$, meaning regularization was inactive. This is because the weights of the best *k*-fold is different than the weights in regression without regularization.

The regularization factor should improve accuracy or at least keep the same performance.

The accuracy of logistic regression is the same as regression with regularization, This is just coincidence, as the predicted values are different.

The simplest model wins in both simplicity and accuracy.

The accuracy around 75% is a good value for regression models with minimum data filtering.

Method	Accuracy
Regression	75.65
Regression w/ regularization	74.78
Logistic	74.78

5.2 Hepatitis

Now the regression result is identical to the regression with regularization. As expected, the logistic regression has the best performance with accuracy at 78.7%

Method	Accuracy
Regression	72.34
Regression w/ regularization	72.34
Logistic	78.72

5.3 Iris

The multi-class dataset has identical performance for regression with and without regularization. With 100% accuracy, setosa has distinct features that sets it apart from other species. Versicolor and virginica have the same accuracy.

The logistic regression achieved higher performance classifying each flower.

Method	Accuracy
Regression - setosa	100.00
Regression - versicolor	82.00
Regression - virginica	82.00
Regression - overall	88.00
Regression w/ regularization - setosa	100.00
Regression w/ regularization - versicolor	82.00
Regression w/ regularization - virginica	82.00
Regression w/ regularization - Overall	88.00
Logistic - setosa	100.00
Logistic - versicolor	98.00
Logistic - virginica	98.00
Logistic - overall	98.67

The regularization

Chapter 6

Conclusion

6.1 Future Developments

The gradient descent algorithm may be improved with the second derivative, or hessian, of the error function and applying the Newton and Levenberg-Marquardt methods.

Reference

M. Kunh, K. Johnson [2016] - Applied Predictive Modeling Yaoliang Yu [2018]- CS480/680–Fall 2018 - LOGISTIC REGRESSION - University of Waterloo Christopher M. Bishop [2006] - Pattern Recognition and Machine Learning G. James, D. Witten, T. Hastie, R. Tibshirani [2017]- An Introduction to Statistical Learning

R.A. Fisher, M. Marshall (1988). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

G.Gong, B. Cestnik (1988). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261–265). IEEE Computer Society Press.