Mathematics for Supply Chain

Msc Supply Chain & Purchasing (2023-2024)

Guillaume Metzler

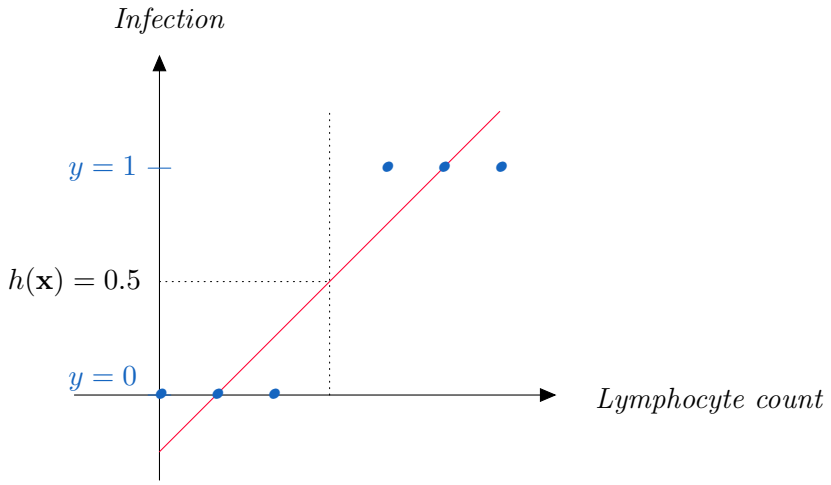
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr

From now on, we were mainly interested in models that are used to estimate the values of real random variable Y: *house price, benefits, average returns, etc.*
Let us place ourselves in a slightly different regression context now and consider the following example.

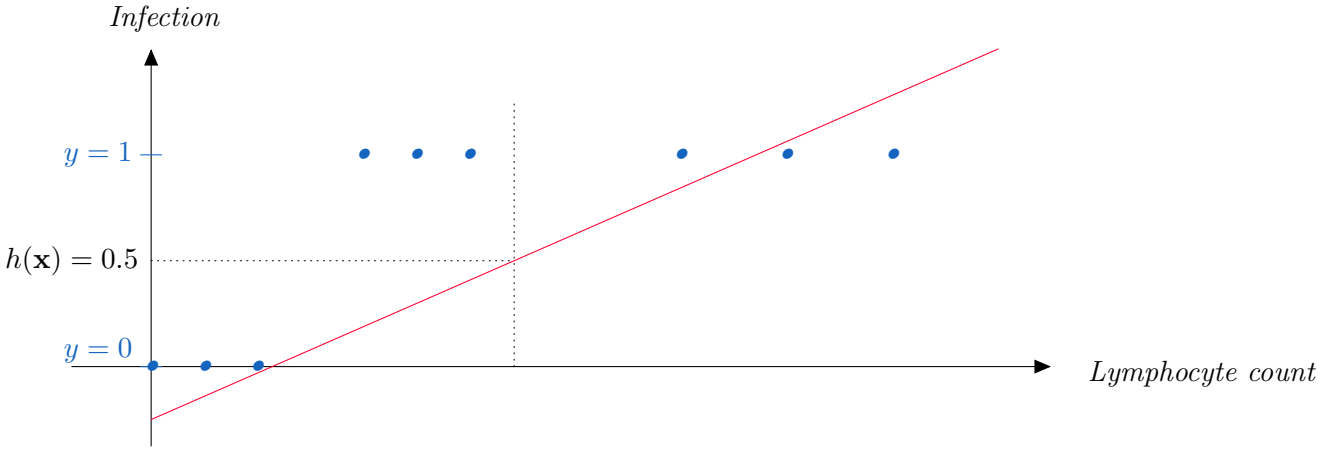
Example. *We are trying to build a regression model capable of determining whether or not an individual has an infection according to his lymphocyte count. The predicted variable can take two values:* 1 *if the person has an infection and* 0 *otherwise.*

At first sight, nothing prevents us from learning a linear model to try to fit our new point cloud, as illustrated below.



We will then simply have to take a threshold, on the values taken by our hypothesis h, beyond which an individual will be considered as sick, e.g. we consider that an example $\mathbf{x}$ belongs to the positive class when the hypothesis h returns a value greater than 0.5 *(i.e. negative on the left part and positive on the right one). In this example it works well.*

Let us now consider another one where the number of Lymphocytes can be extremely high, meaning that the infection is serious. This new dataset is represented below



This time we see that if we use the same threshold, we are missing some positive instances or infected people.

This example shows that the way we model our problem is not well chosen, we need a different structure, *i.e.* a line which is more adapted to the structure of our data. For instance, we need to have a model which be represented as follows:

Such a model takes its values in $[0, 1]$ and we can thus say that it estimates the probability of having an infection. To transform the values predicted by a linear regression model into probabilities, we use the logistic function, *i.e.* we compute:

$$\frac{1}{1 + \exp\left(-h(\mathbf{x})\right)}.$$

We talk about *linear logistic regression.*

Logistic Regression

This first section is dedicated to a general presentation of the logistic regression.

Presentation of the model

The Logistic Regression model, also called the *logit model* has been introduced in the middle of the 20^{th} century [Cox, 1958] but the use of *logit* models dates back to the end of the 19^{th} century [Cramer, 2003].

They are very close to Linear regression models, but the objective is now to predict the class to which an example belongs to, rather than predicting a real number. More precisely, they are used to estimate the probability that an example belongs to a given class, for instance the positive class: $\eta = Pr(Y = 1 \mid X)$. More precisely, the logistic regression aims to compute the logarithm of the *odds*, i.e. the ratio of the probabilities. Then we estimate the log of this ratio using a linear model:

$$\ln\left(\frac{Pr(y=1 \mid \mathbf{x})}{Pr(y=0 \mid \mathbf{x})}\right) = h(\mathbf{w}, b, \mathbf{x}) = b + \langle \mathbf{x}, \mathbf{w} \rangle.$$

Thus, once the parameters of the model are learned, we can compute the probability of being in class 1:

$$Pr(y=1 \mid \mathbf{x}) = \frac{\exp(h(\mathbf{w}, b, \mathbf{x}))}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}))} = \frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}))}.$$

Such function is called a *logistic function* and takes its values in $[0, 1]$. An example $\mathbf{x}_i$ is (usually) predicted in class 1 if $Pr(y = 1 \mid x) > 0.5$, i.e. if $h(\mathbf{w}, b, \mathbf{x}) > 0$. Given a task and an objective, we can choose to modify this threshold as we will see in the next chapter.

To estimate the parameters of the model, we maximize the likelihood of the data $\mathfrak{L}(\mathbf{w}, S)$, where S is a set of m examples.

$$\begin{aligned}
\mathfrak{L}(\mathbf{w}, b, S) &= \prod_{i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i), \\
&\quad \downarrow \text{ separate } y_i = 0 \text{ and } y_i = 1 \\
&= \prod_{i=1, y_i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i) \times \prod_{i=1, y_i=0}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i), \\
&\quad \downarrow \text{ Use the fact that the underlying law is a Bernoulli law} \\
&= \prod_{i=1}^{m} \left(\frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}_i))}\right)^{y_i} \times \left(\frac{1}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}_i))}\right)^{(1-y_i)}.
\end{aligned}$$

Note that we usually prefer to minimize the negative log-likelihood of the data:

$$\begin{aligned}\ell(\mathbf{w}, b, S) = & -\ln\left(\mathfrak{L}(\mathbf{w}, b, S)\right), \\ = & -\sum_{i=1}^{m} y_i \ln\left(\frac{1}{1+\exp\left(-(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)\right)}\right) + (1-y_i)\ln\left(1-\frac{1}{1+\exp\left(-(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)\right)}\right).\end{aligned}$$

By doing so, we find the logistic loss function introduced in the introduction lecture. In the following, for the sake of simplicity, we will set $g(\mathbf{w}, b, \mathbf{x}) = \dfrac{1}{1+\exp\left(-(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)\right)}$. Therefore, the optimization problem becomes:

$$\min_{\mathbf{w}, b\in\mathbb{R}^{d+1}} \quad -\frac{1}{m}\sum_{i=1}^{m} y_i \ln\left(g(\mathbf{w}, b, \mathbf{x}_i)\right) + (1-y_i)\ln\left(1-g(\mathbf{w}, b, \mathbf{x}_i)\right).$$

We divide the loss by a factor m in order to be consistent with the notion of empirical risk previously defined.

Avoiding over-fitting

In order to avoid over-fitting, a regularization term of the form $\lambda\|\mathbf{w}\|$ is usually used in regression tasks. Thus, the optimization problem can be rewritten:

$$\min_{\mathbf{w}, b\in\mathbb{R}^{d+1}} \quad -\frac{1}{m}\sum_{i=1}^{m} y_i \ln\left(g(\mathbf{w}, b, \mathbf{x}_i)\right) + (1-y_i)\ln\left(1-g(\mathbf{w}, b, \mathbf{x}_i)\right) + \lambda\|\mathbf{w}\|^2.$$

In the gaussian linear model, it can be written as

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|\mathbf{y} - h(\boldsymbol{\theta}, \mathbf{X})\|_2^2 + \lambda\|\boldsymbol{\theta}\|^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \sum_{i=1}^{m} (y_i - h(\boldsymbol{\theta}, \mathbf{x}_i))^2 + \lambda\|\boldsymbol{\theta}\|^2$$

In the two preceding expressions, the standard used is not specified but the standards are very often encountered: the L_1 norm $\|\cdot\|_1^2$ and the L_2 norm $\|\cdot\|_2^2$.
The latter will have an important impact on the model and especially on the parameters learned by the model:

- The L_1 norm

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{p} = |w_j|$$

is notably used to induce sparsity in the learned hypothesis, i.e. when we want the learned hypothesis to depend on a minimum of parameters. When we use this regularization term, we talk about *Lasso Regression*. This type of regularization is particularly used in high dimensional models, when there are many variables, as it can be the case in genetics. This regularization will allow to highlight the most important variables for the prediction task. However, it has an important drawback which is its non-differentiability at any point.

- The L_2 norm

$$\|\mathbf{w}\|_2^2 = \sum_{j=1}^{p} = |w_j|^2$$

is used to avoid that some parameters take too important points compared to the other parameters of the model. In this case we speak of *Ridge Regression*.

Let us now use in practice in a data science perspective, *i.e.* to do some Machine Learning.

Machine Learning with Logistic Regression

To illsutrate our purpose, we are going to use the *leukemia* dataset.

```
data = read.csv("../datasets/leukemia.csv",sep=";")
head(data)
```

```
##   REMISS CELL SMEAR INFIL  LI BLAST TEMP
## 1      1  0.8  0.83  0.66 1.9  1.10 1.00
## 2      1  0.9  0.36  0.32 1.4  0.74 0.99
## 3      0  0.8  0.88  0.70 0.8  0.18 0.98
## 4      0  1.0  0.87  0.87 0.7  1.05 0.99
## 5      1  0.9  0.75  0.68 1.3  0.52 0.98
## 6      0  1.0  0.65  0.65 0.6  0.52 0.98
```

We are interested here in predicting the value of the remission variable after treatment (**REMISS**) using the following information

- **CELL** : Cellularity (percentage) of marrow clot,
- **SMEAR** : Differential percentage of cancerous blasts,
- **INFIL** : Percentage of absolute medullary leukemia infiltration,
- **LI** : Percentage labeling index of bone marrow leukemia cells (vs. other cells),
- **BLAST** : Absolute number of cancerous blasts, in thousands,
- **TEMP** : Highest temperature before treatment.

So we want to know whether a treatment administered to a patient will work or not. We can start by looking at the distribution of the variable we're trying to predict **REMISS**.

```
# The function "table", gives a table with the number of occurrences
# of the studied object
prop = table(data$REMISS)
prop

##
##  0  1
## 18  9
```

In our dataset, we therefore have 9 patients who went into remission and 18 patients for whom treatment did not work..

We're now going to build our model that will enable us to perform our predictive task, and we'll then test its effectiveness on an independent dataset. To do this, we start by separating our dataset into two sets (Figure 1) : *train* and *test*, the first will be used to learn the **w** parameters of the model and the second to test our model, *i.e.* to see if the learned model is able to generalize on similar data.

Let us split our sample into two sets

```
# We fix the seed in order to have same results.

set.seed(4)
```

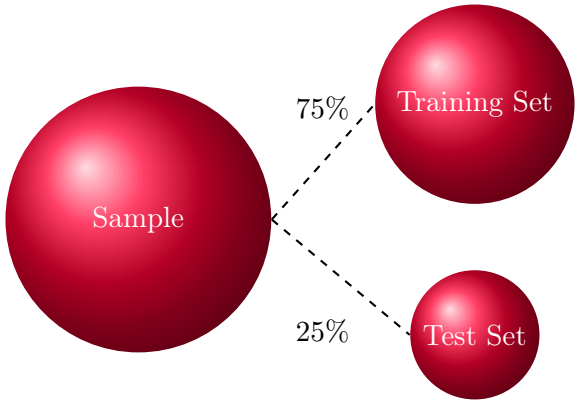


Figure 1: Seperating the dataset into train/test with proportion 75/25 %

```
# Let us split our dataset

library(caret)

## Loading required package:  ggplot2
## Loading required package:  lattice

index_train <- createDataPartition(y=data$REMISS, p = 0.75, list=FALSE)
train <- data[index_train,]
test <- data[-index_train,]
```

The function **createDataPartition** creates an index list containing 75% of the examples here, while preserving the proportion of classes present.

We can now perform the logistic regression using the **glm** from R.

```
mymodel = glm(REMISS ~ ., data = train, family=binomial)
summary(mymodel)


##
## Call:
## glm(formula = REMISS ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7929  -0.6075  -0.1383   0.1845   1.6706
##
## Coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -127.025    172.241  -0.737   0.4608
## CELL         -14.523     22.082  -0.658   0.5107
## SMEAR        -30.502     34.482  -0.885   0.3764
## INFIL         41.887     39.068   1.072   0.2836
## LI             9.476      5.148   1.841   0.0657 .
## BLAST         -5.135      3.552  -1.446   0.1483
## TEMP         128.892    172.806   0.746   0.4557
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26.734  on 20  degrees of freedom
## Residual deviance: 14.189  on 14  degrees of freedom
## AIC: 28.189
##
## Number of Fisher Scoring iterations: 6
```

We use the parameter "*family = binomial*" in the function **glm**. The binomial argument refers to the binomial law and therefore to the fact that the value we are trying to predict can only take two forms (0 or 1).

The significance of the coefficients is interpreted in the same way as for the classic linear model. Here, we note that all the variables are not significant, except for the **LI** variable. Model quality is assessed according to the **AIC** criterion, which is analogous to the **BIC** criterion (note that there is a **AICc**, *i.e.* a **AIC** corrected for small sample sizes).

Note: This analysis needs to be qualified, given the small number of examples available for this analysis. In particular, the **AIC** criterion can be used to perform model selection, i.e. to build a simpler model containing only a limited number of variables.

```
mymodel_bis = glm(REMISS ~ CELL+LI, data=train, family=binomial)
summary(mymodel_bis)
```

```
##
## Call:
## glm(formula = REMISS ~ CELL + LI, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.0137  -0.5909  -0.3507   0.4643   1.5281
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -11.377      8.164  -1.393   0.1635
## CELL           6.704      7.549   0.888   0.3745
## LI             4.309      2.004   2.150   0.0315 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26.734  on 20  degrees of freedom
## Residual deviance: 17.562  on 18  degrees of freedom
## AIC: 23.562
##
## Number of Fisher Scoring iterations: 6
```

this new model has a lower **AIC** compared to the complete model and is thus better.

Prediction abilities of the model We now want to know whether our model is capable of making good predictions. We'll start by looking at (i) whether it has managed to learn anything from our data. So we're testing its performance on the train. Finally, (ii) if it is able to generalize and therefore test its performance on test data it didn't encounter during training. We'll carry out our study on our reduced model.

- **Train**: let us have a look at the prediction on the *train* set

```
predictTrain = predict(mymodel_bis,newdata=train, type="response")

summary(predictTrain)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.001374 0.071327 0.226925 0.333333 0.516728 0.934154

tapply(predictTrain, train$REMISS, mean)

##         0         1
## 0.2060604 0.5878791
```

- **Test** : let us do the same on the *test* set

```
predictTest = predict(mymodel_bis,newdata=test, type="response")

summary(predictTest)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.007655 0.023762 0.285611 0.420586 0.849207 0.971087

tapply(predictTest, test$REMISS, mean)

##         0         1
## 0.2589253 0.7439072
```

We see that the predictions returned are probabilities of being in remission. By default, we assume that if the probability of being in remission is greater than 0.5, then the model predicts that it is more likely to be in remission (than not).

To evaluate the model's performance, we'll now compare the model's predictions with the true values. We'll draw up a so-called *confusion matrix*..

```
true_label = test$REMISS
predicted_label = 1*(predictTest>0.5)
res = table(actual = true_label, predict = predicted_label)
res

##       predict
## actual 0 1
##      0 3 1
##      1 0 2
```

The table can be read as follows: *2 individuals in remission have been found by the model, 1 individual has been predicted as having entered remission when this is not the case, and 3 individuals who are not in remission have been predicted as not having entered remission.*

Obviously, if we change the threshold for class assignment, this contingency table (another name for the confusion matrix) will also be modified.

```
true_label = test$REMISS
predicted_label = 1*(predictTest>0.2)
res_other = table(actual = true_label, predict = predicted_label)
res_other
```

```
##       predict
## actual 0 1
##      0 3 1
##      1 0 2
```

Towards the analysis of performances We can begin by evaluating the model's performance by assessing its rate of correct classification. This is also known as Accuracy.

```
acc = sum(diag(res)) / sum(res)
acc
```

```
## [1] 0.8333333
```

We can also define measures such as the model's specificity or sensitivity. The *sensitivity* or *recall* is a quantity that measures the proportion of positives (in this case, remissions) found by the model. Specificity is a quantity that measures the proportion of negatives (in this case, non-remission) found by the model.

```
sensitivity = res[2,2]/sum(res[2,])
sensitivity
```

```
## [1] 1
```

```
specitivity =  res[1,1]/sum(res[1,])
specitivity
```

```
## [1] 0.75
```

Other criteria, such as the AUC or simply the ROC curve, give an idea of the model's local and global predictive capacities. This measure is particularly useful when the model returns class membership scores.

To avoid overfitting We are going to use another function, which is called *glmnet*. Let us do it together on a little example.

Another case

Try to follow the same procedure in order to establish the best model on the following datasets that can be found in the same repository.

```
### Performing a logistic regression with penalization

# Install the required packages
install.packages("glmnet")
# or
install.packages("glmnet", repos = "https://cran.us.r-project.org")

library(glmnet)

# A toy dataset
data(BinomialExample)
x <- BinomialExample$x
y <- BinomialExample$y

# Perform Cross-validation
cvfit <- cv.glmnet(x, y, type.measure = "AUC", nfolds = 20) #
print(cvfit)

# Make predictions
predict(cvfit, newx = x[1:5,], s = "lambda.min")

# Study the impact of both hyper-parameters
foldid <- sample(1:10, size = length(y), replace = TRUE)
cv1  <- cv.glmnet(x, y, foldid = foldid, alpha = 1)
cv.5 <- cv.glmnet(x, y, foldid = foldid, alpha = 0.5)
cv0  <- cv.glmnet(x, y, foldid = foldid, alpha = 0)

# Draw some graphs
par(mfrow = c(2,2))
plot(cv1); plot(cv.5); plot(cv0)
plot(log(cv1$lambda)   , cv1$cvm , pch = 19, col = "red",
     xlab = "log(Lambda)", ylab = cv1$name)
points(log(cv.5$lambda), cv.5$cvm, pch = 19, col = "grey")
points(log(cv0$lambda) , cv0$cvm , pch = 19, col = "blue")
legend("topleft", legend = c("alpha= 1", "alpha= .5", "alpha 0"),
       pch = 19, col = c("red","grey","blue"))
```

References

[Cox, 1958] Cox, D. R. (1958). The regression analysis of binary sequences (with discussion). *Journal of the Royal Statistical Society*, 20:215–242.

[Cramer, 2003] Cramer, J. (2003). The origins of logistic regression. *SSRN Electronic Journal*.