Linear regression

Diane Lingrand

UNIVERSITÉ
CÔTE D'AZUR
Master Data Science M1

2020 - 2021

Outline

1 Linear regression

2 Logistic regression

3 Evaluation (focused on binary classification)

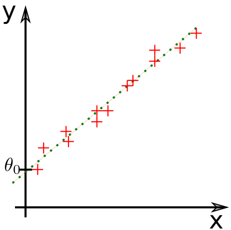
Data

- data :
  - scalar value : x
  - many scalar values : vector : $\mathrm{x} = [x_0 x_1 ... x_n]$
- we want to predict y value
- Examples :
  - amount of rain from altitude in the Alps
  - price of an appartment from size in m^2
  - vote from age, sex, income, residence location, ...
  - risk of a disease from age, weight, result of blood analysis, ...

Linear regression

- Regression : determine value of y with respect to x.
- Linear : the function is a line parameterised by $\theta = [\theta_0 \theta_1]$:

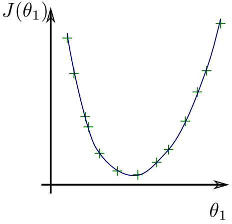
$$y = \theta_0 + x * \theta_1$$



- Learning : determine θ_0 et θ_1
- Regression : compute $y = h_\theta(x) = \theta_0 + x * \theta_1$
- Cost function (or error) :

$$J(\theta) = \frac{1}{2m} \sum (h_\theta(x^i) - y^i)^2$$

- Cost minimisation $J(\theta)$
  - Let us consider cases where $\theta_0 = 0$: determine θ_1 that minimises $J(\theta)$

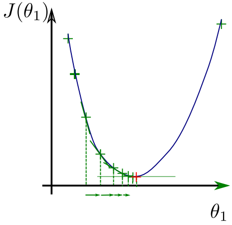


    - Exhaustive search ?
    - Gradient descent

Gradient descent (one variable)

- Find extrema of J : find the zeros of $\frac{dJ}{d\theta_1}$
- Iterative algorithm :
  - pick initial value of θ_1
  - while $J(\theta_1)$ changes (stop when $\frac{dJ}{d\theta_1}(\theta_1) \simeq 0$) :
    - replace θ_1 by $\theta_1 - \alpha \frac{dJ}{d\theta_1}$ ($\alpha > 0$, small)

- α : learning rate
- α has to be chosen carefully :
  - if too small : slow convergence
  - if too big : oscillations
  - $\Rightarrow$ plot $J(\theta)$



- many scalar values : vector x $= [x_1 ... x_n]$
- linear model : $y = \theta_0 + x_1\,\theta_1 + x_2\,\theta_2 + \ldots + x_n\,\theta_n$
- Find extrema of J : find the zeros of $\frac{dJ}{d\theta}$
- Iterative algorithm :
  - pick initial value of $\theta = [\theta_0 ... \theta_n]$
  - while $J(\theta)$ changes (stop when $\frac{dJ}{d\theta}(\theta) \simeq 0$) :
    - replace each θ_i by $\theta_i - \alpha\,\frac{dJ}{d\theta_i}$ ($\alpha > 0$, small)

Gradient descent : many data

- batch gradient descent
  - all training samples for each step
- stochastic gradient descent
  - one training sample for each step (need to shuffle training data)
- mini batch (b=10)
  - a set of b training samples for each step

Outline

1 Linear regression

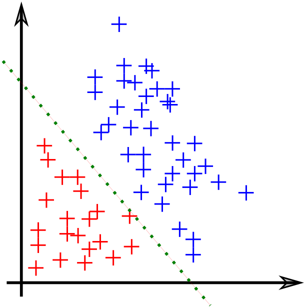
2 Logistic regression

3 Evaluation (focused on binary classification)

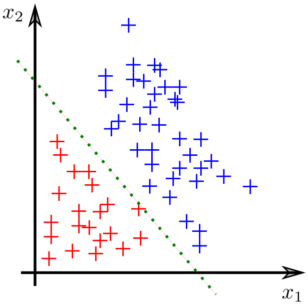
Context

- Supervised learning = learn to predict an output when given an input vector
- We know the class / label y for all training data x
- Logistic regression is a classification method
  - Linear regression leads to values $h_\theta(x) \in \mathcal{R}$
  - the idea : values in [0 1] then thresholding at 0.5
- Also known as logit regression or maximum-entropy classification

- Data separation according to labels (0 or 1).
  - Linear separation : line, plane or hyperplane
  - Non-linear separation : polynomial or gaussian
- Notations :
  - data : $\mathrm{x} = [x_1 \ x_2 ...]$
  - labels : $y \in \{0, 1\}$
  - decision criteria h_θ parameterised by θ
    - $\theta = [\theta_0 \ \theta_1 \ ...]$



Linear separation



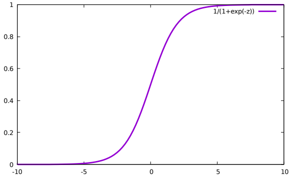
- Decision boundary :
  - line of equation : $\theta_0 + \theta_1\, x_1 + \theta_2\, x_2 = 0$
  - also written as : $\theta^T \mathrm{x} = 0$
- Decision :
  - if $\theta^T \mathrm{x} \geq 0$ then $y = 1$
  - if $\theta^T \mathrm{x} < 0$ then $y = 0$

Logistic function

$$h_\theta(\mathrm{x}) = s(\theta^T \mathrm{x})$$

with :

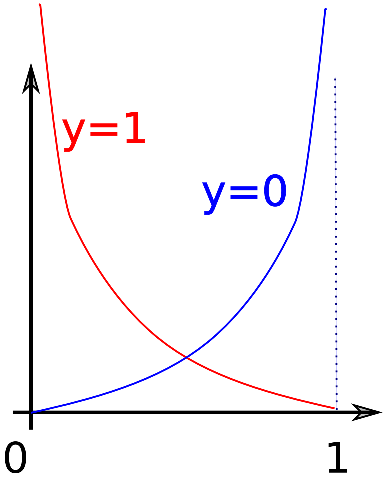
$$s(z) = \frac{1}{1 + e^{-z}}$$



- The decision is :
  - if $h_\theta(\mathrm{x}) \geq 0.5$ then $y = 1$
  - if $h_\theta(\mathrm{x}) < 0.5$ then $y = 0$

- data : $(\mathrm{x}^{(1)}, y^{(1)}), (\mathrm{x}^{(2)}, y^{(2)}), ..., (\mathrm{x}^{(m)}, y^{(m)})$
- m data
- learning aims at finding θ
- method :
  - error minimisation
  - gradient descent (or other minimisation method)

$$J = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(h_\theta(\mathrm{x}^{(\mathrm{i})})) + (1-y^{(i)})\log(1-h_\theta(\mathrm{x}^{(\mathrm{i})})))$$



For all components θ_j de θ :

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

with

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)}) x_j^{(i)}$$

Non linear logistic regression

- polynomial model :

$$x = [x_0 x_1 \dots x_n x_0^2 x_1^2 \dots x_0 x_1 x_0 x_2 \dots$$

- under-fitting
    - add parameters
- over-fitting
    - reduce the number of parameters

Regularisation

- in order to avoid over-fitting
- add $\|\theta\|$ to the cost :

$$J = \|\theta\| - C\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(h_\theta(\mathrm{x}^{(\mathrm{i})})) + (1 - y^{(i)})\log(1 - h_\theta(\mathrm{x}^{(\mathrm{i})})))$$

- norm : $\mathcal{L}_1$ (Lasso), $\mathcal{L}_2$ (Ridge), Elastic-Net ($\frac{1-\rho}{2}\|\theta\|_2^2 + \rho\|\theta\|_1$)
- many solvers :
  - Coordinate Descent (C++ LIBLINEAR library),
  - Stochastic Average Gradient (SAG) or variant SAGA : good for large dataset
  - Broyden–Fletcher–Goldfarb–Shanno algorithm (small data set only) : family of Newton algorithm
- more details on `https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`

Logistic regression using OpenCV

From `https://docs.opencv.org/3.0-last-rst/modules/ml/doc/logistic_regression.html` :
A sample set of training parameters for the Logistic Regression classifier can be initialized as follows :

- `LogisticRegression::Params params;`
- `PARAMS.ALPHA = 0.5;`
- `PARAMS.NUM_ITERS = 10000;`
- `PARAMS.NORM = LOGISTICREGRESSION::REG_L2;`
- `PARAMS.REGULARIZED = 1;`
- `PARAMS.TRAIN_METHOD = LOGISTICREGRESSION::MINI_BATCH;`
- `PARAMS.MINI_BATCH_SIZE = 10;`

Logistic regression using scikit-learn

Description at : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html A toy example :

```
#logistic regression object creation
logisticRegr = LogisticRegression()
#learning
logisticRegr.fit(bows, labels)
#predicted labels computation
labelsPredicted = logisticRegr.predict(bows)
#score computation and display
score = logisticRegr.score(bows, labels)
print("train score = ", score)
#objet saving
with open('sauvegarde.logr', 'wb') as output:
   pickle.dump(logisticRegr, output, pickle.HIGHEST_PROTOCOL)
#object loading
with open('sauvegarde.logr',  'rb') as input:
   logisticRegr = pickle.load(input)
```

Outline

1 Linear regression

2 Logistic regression

3 Evaluation (focused on binary classification)

Splitting the dataset

- Training data : for learning
  - contains both positives and negatives samples
- Validation data
  - for learning stopping
  - for hyperparameters learning
- Test data
  - evaluation of the performances of the classification
  - contains both positives and negatives samples with the same ratio as in the train dataset
- Ratio : usually 60% 20% 20% or 80% 0% 20%
- If the size of the data set is small :
  - cross-validation (k-fold)

True false positives negatives

true positives (TP) : positive samples computed as positives

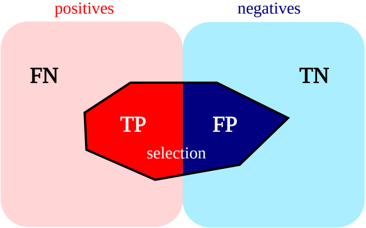
true negatives (TN) : negatives samples computed as negatives

false positives (FP) : negatives samples computed as positives

false negatives (FN) : positive samples computed as negatives

Confusion matrix :

		estimated classes	
		cat	no cat
true classes	cat	9	1
	no cat	1	9



True false positives negatives

true positives (TP) : positive samples computed as positives

true negatives (TN) : negatives samples computed as negatives

false positives (FP) : negatives samples computed as positives

false negatives (FN) : positive samples computed as negatives

sensibility, recall, TP rate : $\frac{TP}{TP+FN}$

specificity, selectivity, TN : $\frac{TN}{TN+FP}$

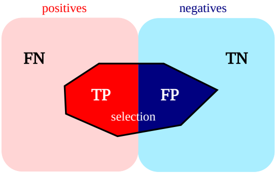
precision, positive predictive value : $\frac{TP}{TP+FP}$

negative predictive value : $\frac{TN}{TN+FN}$

F-mesure (*F1 score*) :

$$2\frac{\text{precision.recall}}{\text{precision}+\text{recall}} = \frac{2TP}{2TP+FP+FN}$$

accuracy : $\frac{TP+TN}{P+N} = \frac{TP+TN}{TP+FP+TN+FN}$



- ROC : Receiver Operating Characteristic
- AUC : Area Under Curve