Linear and Logistic Regression

Ramon Ferrer-i-Cancho
rferrericancho@upc.edu
(Marta Arias)

Dept. CS, UPC

Fall 2021

$$y = a_1 * x_1 + a_2 * x_2 + ... + b$$

x_i are the attributes, y is the target value
a_i and b are the coefficients or parameters of the linear model
For example:

$$house\_price = 2 * area + 0.5 * proximity\_metro + 150$$

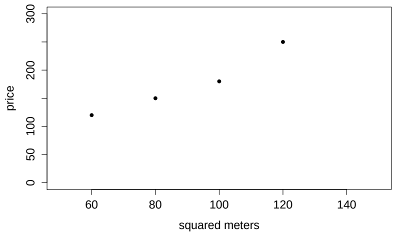
or

$$house\_price = 25 * area - 0.5 * proximity\_metro + 1500$$

Linear regression

Example: housing prices

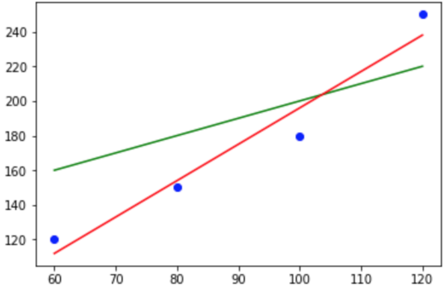
	area	*price*
i	x^i	y^i
1	60	120
2	80	150
3	100	180
4	120	250
	110	?



Linear regression

Example: housing prices

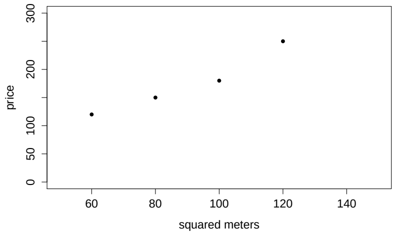
```
In [33]: X = np.array([60, 80, 100,120])
         y = np.array([120,150,180,250])
         plt.plot(X, 1.0*X + 100, 'g')
         plt.plot(X, 2.1*X -14, 'r')
         plt.plot(X, y, 'bo')

Out[33]: [<matplotlib.lines.Line2D at 0x119834ef0>]
```



Linear regression

Example: housing prices

	area	*price*
i	x^i	y^i
1	60	120
2	80	150
3	100	180
4	120	250
	110	?



Want to find the line that best fits the available data

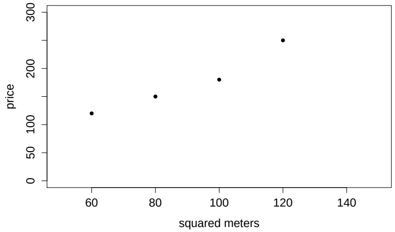
find parameters a and b such that $ax^i + b$ is closest to y^i (for all i simultaneously), e.g. *minimize squared error*:

$$\arg\min_{a,b} \sum_i (ax^i + b - y^i)^2$$

Linear regression

Example: housing prices

	area	price
i	x^i	y^i
1	60	120
2	80	150
3	100	180
4	120	250
	110	?



In this case, we seek parameters (a, b) that minimize $J(a, b) = \sum_i (ax^i + b - y^i)^2$

$$\begin{aligned} J(a,b) &= (60a + b - 120)^2 \\ &+ (80a + b - 150)^2 \\ &+ (100a + b - 180)^2 \\ &+ (120a + b - 250)^2 \end{aligned}$$

$$\begin{aligned} J(a = 2.1, b = -14) &= 480 \\ J(a = 2.1, b = -10) &= 544 \\ J(a = 2.0, b = -14) &= 824 \\ J(a = -2.1, b = -14) &= 607296 \end{aligned}$$

Linear regression

Simple case: $\mathbb{R}^2$

Here is the idea:

1. Got a bunch of points in $\mathbb{R}^2$, $\{(x^i, y^i)\}$.
2. Want to fit a line $y = ax + b$ that describes the trend.
3. We define a *cost function* that computes the total squared error of our predictions w.r.t. observed values y^i $J(a, b) = \sum_i (ax^i + b - y^i)^2$ that we want to minimize.
4. See it as a function of a and b: compute both derivatives, force them equal to zero, and solve for a and b.
5. The coefficients you get give you the minimum squared error.
6. More general version in $\mathbb{R}^n$.

Linear regression

OK, so let's find those minima

Find parameters (a, b) that minimize $J(a, b)$

$$
\begin{aligned}
J(a,b) &= (60a+b-120)^2 && +(80a+b-150)^2 && +(100a+b-180)^2 && +(120a+b-250)^2 \\
\frac{\partial J(a,b)}{\partial a} &= 2(60a+b-120)60 && +2(80a+b-150)80 && +2(100a+b-180)100 && +2(120a+b-250)120 \\
\frac{\partial J(a,b)}{\partial b} &= 2(60a+b-120) && +2(80a+b-150) && +2(100a+b-180) && +2(120a+b-250)
\end{aligned}
$$

Linear regression

OK, so let's find those minima

Set $\frac{\partial J(a,b)}{\partial a} = 0$

$$\frac{\partial J(a,b)}{\partial a} = 0 \iff$$
$$2\{(60a+b-120)60 + (80a+b-150)80 + (100a+b-180)100 + (120a+b-250)120\} = 0 \iff$$
$$(60a+b-120)60 + (80a+b-150)80 + (100a+b-180)100 + (120a+b-250)120 = 0 \iff$$

$$(60a+b)60 + (80a+b)80 + (100a+b)100 + (120a+b)120 = 120*60 + 150*80 + 180*100 + 250*120 \iff$$
$$(60^2+80^2+100^2+120^2)a + (60+80+100+120)b = 120*60 + 150*80 + 180*100 + 250*120 \iff$$
$$34400a + 360b = 67200$$

Linear regression

OK, so let's find those minima

Set $\frac{\partial J(a,b)}{\partial b} = 0$

$$\begin{aligned}
\frac{\partial J(a,b)}{\partial b} = 0 &\iff \\
2\{(60a + b - 120) + (80a + b - 150) + (100a + b - 180) + (120a + b - 250)\} = 0 &\iff \\
(60a + b - 120) + (80a + b - 150) + (100a + b - 180) + (120a + b - 250) = 0 &\iff
\end{aligned}$$

$$\begin{aligned}
(60a + b) + (80a + b) + (100a + b) + (120a + b) = 120 + 150 + 180 + 250 &\iff \\
(60 + 80 + 100 + 120)a + (1 + 1 + 1 + 1)b = 120 + 150 + 180 + 250 &\iff \\
360a + 4b = 700
\end{aligned}$$

Linear regression

OK, so let's find those minima

Finally, solve system of linear equations

$$
\begin{aligned}
34400a + 360b &= 67200 \\
360a + 4b &= 700
\end{aligned}
$$

Linear regression

Simple case: $\mathbb{R}^2$ now in general!

Let $h(x) = ax + b$, and $J(a,b) = \sum (h(x^i) - y^i)^2$

$$\begin{aligned}
\frac{\partial J(a,b)}{\partial a} &= \frac{\partial \sum_i (h(x^i) - y^i)^2}{\partial a} \\
&= \sum_i \frac{\partial (ax^i + b - y^i)^2}{\partial a} \\
&= \sum_i 2(ax^i + b - y^i) \frac{\partial (ax^i + b - y^i)}{\partial a} \\
&= 2 \sum_i (ax^i + b - y^i) \frac{\partial (ax^i)}{\partial a} \\
&= 2 \sum_i (ax^i + b - y^i) x^i
\end{aligned}$$

Linear regression

Simple case: $\mathbb{R}^2$

Let $h(x) = ax + b$, and $J(a,b) = \sum (h(x^i) - y^i)^2$

$$
\begin{aligned}
\frac{\partial J(a,b)}{\partial b} &= \frac{\partial \sum_i (h(x^i) - y^i)^2}{\partial b} \\
&= \sum_i \frac{\partial (ax^i + b - y^i)^2}{\partial b} \\
&= \sum_i 2(ax^i + b - y^i) \frac{\partial (ax^i + b - y^i)}{\partial b} \\
&= 2 \sum_i (ax^i + b - y^i) \frac{\partial (b)}{\partial b} \\
&= 2 \sum_i (ax^i + b - y^i)
\end{aligned}
$$

Linear regression

Simple case: $\mathbb{R}^2$

Normal equations

Given $\{(x^i, y^i)\}_i$, solve for a, b:

$$\begin{aligned} \sum_i (ax^i + b)x^i &= \sum_i x^i y^i \\ \sum_i (ax^i + b) &= \sum_i y^i \end{aligned}$$

In our example:

$\{(x^i, y^i)\}_i = \{(60, 120), (80, 150), (100, 180), (120, 250)\}$ and so the normal equations are:

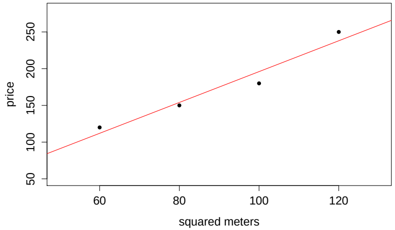
$$\begin{aligned} 34.400a + 360b &= 67.200 \\ 360a + 4b &= 700 \end{aligned}$$

solving for a and b gives: $a = 2.1$ and $b = -14$.

Linear regression

Example: housing prices

i	*area in* m^2	*price*
1	60	120
2	80	150
3	100	180
4	120	250
	110	217



Best linear fit: $a = 2.1$, $b = -14$

So best guessed price for a home of 110 sq m is

$2.1 \times 110 - 14 = 217$

Linear regression

General case: $\mathbb{R}^n$

i	*area in* m^2	*location quality*	*distance to metro*	*price*
1	60	75	0.3	120
2	80	60	2	150
3	100	48	24	180
4	120	97	4	250

- Now, each $\mathrm{x}^i = \langle x_1^i, x_2^i, .., x_n^i \rangle$ so e.g. $\mathrm{x}^1 = \langle 60, 75, 0.3 \rangle$
- So: $\mathrm{X} = \begin{pmatrix} 60 & 75 & 0.3 \\ 80 & 60 & 2 \\ 100 & 48 & 24 \\ 120 & 97 & 4 \end{pmatrix}$ and $\mathrm{y} = \begin{pmatrix} 120 \\ 150 \\ 180 \\ 250 \end{pmatrix}$
- Model parameters are $a_1, .., a_n, b$ and so prediction is $a_1 * x_1 + \ldots a_n * x_n + b$, in short $\mathrm{ax} + b$
- Cost function is $J(\mathrm{a}, b) = \sum_i (\mathrm{ax}^i + b - y^i)^2$

Linear regression

Practical example with *scikit-learn*

We have a dataset with data for 20 cities; for each city we have information on:

- Nr. of inhabitants (in 10^3)
- Percentage of families' incomes below 5000 USD
- Percentage of unemployed
- Number of murders per 10^6 inhabitants per annum

	inhabitants	*income*	*unemployed*	*murders*
1	587	16.50	6.20	11.20
2	643	20.50	6.40	13.40
3	635	26.30	9.30	40.70
4	692	16.50	5.30	5.30
⋮	⋮	⋮	⋮	⋮
20	3353	16.90	6.70	25.70

We wish to perform regression analysis on the number of murders based on the other 3 features.

Linear regression

Practical example with *scikit-learn*

In [56]:
```python
import pandas as pd
murders = pd.read_csv('data/murders.txt', sep=" ")
```

In [57]:
```python
murders.head()
```

Out[57]:

	inhabitants	income	unemployment	murders
0	587	16.5	6.2	11.2
1	643	20.5	6.4	13.4
2	635	26.3	9.3	40.7
3	692	16.5	5.3	5.3
4	1248	19.2	7.3	24.8

In [58]:
```python
attributes = ['inhabitants','income','unemployment']
X = murders[attributes]
y = murders['murders']
```

In [59]:
```python
from sklearn.linear_model import LinearRegression
linreg = LinearRegression().fit(X, y)
```

In [60]:
```python
print("Linear model intercept: {}".format(linreg.intercept_))
print("Linear model coefficients: {}".format(linreg.coef_))
```

```
Linear model intercept: -36.764925281988475
Linear model coefficients: [7.62936937e-04 1.19217421e+00 4.71982137e+00]
```

Ridge regression

Introducing *regularization*

We modify the cost function so that linear models with very large coefficients are penalized:

$$J_{ridge}(\mathrm{a}, b) = \underbrace{\sum_i (\mathrm{a}\mathrm{x}^i + b - y^i)^2}_{\text{fit to data}} + \quad \alpha * \quad \underbrace{\sum_j a_j^2}_{\text{model complexity}}$$

- Regularization helps in preventing *overfitting* since it controls model complexity.
- α is a hyperparameter controlling how much we regularize: higher α means more regularization and simpler models

Ridge regression

Practical example with *scikit-learn*

```
from sklearn.linear_model import Ridge
linridge10 = Ridge(alpha = 10).fit(X,y)
print("Linear model intercept: {}".format(linridge10.intercept_))
print("Linear model coefficients: {}".format(linridge10.coef_))
```

```
Linear model intercept: -32.34208354746895
Linear model coefficients: [5.74094699e-04 1.75869043e+00 2.51017024e+00]
```

```
for a in [0,10,100,1000]:
    lr = Ridge(alpha = a).fit(X, y)
    print("Linear model intercept with alpha = {}: {}".format(a, lr.intercept_))
    print("Linear model coefficients with alpha = {}: {}".format(a, lr.coef_))
```

```
Linear model intercept with alpha = 0: -36.764925282007134
Linear model coefficients with alpha = 0: [7.62936937e-04 1.19217421e+00 4.71982137e+00]
Linear model intercept with alpha = 10: -32.34208354746895
Linear model coefficients with alpha = 10: [5.74094699e-04 1.75869043e+00 2.51017024e+00]
Linear model intercept with alpha = 100: -16.04743760553152
Linear model coefficients with alpha = 100: [2.29738002e-04 1.55411861e+00 8.13410812e-01]
Linear model intercept with alpha = 1000: 11.591883435274537
Linear model coefficients with alpha = 1000: [-2.32827795e-04  4.14453690e-01  1.64200724e-01]
```

Remember that the cost function in ridge regression is:

$$J_{ridge}(\mathrm{a}, b) = \sum_i (\mathrm{ax}^i + b - y^i)^2 + \quad \alpha * \sum_j a_j^2$$

If features x_j are in different *scales* then they will contribute differently to the penalty of this cost function, so we want to bring them to the *same scale* so that this does not happen (this also true for many other learning algorithms)

Feature normalization with *scikit-learn*

Example using the *MinMaxScaler* (there are others, of course)

One possibility is to turn all features into 0-1 range by doing the following transformation: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
pd.DataFrame(data = X_scaled, columns = list(X)).head()
```

	inhabitants	income	unemployment
0	0.000000	0.183333	0.295455
1	0.007663	0.516667	0.340909
2	0.006568	1.000000	1.000000
3	0.014368	0.183333	0.090909
4	0.090449	0.408333	0.545455

Lasso regression

We modify again the cost function so that linear models with very large coefficients are penalized:

$$J_{lasso}(\mathsf{a}, b) = \underbrace{\sum_i (\mathsf{a}x^i + b - y^i)^2}_{\text{fit to data}} + \quad \alpha * \quad \underbrace{\sum_j |a_j|}_{\text{model complexity}}$$

- Note that the penalization uses *absolute value* instead of squares.
- This has the effect of setting parameter values to 0 for the least influential variables (like doing some *feature selection*)

Lasso regression

Practical example with *scikit-learn*

```
from sklearn.linear_model import Lasso
lasso10 = Lasso(alpha = 10).fit(X,y)
print("Linear model intercept: {}".format(lasso10.intercept_))
print("Linear model coefficients: {}".format(lasso10.coef_))
```

```
Linear model intercept: -10.453255579994416
Linear model coefficients: [9.53798931e-05 1.56625640e+00 0.00000000e+00]
```

```
for a in [0,10,100,1000]:
    lr = Lasso(alpha = a).fit(X, y)
    print("Linear model intercept with alpha = {}: {}".format(a, lr.intercept_))
    print("Linear model coefficients with alpha = {}: {}".format(a, lr.coef_))
```

```
Linear model intercept with alpha = 0: -36.764925282007134
Linear model coefficients with alpha = 0: [7.62936937e-04 1.19217421e+00 4.71982137e+00]
Linear model intercept with alpha = 10: -10.453255579994416
Linear model coefficients with alpha = 10: [9.53798931e-05 1.56625640e+00 0.00000000e+00]
Linear model intercept with alpha = 100: 21.075703617803477
Linear model coefficients with alpha = 100: [-0.0003529  0.         0.        ]
Linear model intercept with alpha = 1000: 20.608005656037506
Linear model coefficients with alpha = 1000: [-2.65217418e-05  0.00000000e+00  0.00000000e+00]
```

Logistic regression

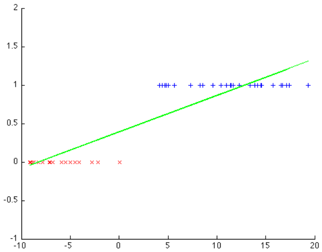
What if $y^i \in \{0, 1\}$ instead of continuous real value?

Disclaimer

Even though logistic regression carries *regression* in its name, it is specifically designed for *classification*

Binary classification

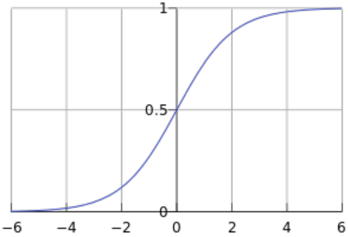
Now, datasets are of the form $\{(\mathrm{x}^1, 1), (\mathrm{x}^2, 0), ..\}$. In this case, linear regression will not do a good job in classifying examples as *positive* ($y^i = 1$), or *negative* ($y^i = 0$).



Logistic regression

Hypothesis space

- $h_{\mathrm{a},b}(\mathrm{x}) = g(\sum_{j=1}^{n} a_j x_j + b) = g(\mathrm{ax} + b)$
- $g(z) = \frac{1}{1+e^{-z}}$ is sigmoid function (a.k.a. logistic function)
  - $0 \leqslant g(z) \leqslant 1$, for all $z \in \mathbb{R}$
  - $\lim_{z \to -\infty} g(z) = 0$ and $\lim_{z \to +\infty} g(z) = 1$
  - $g(z) \geqslant 0.5$ iff $z \geqslant 0$
- Given example x
  - predict *positive* iff $h_{\mathrm{a},b}(\mathrm{x}) \geqslant 0.5$ iff $g(\mathrm{ax} + b) \geqslant 0.5$ iff $\mathrm{xa} + b \geqslant 0$



Logistic regression

Optimization for logistic regression

Let us assume that

- $P(y = 1|x; \mathrm{a}, b) = h_{\mathrm{a},b}(\mathrm{x})$, and so
- $P(y = 0|x; \mathrm{a}, b) = 1 - h_{\mathrm{a},b}(\mathrm{x})$

Given m training examples $\{(\mathrm{x}^i, y^i)\}_i$ where $y^i \in \{0, 1\}$ we compute the likelihood (assuming independence of training examples)

$$\begin{aligned} L(\mathrm{a}, b) &= \prod_i p(y^i|\mathrm{x}^i; \mathrm{a}, b) \\ &= \prod_i h_{\mathrm{a},b}(\mathrm{x}^i)^{y^i} \, (1 - h_{\mathrm{a},b}(\mathrm{x}^i))^{1-y^i} \end{aligned}$$

Our strategy will be to maximize the log likelihood:

$$\begin{aligned} \log L(\mathrm{a}, b) &= \sum_i y^i \log h_{\mathrm{a},b}(\mathrm{x}^i) + (1 - y^i) \log(1 - h_{\mathrm{a},b}(\mathrm{x}^i)) \\ &= \sum_i y^i \log g(\mathrm{a}\mathrm{x}^i + b) + (1 - y^i) \log(1 - g(\mathrm{a}\mathrm{x}^i + b)) \end{aligned}$$

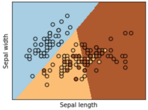
Logistic regression

Practical example with *scikit-learn*[1]

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

logreg = LogisticRegression(C=1e5, solver='lbfgs', multi_class='multinomial')

# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X, Y)
```

```
LogisticRegression(C=100000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='multinomial', n_jobs=None, penalty='l2',
          random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
          warm_start=False)
```

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



1