Logistic Regression

Michela Papandrea

University of Applied Sciences and Arts of Southern Switzerland

michela.papandrea@supsi.ch

Overview

1. Logistic Regression
   - Estimating Probabilities
   - Training and Cost Functions
   - Decision Boundaries
   - Softmax Regression

Logistic Regression

Logistic Regression

- some regression algorithms can be used for classification as well.
- **Logistic Regression** (also called **Logit Regression**) is commonly used to *estimate the probability that an instance belongs to a particular class*

  $(\hat{p} > 50\%) \Rightarrow$ TRUE CLASS

  - if the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled "1")
  - else it predicts that it does not (i.e., it belongs to the negative class, labeled "0").
  - This makes it a binary classifier. $\rightarrow$ 1/0 TRUE/FALSE

Estimating Probabilities

- Logistic Regression computes a weighted sum of the input features (plus a bias term)
- instead of outputting the result directly like the Linear Regression model does, it outputs the *logistic of this result*

$$\overset{[0,1]}{\hat{p}} = h_{\theta}(\mathbf{x}) = \sigma(\overset{[-\infty,+\infty]}{\boxed{\mathbf{x}^T\theta}})$$

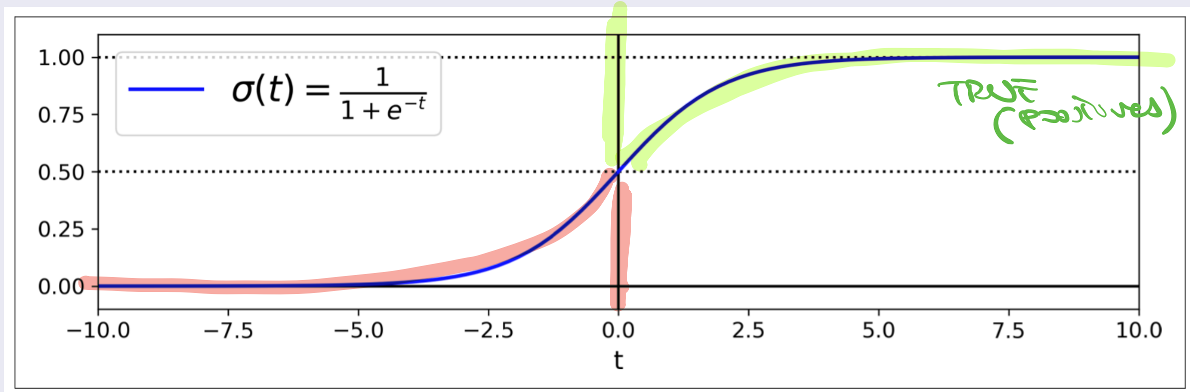
$$\hat{y} = [1\ x_1\ x_2 \ldots x_n] \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \hat{y}$$

Sigmoid Function

Sigmoid Function

S-shaped function that outputs a number between 0 and 1

$$\hat{p} = \sigma(t) = \frac{1}{1 + e^{-t}}$$



Once the Logistic Regression model has estimated the probability $p = h_{\theta}(\mathbf{x})$ that an instance $\mathbf{x}$ belongs to the positive class, it can make its prediction $\hat{y}$ easily

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

Sigmoid function

Notice that

- $\sigma(t) < 0.5$ when $t < 0$
- $\sigma(t) \geq 0.5$ when $t \geq 0$

so a Logistic Regression model predicts

- 1 if $\mathbf{x}^T\boldsymbol{\theta}$ is positive
- 0 if $\mathbf{x}^T\boldsymbol{\theta}$ is negative.

$\boxed{t = x^T\theta}$

sigmoid

$$t \underset{\log\left(\frac{p}{1-p}\right)}{\overset{\frac{1}{1+e^{-t}}}{\rightleftarrows}} \hat{p}$$

logit

- The score t is often called the **logit**: this name comes from the fact that the logit function, defined as $logit(p) = log(\frac{p}{(1-p)})$, is the inverse of the logistic function
- If you compute the *logit* of the estimated probability p, you will find that the result is t.
- The *logit* is also called the $log - odds$
  $\rightarrow$ it is the *log of the ratio between the estimated probability for the positive class and the estimated probability for the negative class.*

Training and Cost Function

Training

The objective of training is to set the parameter vector $\boldsymbol{\theta}$ so that the model estimates:

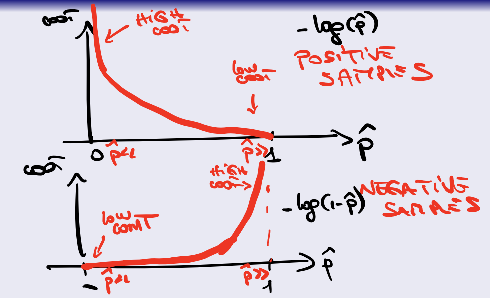
- high probabilities for positive instances ($y = 1$)
- low probabilities for negative instances ($y = 0$).

SPAM → 1 spam
→ 0 non spam

diagnosis → 1 diagnosis TRUE
→ 0 NO diagnosis!

Cost Function for a single training instance **x**

$$c(\boldsymbol{\theta}) = \begin{cases} -log(\hat{p}) & \text{if } y = 1 \\ -log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$



- $-log(t)$ grows very large when t approaches 0, so the cost will be large if the model estimates a probability close to 0 for a positive instance
- it will also be very large if the model estimates a probability close to 1 for a negative instance.
- $-log(t)$ is close to 0 when t is close to 1

$\Rightarrow$ the cost will be close to 0 if the estimated probability is close to 0 for a negative instance or close to 1 for a positive instance

Cost Function

START 7/11/23

Cost Function for the whole training set - *log-loss*

The cost function over the whole training set is the average cost over all training instances

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[\overbrace{y^{(i)} log(\underbrace{\hat{p}^{(i)})}_{P(\text{posit.})}}^{\text{if } y=1} + \overbrace{(1-y^{(i)}) log(\underbrace{1-\hat{p}^{(i)}}_{P(\text{negative})})}^{\text{if } y=0}\right]$$

$y \rightarrow 1$, $y \rightarrow 0$

(−) There is no known closed-form equation to compute the value of $\boldsymbol{\theta}$ that minimizes this cost function

(+) This cost function is convex, so Gradient Descent (or any other optimization algorithm) is guaranteed to find the global minimum

- if the learning rate is not too large and you wait long enough

Cost Function

The partial derivatives of the cost function with regards to the j^{th} model parameter θ_j is

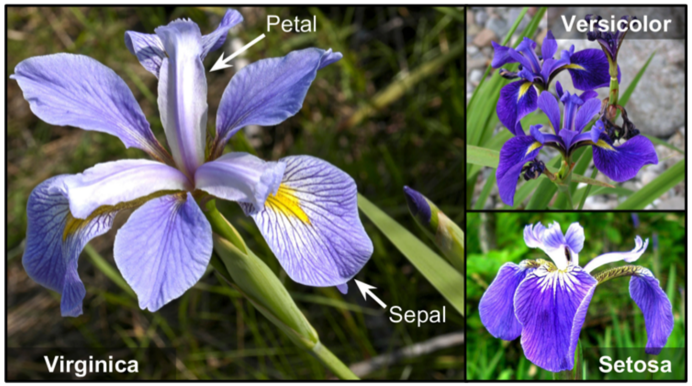
$y = 1 \rightarrow \hat{p} \gg \Rightarrow$ RESIDUAL SMALL
0,98
$\hat{p} \ll \Rightarrow$ RESIDUAL HIGH
0,1

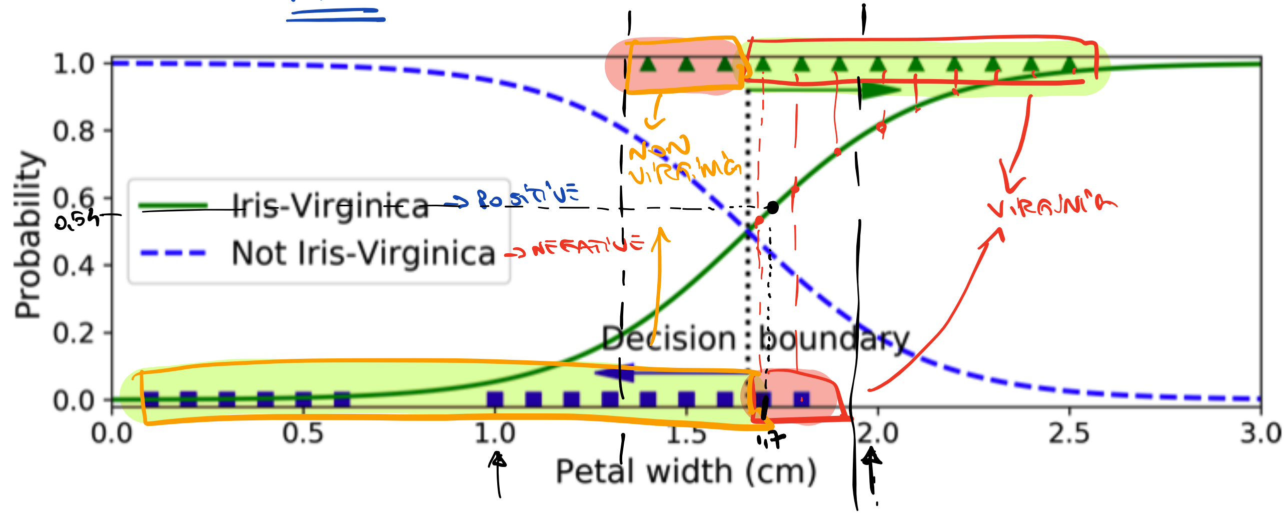
$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (\underbrace{\sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)})}_{(\hat{p}} - \underbrace{y^{(i)}}_{y)}) x_j^{(i)}$$

- For each instance it computes the prediction error and multiplies it by the i^{th} feature value, and then it computes the average over all training instances.
- Once you have the gradient vector containing all the partial derivatives you can use it in the Batch Gradient Descent algorithm.
- For Stochastic GD you would of course just take one instance at a time
- For Mini-batch GD you would use a mini-batch at a time.

Decision Boundaries

TASK: is the flower an Iris-Virginica?






```python
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

# load the datset
iris = datasets.load_iris()
X = iris["data"][:, 3:] # petal width
y = (iris["target"] == 2).astype(np.int) # 1 if Iris-Virginica, else 0

#train a logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X, y)

#make a prediction
print("Class prediction = {}".format(log_reg.predict([[1.7]])))
print("Probability prediction for all classes = {}".format(log_reg.predict_proba([[1.7]])))
```

$\hat{p} = \sigma(X^T\theta)$

sigmoid

linear regr

```
Class prediction = [1]
Probability prediction for all classes = [[0.45722097 0.54277903]]
```

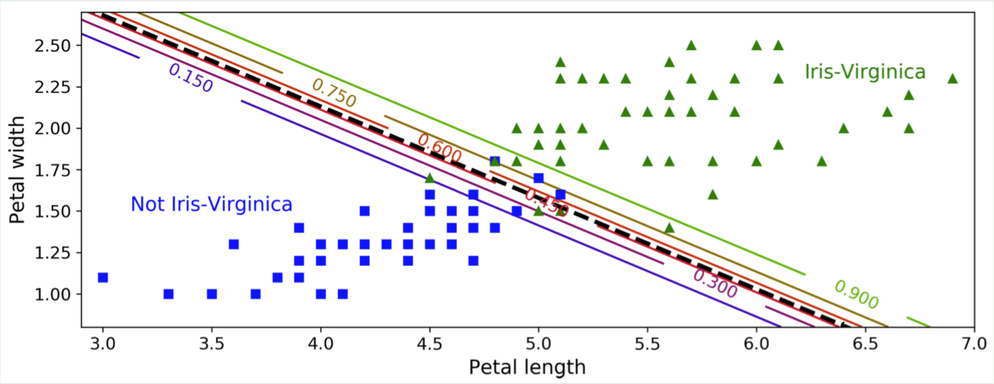
P(false) P(True)

P(0) P(1)

Decision Boundaries

- The petal width of Iris-Virginica flowers (represented by triangles) ranges in $[1.4 - 2.5]$ cm
- The other iris flowers (represented by squares) generally have a smaller petal width, ranging in $[0.1, 1.8]$ cm
  - There is however a bit of overlap.
- Above about 2 cm the classifier is highly confident that the flower is an Iris-Virginica (it outputs a high probability to that class)
- Below 1 cm it is highly confident that it is not an Iris-Virginica (high probability for the "Not Iris-Virginica" class).
- In between these extremes, the classifier is unsure. However it will always predict the class which is the most likely.
- There exist a **decision boundary** at around 1.6 cm where both probabilities are equal to 50%
  - if petal width > 1.6 cm $\Rightarrow$ the classifier will predict that the flower is an Iris-Virginica
  - otherwise it will predict that it is not (even if it is not very confident)

Decision Boundaries



- The figure refers to the same dataset but this time displaying two features: *petal width* and *petal length*
- The Logistic Regression classifier is trained in order to estimate the probability that a new flower is an Iris-Virginica based on these two features.
- **Dashed line** $\rightarrow$ points where the model estimates a 50% probability (**model's decision boundary**, linear boundary in this case) $\rightarrow$set of values **x** such that $\boxed{\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0}$
- Each parallel line represents the points where the model outputs a specific probability, from 15% (bottom left) to 90% (top right).
  - All the flowers beyond the top-right line have an over 90% chance of being Iris-Virginica according to the model.
- Just like the other linear models, Logistic Regression models can be regularized using ℓ_1 or ℓ_2 penalties

Softmax Regression

Softmax Regression (or *Multinomial Logistic Regression*)

- Generalization of the Logistic Regression model to support multiple classes
- When given an instance $\mathbf{x}$, the Softmax Regression model:
  1. computes a **score** $s_k(\mathbf{x})$ for each class k

     $$s_k(\mathbf{x}) = \underbrace{\mathbf{x}^T \boldsymbol{\theta}^{(k)}}$$

     $$x \qquad \hat{S} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{bmatrix} \begin{array}{l} \leftarrow [\theta_0\ \theta_1 \ldots \theta_m]\ k=1 \\ \leftarrow [\theta_0\ \theta_1 \ldots \theta_m]\ k=2 \\ \\ \vdots \\ \leftarrow [\theta_0\ \theta_1 \ldots \theta_m]\ k=n \end{array}$$

     - each class has its own dedicated parameter vector $\boldsymbol{\theta}^{(k)}$
     - all these vectors are typically stored as rows in a *parameter matrix* $\boldsymbol{\Theta}$
  2. estimates the probability $\hat{p}_k$ that the instance $\mathbf{x}$ belong to each of the classes k by applying the **softmax function** (aka **normalized exponential**) to the scores

Softmax Regression

Softmax Function

$[x^T\theta]$ → sigmoid → [binary classif task]
→ softmax → [multiclass class task]

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{e^{s_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{s_j(\mathbf{x})}}$$

$$\hat{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{bmatrix} \quad e^{s_k} = \begin{bmatrix} e^{s_1} \\ e^{s_2} \\ \vdots \\ e^{s_n} \end{bmatrix} \Rightarrow P_k = \begin{bmatrix} e^{s_1}/\Sigma \\ e^{s_2}/\Sigma \\ \vdots \\ e^{s_n}/\Sigma \end{bmatrix}$$

$\sum = \ldots = \Sigma$ $\quad \Sigma = 1$

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Softmax Regression classifier prediction

The Softmax Regression classifier predicts the class with the highest estimated probability

$$\hat{y} = \underset{k}{argmax}(\sigma(\mathbf{s}(\mathbf{x}))_k) = \underset{k}{argmax}(s_k(\mathbf{x})) = \underset{k}{argmax}\left(\left(\boldsymbol{\theta}^{(k)}\right)^T \mathbf{x}\right)$$

$$\begin{array}{c c} & \hat{p} \\ \begin{array}{c} A \\ B \\ C \end{array} & \begin{bmatrix} 0,8 \\ 0,1 \\ 0,1 \end{bmatrix} \end{array} \Rightarrow \text{predict } \boxed{\text{class } A}$$

✓ TRUE value $\boxed{\text{class } A}$

$$\begin{array}{c c} & \hat{p} \\ \begin{array}{c} A \\ B \\ C \end{array} & \begin{bmatrix} 0,4 \\ 0,3 \\ 0,3 \end{bmatrix} \end{array} \Rightarrow \text{predict } \boxed{\text{class } A}$$

Softmax Regression

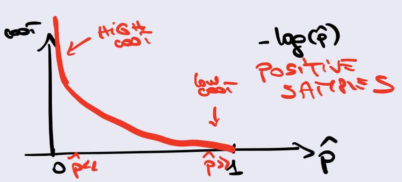
Cross Entropy

- During training, the objective is to have a model that estimates a high probability for the target class (and consequently a low probability for the other classes).
- Minimizing the cost function called the **cross entropy**, should lead to this objective because it penalizes the model when it estimates a low probability for a target class.
- Cross entropy is frequently used to measure how well a set of estimated class probabilities match the target classes

$$K \begin{bmatrix} [\theta_0 & \theta_1 & \dots & \theta_n] \\ [\theta_0 & \theta_1 & \dots & \theta_n] \\ & & \vdots & \\ [\theta_0 & \theta_1 & \dots & \theta_n] \end{bmatrix}$$

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} \overbrace{y_k^{(i)}}^{\text{1 for the true class}} \log\left(\underbrace{\hat{p}_k^{(i)}}_{P(\text{true class})}\right)$$

$$\begin{array}{c} A \\ B \\ C \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



- $y_k^{(i)}$ is the target probability that the i^{th} instance belongs to class k. In general, it is either equal to 1 or 0, depending on whether the instance belongs to the class or not.
- when there are just two classes ($K = 2$), this cost function is equivalent to the Logistic Regression's cost function (log-loss)

Softmax Regression

Cross entropy gradient vector for class k

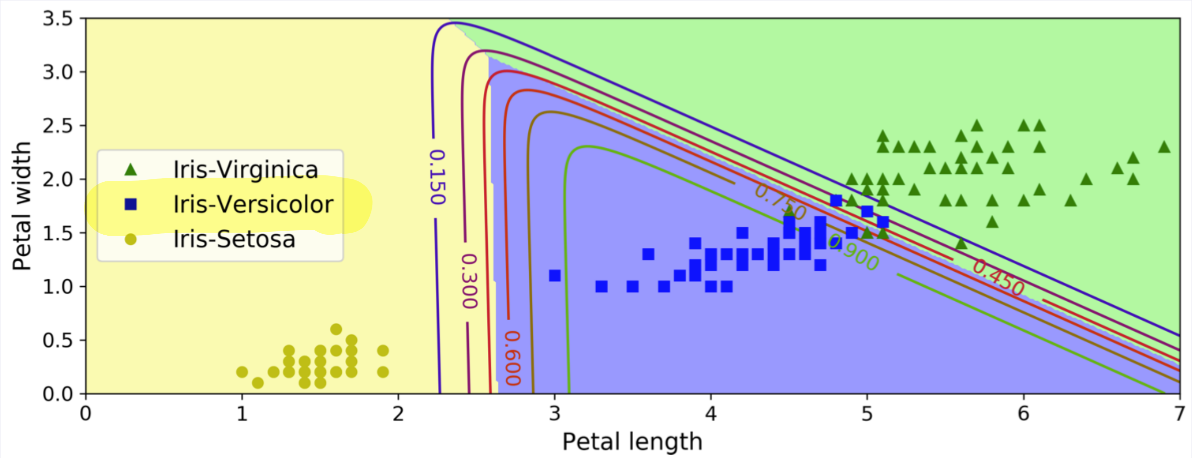
Gradient vector of the cross entropy cost function with regards to $\boldsymbol{\theta}^{(k)}$

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - \hat{y}_k^{(i)} \right) \mathbf{x}^{(i)}$$

1. compute the gradient vector for every class
2. use Gradient Descent (or any other optimization algorithm) to find the parameter matrix $\boldsymbol{\Theta}$ that minimizes the cost function.

Softmax Regression

Softmax Regression decision boundaries



- The decision boundaries between any two classes are linear.
- The figure also shows the probabilities for the Iris-Versicolor class, represented by the curved lines (e.g., the line labeled with 0.450 represents the 45% probability boundary).
- the model can predict a class that has an estimated probability below 50%
    - at the point where all decision boundaries meet, all classes have an equal estimated probability of 33%.

References

Aurélien Géron (2019)
Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow
Chapter 4: Training Models (pp. 113 – 153)
Published by O'Reilly Media, Inc..