Logistic Regression

Some images and notes were retrieved from Aurélien Géron - Hands-on Machine Learning with Scikit-Learn and TensorFlow (2019, O'Reilly)

Logistic Regression

- Remind that
  - Regression → predicting values
  - Classification → predicting classes

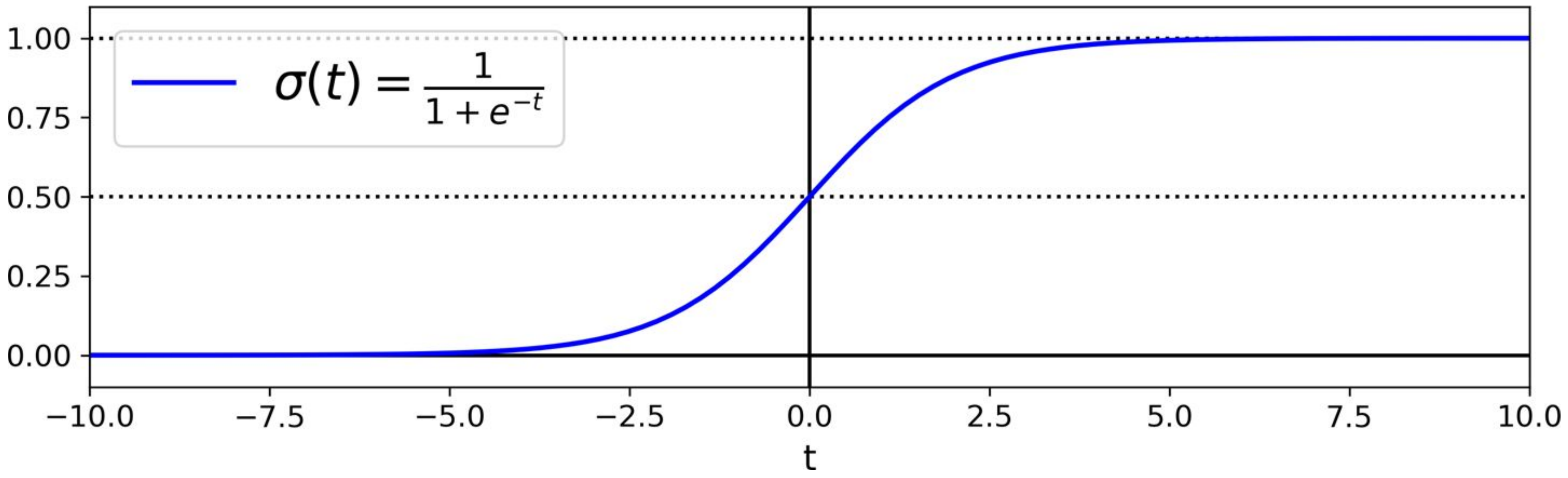
- It’s possible to convert a regression technique into a classification technique, and the idea is very simple.

- Instead of using the regression prediction as output, just give **logistic** (**sigmoid function** - S shaped) of this result.

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \underset{\text{Sigmoid Function}}{\sigma}\left(\underbrace{\boldsymbol{\theta}^\intercal \mathbf{x}}_{\text{Regression Output}}\right)$$

Sigmoid Function

- Sigmoid function (referred with σ) is a S shaped function, which always output a numeric value between 0 and 1.



Sigmoid Function

- Actually, sigmoid function calculates the probability of instance x belongs to positive class (assuming it's a binary classification task).

- If the probability (output of sigmoid function) is greater or equal to 0.5, then the prediction will be 1 (positive class). Otherwise, the prediction will be 0 (negative class).

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

- Notice that
  - $\sigma(t) < 0.5$ if $t < 0$
  - $\sigma(t) \geq 0.5$ if $t \geq 0$

- The score t is called as *logit*.

Training and Cost Function

- Remind that we want to calculate the parameter vector θ to be able to make accurate predictions.

- Accurate predictions → sigmoid function should output
  - High probabilities for positive classes
  - Low probabilities for negative classes

- The cost function of a single instance is as follows

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

- The cost function of whole dataset (called as **log loss**) is as follows:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right)\right]$$

Training and Cost Function

- Unfortunately, there is no closed form to calculate vector θ which minimizes the cost function.

- Luckily, the cost function is convex → means that GD can will global minimum with small learning rate and enough number of iterations.

- `LogisticRegression` function from `sklearn.linear_model` can be used.

- `LogisticRegression` can be regularized with ℓ_1 and ℓ_2 penalties (by default it's ℓ_2).

- The hyperparameter controlling the regularization strength of `LogisticRegression` is **NOT** alpha (as in other linear models), but its inverse: C. The higher the value of C, the less the model is regularized.

Multiclass Classification

- Binary classification → 2 classes

- Multiclass classification → Strictly more than 2 classes
  - One versus the rest (OvR) strategy: It composes N different binary classifiers like “classA vs others”, “classB vs others” etc. Then, it predicts the class with the highest score.

  - One versus one (OvO) strategy: It composes N*(N-1)/2 different binary classifiers like “classA vs classB”, “classA vs classC”, “classB vs classC” etc. Then, it predicts the class wins the most duels.

- However, some classifiers like SVMs prefers to use OvO since its (SVMs) time performance is poor. Although, this strategy uses more number of binary classifiers, the training data for each classifier will be much smaller. Thus, it runs much faster with big scales.

Multiclass Classification

- The Logistic Regression model can be generalized to support multiple classes directly, without using OvR and OvO strategies (that's why it's called as Multinomial Logistic Regression).

- But we can still use OvR if we want by setting multi_class parameter to "ovr" (reference).

- This time, model keeps a parameter matrix θ that contains parameter vectors for each class.

Multiclass Classification

- For a given instance x, the model calculates the logit scores ***$s_k(x)$*** for each class.

$$s_k(\mathbf{x}) = \left(\boldsymbol{\theta}^{(k)}\right)^\intercal \mathbf{x}$$

- Then, it uses **softmax function** to calculate probability of each class for instance ***x***.

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

- Finally, it predicts the **class with highest probability**.

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

Multiclass Classification (Cost Function)

- For the cost function, **cross entropy** is used during the training. Check this video for detailed information.

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- The objective is the same, we want to minimize the cost function during training (ie. producing high probability for correct classes, low probability for incorrect classes).

- In LogisticRegression function, we need to set `multi_class` hyperparameter to "`multinomial`" to switch it to Softmax Regression. Additionally, `solver` parameter should be "`lbfgs`".

```
softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10)
softmax_reg.fit(X, y)
```