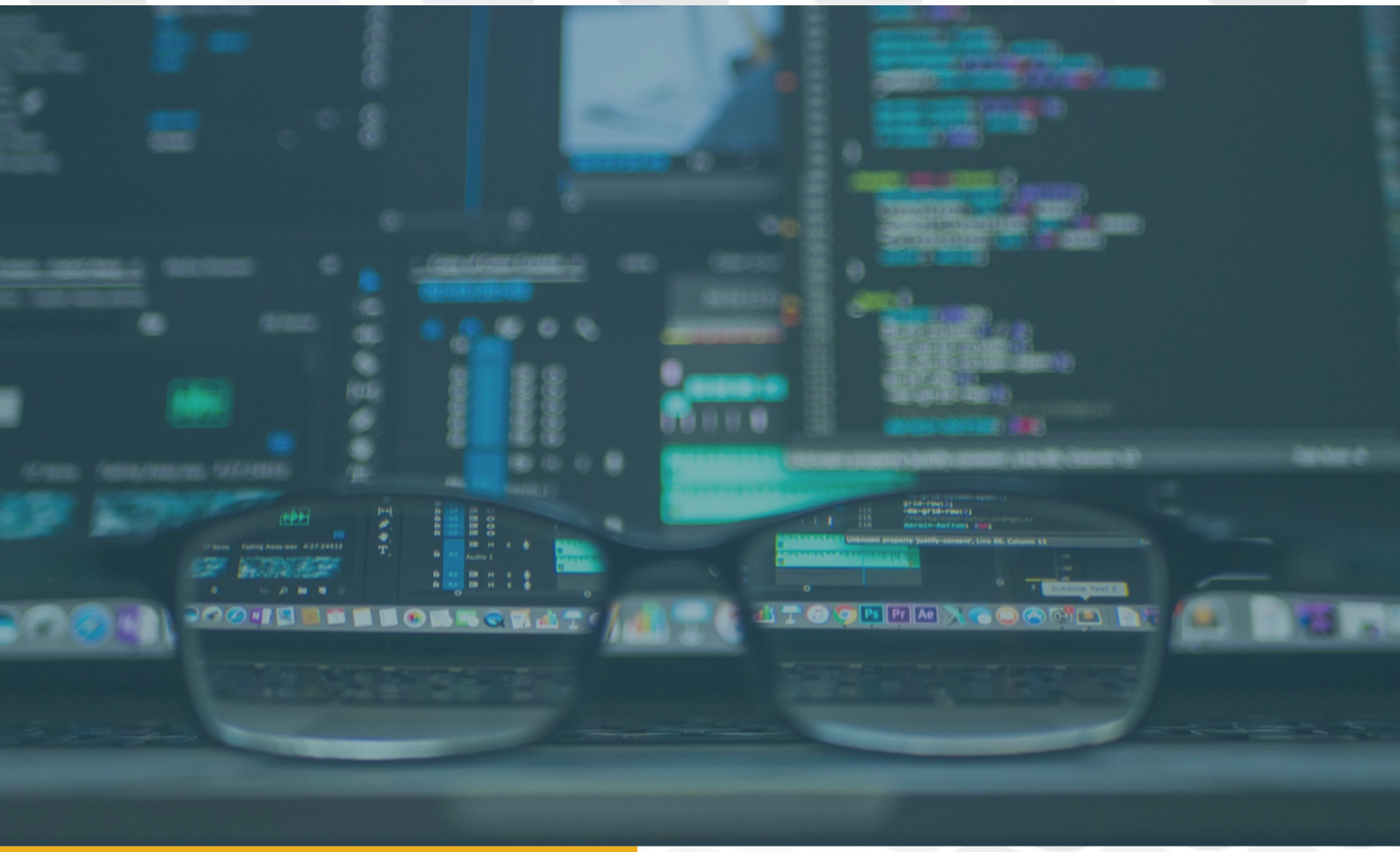
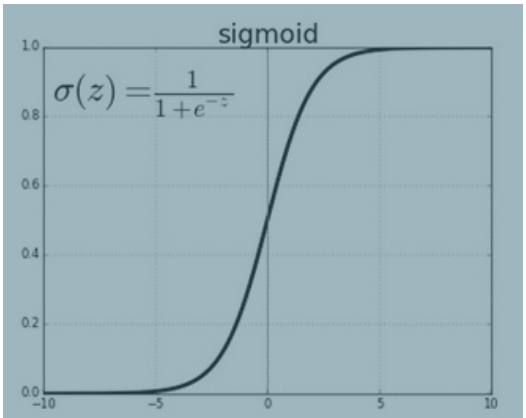


06

Classification: Logistic Regression

Logistic Function

A Sigmoid function is a mathematical function which has a characteristic S- shaped curve. There are a number of common sigmoid functions, such as the logistic function, the hyperbolic tangent, and the arctangent Logistic function is a type of sigmoid function that squishes values between 0 and 1.



Sigmoid Function

Logistic Regression

One important characteristic of the logistic function is that it has two horizontal asymptotes at 0 and 1:

$$\lim_{z \to -\infty} logistic(z) = 0 \qquad \lim_{z \to \infty} logistic(z) = 1$$

Because logistic function transforms any real number into a value between 0 to 1. If z goes to infinity, logistic function will become 1 and if z goes to negative infinity, the function will become 0.

Logistic Regression

Logistic regression is a binary classification algorithm despite the name contains the word 'regression'. For binary classification, we have two target classes we want to predict. Let's refer to them as positive (y=1) and negative (y=0) classes. When we combine linear regression and logistic function, we get the logistic regression equation:

$$\hat{p}(y=1) = logistic(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n)$$

where:
$\hat{p}(y=1)$: predicted probability of belonging to the positive class
θ_0 : intercept (aka bias term)
$\theta_1, \theta_2, ..., \theta_n$: coefficients (aka weights)
$x_1, x_2, ..., x_n$: features

$$\hat{p}(y=1) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n)}}$$

Logistic regression predicts the probability of a record belonging to the positive class given features.
Note that inside of logistic function is basically a formula of linear regression!

Since we have two classes, finding the probability of belonging to the negative class is simple:

Once we have probability values, it's easy to convert them to a predicted class. Using a threshold of 0.5 (i.e. probability of 50%), we can convert the probability values to classes:

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

where:
$\hat{y}$: predicted class

This is equivalent to:

$$\hat{y} = \begin{cases} 0 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n < 0 \\ 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \geq 0 \end{cases}$$

Here's a simple example on a small toy dataset with a binary target and two features:

```
from sklearn.linear_model import LogisticRegression
# Create sample data
df = pd.DataFrame({'x1': [1, 2, 2, 3, 2, 4, 3, 4, 3, 4],
                   'x2': [2, 3, 4, 2, 1, 3, 5, 2, 3, 6],
                   'y': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]})
# Train a logistic regression
target = 'y'
features = ['x1', 'x2']
model = LogisticRegression()
model.fit(df[features], df[target])
# Predict
df['z'] = model.intercept_[0] + df['x1']*model.coef_[0][0] + df['x2']*model.coef_[0][1]
df['p_hat'] = model.predict_proba(df[features])[:,1]
df['y_hat'] = model.predict(df[features])
df
```

	x1	x2	y	z	p_hat	y_hat
0	1	2	0	-2.978560	0.048404	0
1	2	3	0	-1.114241	0.247081	0
2	2	4	0	-0.557769	0.364064	0
3	3	2	0	-0.362868	0.410266	0
4	2	1	0	-2.227187	0.097336	0
5	4	3	1	1.501451	0.817791	1
6	3	5	1	1.306550	0.786935	1
7	4	2	1	0.944979	0.720104	1
8	3	3	1	0.193605	0.548251	1
9	4	6	1	3.170869	0.959723	1

We can see that for negative values of z, probability values are under 0.5 and for positive values of z, the probability values are above 0.5.

Decision Boundary

Decision boundary for logistic regression is given by:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = 0$$

In our example, since we only have two features, The equation becomes:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

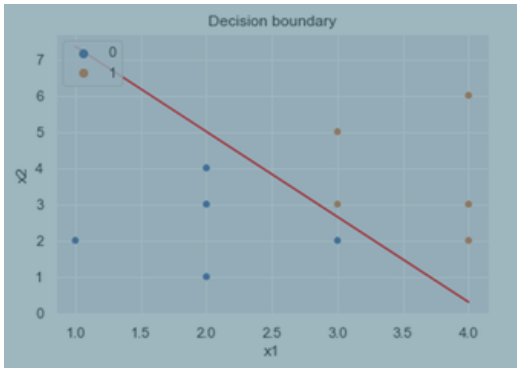
Let's plot x1 on the horizontal axis and x2 on the vertical axis along with the decision boundary. Rearrange the above equation such that x2 is expressed in terms of x1

$$\theta_2 x_2 = -(\theta_0 + \theta_1 x_1)$$

$$x_2 = -\frac{\theta_0 + \theta_1 x_1}{\theta_2}$$

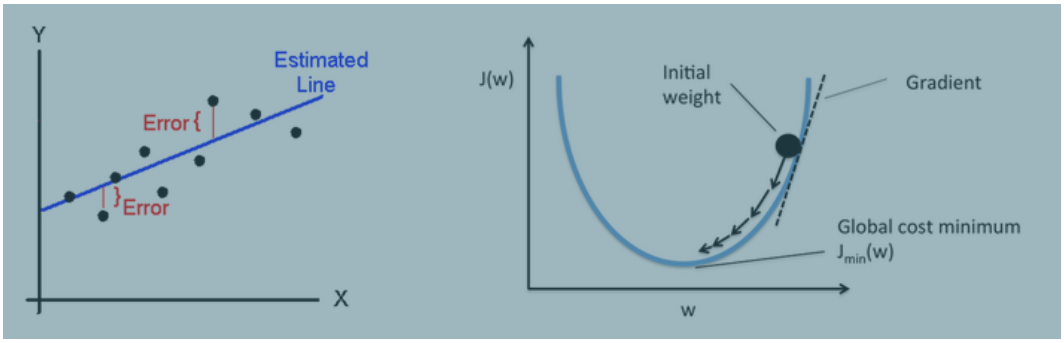
```
theta0 = model.intercept_[0]
theta1 = model.coef_[0][0]
theta2 = model.coef_[0][1]

plt.figure(figsize=(6,4))
sns.set()
sns.scatterplot(data=df,
                x='x1', y='x2',
                hue='y')
sns.lineplot(x=df['x1'],
             y=-(theta0 + theta1*df['x1'])/theta2,
             color='red')
plt.legend(loc='upper left')
plt.title('Decision boundary')
plt.show()
```



Cost Function

In ML, cost functions (you may also see this referred to as loss or error) are used to estimate how badly models are performing. A cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y. This is typically expressed as a difference or distance between the predicted value and the actual value.



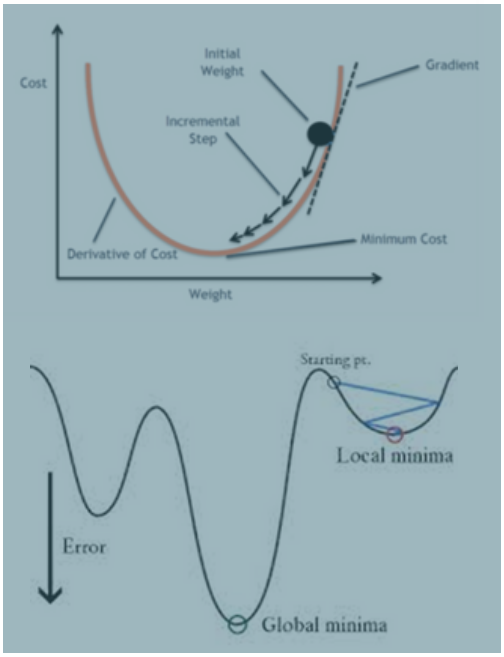
In linear regression, we can use MSE as its cost function

$$J = \frac{\sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2}{n}$$

Gradient Descent is an algorithm that is used to optimize the cost function or the error of the model. It is used to find the minimum value of error possible in your model. This can be done by the help of 'gradient'

Gradient descent will converge into global minimum only if the function is convex.

Cost function in Linear Regression:



In Logistic Regression Yi is a nonlinear function (Y=1/1+e-z), if we put this in the above cost function it will give a non-convex function. There will exist many local optima on which optimization algorithms might prematurely converge before finding the true minimum.

Log Loss

Using the Maximum Likelihood Estimator from statistics, we can obtain the cost function which produces a convex space friendly for optimization. This function is known as the binary cross-entropy loss or Log loss.

Log loss compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. Formal definition of log loss: The negative average of the log of corrected predicted probabilities.

ID	Actual	Predicted probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.0268721464
ID1	1	0.90	0.90	-0.0457574906
ID7	1	0.78	0.78	-0.1079053973
ID8	0	0.56	0.44	-0.3565473235
ID2	0	0.51	0.49	-0.30980392
ID3	1	0.47	0.47	-0.3279021421
ID4	1	0.32	0.32	-0.4948500217
ID5	0	0.10	0.90	-0.0457574906

- Actual: It is the class the object originally belongs to
- Predicted probabilities: the probability that particular object belongs to class 1.
- Corrected probabilities: the probability that a particular observation belongs to its original class

For Class 1, the corrected prob. = predicted prob
For Class 0, the corrected prob. = 1 – predicted prob

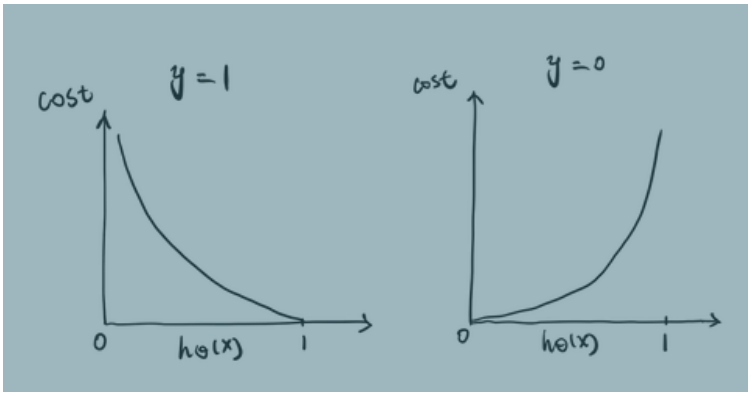
Since all the corrected probabilities lie between 0 and 1, their log values are negative. In order to compensate for this negative value, we will use a negative average of the value

$$-\frac{1}{N}\sum_{i=1}^{N}(\log(p_i))$$

The value of the negative average of corrected probabilities we calculate comes to be 0.214 which is our Log loss for this particular example.

$$\text{Log loss} = \frac{1}{N}\sum_{i=1}^{N} -(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Here, pi is the probability of class 1, and (1-pi) is the probability of class 0. You will notice that the first half of the equation is 0 when y=0 (when observation belongs to class 0) whereas the second half of the equation is 0 when y=1 (when observation belongs to class 1). Now we know how to measure the error for a given set of parameters, the next question is how to find the best parameters that minimize the error. This is where optimization algorithms such as gradient descent come into play.



Note: h(x) is the same as pi we have calculated

Advantages of Logistic Regression

- It makes no assumptions about distributions of classes in feature space.
- It is very fast at classifying unknown records.
- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable.
- Simplicity and transparency. Logistic Regression is just a bit more involved than Linear Regression, which is one of the simplest predictive algorithms out there. It is also transparent, meaning we can see through the process and understand what is going on at each step, contrasted to the more complex ones (e.g. SVM, Deep Neural Nets) that are much harder to track
- Giving probabilistic output. Some other algorithms (e.g. Decision Tree) only produce the most seemingly matched label for each data sample, meanwhile, Logistic Regression gives a decimal number ranging from 0 to 1, which can be interpreted as the probability of the sample to be in the Positive Class. With that, we know how confident the prediction is, leading to a wider usage and deeper analysis.

Disadvantages of Logistic Regression

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.
- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.
- Logistic Regression requires average or no multicollinearity between independent variables.
- It is tough to obtain complex relationships using logistic regression. More powerful and compact algorithms such as Neural Networks can easily outperform this algorithm.

Picture Source

- pexels.com