Logistic Regression From Scratch

Koushik · Follow
6 min read · Aug 28, 2023

Logistic regression is often mentioned in connection to classification tasks. The model is simple and one of the easy starters to learn about generating probabilities, classifying samples, and understanding gradient descent. This tutorial walks you through some mathematical equations and pairs them with practical examples in Python so that you can see exactly how to train your own custom binary logistic regression model.

Implementing logistic regression from scratch in Python

Import Libraries

```
import numpy as np
from utils import *
```

First we Initialize some parameters.In machine learning we call those parameters ,hyperparameter.

1. Learning rate
2. Batch size
3. Number of iteration

weight : If not scratch this param generate automatically in Keras/ Tensorflow or can choose manually.

bias : if not scratch this param generate automatically in Keras/Tensorflow

```
def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
```

Activation Function : Why Sigmoid ,why not linear function?

However, a linear activation function has two major problems :

- It's not possible to use backpropagation as the derivative of the function is a constant and has no relation to the input x.
- All layers of the neural network will collapse into one if a linear activation function is used. No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer. So, essentially, a linear activation function turns the neural network into just one layer.

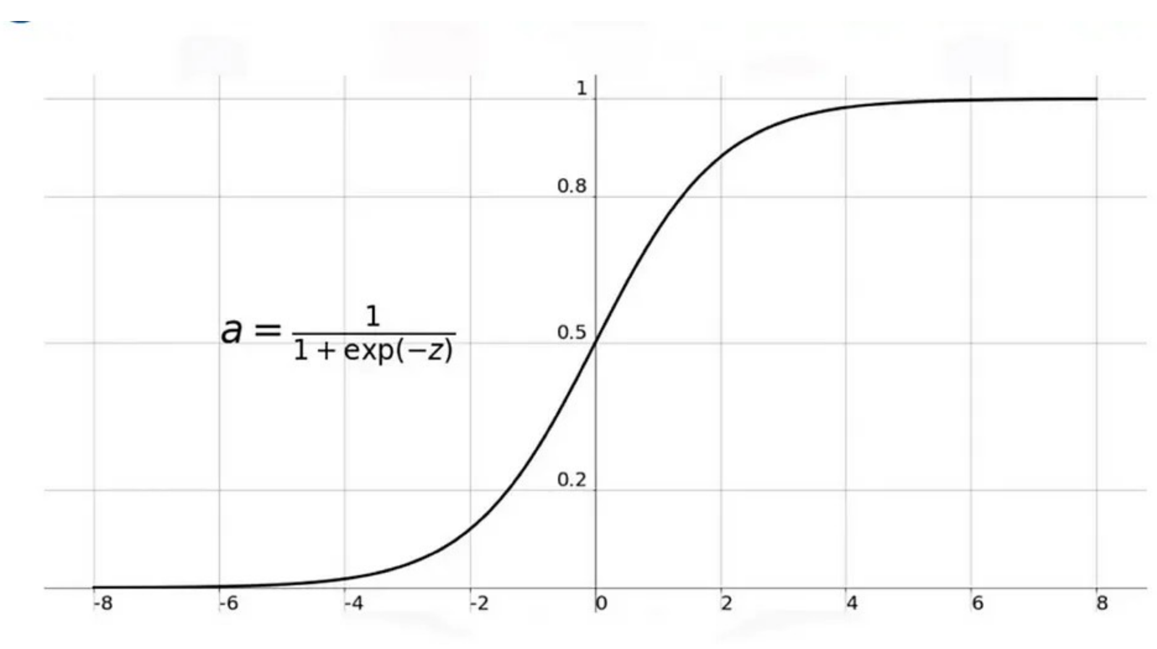
Non Linear Activation

The linear activation function shown above is simply a linear regression model. Because of its limited power, this does not allow the model to create complex mappings between the network's inputs and outputs.

Non-linear activation functions solve the following limitations of linear activation functions:

- They allow backpropagation because now the derivative function would be related to the input, and it's possible to go back and understand which weights in the input neurons can provide a better prediction.
- They allow the stacking of multiple layers of neurons as the output would now be a non-linear combination of input passed through multiple layers. Any output can be represented as a functional computation in a neural network.

```
def sigmoid( x):
    return 1 / (1 + np.exp(-x))
```



Sigmoid Function

This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.

- The derivative of the function is f’(x) = sigmoid(x)*(1-sigmoid(x)).

**Reference

Exp Normalization Trick: Numerically stable sigmoid function

Cross Entropy Loss

```
def compute_loss(self, y_true, y_pred):
        # binary cross entropy
        epsilon = 1e-9
        y1 = y_true * np.log(y_pred + epsilon)
        y2 = (1-y_true) * np.log(1 - y_pred + epsilon)
        return -np.mean(y1 + y2)
```

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

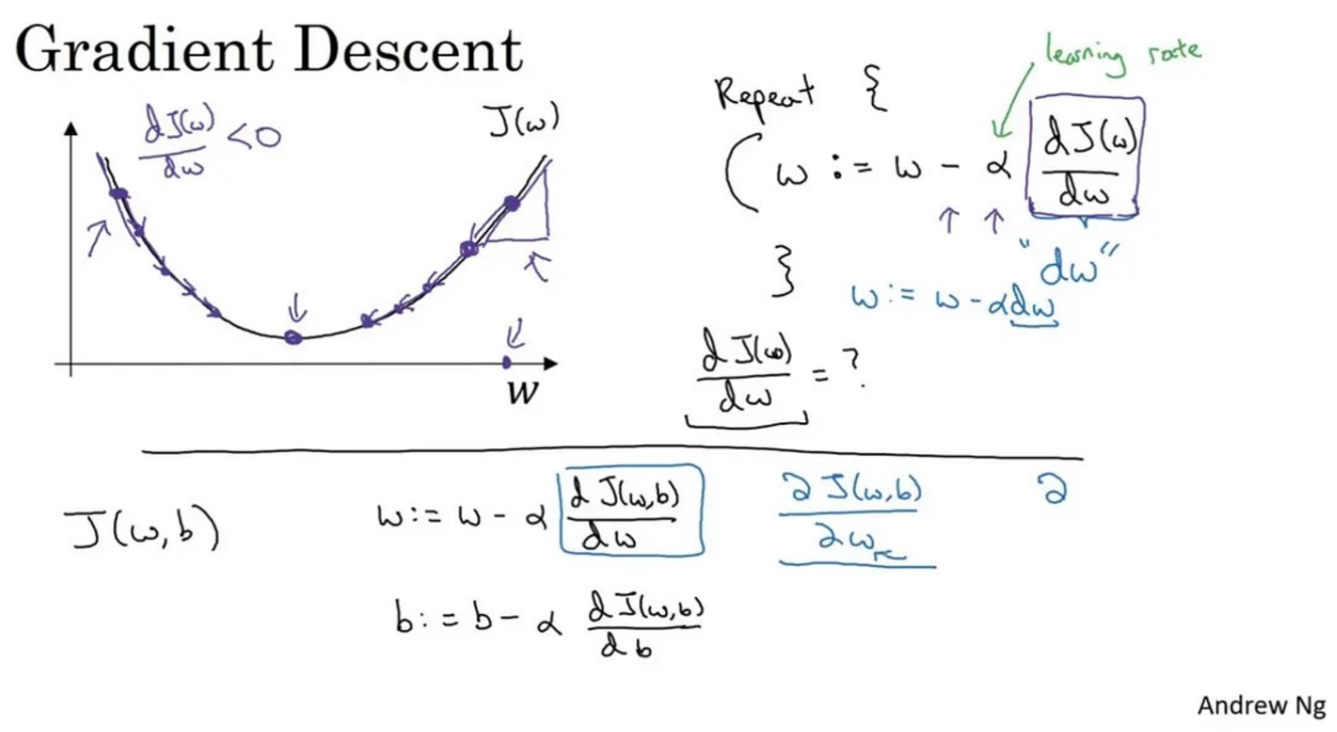
Binary Cross Entropy Loss

It’s a method of evaluating how well your algorithm models your dataset. If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll output a lower number. As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you're getting anywhere.You are essentially finding all of the errors by comparing your ground truth y_true to your predictions y_pred (also known as y hat from your explanation section).

**Reference

Deep Learning, by Ian Goodfellow, Yoshua Bengio and Aaron Courville.

Gradient Descent and (Forward + BackProp)


Gradient Descent
$\frac{dJ(w)}{dw} < 0$
$J(w)$
w
learning rate
Repeat {
$w := w - \alpha \frac{dJ(w)}{dw}$
}
"dw"
$w := w - \alpha dw$
$\frac{dJ(w)}{dw} = ?$
$J(w,b)$
$w := w - \alpha \frac{dJ(w,b)}{dw}$
$\frac{\partial J(w,b)}{\partial w}$
∂
$b := b - \alpha \frac{dJ(w,b)}{db}$
Andrew Ng

Gradient Descent , Andrew ng ,Coursera Machine Learning

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. Gradient Descent can be thought of as the direction you have to take to reach the least possible error.

So here's the definition: when you derive a function, you get an equation that tells you what the gradient of your function will be at any given value for x. Looking back at the graph of the cost function, if we could therefore derive the cost function, we could find out what the gradient is.

Here 'w' sets the direction and 'b' move the direction back and forth.

***References

Stanford CS229.2018,Andrew Ng

Stanford Lecture :CS229

Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville.

```
def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):
            A = self.feed_forward(X)
            dz = A - y # derivative of sigmoid and bce X.T*(A-y)

            # compute gradients
            dw = (1 / n_samples) * np.dot(X.T, dz)
            db = (1 / n_samples) * np.sum(A - y)
            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db
```

Here is the full code:

```
class LogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
        self.losses = []

    #Sigmoid method
    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def compute_loss(self, y_true, y_pred):
        # binary cross entropy
        epsilon = 1e-9
        y1 = y_true * np.log(y_pred + epsilon)
        y2 = (1-y_true) * np.log(1 - y_pred + epsilon)
        return -np.mean(y1 + y2)

    def feed_forward(self,X):
        z = np.dot(X, self.weights) + self.bias
        A = self._sigmoid(z)
        return A

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):
            A = self.feed_forward(X)
            self.losses.append(compute_loss(y,A))
            dz = A - y # derivative of sigmoid and bce X.T*(A-y)
            # compute gradients
            dw = (1 / n_samples) * np.dot(X.T, dz)
            db = (1 / n_samples) * np.sum(dz)
            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        threshold = .5
        y_hat = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(y_hat)
        y_predicted_cls = [1 if i > threshold else 0 for i in y_predicted]

        return np.array(y_predicted_cls)
```

Output of the Regression

Here we use sklearn toy dataset (Brest Cancer) to test our regression model.As compare to Sklearn Logistic Resression model it gives a good output with 93% sccuracy with 91.7% precision.

```
from sklearn.model_selection import train_test_split
from sklearn import datasets

dataset = datasets.load_breast_cancer()
X, y = dataset.data, dataset.target

X, y = dataset.data, dataset.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1234
)

regressor = LogisticRegression(learning_rate=0.0001, n_iters=1000)
regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)
cm ,accuracy,sens,precision,f_score  = confusion_matrix(np.asarray(y_test), np.a
print("Test accuracy: {0:.3f}".format(accuracy))
print("Confusion Matrix:",np.array(cm))
```

Thank You for reading the artice.Don't forget to follow me in github and Medium.

Here is the full code

Machine_learning_from_scratch/LogisticRegression at main · Koushikl0l/Machine_learning_from_scratch

Contribute to Koushikl0l/Machine_learning_from_scratch development by creating an account on GitHub.

github.com

References

CS229: Machine Learning

Time and Location Lectures: Monday, Friday 4:30 PM - 7:00 PM (PST) in NVIDIA Auditorium Quick Links (You may need to…

cs229.stanford.edu

What are Loss Functions?

After the post on activation functions, we will dive into the second part, the loss, or objective function for neural...

towardsdatascience.com

Supervised Machine Learning: Regression and Classification

In the first course of the Machine Learning Specialization, you will: * Build machine learning models in Python using...

www.coursera.org

Deep Learning

The Deep Learning textbook is a resource intended to help students and practitioners enter the field of machine...

www.deeplearningbook.org

Exp-normalize trick

This trick is the very close cousin of the infamous log-sum-exp trick (scipy.misc.logsumexp). Supposed you'd like to...

timvieira.github.io

Machine Learning Artificial Intelligence Data Science Deep Learning Logistic Regression

Written by Koushik

234 Followers · 3 Following

Machine Learning Engineer

Follow

Responses (1)

Zs Veres-Lakos Zsombor

What are your thoughts?

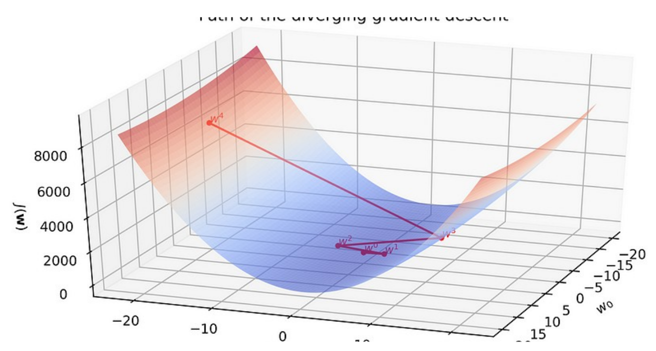
Sayied
Dec 28, 2023 (edited)

> we call those parameters ,hyperparameter

Why hyperparameters

2 replies Reply

More from Koushik

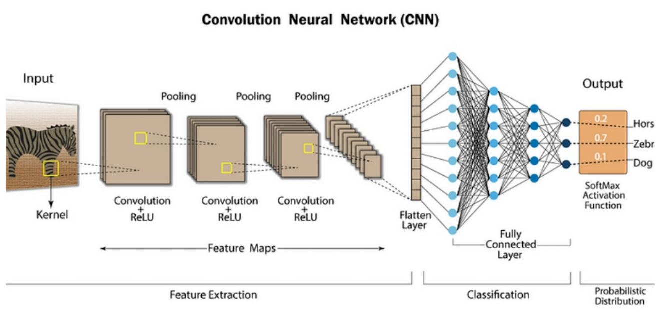


Koushik

Optimization Algorithms in Machine Learning: A...

Optimizer, the engine of machine learning.

Dec 7, 2023 60

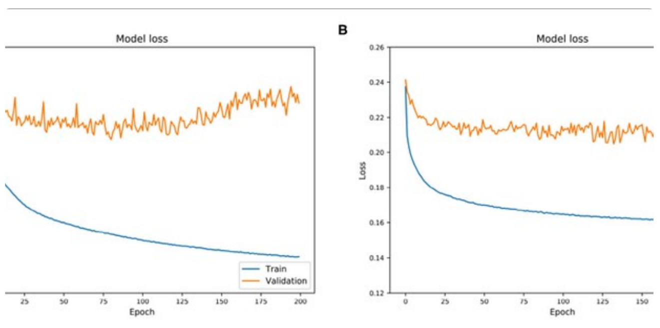


Koushik

Understanding Convolutional Neural Networks (CNNs) in Depth

Convolutional Neural Networks skillfully capturing and extracting patterns from data...

Nov 28, 2023 618 8

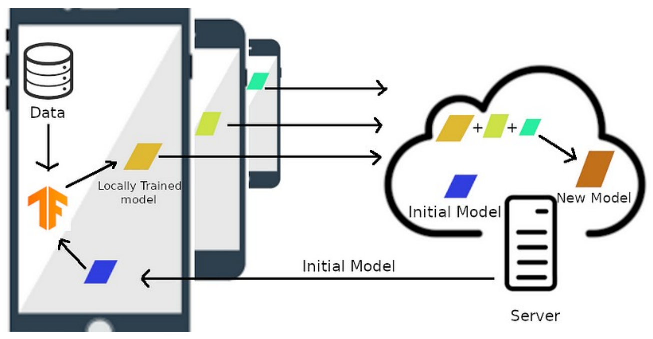


Koushik

What is Overfitting and Underfitting , and how to deal wit...

In machine learning, it is common to face a situation when the accuracy of models on th...

Sep 20, 2023 108 1



Koushik

Federated Learning: Advancing Machine Learning While Protecti...

Revolutionizing Machine Learning Through Decentralized Collaboration and Data...

Dec 7, 2023 10

See all from Koushik

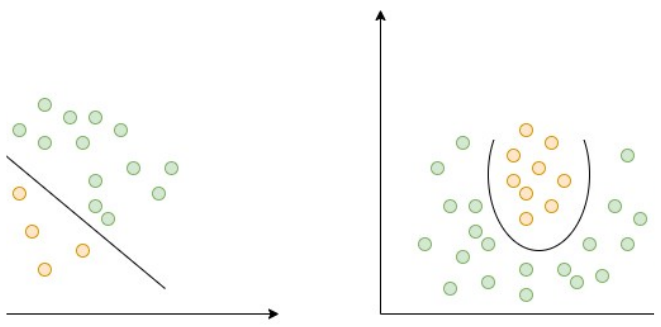
Recommended from Medium

CodexRushi

Gradient Descent: Explanation with Python Code

Gradient descent is one of the algorithms considered to form the basis of machine…

Oct 24, 2024



Okeshakarunarathne

Understanding Decision Boundaries in Machine Learning

When training a machine learning model for classification tasks, one of the most…

Sep 27, 2024 59

Lists

Predictive Modeling w/ Python
20 stories · 1848 saves

Practical Guides to Machine Learning
10 stories · 2217 saves

Natural Language Processing
1969 stories · 1614 saves

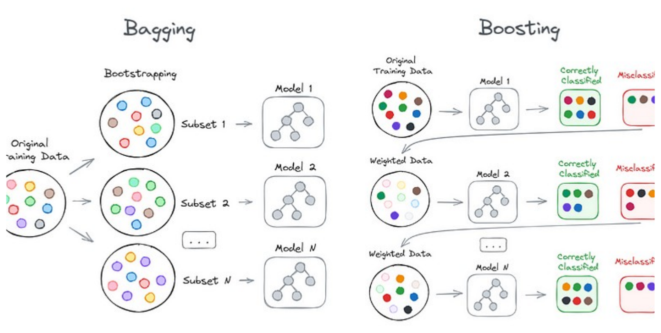
data science and AI
40 stories · 338 saves

D.H. Jang

Interpreting Support Vector Machine Coefficients: A…

In the rapidly advancing landscape of artificial intelligence (AI) and machine learning (ML),…

Nov 3, 2024

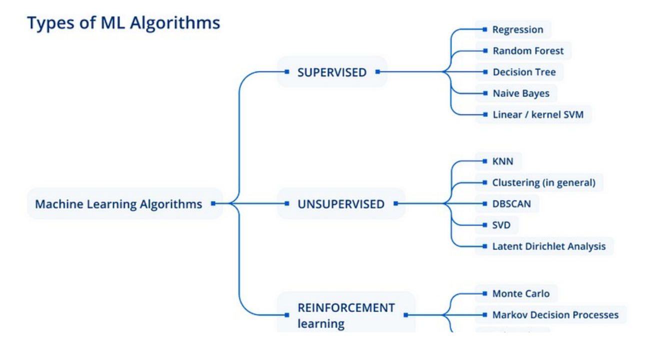


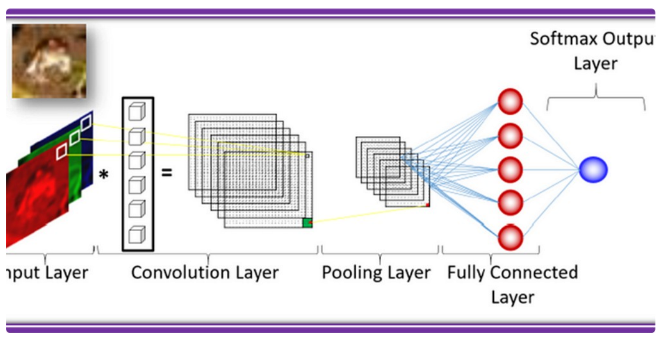
In Towards AI by Thomas A Dorfer

Bagging vs. Boosting: The Power of Ensemble Methods in Machine…

How to maximize predictive performance by creating a strong learner from multiple weak…

Jun 16, 2023 608 4





John Vastola

10 Must-Know Machine Learning Algorithms for Data Scientists

Machine learning is the science of getting computers to act without being explicitly…

Dec 6, 2022 1.3K 25

In Python in Plain English by Jyoti Dabass, Ph.D.

Friendly Introduction to Deep Learning Architectures (CNN, RN…

This blog aims to provide a friendly introduction to deep learning architectures…

Apr 1, 2024 1.5K 14

See more recommendations

Help Status About Careers Press Blog Privacy Terms Text to speech Teams