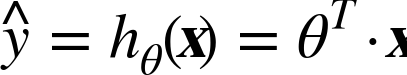
Linear regression

Linear regression implementation with Numpy

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the bias term.

Equation y = θ+ X1θ1 + X2θ2 + ...

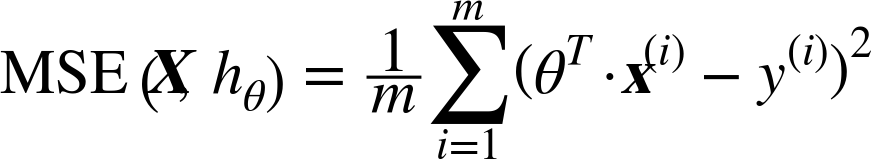
vectorized form of above equation



Therefore, to train a Linear Regression model, you need to find the value of θ that minimizes the RMSE. In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result (because the value that minimizes a function also minimizes its square root)

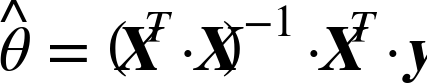
The MSE of a Linear Regression hypothesis hθ on a training set X is calculated using [Equation 4-3](https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html#mse_cost_function).

##### *MSE cost function for a Linear Regression model*

**

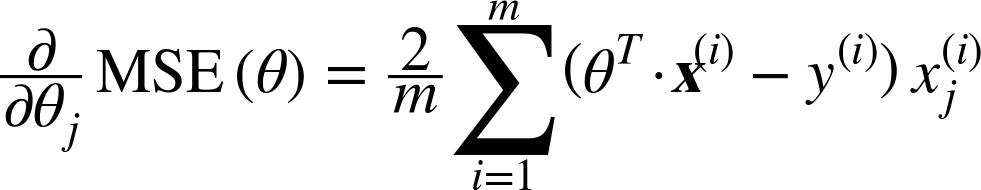
*The reason for the operations is that the summation of the squared errors gives us a unique and simple global number, the difference between expected and real number gives us the proper distance, and the square power gives us a positive number, which penalizes distances in a more-than-linear fashion.*

To find the value of *θ* that minimizes the cost function, there is a *closed-form solution*

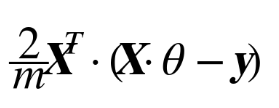
**

Linear regression with batch gradient descent

To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter *θj*. In other words, you need to calculate how much the cost function will change if you change *θj* just a little bit. This is called a *partial derivative*.



Instead of calculating gradient individually, will use following equation.



Linear regression with *Stochastic Gradient Descent*

*Stochastic Gradient Descent* just picks a random instance in the training set at every step and computes the gradients based only on that single instance.

## *Mini-batch Gradient Descent*

Mini-batch GD computes the gradients on small random sets of instances called *mini-batches*.

Polynomial features refer this

[Polynomial feature implementation](https://render.githubusercontent.com/view/ipynb?commit=0f60305dd7a3fe98056858a13a30c75c247d0242&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f616765726f6e2f68616e64736f6e2d6d6c2f306636303330356464376133666539383035363835386131336133306337356332343764303234322f30345f747261696e696e675f6c696e6561725f6d6f64656c732e6970796e62&nwo=ageron%2Fhandson-ml&path=04_training_linear_models.ipynb&repository_id=51863547&repository_type=Repository#Polynomial-regression)

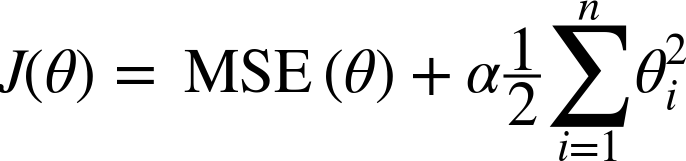
Regularized Linear Model

Fewer degrees of freedom it has, the harder it will be for it to overfit the data. For example, a simple way to regularize a polynomial model is to reduce the number of polynomial degrees.

**Ridge regression :**

Ridge regularization term equal to  is added to the cost function

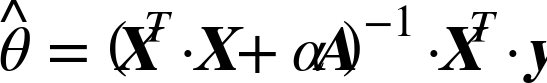
*Note that the regularization term should only be added to the cost function during training. Once the model is trained, you want to evaluate the model’s performance using the unregularized performance measure*

**

*Note that the bias term θ0 is not regularized (the sum starts at i = 1, not 0). If we define* ***w*** *as the vector of feature weights (θ1 to θn), then the regularization term is simply equal to ½(∥* ***w*** *∥2)2, where ∥ · ∥2 represents the ℓ2 norm of the weight vector.*[*12*](https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html#idm140310483341168) *For Gradient Descent, just add α****w*** *to the MSE gradient vector*

*It is important to scale the data (e.g., using a StandardScaler) before performing Ridge Regression, as it is sensitive to the scale of the input features. This is true of most regularized models.*

Closed form of ridge equation

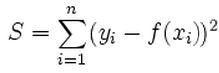
**

## Quick look: L1 and L2 norm

L1-norm is also known as least absolute deviations (LAD), least absolute errors (LAE). It is basically minimizing the sum of the absolute differences **(S)** between the target value (**Yi**) and the estimated values (**f(xi)**):

l1-norm formula

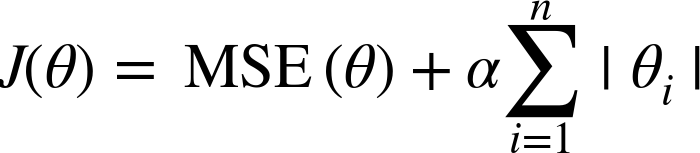
L2-norm is also known as least squares. It is basically minimizing the sum of the square of the differences **(S)** between the target value (**Yi**) and the estimated values (**f(xi):**

****

## **Lasso Regression**

Ridge Regression, it adds a regularization term to the cost function, but it uses the ℓ1 norm of the weight vector instead of half the square of the ℓ2 norm.

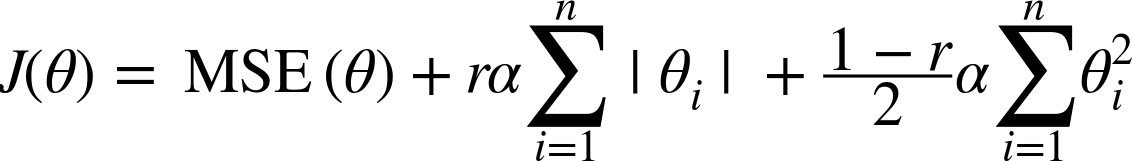
Lasso cost function



An important characteristic of Lasso Regression is that it tends to completely eliminate the weights of the least important features (i.e., set them to zero). For example, the dashed line in the right plot on [Figure 4-18](https://www.safaribooksonline.com/library/view/Hands-On+Machine+Learning+with+Scikit-Learn+and+TensorFlow/9781491962282/ch04.html#lasso_regression_plot) (with *α* = 10-7) looks quadratic, almost linear: all the weights for the high-degree polynomial features are equal to zero. In other words, Lasso Regression automatically performs feature selection and outputs a *sparse model* (i.e., with few nonzero feature weights).

## **Elastic Net**

Elastic Net is a middle ground between Ridge Regression and Lasso Regression. The regularization term is a simple mix of both Ridge and Lasso regularization terms, and you can control the mix ratio *r*. When *r* = 0, Elastic Net is equivalent to Ridge Regression, and when *r* = 1, it is equivalent to Lasso Regression

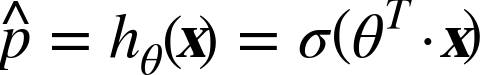
**

*So when should you use Linear Regression, Ridge, Lasso, or Elastic Net? It is almost always preferable to have at least a little bit of regularization, so generally you should avoid plain Linear Regression. Ridge is a good default, but if you suspect that only a few features are actually useful, you should prefer Lasso or Elastic Net since they tend to reduce the useless features’ weights down to zero as we have discussed. In general, Elastic Net is preferred over Lasso since Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.*

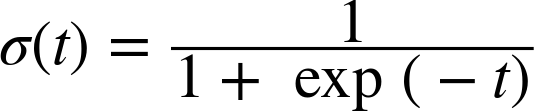
Logistic regression

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

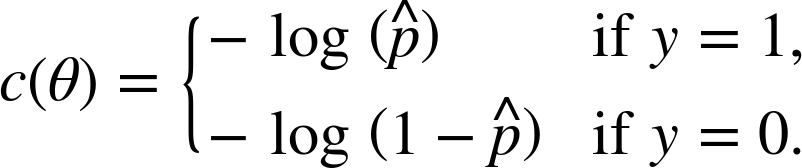
Logistic Regression model estimated probability (vectorized form)



Logistic function

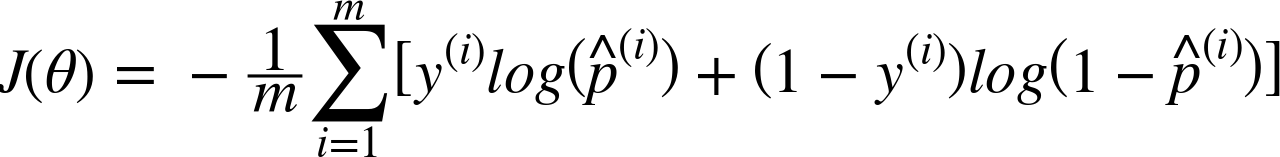


Cost function of a single training instance



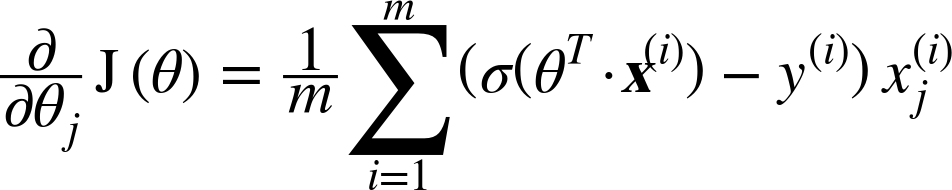
This cost function makes sense because – log(t) grows very large when t approaches 0, so the cost will be large if the model estimates a probability close to 0 for a positive instance, and it will also be very large if the model estimates a probability close to 1 for a negative instance. On the other hand, – log(t) is close to 0 when t is close to 1, so 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, which is precisely what we want.

Logistic Regression cost function (log loss)



The bad news is that there is no known closed-form equation to compute the value of θ that minimizes this cost function (there is no equivalent of the Normal Equation). But the good news is that 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). The partial derivatives of the cost function with regards to the jth model parameter θj is given by [Equation 4-18](https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html#logistic_cost_function_partial_derivatives).

Equation 4-18. Logistic cost function partial derivatives

**

Hyperparameter tuning

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,  
 penalty='l2', random\_state=42, solver='liblinear', tol=0.0001,  
 verbose=0, warm\_start=False)

multi\_class="multinomial" -- for softmax regressor.

Detail info : [http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html\](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html%5C)