Training models

Jens Peter Andersen, Assistant Professor, Roskilde

26-03-2020

Linear model – refreshing

Hypothesis function in general:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots\ldots\ldots + \theta_n x_n$$

Where:

- $\hat{y}$ is the predicted value - e.g. We want to predict ‘total clicks per day’
- n is the number of features - e.g. we may have only 1 feature, ‘cost per click’
- x_i is the i^{th} feature value – e.g. x_1 may represent the value of a ‘cost per click’
- θ_j is the j^{th} model parameter – e.g. only θ_0 and θ_1 are relevant with only one feature

Computational complexity – What does it mean?

Computational complexity:

- Expresses something about computer resources needed to solve a problem.
- That is resources needed when learning from a feature set with *n* features and *m* feature instances (learning set)
- Or resources needed to predict or classify using the model

Expressing computational complexity:

- As a function of number of elements – number of features or number of feature instances
- We are most interested in knowing how resources needed as the number of elements grows

Examples:

- $O(m)$ expresses that resources needed are proportionally or linear dependent on elements processed. That is increasing number of elements processed by a factor 2 increases resources needed by a factor 2
- $O(n^2)$ expresses that resources needed are quadratic dependent on elements processed. That is increasing number of elements processed by a factor 2 increases resources needed by a factor $2^2=4$

Computational complexity – Closed form

We have learned about the closed form linear regression so far.

Typical complexity for closed form computations:

- Training the model with *m* feature instances is complexity $O(m)$ – proportionally or linear dependent on elements processed
- Training the model with *n* features is complexity $O(n^{(>2)})$ – e.g. increasing *n* by 2 could increase processing resources needed by $2^3=8$

Computational complexity – Examples

In one of the videos a huge number of features is mentioned, when it comes to predicting on basis of the human genom. So closed form computation would definitely <u>not</u> be a good idea for learning in this case.

Any other examples where the number of featuree is huge?

Gradient descend – General learning approach

- Applicable for various kinds of models, which includes:
  - Linear
  - Polynomial
  - Logistic regression (classification)
- Out of core learning is possible – can be fast when *m* – the of the feature set increases
- Fast when *n* - the number of model paramters $\theta_1, \ldots, \theta_n$ - increases

Gradient descend – minimizing MSE

- Applied in order to <u>optimize</u> by <u>minimizing</u> a so-called cost function – typically MSE in machine learning.
- Here the Mean Square Error is a function of the model parameters – that is $MSE(\theta_1, \ldots, \theta_n)$
- 'Gradient' refers to observing on the slope of the cost function – negative, zero or positive.
- We want to end up in a set of values for $\theta_1, \ldots, \theta_n$ where the slope of the cost function is zero – that means minimum reached.
- 'Descend' refers to that we are adjusting the values of $\theta_1, \ldots, \theta_n$ in the direction where the cost function diminishes

Linear model - Mean squared error (MSE)

Problem: Finding the model parameters θ_0 and θ_1

$\hat{y} = \theta_0 + \theta_1 x_1$

Solution: Finding the model parameters θ_0 and θ_1 by minimizing MSE:

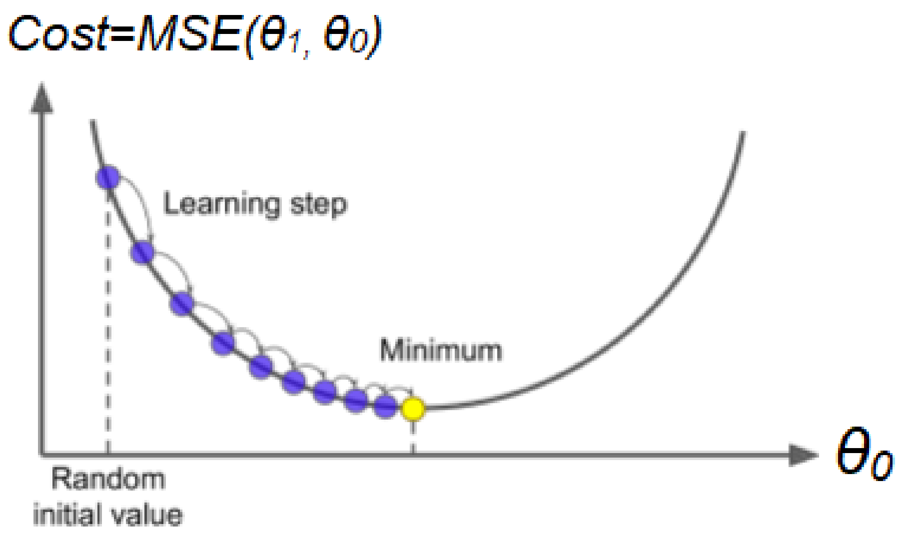
$MSE(\theta_1, \theta_0) = 1/N \; \Sigma_{i=1..N} \; (y_i - (\theta_1 x_i + \theta_0))^2$

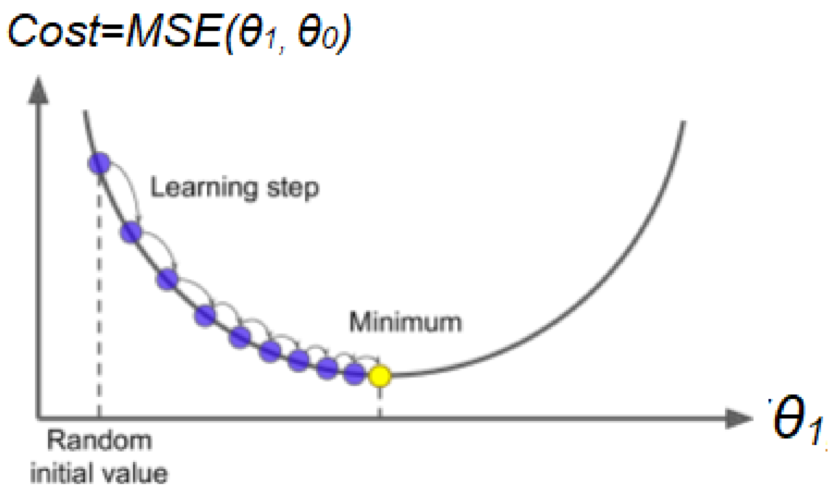
Linear model - Mean squared root error

- Mathematically we want to calculate the so-called partially derivative with respect to all model parameters in order to approach the minimum for MSE.
- With 2 model parameters the partially derivatives are expressed like this :
  - $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_1$ - Expresses slope in the direction of θ_1
  - $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_0$ - Expresses slope in the direction of θ_0
- Don’t worry we will look at the curves soon

Performing Gradient Descent - Exercise

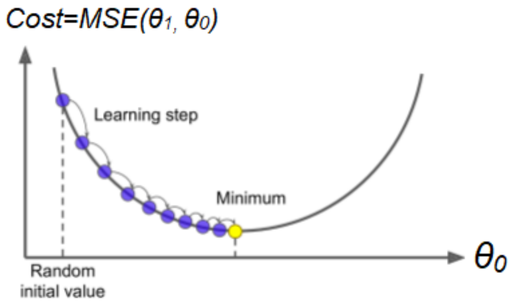
Explain to yourselves in the groups the process on how the cost function Mean Sqaure Error (MSE) goes towards the minimum, use curves below for help.

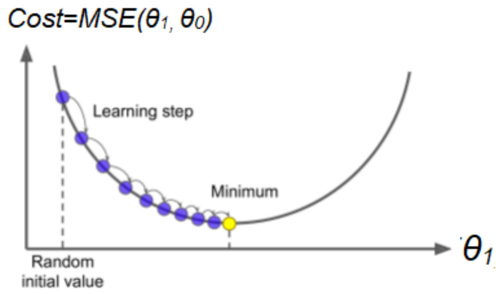




Performing Gradient Descent – the principle

While (Minimum <u>not</u> reached)
{
Based on the learning set:
$\theta_0 = \theta_0$ - LearningRate * $\partial MSE(\theta_1, \theta_0) / \partial\theta_0$
$\theta_1 = \theta_1$ - LearningRate * $\partial MSE(\theta_1, \theta_0) / \partial\theta_1$
}

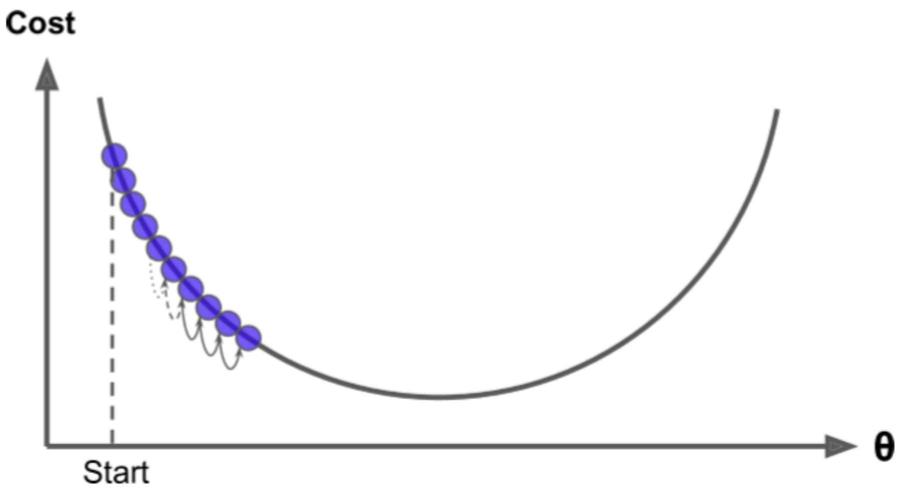




latest.webp

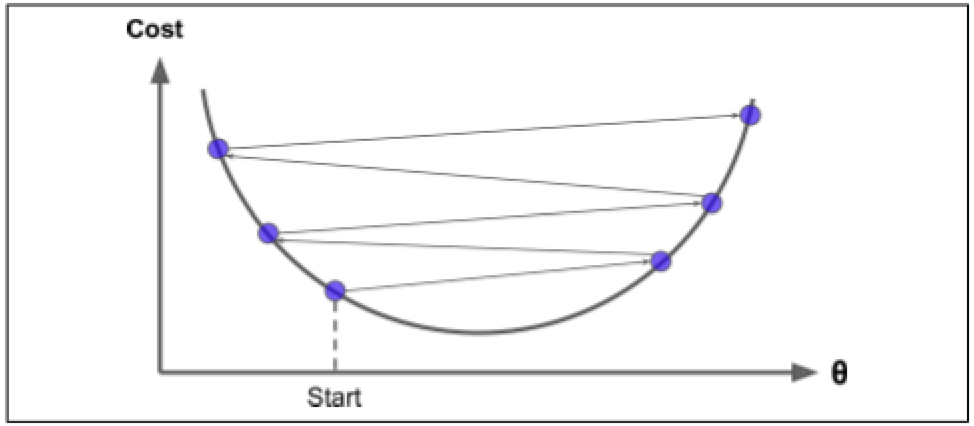
Being too ‘scrooge’ with the learning rate

- Learning rate is too small will make gradient descend too slow
- That is, the model parameters $\theta_{0\ \ldots}\ \theta_1$ are changing in small steps that slows down the algorithm
- Eventually $MSE(\theta_{0,\ \ldots.,}\ \theta_n)$ will converge towards a minimum



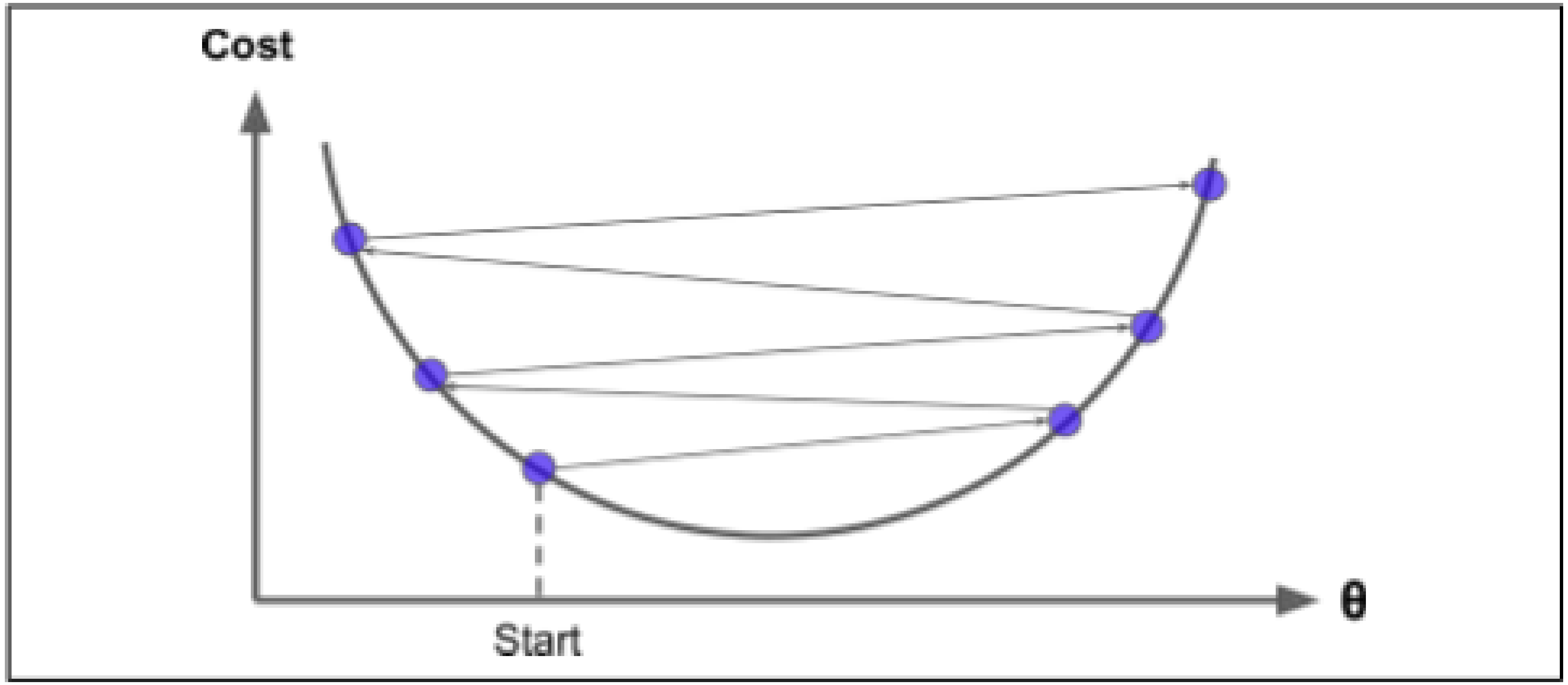
Being too ‘gready’ with the learning rate

- Learning rate is too big will make gradient descend diverge away from finding the minimum $MSE(\theta_{0, \ldots, } \theta_n)$
- That is, the model parameters $\theta_{0 \ldots} \theta_1$ are changing in big steps that makes the learning algorithm get lost



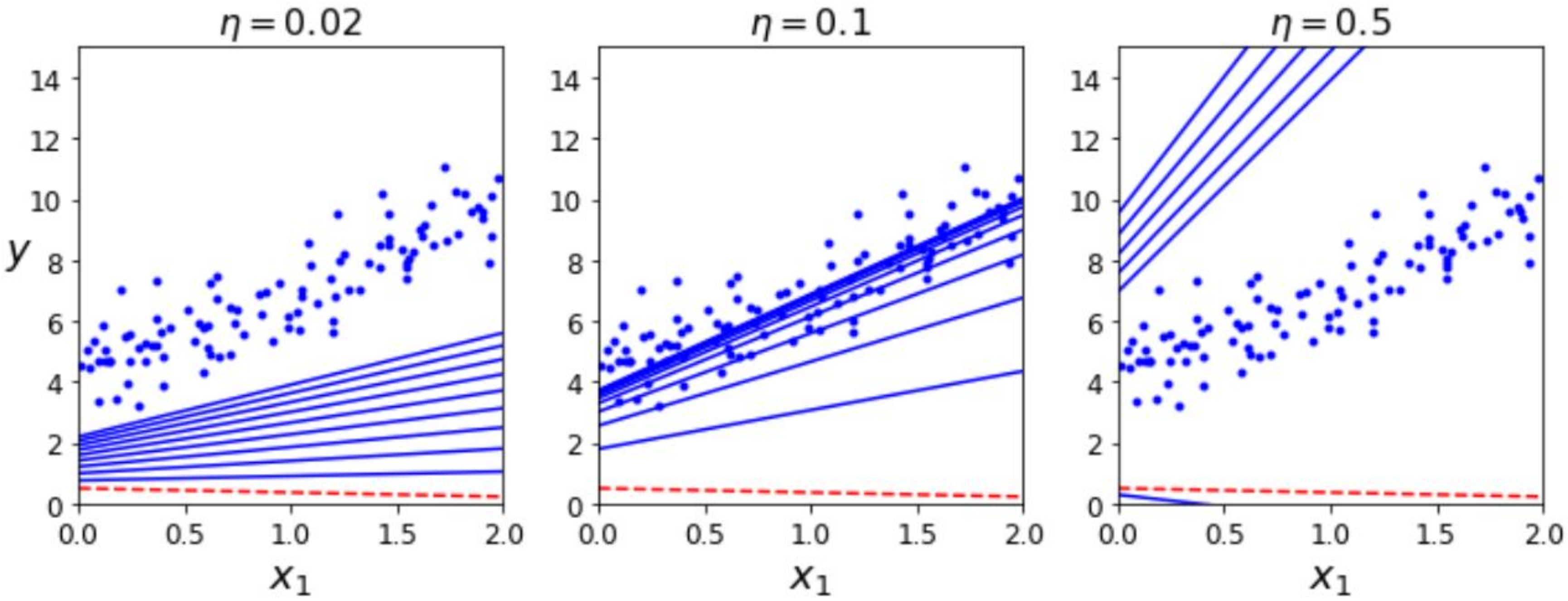
Being too ‘gready’ with the learning rate - Exercise

- Explain to yourselves in your group, the process on how the too ‘gready’ scenario evolves, use the curve below for help.



Adjusting the learning rate - example

- To the <u>left</u>: ‘Scrooge’ learning rate η = *0.02* – too small approaching MSE optimum slowly
- In the <u>middle</u>: ‘Appropriate’ learning rate η = *0.1* – approaches MSE optimum in reasonable time
- To the right: ‘’



Learning rate – SGDRegressor example

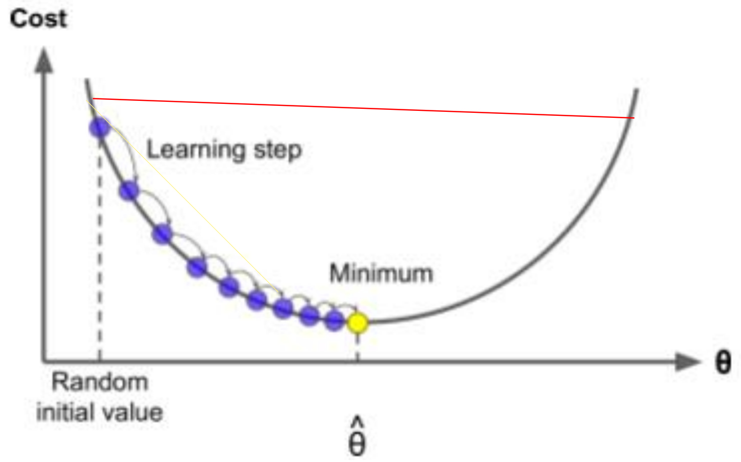
Available parameters:

- **learning_rate:string, default='invscaling'**
  The learning rate schedule:
  - 'constant': eta = eta0
  - 'optimal': eta = 1.0 / (alpha * (t + t0)) where t0 is chosen by a heuristic proposed by Leon Bottou.
  - 'invscaling': [default]: eta = eta0 / pow(t, power_t)
  - 'adaptive': eta = eta0, as long as the training keeps decreasing.
    Each time n_iter_no_change consecutive epochs fail to decrease the training loss by tol or fail to increase validation score by tol if early_stopping is True, the current learning rate is divided by 5.
- **eta0:double, default=0.01**
  The initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules. The default value is 0.01.
- **power_t:double, default=0.25**
  The exponent for inverse scaling learning rate.

Source: scikit-learn.org

Nice linear regression properties with MSE cost function

- The MSE cost function for a Linear Regression model a so-called convex function
- Convex: Red line segment will never cross the curve below
- This means that is has only one global minimum.
- It is a continuous function with a slope that never changes abruptly
- Gradient Descent is then guaranteed to approach arbitrarily close the global minimum.



Performing Gradient Descent Stochastic – the principle

- Instead of processing the entire training set, we pick one training set instance at a time
- Faster than the Batch Gradient Descend
- But more erractic

While (Minimum <u>not</u> reached)

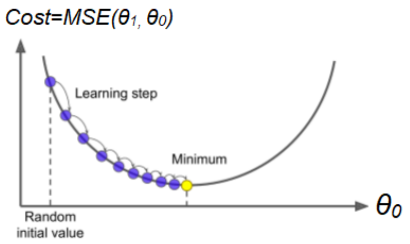
{

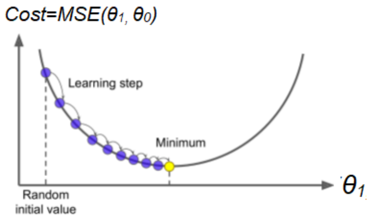
Based on picking one element at a time in the learning set randomly:

$\theta_{0\ =} = \theta_0 - LearningRate * \partial MSE(\theta_1, \theta_0) / \partial\theta_0$

$\theta_{1\ =} = \theta_1 - LearningRate * \partial MSE(\theta_1, \theta_0) / \partial\theta_1$

}

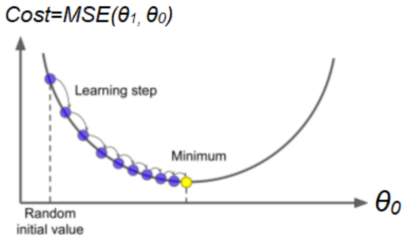


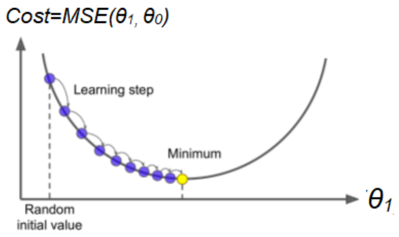


Performing Gradient Descent Mini-batch – the principle

- Instead of processing the entire training set, we pick a batch training set instance at a time
- Trade of between batch and stochastic gradient descend
- Fast and less erractic

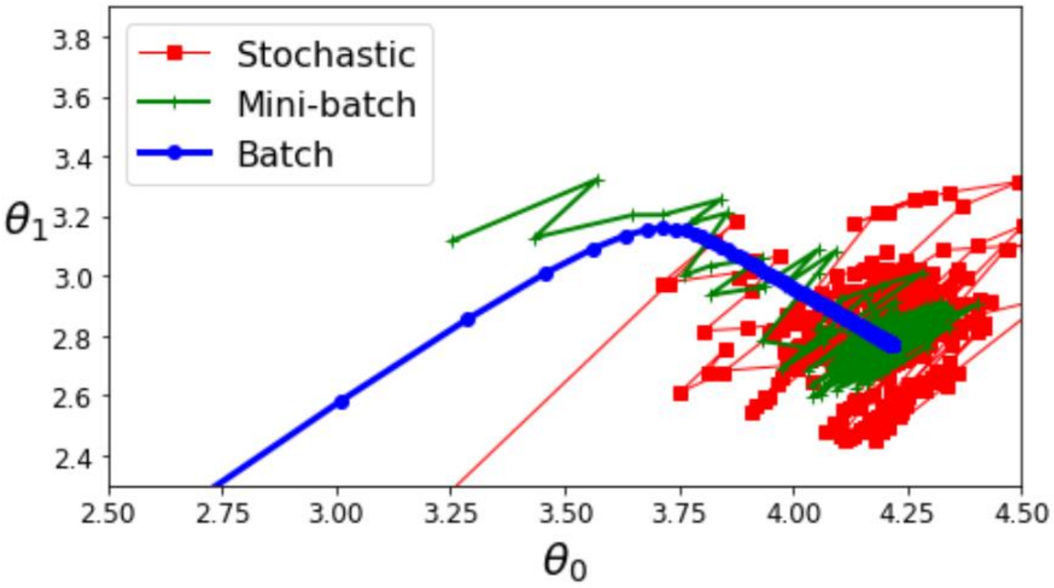
While (Minimum <u>not</u> reached)
{
Based on picking batch of elements at a time in the learning set randomly:
$\theta_0 {}_= = \theta_0$ - LearningRate * $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_0$
$\theta_1 {}_= = \theta_1$ - LearningRate * $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_1$
}





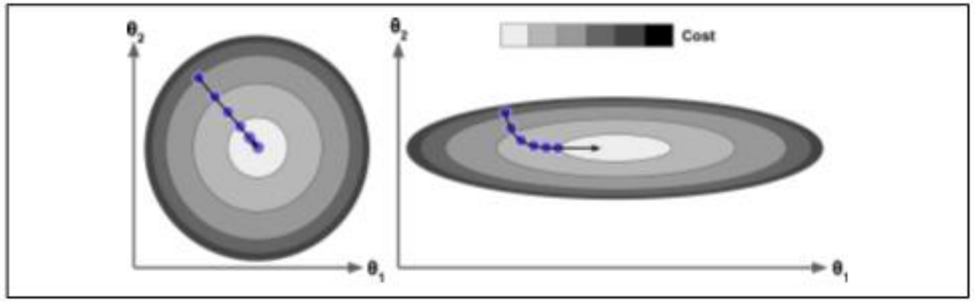
Comparing gradient descend approaches

Different paths in development in model parameters



Gradient Descend – Feature scaling needed

- Feature scaling: Features (learning input) have same order of magnitude – e.g. in the the area -1 to 1
- To the left: Feature scaling applied -> Minimum of cost function approached faster
- To the right: Feature scaling not applied -> Minimum of cost function approached slower
- Feature scaling can be obtained by the Scikit-Learn's StandardScaler
- Feature scaling is recommended for gradient descent algorithms



Some comparison on linear regression algorithms

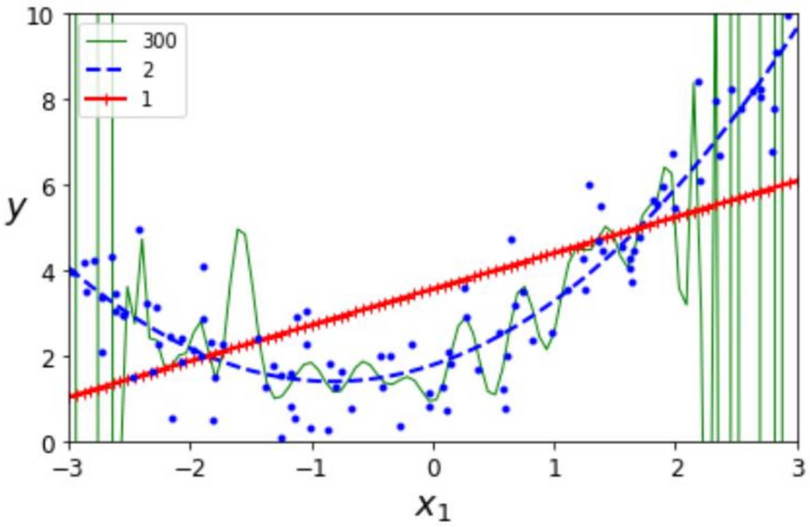
Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	n/a
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

Learning curves

- **Purpose:** Evaluating a model by comparing performance RMSE on both the training and validation sets
- **Focus:** Overfit and underfit situations

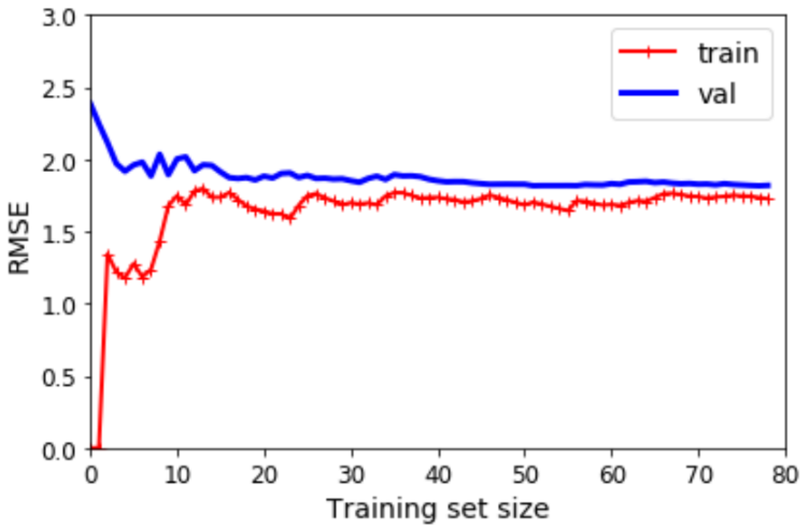
Learning curves - examples

- **Overfit:** Green curve, polynomium degree 300. Performs well on the training set. Will it also perform well on the validation set?
- **Underfit:** Red curve, straight line (polynomium degree 1). Comparable lower performance on both training and validation sets.
- **Good fit:** Blue curve. Polynomium degree 2. Good performance on both training and validation sets.



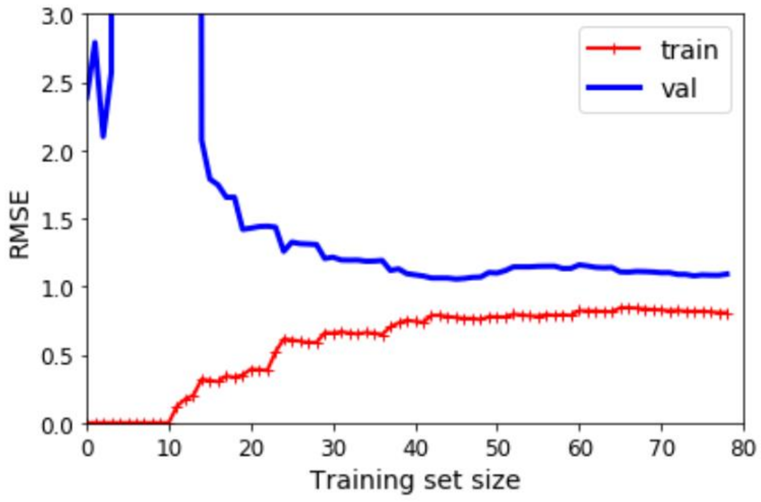
Learning curves – recognizing underfit

- Relatively poor performance RMSE on both validation and training sets
- Performance RMSE on both validation and training sets are compareable



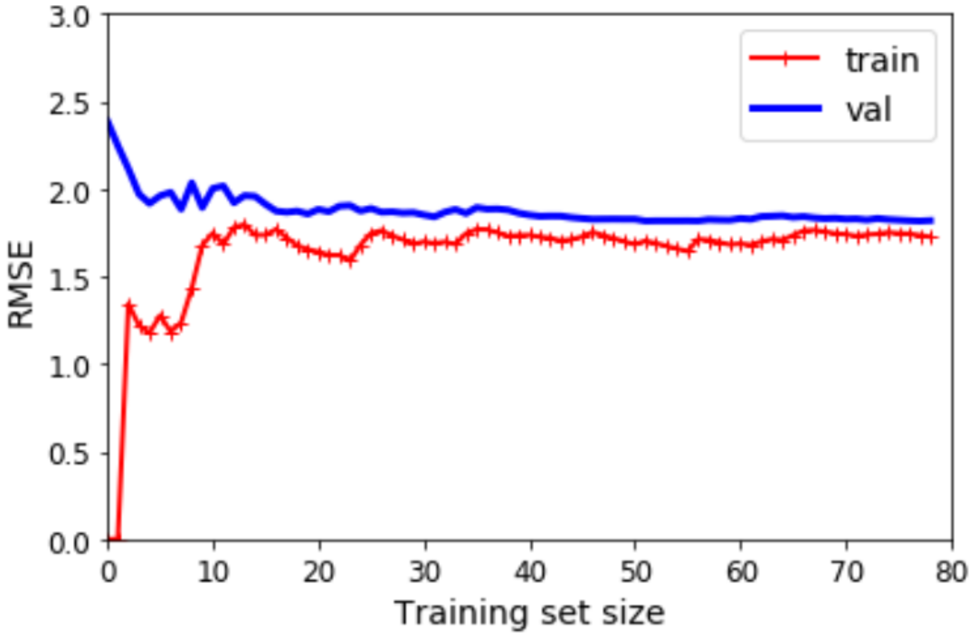
Learning curves – recognizing overfit

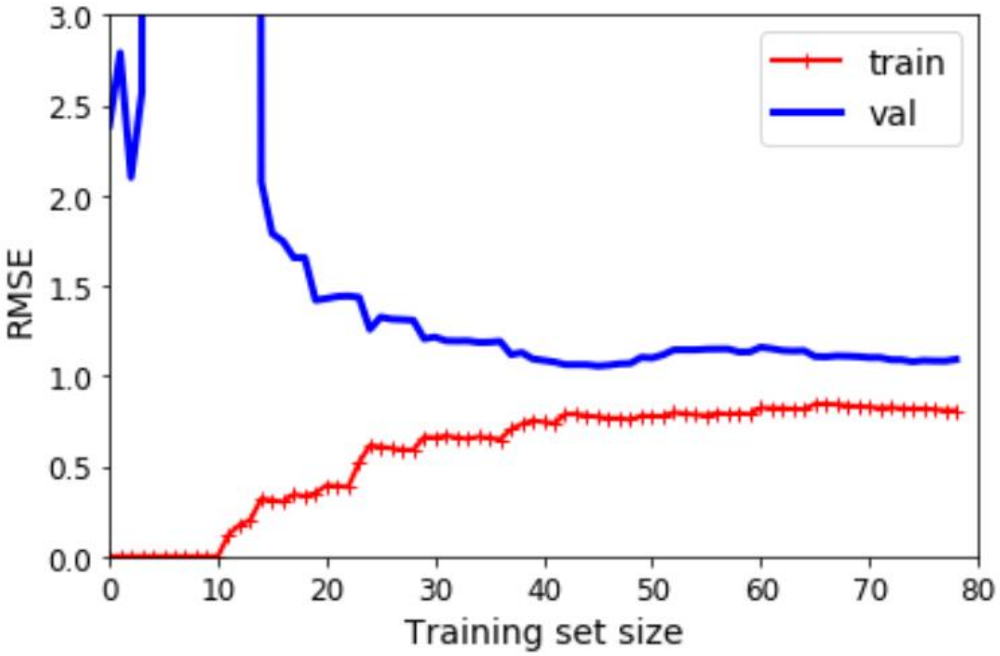
- Relatively good performance RMSE on the training set and a lot worse both the validation set
- Performance RMSE on both validation and training sets are less compareable



Learning curves – comparing underfit and overfit

- To the left: Underfit situation - aka high bias
- To the right: Overfit situation – aka high variance



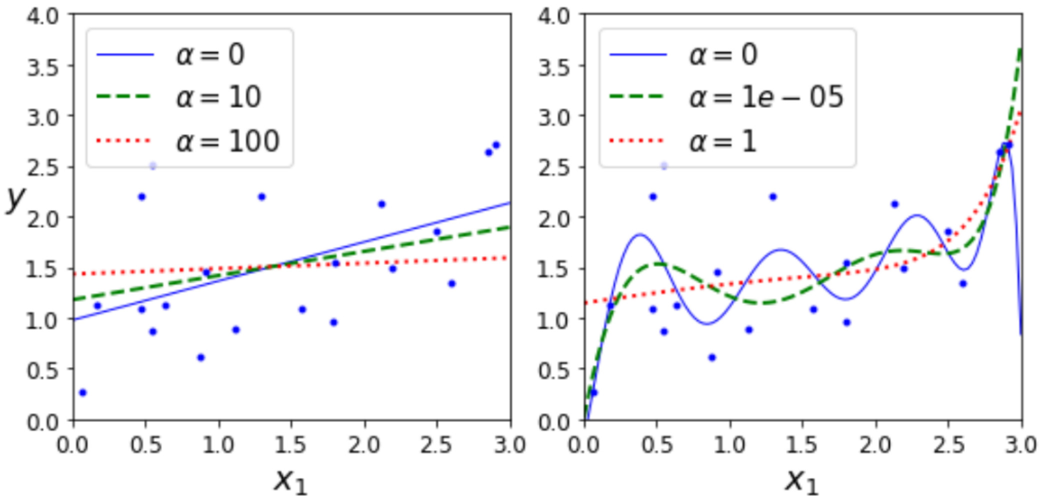


Regularized models

- **Purpose**: Avoiding the overfitting situation
- **Overfitting**: Model fits training set well, but fits the validation set badly
- **Polynomial models:** Reduce polynomial degrees
- **Linear models**: Constrain the model parameters $\theta_1, \ldots, \theta_n$ - That is reducing slopes

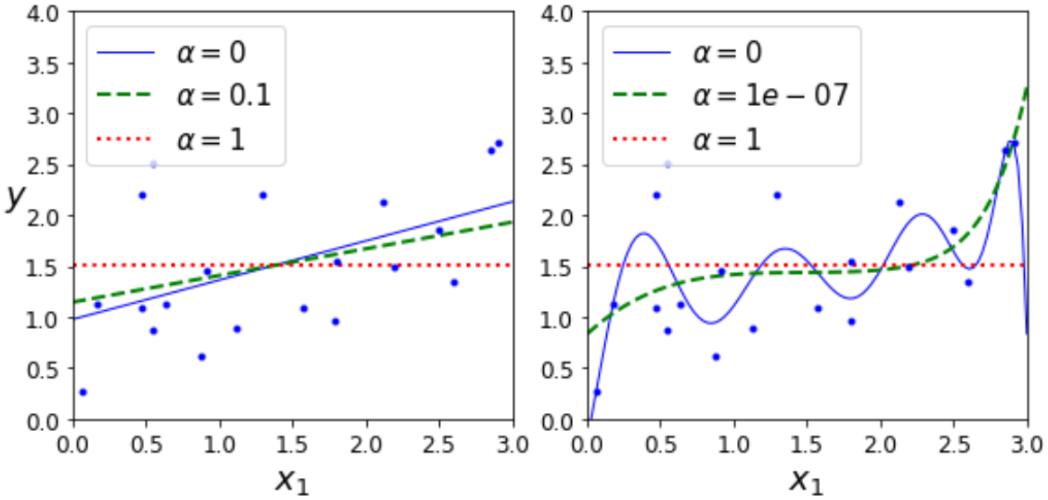
Ridge Regression

- Adding a penalty to the cost function MSE during learning only
- Keeps models weights as small as possible
- Different learning conditions depending ‘penalty factor’ α
- Linear model to the left – Polynomial model to the right



Lasso Regression

- Adding a penalty to the cost function MSE during <u>learning only</u>
- Eliminates the <u>least important features</u>
- Different learning conditions depending 'penalty factor' α
- Linear model to the left – Polynomial model to the right

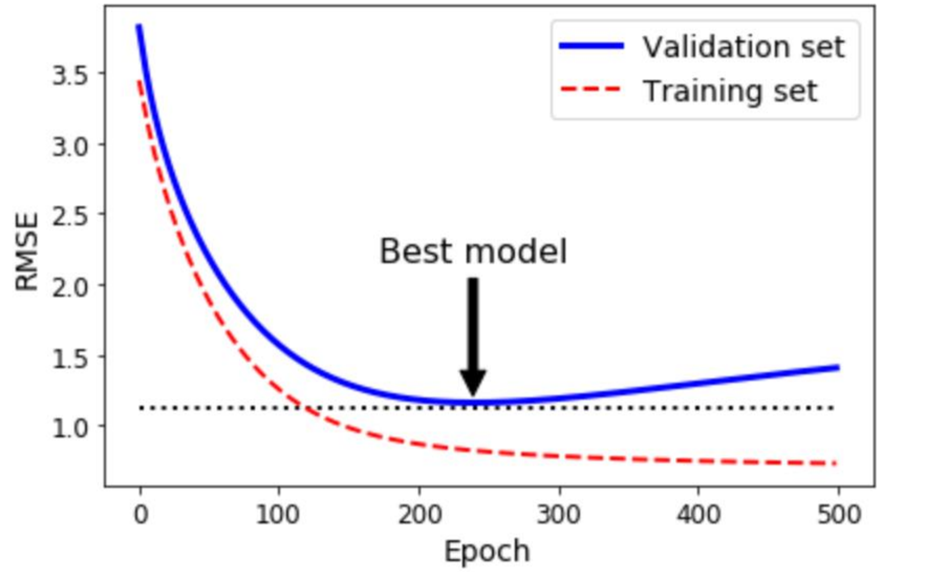


Elastic Net

- Is a combination of the Ridge and Lasso regression

Early Stopping – stop learning when validation is best

- Error RMSE when predicting on training set approaches zero
- Error RMSE when predicting on the validation set reaches the minimum
- The model is best at this minimum
- If proceeding further, we will recognize the overfit situation



Early stopping – SGDRegressor example

Available parameters:

- **early_stopping : bool**, default=False
  Whether to use early stopping to terminate training when validation score is not improving. If set to True, it will automatically set aside a fraction of training data as validation and terminate training when validation score is not improving by at least tol for n_iter_no_change consecutive epochs.
- **n_iter_no_change** : int, default=5
  Number of iterations with no improvement to wait before early stopping.
- **validation_fraction : float**, default=0.1
  The proportion of training data to set aside as validation set for early stopping. Must be between 0 and 1. Only used if early_stopping is True.

Source: scikit-learn.org