Chapter 2

Supervised Machine Learning: Linear Models and Fundamentals

Contents: Supervised Learning: Linear Models and Fundamentals

- Linear Regression
- Linear Classification/ Logistic Regression
- Regularization, Overfitting and Underfitting
- High dimensional data, Multivariate
- Parametric and non-parametric methods

Linear Regression

- Regression is a technique used to predict the value of a response (dependent) variables, from one or more predictor (independent) variables.
- Most commonly used regressions techniques are: **Linear Regression** and **Logistic Regression**.
- In linear regression problems, the goal is to predict a real-value variable y from a given pattern X. In the case of linear regression the output is a linear function of the input.

Simple Linear Model

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R} \times \mathbb{R}$ called **training data**,

compute the line that describes the data generating process best.

For given predictor/input $x \in \mathbb{R}$, the **simple linear model** predicts/outputs

$$\hat{y}(x) := \hat{\beta}_0 + \hat{\beta}_1 x$$

with **parameters** $(\hat{\beta}_0, \hat{\beta}_1)$ called

$\hat{\beta}_0$ **intercept / bias / offset**

$\hat{\beta}_1$ **slope**

```
1 predict-simple-linreg(x ∈ ℝ, β̂0, β̂1 ∈ ℝ):
2     ŷ := β̂0 + β̂1 x
3     return ŷ
```

- Such problems can be solved analytically using the least squares estimates i.e the linear line that best predicts the data :

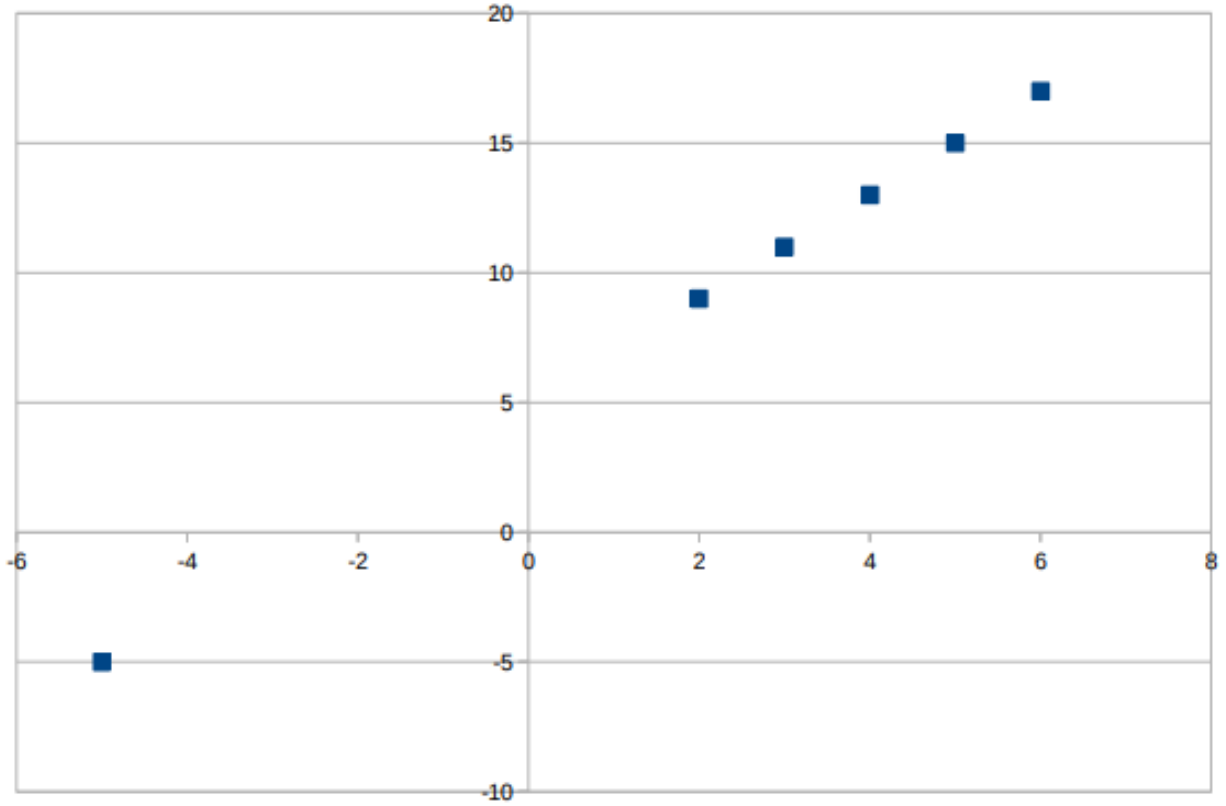
$$\hat{\beta}_1 = \frac{\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N}(x_n - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

1 learn $-$simple$-$linreg($\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\} \in \mathbb{R} \times \mathbb{R}$):
2 $\quad \bar{x} := \frac{1}{N} \sum_{n=1}^{N} x_n$
3 $\quad \bar{y} := \frac{1}{N} \sum_{n=1}^{N} y_n$
4 $\quad \hat{\beta}_1 := \frac{\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N}(x_n - \bar{x})^2}$
5 $\quad \hat{\beta}_0 := \bar{y} - \hat{\beta}_1 \bar{x}$
6 $\quad$ return $(\hat{\beta}_0, \hat{\beta}_1)$

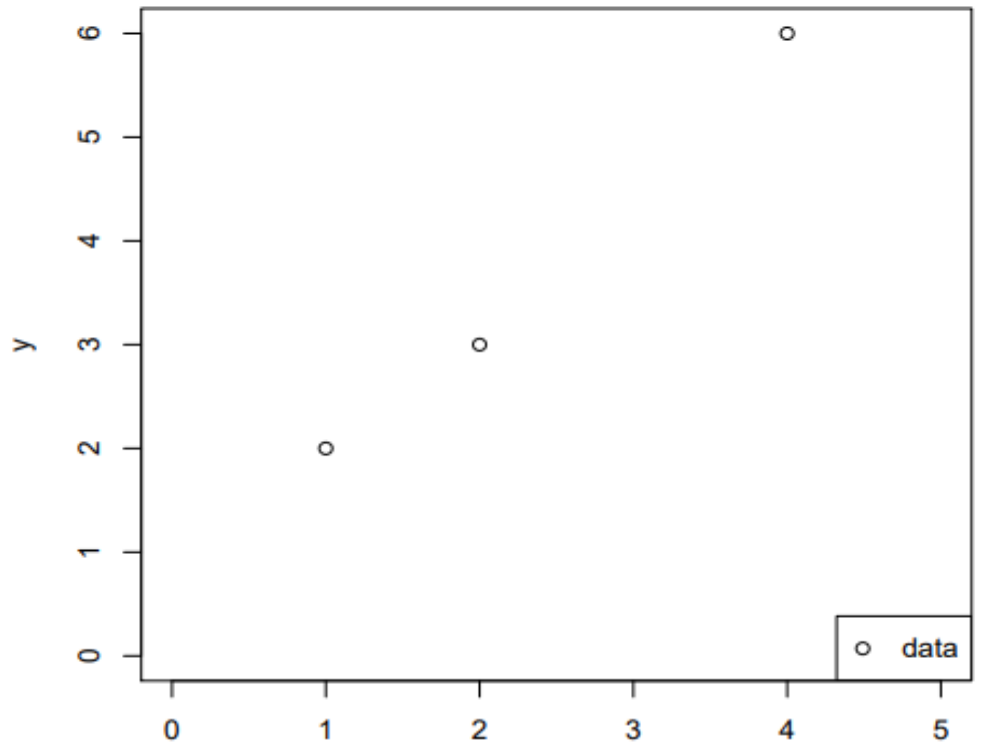
- Example: find `m` and `b` to get the function `y=mx+b` using the formular for $(\hat{\beta}_0, \hat{\beta}_1)$

x	y
-5	-5
2	9
3	11
4	13
5	15
6	17



Exercise

Given the data $\mathcal{D} := \{(1,2),(2,3),(4,6)\}$, predict a value for $x = 3$.



When is a model good?

We still need to specify what "describes the data generating process best" means. — What are good predictions $\hat{y}(x)$?

Predictions are considered better the smaller the difference between

- an **observed** y_n (for predictors x_n) and
- a **predicted** $\hat{y}_n := \hat{y}(x_n)$

is on average, e.g., the smaller the (pointwise) **L2 loss** / **squared error**:

$$\ell(y_n, \hat{y}_n) := (y_n - \hat{y}_n)^2$$

Note: Other error measures such as absolute error $\ell(y_n, \hat{y}_n) = |y_n - \hat{y}_n|$ are also possible, but more difficult to handle.

Cont....

Pointwise losses are usually averaged over a dataset $\mathcal{D}$, then just called **error**, e.g.,

$$\text{err}(\hat{y}; \mathcal{D}) := \frac{1}{N}\text{RSS}(\hat{y}; \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}(x_n))^2$$

$$\text{or } \text{err}(\hat{y}; \mathcal{D}) := \text{RSS}(\hat{y}; \mathcal{D}) := \sum_{n=1}^{N}(y_n - \hat{y}(x_n))^2$$

called **residual sum of squares** (RSS) or also **L2 loss**.

Equivalently, often **Root Mean Square Error** (RMSE) is used:

$$\text{err}(\hat{y}; \mathcal{D}) := \text{RMSE}(\hat{y}; \mathcal{D}) := \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}(x_n))^2}$$

Note: RMSE has the same scale level / unit as the original target y, e.g., if y is measured in meters so is RMSE.

- Models should not just reproduce the data, but **generalize**, i.e., predict well on fresh / unseen data (called **test data**).

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R} \times \mathbb{R}$ called **training data**,

compute the **parameters** $(\hat{\beta}_0, \hat{\beta}_1)$ of a linear **regression function**

$$\hat{y}(x) := \hat{\beta}_0 + \hat{\beta}_1 x$$

s.t. for a set $\mathcal{D}^{\text{test}} \subseteq \mathbb{R} \times \mathbb{R}$ called **test set** the **test error**

$$\text{err}(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|D^{\text{test}}|} \sum_{(x,y) \in \mathcal{D}^{\text{test}}} (y - \hat{y}(x))^2$$

is minimal.

Note: $\mathcal{D}^{\text{test}}$ has (i) to be from the same data generating process and (ii) not to be available during training.

Linear Classification/ Logistic Regression

- The classification Problem

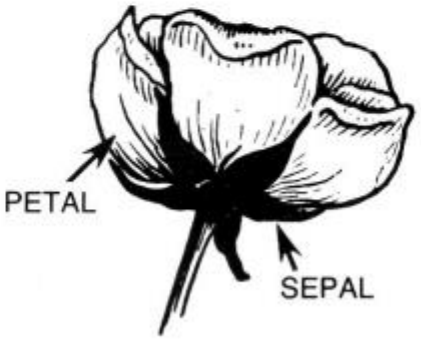
Example: classifying iris plants (Anderson 1935).

iris setosa

iris versicolor

150 iris plants (50 of each species):

- 3 species:
  setosa, versicolor, virginica
- length and width of sepals (in cm)
- length and width of petals (in cm)



iris virginica

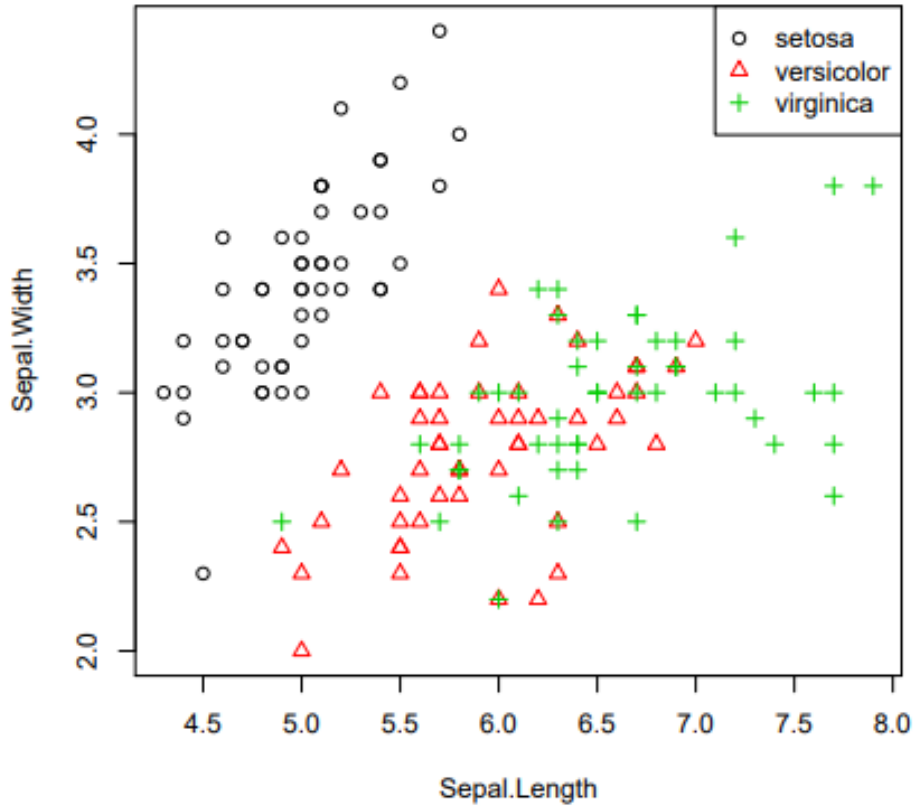
Given the lengths and widths of sepals and petals of an instance, which iris species does it belong to?

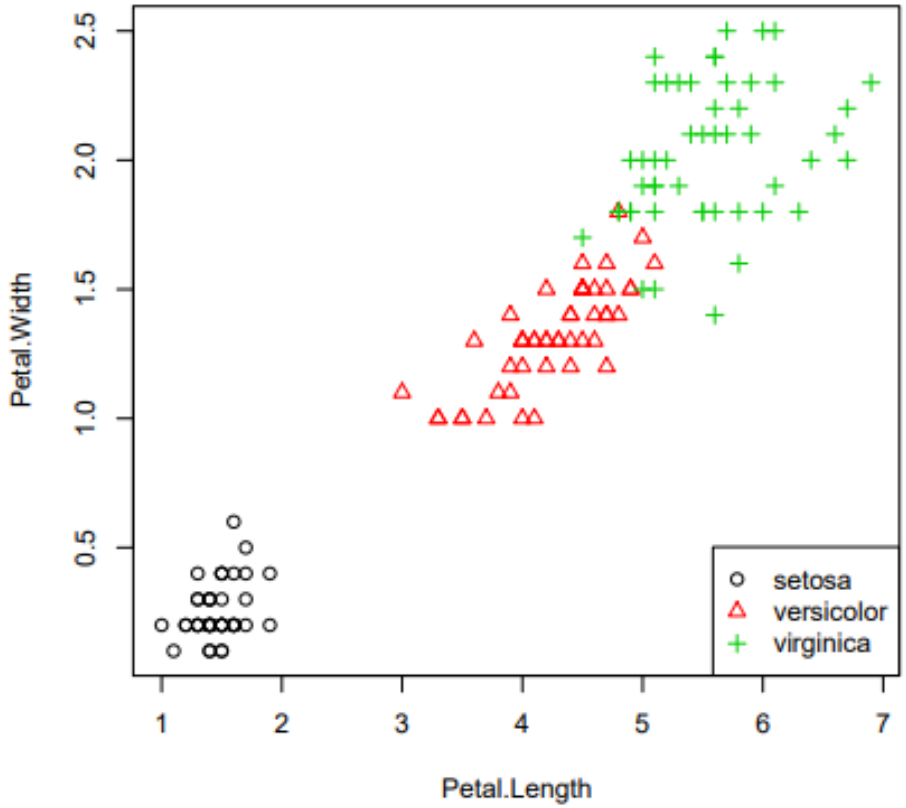
[source: iris species database, http://www.badbear.com/signa]

Cont…. the data:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
⋮	⋮	⋮	⋮	⋮	
51	7.00	3.20	4.70	1.40	versicolor
52	6.40	3.20	4.50	1.50	versicolor
53	6.90	3.10	4.90	1.50	versicolor
⋮	⋮	⋮	⋮	⋮	
101	6.30	3.30	6.00	2.50	virginica
⋮	⋮	⋮	⋮	⋮	
150	5.90	3.00	5.10	1.80	virginica

Cont….


setosa
versicolor
virginica
Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


setosa
versicolor
virginica
Petal.Width
Petal.Length
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Binary Classification

Let us start simple and consider two classes only,
e.g., target space $\mathcal{Y} := \{0, 1\}$.

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R}^M \times \mathcal{Y}$ called **training data**,

we want to estimate a model $\hat{y}(x)$ s.t. for a set $\mathcal{D}^{\text{test}} \subseteq \mathbb{R}^M \times \mathcal{Y}$ called **test set** the **test error** (here: **misclassification rate**)

$$\text{err}(\hat{y}; \mathcal{D}^{\text{test}}) := \text{mcr}(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|D^{\text{test}}|} \sum_{(x,y) \in \mathcal{D}^{\text{test}}} I(y \neq \hat{y}(x))$$

is minimal.

Note: $I(A) := 1$ if statement A is true, $I(A) := 0$ otherwise (**indicator function**).
$\mathcal{D}^{\text{test}}$ has (i) to be from the same data generating process and (ii) not to be available during training.

Cont.... the data:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species setosa
1	5.10	3.50	1.40	0.20	1
2	4.90	3.00	1.40	0.20	1
3	4.70	3.20	1.30	0.20	1
⋮	⋮	⋮	⋮	⋮	
51	7.00	3.20	4.70	1.40	0
52	6.40	3.20	4.50	1.50	0
53	6.90	3.10	4.90	1.50	0
⋮	⋮	⋮	⋮	⋮	
101	6.30	3.30	6.00	2.50	0
⋮	⋮	⋮	⋮	⋮	
150	5.90	3.00	5.10	1.80	0

Binary classification with Linear Regression

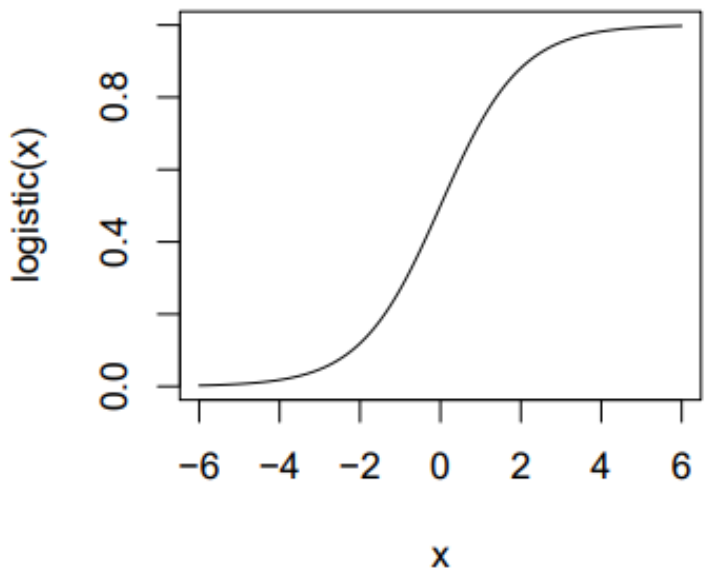
- One idea could be to optimize the linear regression model
  - $Y = mx + b$
- This has several problems
  - It is not suited for predicting y as it can assume all kinds of intermediate values.
  - It is optimizing for the wrong loss.
- Instead of predicting Y directly, we predict
  - $p(Y = 1|X; \beta)$ — the probability of Y being 1 knowing X.
  - But linear regression is also not suited for predicting probabilities, as its predicted values are principally unbounded.
  - Use a trick and transform the unbounded target by a function that forces it into the unit interval [0, 1]

Logistic function:

$$\text{logistic}(x) := \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$

Basic properties:

- has values between 0 and 1,
- converges to 1 when approaching $+\infty$,
- converges to 0 when approaching $-\infty$,
- is smooth and symmetric at $(0, 0.5)$.



- Now coming back to our logistic regression problem, Let us assume that z is a linear function of a single explanatory variable x. We can then express z as follows:

$$z = w_0 + w_1 x$$

- And the logistic function can now be written as:

$$g(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

- Note that g(x) is interpreted as the probability of the dependent variable.
- g(x) = 0.7, gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

- The input to the sigmoid function ‘g’ doesn’t need to be linear function. It can very well be a circle or any shape.

$$z=(w_0+w_1x_1^2+w_2x_2^2)$$

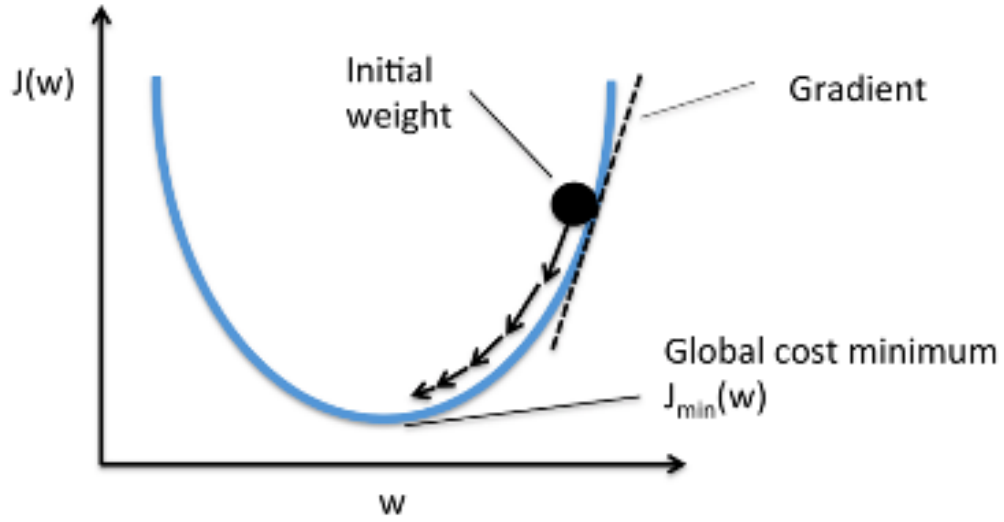
- Loss function:

Misclassification rate

$$\begin{aligned}\text{mcr}(\hat{\beta};\mathcal{D}^{\text{test}}) &:= \text{mcr}(\hat{y}(.;\hat{\beta});\mathcal{D}^{\text{test}}) \\ &= \frac{1}{|D^{\text{test}}|}\sum_{(x,y)\in\mathcal{D}^{\text{test}}} I(y \neq \hat{y}(x;\hat{\beta})) \\ &= \frac{1}{|D^{\text{test}}|}\sum_{(x,y)\in\mathcal{D}^{\text{test}}} I(y \neq I(\text{logistic}(\hat{\beta}^T x) \geq 0.5))\end{aligned}$$

Gradient descent Algorithm:

- When we plot the cost/loss function `J(w)` vs `w`. It is represented as below:



- As we see from the curve, there exists a value of parameters `W` which has the minimum cost `Jmin`. Now we need to find a way to reach this minimum cost.

Cont….

- In the gradient descent algorithm, we start with random model parameters and calculate the error for each learning iteration, keep updating the model parameters to move closer to the values that results in minimum cost.

repeat until minimum cost: {

$$w_j = w_j - \alpha \partial / \partial w_j J(W)$$

}

- In the above equation we are updating the model parameters after each iteration. The second term of the equation calculates the slope or gradient of the curve at each iteration.

Cont….

- The gradient of the cost function is calculated as partial derivative of cost function `J` with respect to each model parameter `wj, j` takes value of number of features `[1 to n].`
- `α,` alpha, is the learning rate, or how quickly we want to move towards the minimum. If `α` is too large, we can overshoot. If `α` is too small, means small steps of learning hence the overall time taken by the model to observe all examples will be more.

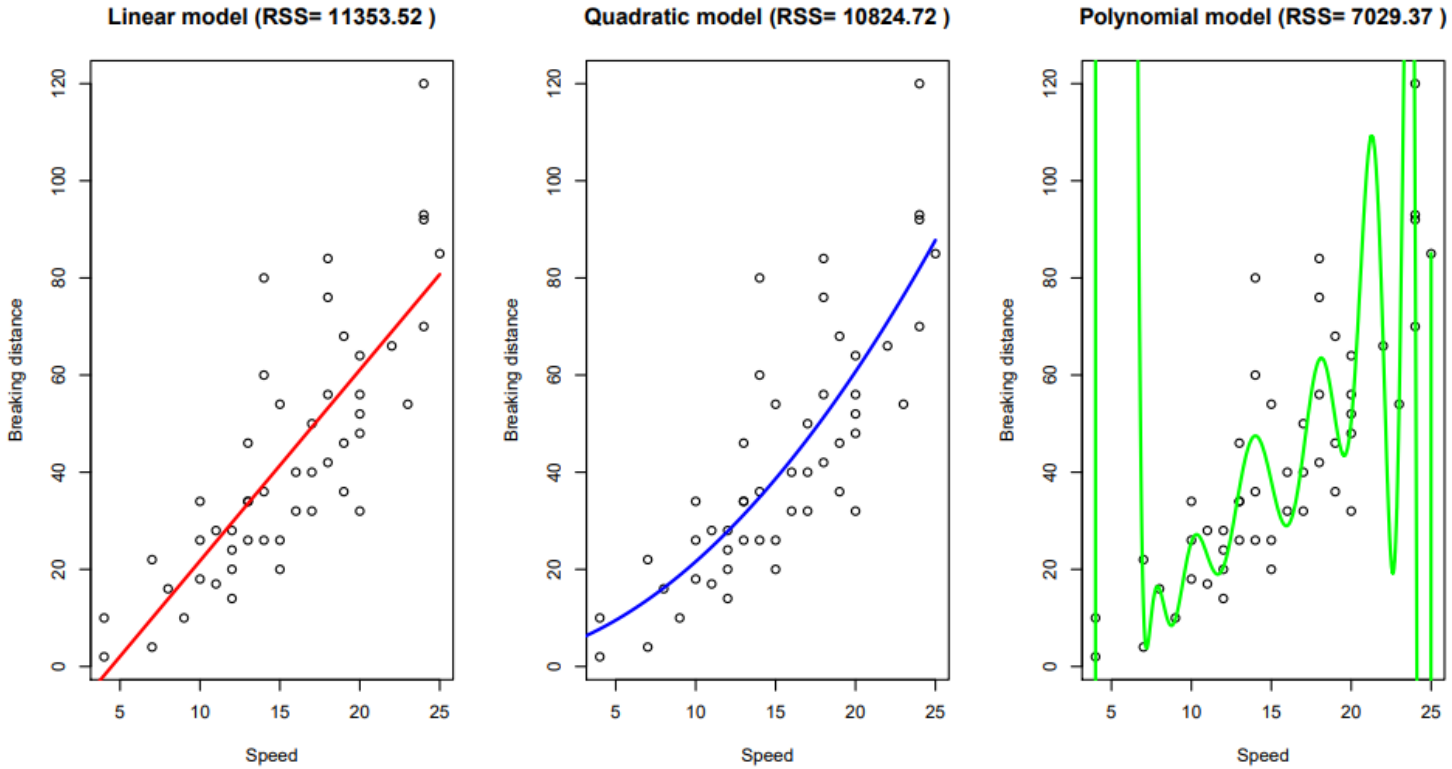
Cont….

There are three ways of doing gradient descent:

- **Batch gradient descent:** Uses all of the training instances to update the model parameters in each iteration.
- **Mini-batch Gradient Descent:** Instead of using all examples, Mini-batch Gradient Descent divides the training set into smaller size called batch denoted by ‘b’. Thus a mini-batch ‘b’ is used to update the model parameters in each iteration.
- **Stochastic Gradient Descent (SGD):** updates the parameters using only a single training instance in each iteration. The training instance is usually selected randomly. Stochastic gradient descent is often preferred to optimize cost functions when there are hundreds of thousands of training instances or more, as it will converge more quickly than batch gradient descent

Overfitting and Underfitting

- Fitting of models



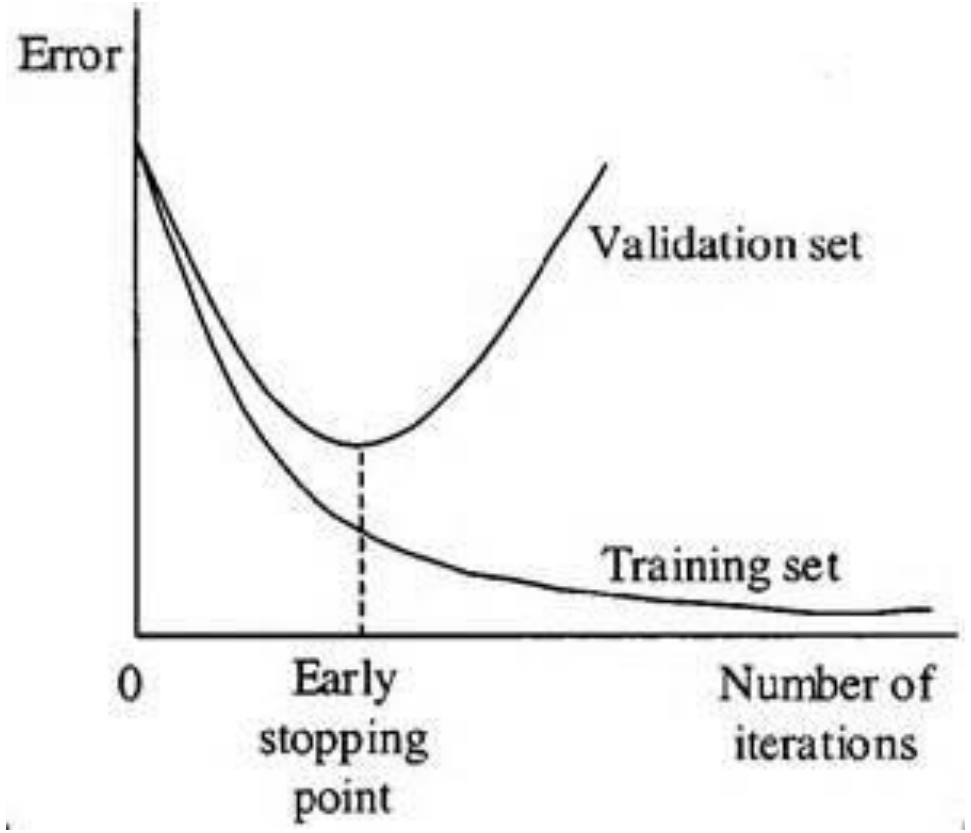
Cont....

- Underfitting:
  - the model is not complex enough to explain the data well.
  - results in poor predictive performance.
  - When the model has fewer features and hence not able to learn from the data very well. This model has high bias
- Overfitting:
  - the model is too complex, it describes the
    - noise, inherent random variations of the data generating process, instead of the
    - signal, the underlying relationship between target and predictors.
  - results in poor predictive performance as well.
  - When the model has complex functions and hence able to fit the data very well but is not able to generalize to predict new data. This model has high variance.

There are three main options to address the issue of over-fitting:

1. **Reduce the number of features:** Manually select which features to keep. Doing so, we may miss some important information, if we throw away some features.
2. **Regularization:** Keep all the features, but reduce the magnitude of weights W. Regularization works well when we have a lot of slightly useful feature.
3. **Early stopping:** When we are training a learning algorithm iteratively such as using gradient descent, we can measure how well each iteration of the model performs. Up to a certain number of iterations, each iteration improves the model. After that point, however, the model's ability to generalize can weaken as it begins to over-fit the training data.

Error curve helps to control over-fitting through early stopping



Regularization

- Regularization can be applied to both linear and logistic regression by adding a penalty term to the error function in order to discourage the coefficients or weights from reaching large values.
- Dropping a feature/variable x_m from the model is equivalent to forcing its model parameter β_m to be 0.
- The two most common types of regularization are L1 and L2 regularization:

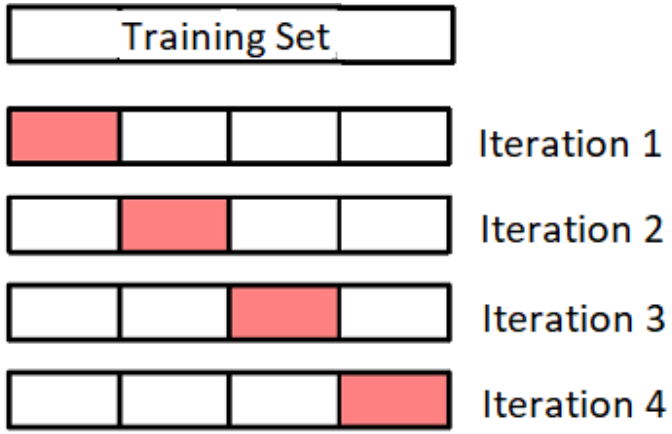
L1 = $\frac{\lambda}{2n}\sum_{j=1}^{n}|w_j|$

L2 = $\frac{\lambda}{2n}\sum_{j=1}^{n}w_j{}^2$

Cont….

- These two types of regularization are added at the end of the loss/cost function to control overfitting.
- L2 shrinks all the coefficient by the same proportions but eliminates none, while L1 can shrink some coefficients to zero, thus performing feature selection.
- λ is a hyperparameter that cannot be learned by direct loss minimization.
- **Hyper-parameters** are “higher-level” parameters that describe structural information about a model that must be decided before fitting model parameters, examples of hyper-parameters include the learning rate, alpha and the Regularization term lambda (λ).

- In machine learning, the overall data set is divided into three:
  - the training data set: which is used to fit the different models
  - validation data set: which is used for model selection including the searching of optimal hyper-parameters (hyper-parameter tuning)
  - test data set: which is unseen data used to report the final performance of the model
- **Cross validation** is used when we have very limited dataset for doing the above split. That is to use slight portions of the data for validation and exchange it through iterations so that we use as much of the available data as possible for training



High dimensional data, Multivariate

High-dimensional data occurs in different situations:

1. Data that comes naturally with many predictors.
   - e.g., text classification
     (# predictors = # words in the bag-of-words representation, e.g., 30.000)

2. Models that extract many predictor variables from objects to classify.
   - variable interactions
   - derived variables
   - complex objects such as graphs, texts, etc.
     - Situation 1 often really is a special case of this one.

3. Data with few examples compared to the number of variables ("small n, large p").
   - gene expression / microarray data

Variable types and coding

The most common variable types:

numerical / interval-scaled / quantitative

- differences and quotients etc. are meaningful,
- usually with domain $\mathcal{X} := \mathbb{R}$,
- e.g., temperature, size, weight.

nominal / discrete / categorical / qualitative / factor

- differences and quotients are **not** defined,
- usually with a finite, enumerated domain,
- e.g., $\mathcal{X} := \{\text{red}, \text{green}, \text{blue}\}$
  or $\mathcal{X} := \{\text{a}, \text{b}, \text{c}, \ldots, \text{y}, \text{z}\}$.

ordinal / ordered categorical

- levels are ordered, but differences and quotients are not defined,
- usually with a finite, enumerated domain,

Cont....

Nominals are usually encoded as a set of binary **dummy variables** (aka **indicator variables**, **one hot encoding**):

$$\delta_{x_0}(X) := \begin{cases} 1, & \text{if } X = x_0, \\ 0, & \text{else} \end{cases}$$

one for each $x_0 \in \mathcal{X}$ (but one).

Example: $\mathcal{X} := \{\text{red}, \text{green}, \text{blue}\}$

one variable X with 3 levels: red, green, blue

$\downarrow$ replace by

two variables $\delta_{\text{red}}(X)$ and $\delta_{\text{green}}(X)$ with 2 levels each: 0, 1

X	$\delta_{\text{red}}(X)$	$\delta_{\text{green}}(X)$
red	1	0
green	0	1
blue	0	0
—	1	1

Multivariate

- Multivariate refers to multiple dependent variables that result in one outcome.
- Real regression problems are more complex than simple linear regression in many aspects:
  - There is more than one predictor.
  - The target may depend non-linearly on the predictors.
- Multivariate can be computed using Multiple linear regression

The (Multiple) Linear Regression Problem

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R}^M \times \mathbb{R}$ called **training data**,

compute the **parameters** $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_M)$ of a linear **regression function**

$$\hat{y}(x) := \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_M x_M$$

s.t. for a set $\mathcal{D}^{\text{test}} \subseteq \mathbb{R}^M \times \mathbb{R}$ called **test set** the **test error**

$$\text{err}(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|D^{\text{test}}|} \sum_{(x,y) \in \mathcal{D}^{\text{test}}} (y - \hat{y}(x))^2$$

is minimal.

Note: $\mathcal{D}^{\text{test}}$ has (i) to be from the same data generating process and (ii) not to be available during training.

Several Predictors

Several predictor variables $x_{n,1}, x_{n,2}, \ldots, x_{n,M}$:

$$\begin{aligned}\hat{y}_n &= \hat{\beta}_0 + \hat{\beta}_1 x_{n,1} + \hat{\beta}_2 x_{n,2} + \cdots \hat{\beta}_M x_{n,M} \\ &= \hat{\beta}_0 + \sum_{m=1}^{M} \hat{\beta}_m x_{n,m}\end{aligned}$$

with $M + 1$ parameters $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_M$.

- Note: n ∈ 1:N denotes the sample/example/instance/case.

Parametric and non-parametric methods

- **Parametric Machine Learning Algorithms**
  - Assumptions can greatly simplify the learning process, but can also limit what can be learned.
  - This are algorithms that simplify the function to a known form
  - A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model.
  - No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.
  - The algorithms involve two steps:
    1. Select a form for the function.
    2. Learn the coefficients for the function from the training data.

Cont….

- Examples of parametric machine learning algorithms include:
  - Logistic Regression
  - Linear Discriminant Analysis
  - Perceptron
  - Naive Bayes
  - Simple Neural Networks
- **Nonparametric Machine Learning Algorithms**
  - Are algorithms that do not make strong assumptions about the form of the mapping function
  - By not making assumptions, they are free to learn any functional form from the training data.
  - Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don’t want to worry too much about choosing just the right features.

Cont….

- Nonparametric methods seek to best fit the training data in constructing the mapping function, whilst maintaining some ability to generalize to unseen data.
- Examples of nonparametric machine learning algorithms include:
  - k-Nearest Neighbors
  - Decision Trees like CART and C4.5
  - Support Vector Machines