Machine Learning L+Pr

Béla J. Szekeres, PhD

Lecture 2 / I

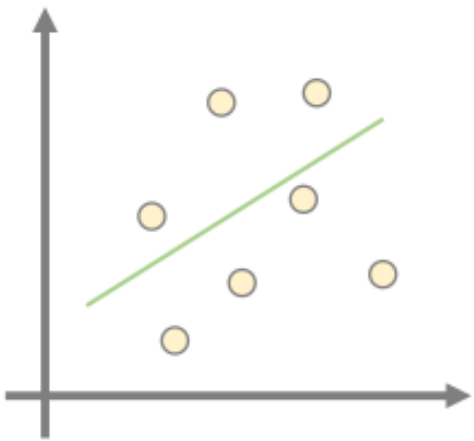
ELTE Faculty of Informatics, Szombathely, Hungary

Topics of this day

- Linear regression
- Logistic regression
- Naive Bayes Classifier
- Bias-Variance trade-off
- Confusion matrix, Precision, Recall, F1-Score
- Practice: Scikit-learn

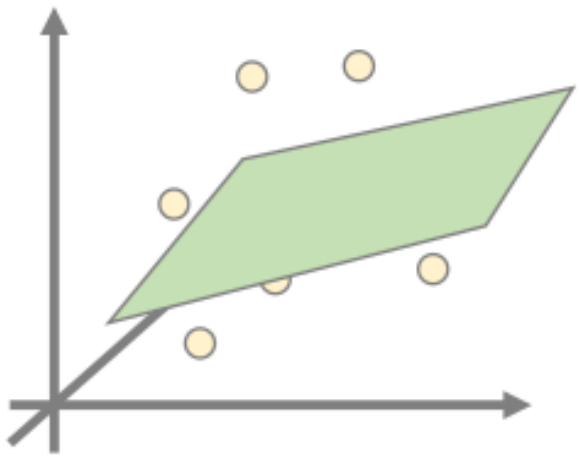
Linear regression

Simple Linear Regression



- single explanatory variable x
- we want to approximate the price of houses if we know the size

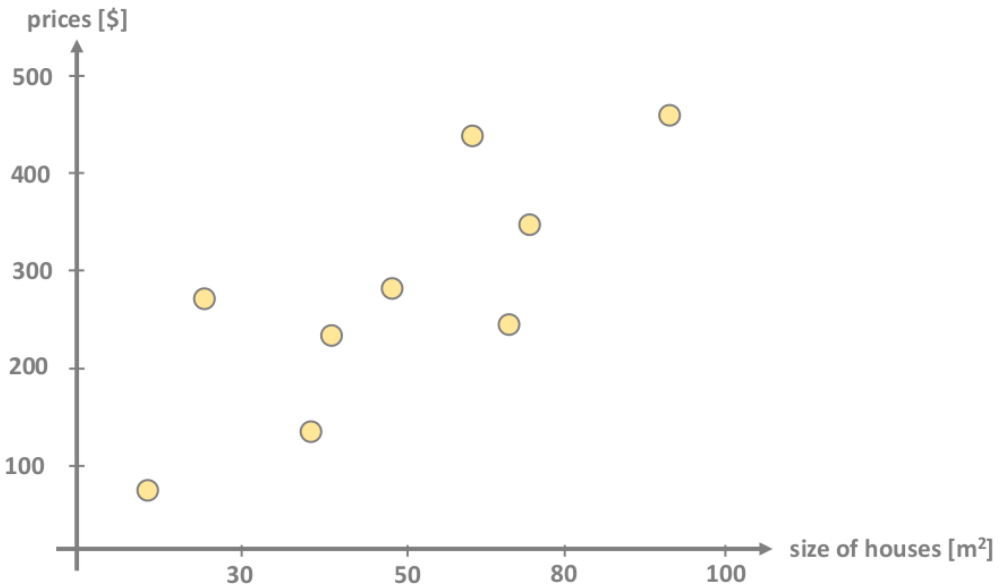
Multiple Linear Regression



- several explanatory variables $\vec{x}$
- we want to approximate the price of houses if we know the size and the number of rooms

Simple Linear regression

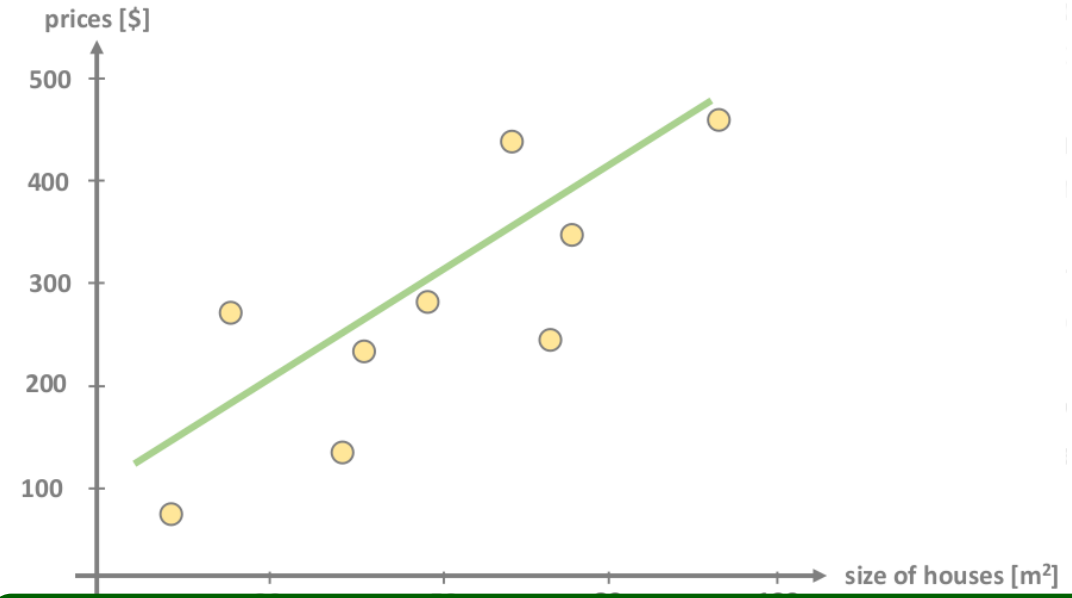
What is the aim?



- want to find some **linear relationship** between the feature and the target variable
- x: feature (size)
- t: target (price)
- $H(x) = b_0 + b_1 x$: hypothesis
- linear relationship: this is our model now

Simple Linear regression

Our model (or hypothesis) is linear, so the result is a **linear line**.



- the model defines the relationship between the variables
- variables: size (feature) and price (target)

What does it mean?

If we have a new x feature (size of the house) we can get the $H(x)$ price of the house accordingly.

General steps to building a machine learning model

we have a **dataset** - $(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, N$

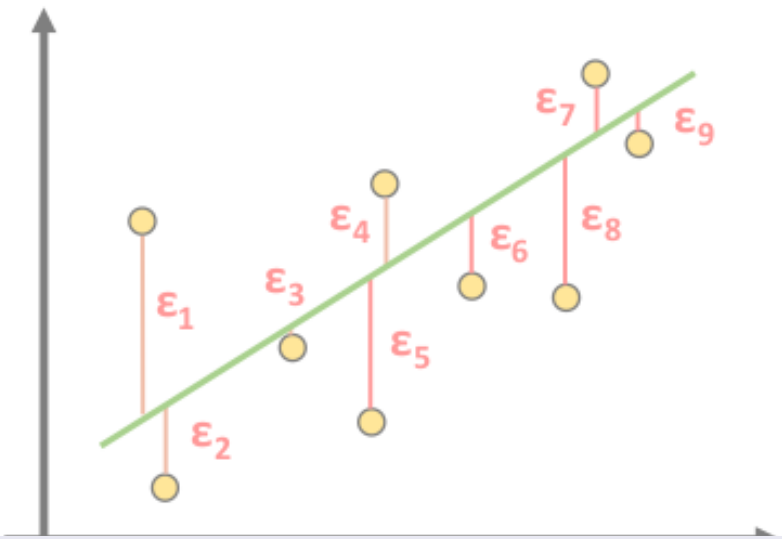
we **train the model** - free parameters in H - optimization

$$H(x) = b_0 + b_1 x$$

after the training we can **make predictions** with the model H

Simple Linear regression

How good is the model?



- Mean Squared Error (MSE)
- **cost function - loss function**

→ if this is small: the predictions are close to the target values

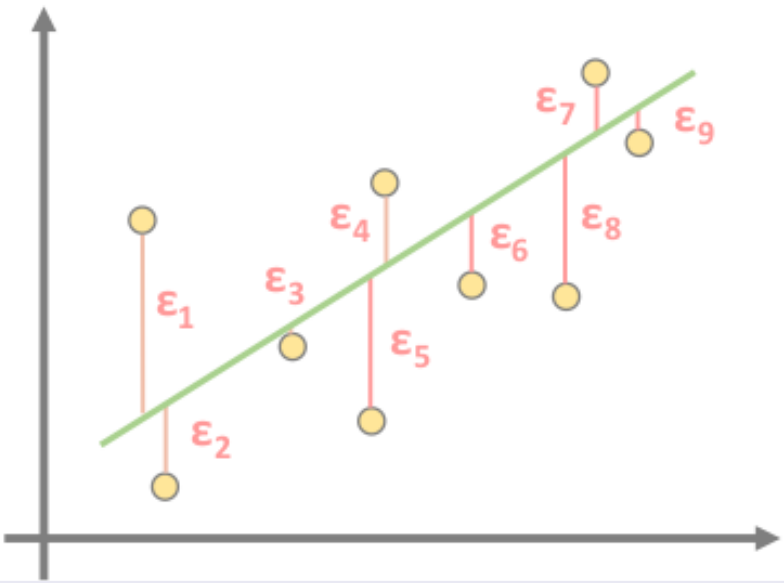
→ if this is large: the predictions differ from the target values

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2 \quad \leftarrow \textbf{cost function}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (H(x^{(i)}) - y^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N} (b_0 + b_1 x^{(i)} - y^{(i)})^2$$

Simple Linear regression

How good is the model?



- Mean Squared Error (MSE)
- **cost function - loss function**

→ if this is small: the predictions are close to the target values

→ if this is large: the predictions differ from the target values

$MSE = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2$ ← **cost function**

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (H(x^{(i)}) - y^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N} (b_0 + b_1 x^{(i)} - y^{(i)})^2$$

Simple Linear regression

Optimalization

How to train our model? How to find the optimal b_0 and b_1 parameters?

Normal equation

- we can transform the problem into **linear equations** and use the following formula
- $\vec{b} = [b_0, b_1]^T = (X^T X)^{-1} X^T \vec{y}$
- Okay, but what is X and y? X is a matrix, its i-th row is $[1, x^{(i)}]$.

Gradient descent

- It is a first-order **iterative optimizaton algorithm** for finding the minimum of a function.

Simple Linear regression

Cost function

$$C(b_0, b_1) = \frac{1}{N} \sum_{i=1}^{N} (b_0 + b_1 x^{(i)} - y^{(i)})^2$$

Gradient descent in general

Do while ...

- $\vec{b}_{new} = \vec{b}_{old} - \alpha \nabla C$

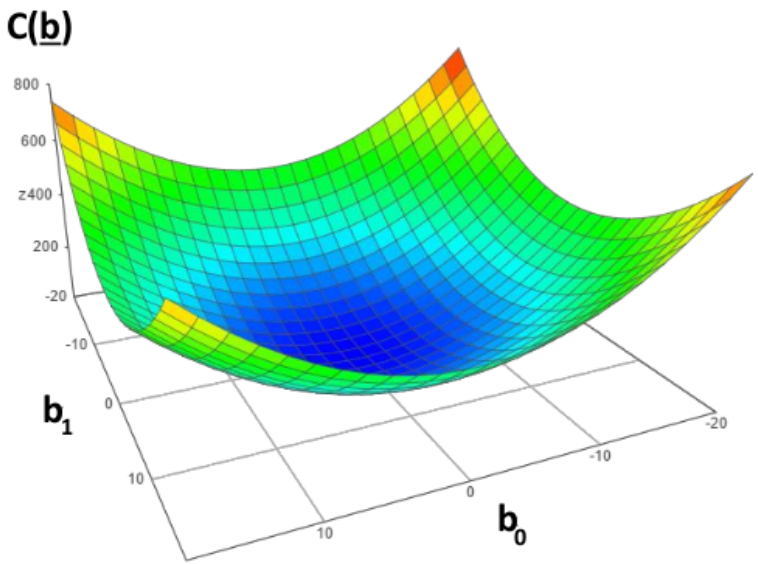
Gradient (∇C) in our case

$$\nabla C = [\frac{\partial C}{\partial b_0}, \frac{\partial C}{\partial b_1}] = ?$$

- $\frac{\partial C}{\partial b_0} = \frac{2}{N} \sum_{i=1}^{N} (b_0 + b_1 x^{(i)} - y^{(i)})$
- $\frac{\partial C}{\partial b_1} = \frac{2}{N} \sum_{i=1}^{N} (b_0 + b_1 x^{(i)} - y^{(i)}) \cdot x^{(i)}$

Simple Linear regression

What is α? What about ∇C?



- ∇C, the gradient of the function C is pointing in the direction of a local maximum $\Rightarrow$ we are going to the opposite direction $-\nabla C$, the direction of steepest descent
- α - **Learning rate**

$\rightarrow$ if small, then the algorithm takes small steps towards the minimum

$\rightarrow$ if large, then the algorithm takes large steps towards the minimum

General steps to building a machine learning model

we have a **dataset** - $(\vec{x}^{(i)}, y^{(i)}), \quad i = 1, \ldots, N$ and $\vec{x}^{(i)} \in \mathbb{R}^K$

number of features: K

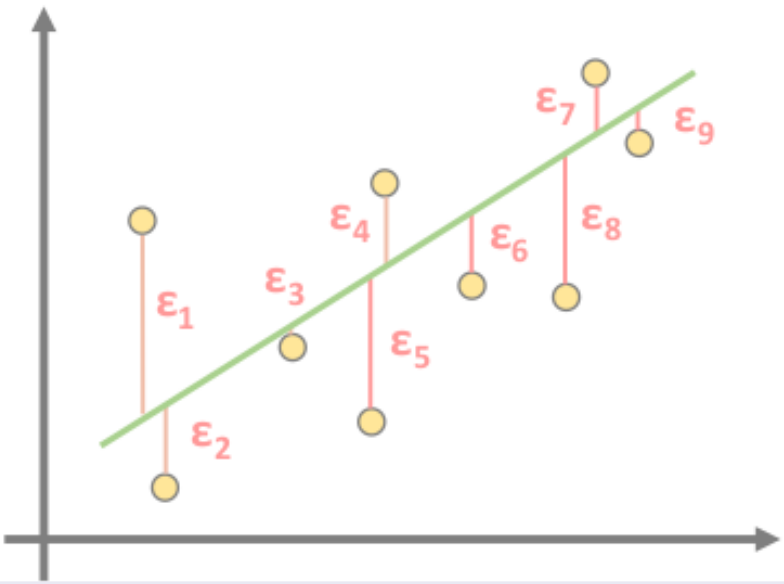
we **train the model** - free parameters in H - optimization

$$H(\vec{x}) = b_0 + b_1 x_1 + \cdots + b_K x_K$$

after the training we can **make predictions** with the model H

Multiple Linear regression

How good is the model?



- Mean Squared Error (MSE)
- **cost function - loss function**

→ if this is small: the predictions are close to the target values

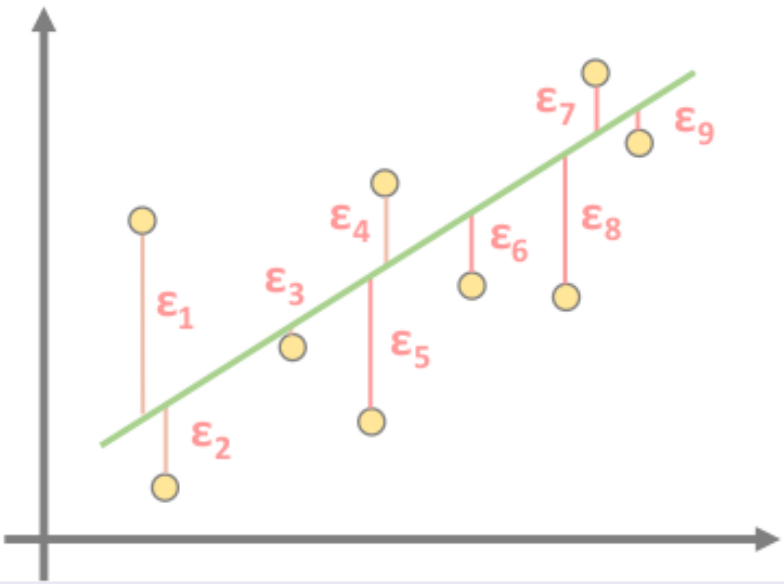
→ if this is large: the predictions differ from the target values

$MSE = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2 =$ ← **cost function**

$$\frac{1}{N}\sum_{i=1}^{N}(H(\vec{x}^{(i)}) - y^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N}(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)} - y^{(i)})^2$$

Multiple Linear regression

How good is the model?



- Mean Squared Error (MSE)
- **cost function - loss function**

→ if this is small: the predictions are close to the target values

→ if this is large: the predictions differ from the target values

$MSE = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2$ ← **cost function**

$= \frac{1}{N}\sum_{i=1}^{N} (H(\vec{x}^{(i)}) - y^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N} (b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)} - y^{(i)})^2$

Multiple Linear regression

Optimalization

How to find the optimal $b_0, b_1, \ldots, b_K$ parameters?

Normal equation

- transform the problem into **linear equations** and
- $\vec{b} = [b_0, b_1, \ldots, b_K]^T = (X^T X)^{-1} X^T \vec{y}$
- $X \in \mathbb{R}^{N \times K+1}$, its i-th row is $[1, \vec{x}^{(i)}] = [1, x_1^{(i)}, \ldots, x_K^{(i)}]$.

Or we can use **gradient descent** again. How?

Multiple Linear regression

Cost function

$$C(b_0, b_1, \ldots, b_K) = \frac{1}{N}\sum_{i=1}^{N}(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)} - y^{(i)})^2$$

Gradient descent in general

Do while ...

- $\vec{b}_{new} = \vec{b}_{old} - \alpha \nabla C$

Gradient (∇C) in our case

$$\nabla C = [\frac{\partial C}{\partial b_0}, \frac{\partial C}{\partial b_1}, \ldots, \frac{\partial C}{\partial b_K}] = ?$$

- $\frac{\partial C}{\partial b_0} = \frac{2}{N}\sum_{i=1}^{N}(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)} - y^{(i)})$
- $\frac{\partial C}{\partial b_j} = \frac{2}{N}\sum_{i=1}^{N}(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)} - y^{(i)})x_j^{(i)}$, if $j > 0$

Discussion about the Optimization

Normal equation

- for low-dimensional problems
- low-dimensional = few features
- expensive: $O(N^3)$

Gradient descent

- works fine in higher dimensions
- iterative approach

Polynomial Linear Regression

we have a **dataset** - $(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, N$

we **train the model** - free parameters in H - optimization

$$H(x) = b_0 + b_1 x + \cdots + b_K x^K$$

after the training we can **make predictions** with the model H

So we get a **multiple linear regression** task.

Linear Regression - Quality of the fitting

R^2 statistic: $R^2 = 1 - \frac{RSS}{TSS}$

- RSS: residual sum of squares - $\sum_{i=1}^{N}(H(\vec{x}^{(i)}) - y^{(i)})^2$
  it measures the variability left unexplained after performing the regression (cost function)
- TSS: total sum of squares - $\sum_{i=1}^{N}(\overline{\vec{y}} - y^{(i)})^2$
  it measures the total variance in $\vec{y}$ (variance in target data)

How to use it?

- $R^2 = 1$: the fitted model explains all variability (perfect fit)
- $R^2 = 0$: no "linear relationship" between the features and the target
- $R^2 = 0.7$: "Seventy percent of the variance in the response variable can be explained by the explanatory variables."

Logistic Regression - Classification

Linear Regression solves regression problem: the prediction is a value

For example: we have the features (size, number of rooms) and this model predicts the target (price)

Logistic regression

it solves classification problems

at first, binary classification

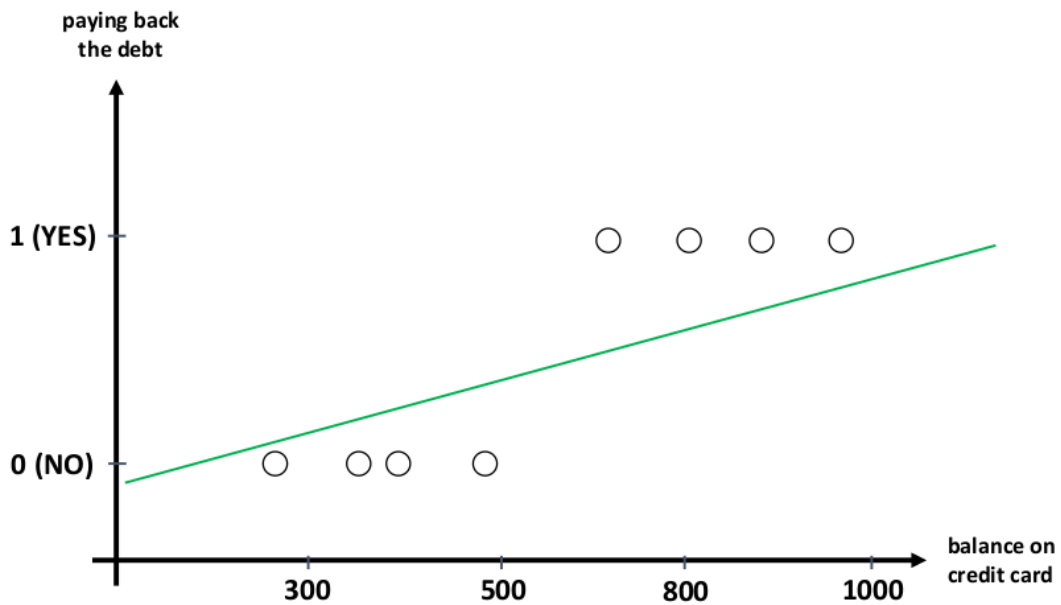
- for example: spam detection for emails
- the output variable is discrete
- it assigns probabilities to given outcomes

  so the output is a probability that the given input belongs to a certain class

Logistic Regression - Motivation

What is the problem with linear regression for a classification task?

Applying LR is not a good idea.

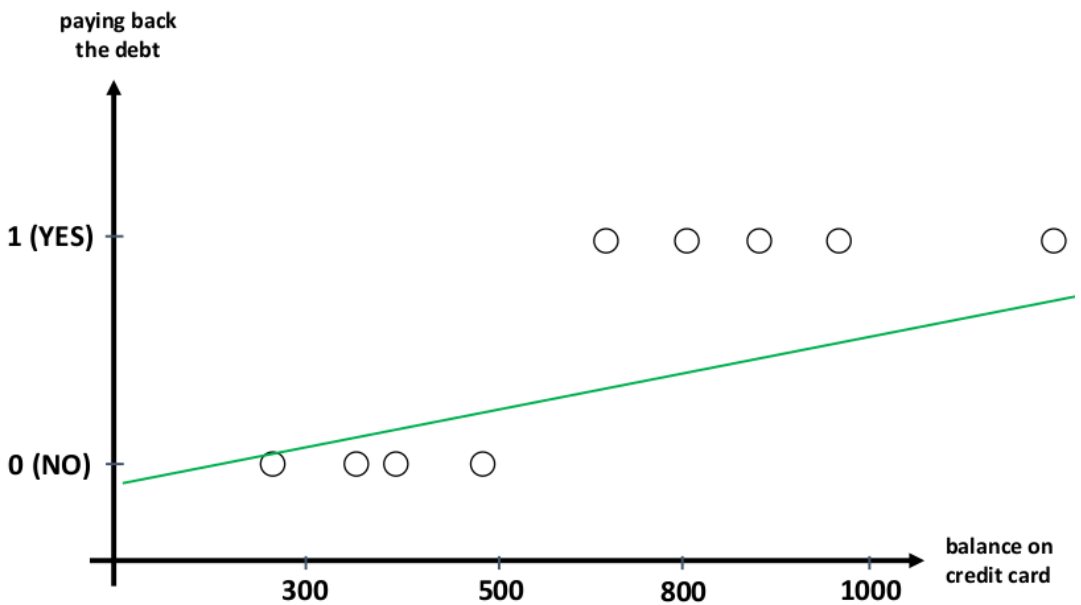


- predicts even outside the $[0, 1]$ range $\rightarrow$ we want to deal with probabilities
- sensitive to outliers

Logistic Regression - Motivation

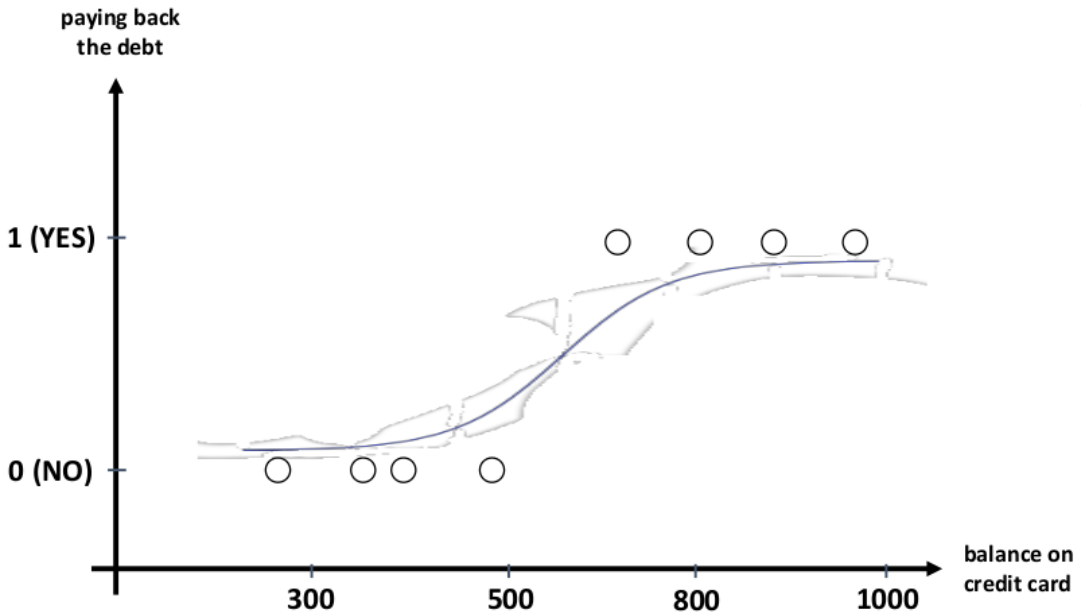
What is the problem with linear regression for a classification task?

Applying LR is not a good idea.



- predicts even outside the $[0, 1]$ range $\rightarrow$ we want to deal with probabilities
- sensitive to outliers

Logistic Regression - Motivation



$\rightarrow$ ideal prediction should look like

Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = P(y = 1|x = balance)$$

Logistic Regression - Loss function

We also need a loss function

- $L(y, t)$ = How much the predicted value y differs from the true value (or target) t? MSE is bad for this, see the previous figures
- We do this via **conditional maximum likelihood** estimation: we choose the model parameters that maximize the probability of the true labels t in the training data given the observations x.
  Optimalization procedure: we'll look for the free parameters in the model, to maximize the probability of the correct label $p(t|x)$.
- We need a **likelihood function**.

Logistic Regression - Loss function

Bernoulli

- since there are only two possible outcomes (1 or 0)
- $p(t|x) = y^t(1-y)^{1-t}$
- $\Rightarrow$ we want to maximize this probability
- $\Rightarrow$ take its log: $\log p(t|x) = t \log y + (1-t)\log(1-y)$
- $\Rightarrow$ maximizing this $\Leftrightarrow$ minimizing
  $L(t,y) = -\left(t \log y + (1-t)\log(1-y)\right)$

General steps to building a machine learning model

we have a **dataset** - $(\vec{x}^{(i)}, t^{(i)}), \quad i = 1, \ldots, N \quad$ and $\vec{x}^{(i)} \in \mathbb{R}^K$

number of features: K

we **train the model** - free parameters in H - optimization

$$H(\vec{x}) = \sigma\left(b_0 + b_1 x_1 + \cdots + b_K x_K\right)$$

minimizing the **Loss function**, which is

$$\frac{1}{N} \sum_{i=1}^{N} L\left(t^{(i)}, H(\vec{x}^{(i)})\right)$$

after the training we can **make predictions** with the model H

Gradient descent

Cost function

$$C(b_0, b_1, \ldots, b_K) = \frac{1}{N} \sum_{i=1}^{N} L\left[t^{(i)}, \sigma\left(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)}\right)\right]$$

Gradient descent in general

Do while ...

- $\vec{b}_{new} = \vec{b}_{old} - \alpha \nabla C$

Gradient (∇C) in our case

$\nabla C = [\frac{\partial C}{\partial b_0}, \frac{\partial C}{\partial b_1}, \ldots, \frac{\partial C}{\partial b_K}] =?$

- $\frac{\partial C}{\partial b_0} = \frac{1}{N} \sum_{i=1}^{N} \left[\sigma(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)}) - t^{(i)}\right]$
- $\frac{\partial C}{\partial b_j} = \frac{1}{N} \sum_{i=1}^{N} \left[\sigma(b_0 + b_1 x_1^{(i)} + \cdots + b_K x_K^{(i)}) - t^{(i)}\right] x_j^{(i)}$, if $j > 0$

Naive Bayes Classifier (NBC)

Advantages

- very efficient supervised learning algorithm
- is scales well even in high dimensions
- it is able to compete with **SVM** or **random forest** classifiers
- it is able to make good predictions even when the training set is small

Why is it naive?

The naive assumption is that every pair of features are independent.

- a fruit can be consideres to be an apple if it is red, rounded and about $8cm$ in diameter
- **NBC** considers each of these features contribute independently to the probability that this fruit is an apple

Naive Bayes Classifier - Math

Abstract

- features: $x_1, \ldots, x_n$ targets: $C_1, \ldots, C_K$
- calculate $P(C_k|x_1, \ldots, x_n)$ for each classes
- choose that k when $P(C_k|x_1, \ldots, x_n)$ is maximal $\Rightarrow$ that will be the predicted class

How exactly?

Use **Bayes theorem** for every C_k classes

- $P(C_k|x_1, \ldots, x_n) = \frac{P(C_k)P(x_1, \ldots, x_n|C_k)}{P(x_1, \ldots, x_n)}$
- we don't have to care about the divisior (it is the same for all classes)
- use the **independency** assumption $\Rightarrow P(x_1, \ldots, x_n|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$

Naive Bayes Classifier - Why?

Advantages

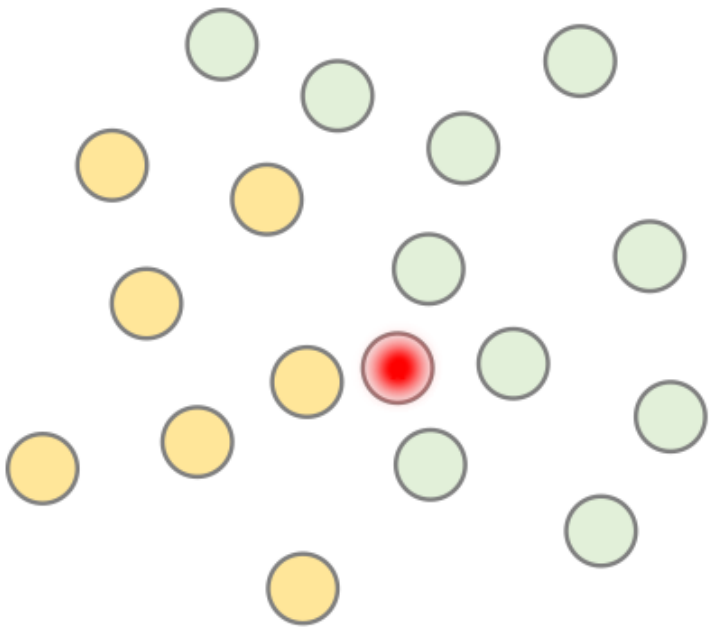
- relatively simple
- it can be trained on small datasets well
- fast
- it is not sensitive to irrelevant features
- powerful tool for **text classification**

Disadvantages

It assumes that every feature is independent: of course this is not usually true.

Naive Bayes Classifier - Example

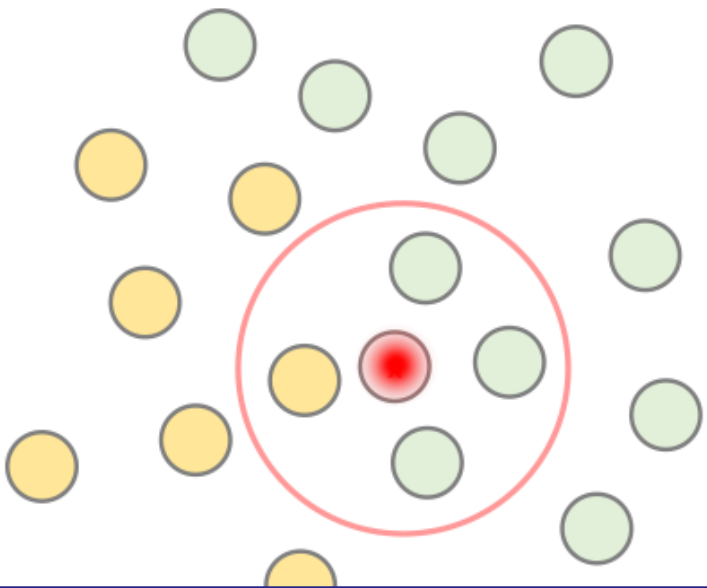
We would like to decide: the red point which class belongs to.



- $P(yellow) = \frac{7}{17}$
- $P(green) = \frac{10}{17}$

Naive Bayes Classifier - Example

We would like to decide: the red point which class belongs to.



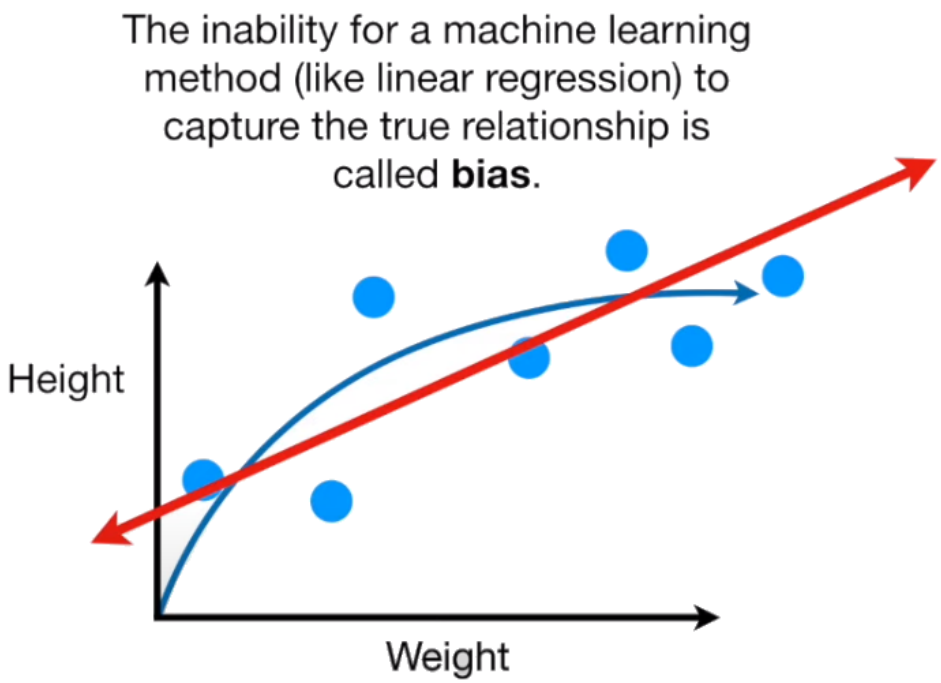
- $P(yellow) = \frac{7}{17}$
- $P(green) = \frac{10}{17}$
- $P(red\ is\ there|green) = \frac{3}{10}$
- $P(red\ is\ there|yellow) = \frac{1}{7}$

Posterior probabilities

- $P(green) = P(green)P(red\ is\ there|green) = \frac{10}{17}\frac{3}{10} = \frac{30}{170}$
- $P(yellow) = P(yellow)P(red\ is\ there|yellow) = \frac{7}{17}\frac{1}{7} = \frac{7}{119}$

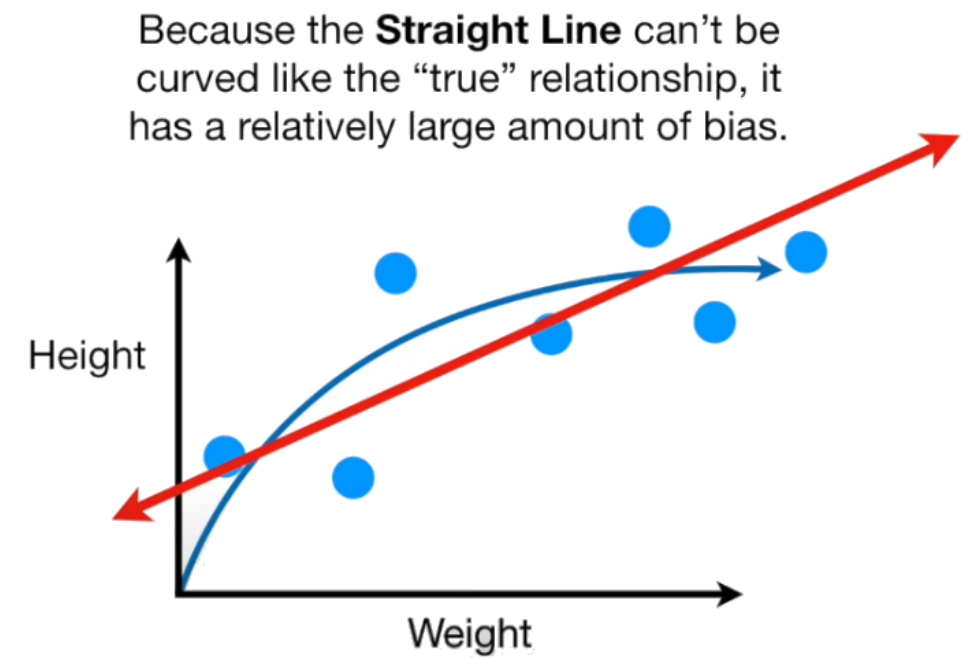
Bias and variance

We would like to decide that our model how good is. We divide our dataset to **Training** and **Testing** sets.



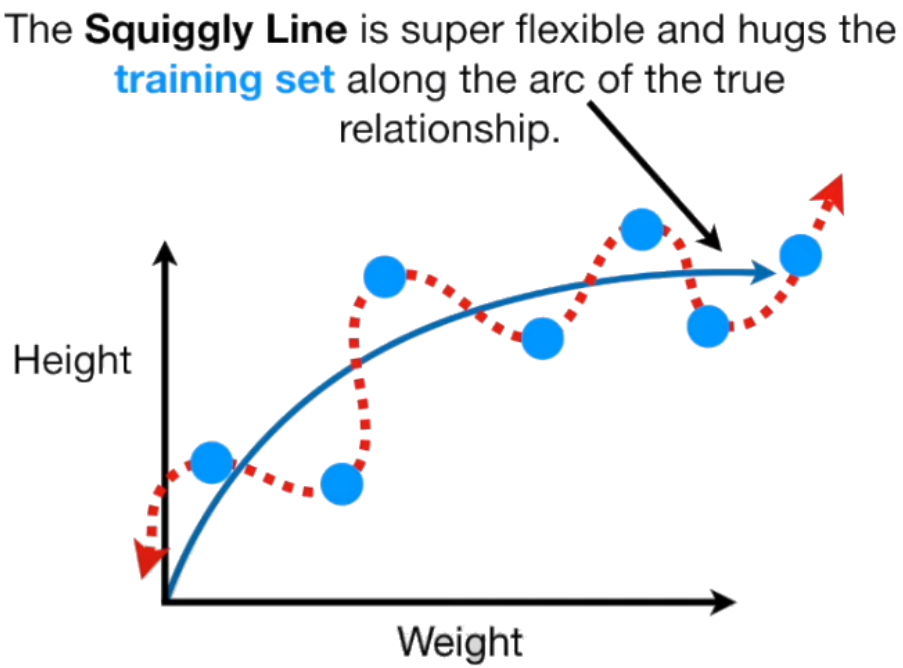
Bias and variance

We would like to decide that our model how good is. **Linear Regression**



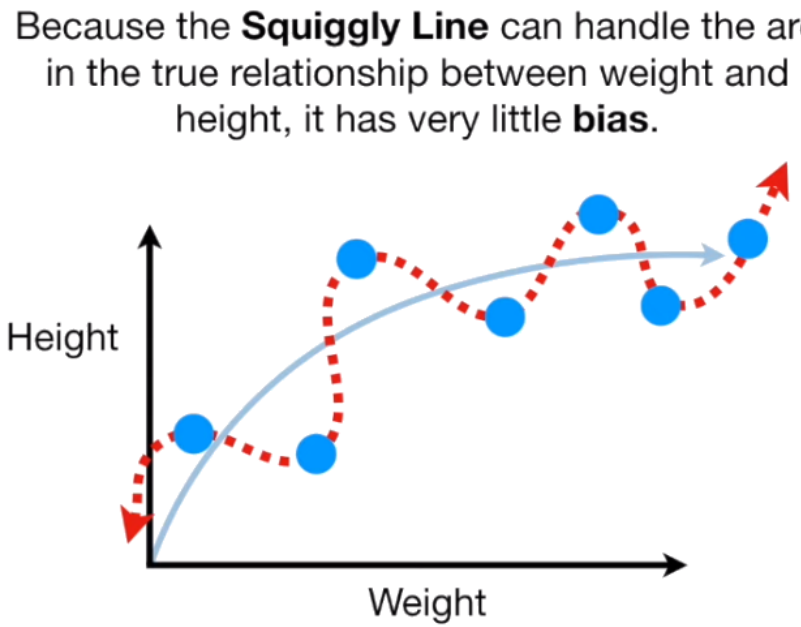
Bias and variance

We would like to decide that our model how good is. **Polynomial Regression**



Bias and variance

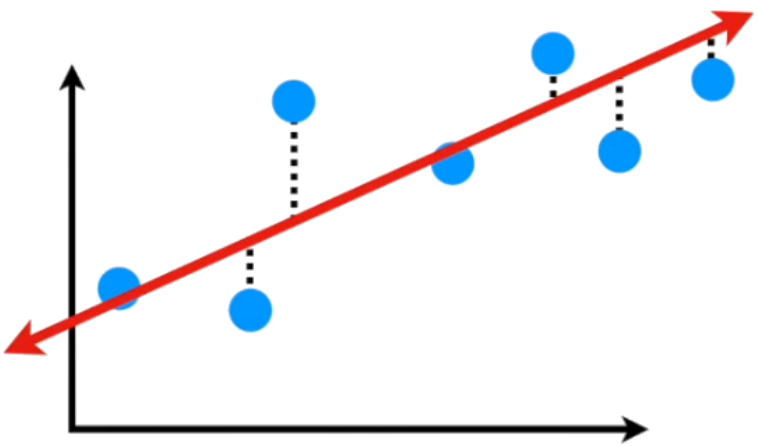
What is bias?

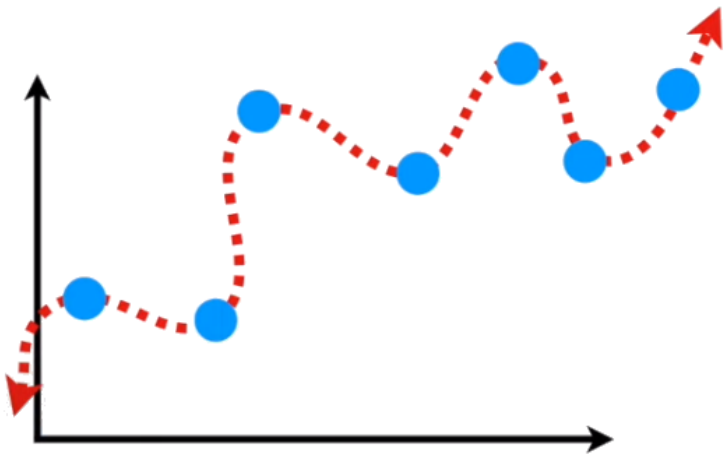


Bias and variance

We would like to decide that our model how good is.

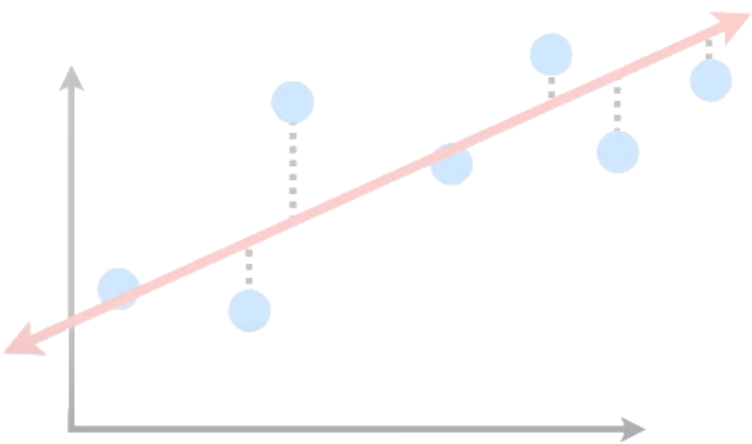
We can compare how well the **Straight Line** and the **Squiggly Line** fit the **training set** by calculating their sums of squares.



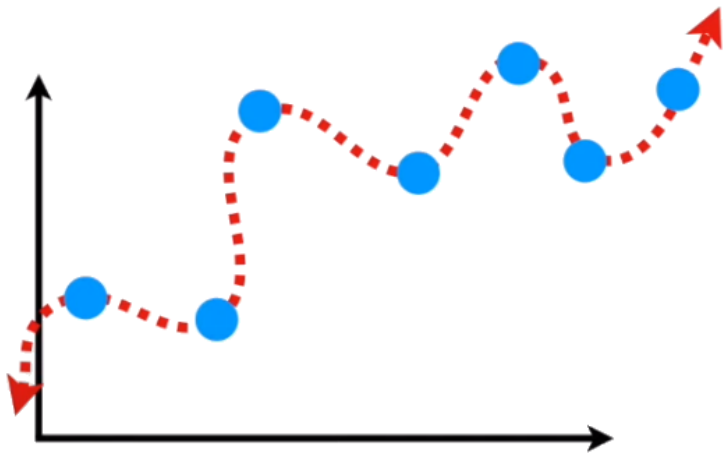


Bias and variance

We would like to decide that our model how good is.

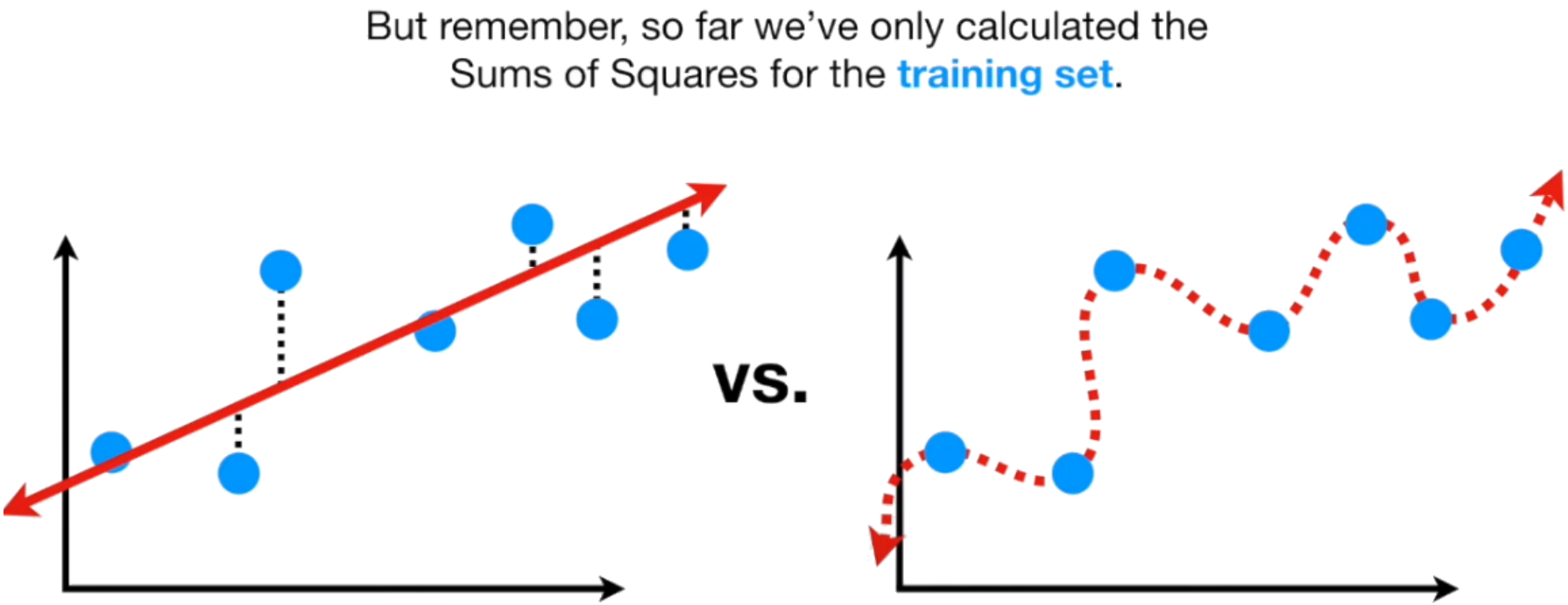


Notice how the **Squiggly Line** fits the data so well that the distances between the line and the data are all 0.



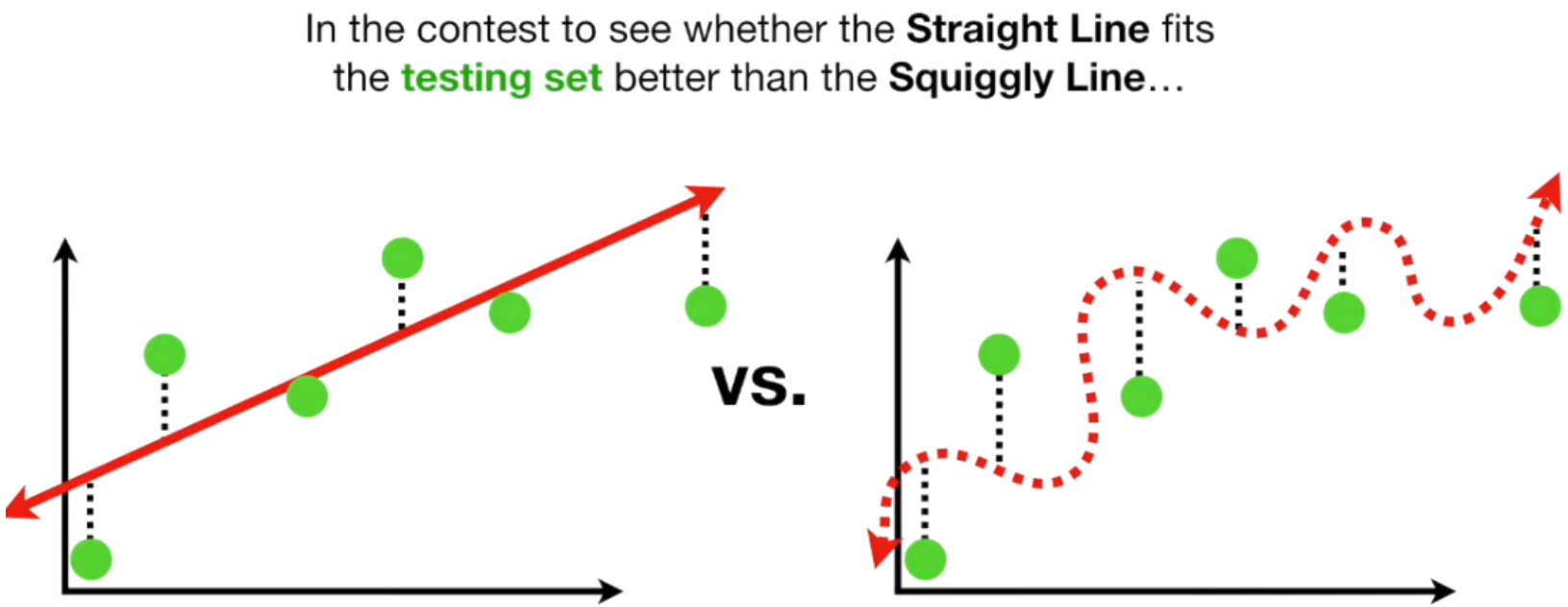
Bias and variance

We would like to decide that our model how good is.



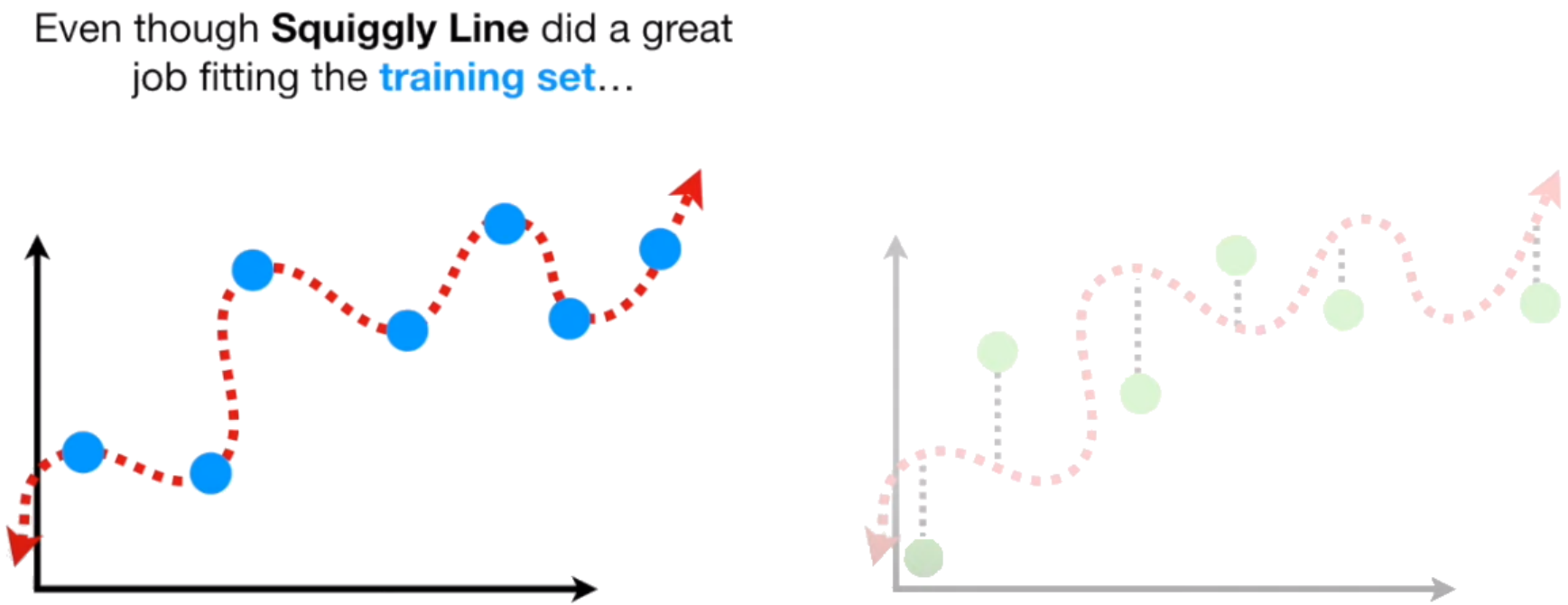
Bias and variance

We would like to decide that our model how good is.



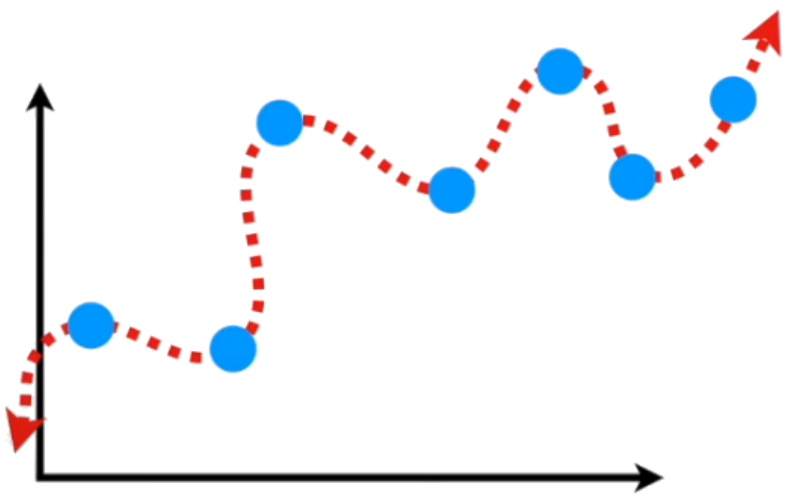
Bias and variance

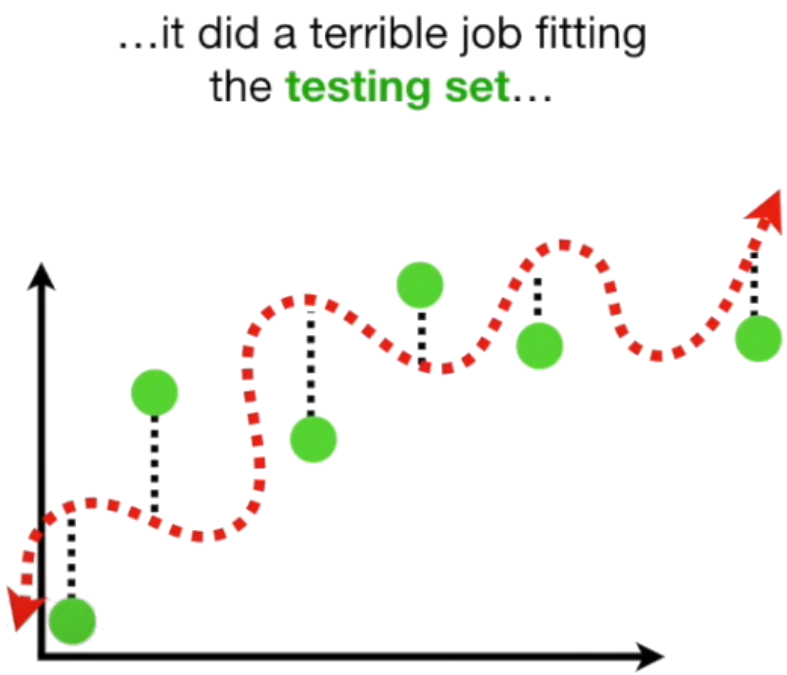
We would like to decide that our model how good is.



Bias and variance

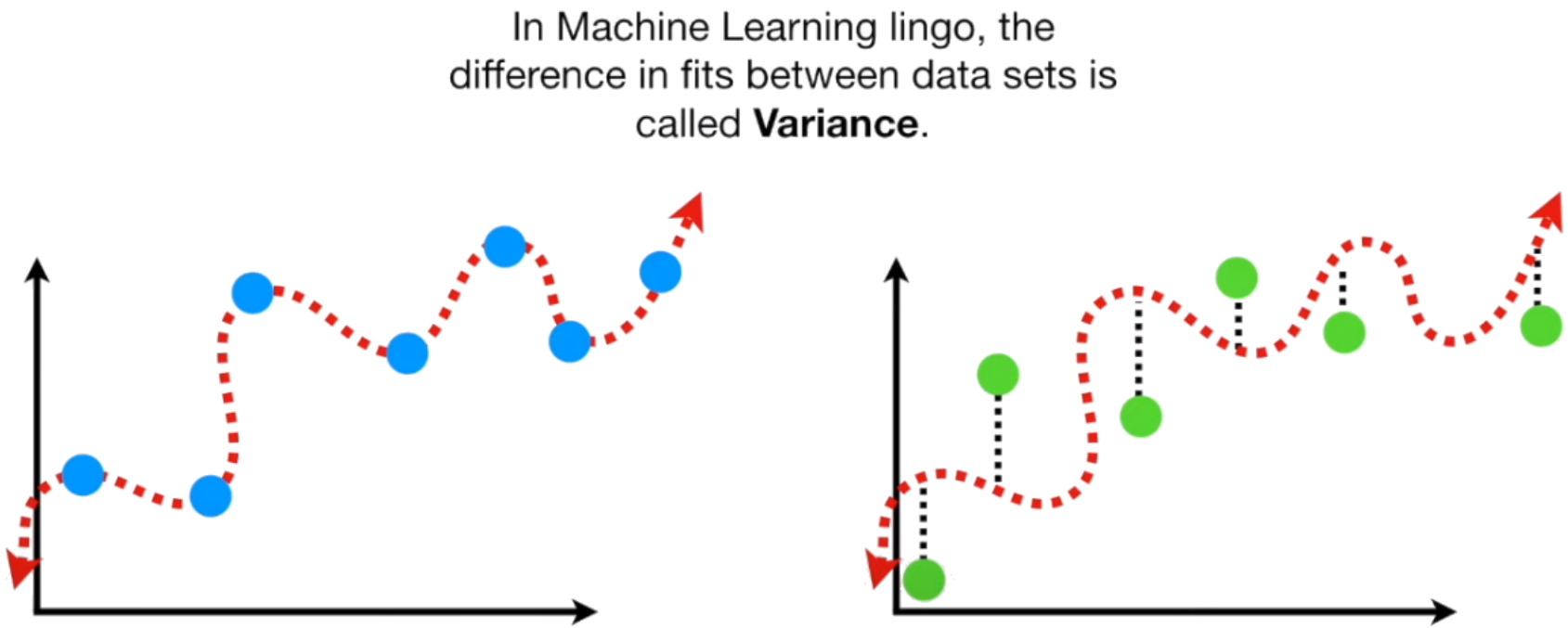
We would like to decide that our model how good is.





Bias and variance

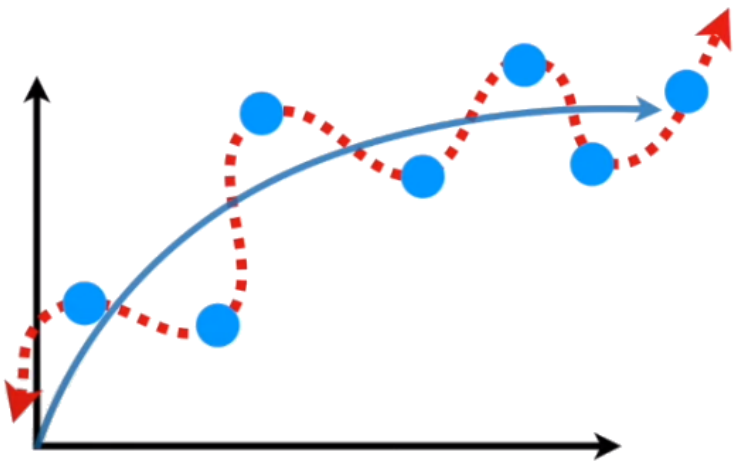
We would like to decide that our model how good is.

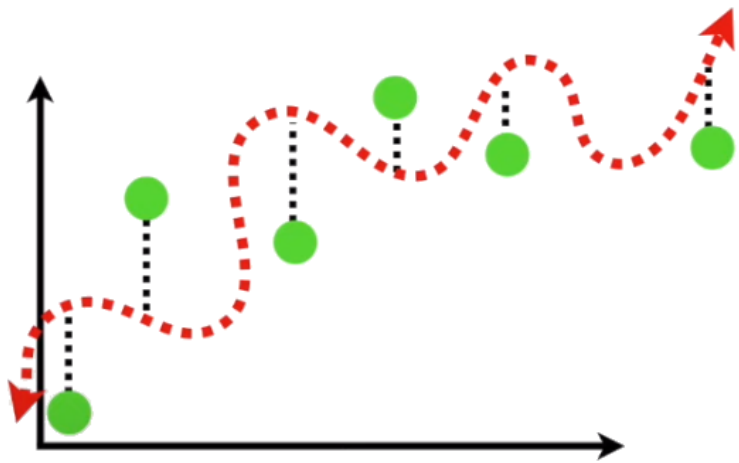


Bias and variance

We would like to decide that our model how good is.

In other words, it's hard to predict how well the **Squiggly Line** will perform with future data sets. It might do well sometimes, and other times it might do terribly.

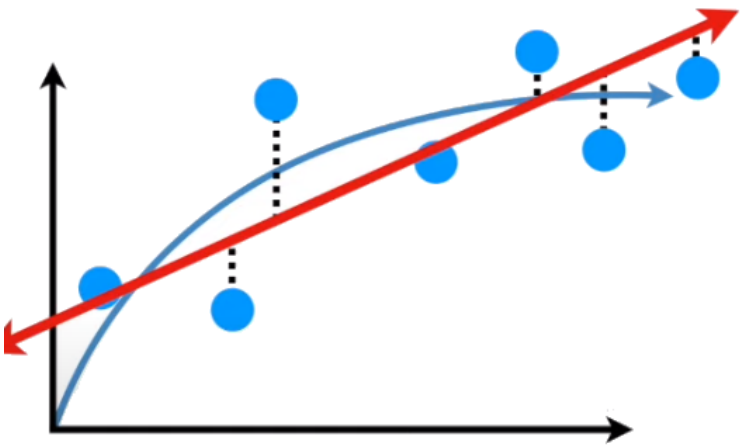


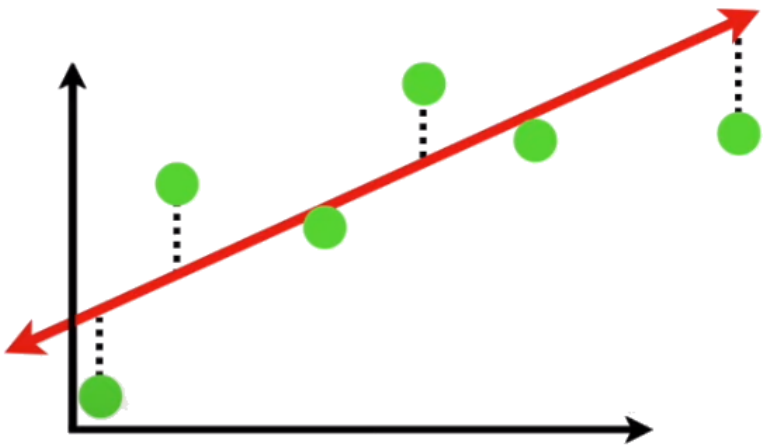


Bias and variance

We would like to decide that our model how good is.

In other words, the **Straight Line** might only give good predictions, and not great predictions. But they will be consistently good predictions.



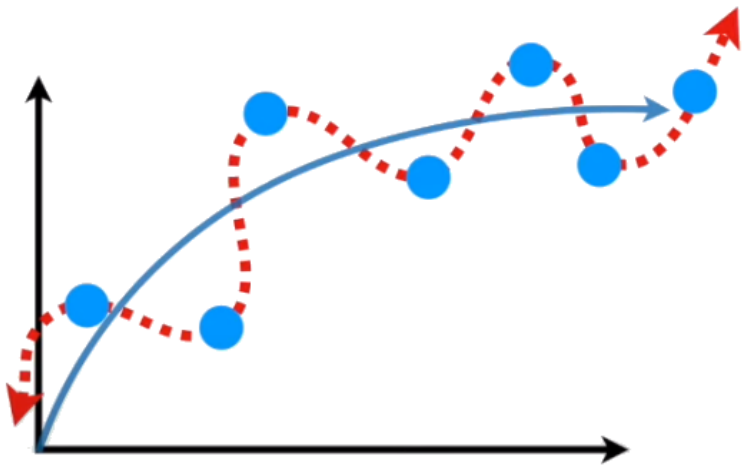


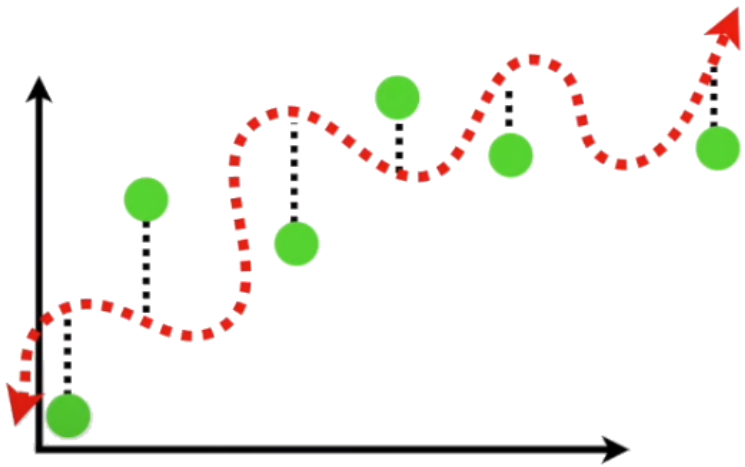
Bias and variance

We would like to decide that our model how good is.

Terminology Alert!!!

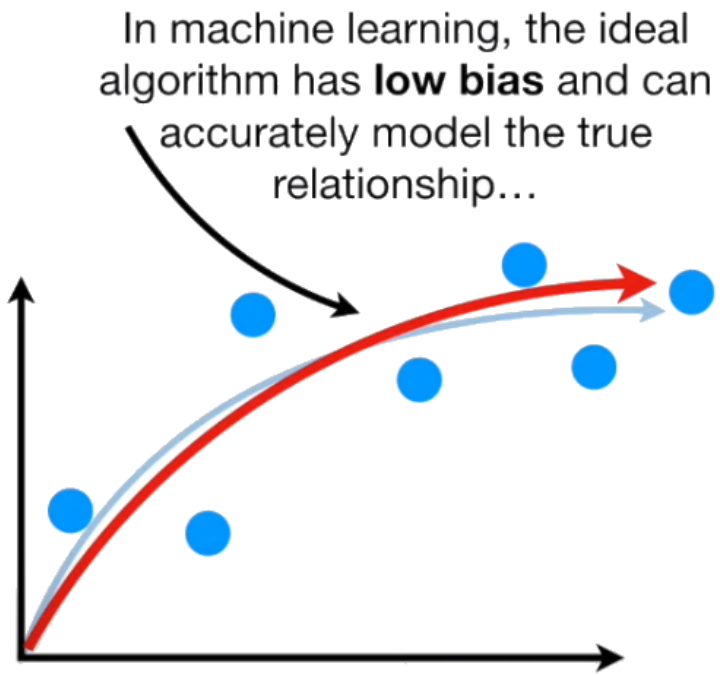
Because the **Squiggly Line** fits the **training set** really well, but not the **testing set**, we say that the **Squiggly Line** is **overfit**.





Bias and variance

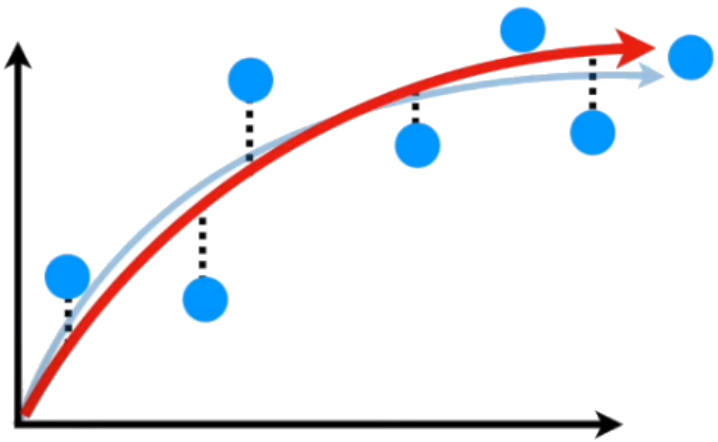
We would like to decide that our model how good is.

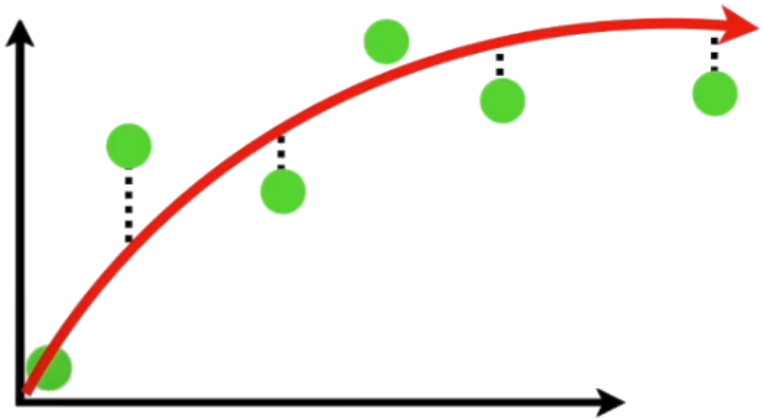


Bias and variance

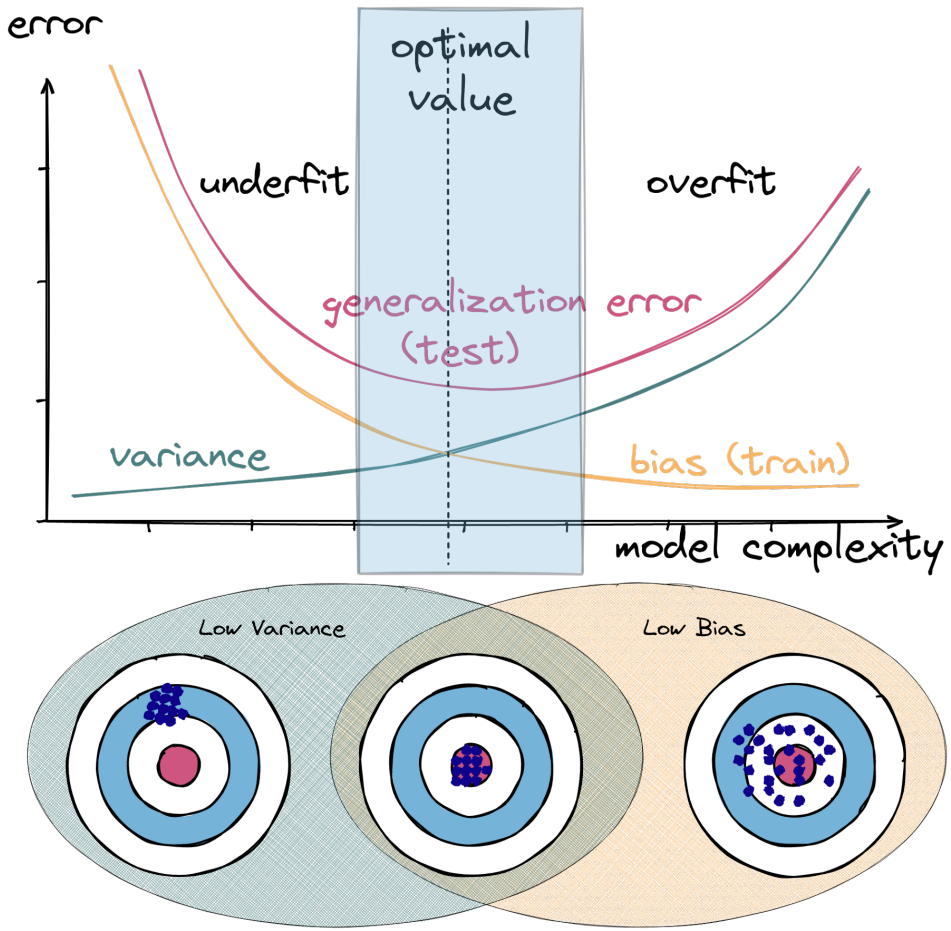
We would like to decide that our model how good is.

...and it has **low variability**, by producing consistent predictions across different datasets.





Bias and variance: Summary



Confusion matrix

What is a confusion matrix?

It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.

		ACTUAL: Negative	ACTUAL: Positive
PREDICTION	Negative	TRUE NEGATIVE	FALSE NEGATIVE
	Positive	FALSE POSITIVE	TRUE POSITIVE

Confusion matrix: Example

A machine learning model is trained to predict tumor in patients. The test dataset consists of 100 people. We get the following results.

		ACTUAL	
		Negative	Positive
PREDICTION	Negative	60	8
	Positive	22	10

Example

- **True Positive (TP)** – model correctly predicts the positive class (prediction and actual both are positive). In the above example, 10 people who have tumors are predicted positively by the model.
- **True Negative (TN)** – model correctly predicts the negative class (prediction and actual both are negative). In the above example, 60 people who don't have tumors are predicted negatively by the model.
- **False Positive (FP)** – model gives the wrong prediction of the negative class (predicted-positive, actual-negative). In the above example, 22 people are predicted as positive of having a tumor, although they don't have a tumor. FP is also called a TYPE I error.
- **False Negative (FN)** – model wrongly predicts the positive class (predicted-negative, actual-positive). In the above example, 8 people who have tumors are predicted as negative. FN is also called a TYPE II error.

Rates

- **TPR** – $\frac{TP}{\text{Actual Positive}} = \frac{TP}{TP+FN}$ (Recall)
- **FNR** – $\frac{FN}{\text{Actual Positive}} = \frac{FN}{TP+FN}$
- **TNR** – $\frac{TN}{\text{Actual Negative}} = \frac{TN}{TN+FP}$
- **FPR** – $\frac{FP}{\text{Actual Negative}} = \frac{FP}{TN+FP}$

Even if data is **imbalanced**, we can figure out that our model is working well or not. For that, the values of TPR and TNR should be high, and FPR and FNR should be as low as possible.

With the help of TP, TN, FN, and FP, other performance metrics can be calculated.

Precision and Recall

Both precision and recall are crucial for information retrieval, where positive class mattered the most as compared to negative. Why?
While searching something on the web, the model does not care about something irrelevant and not retrieved (this is the true negative case).
Therefore only TP, FP, FN are used in Precision and Recall.

Precision

Out of all the positive predictions, what percentage is truly positive.

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall

Out of the total truly positive, what percentage are predicted truly positive. It is the same as TPR (true positive rate). $\text{Recall} = \frac{TP}{TP+FN}$

How are precision and recall useful? Let's see through examples.

Example: Credit card fraud detection

We do not want to miss any fraud transactions. Therefore, we want False-Negative to be as low as possible. **In these situations, we can compromise with the low precision, but recall should be high.**

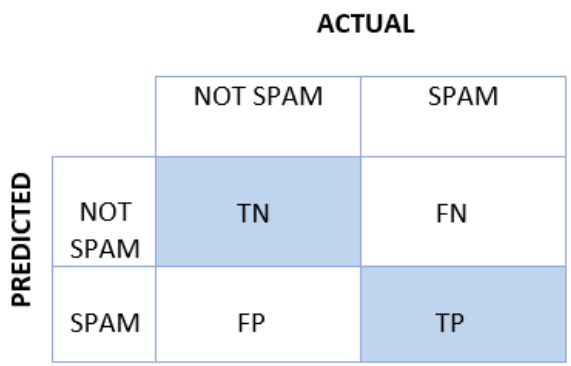
We do not want to falsely accuse anyone. Previous one is better example (you don't want to miss any actual positive → minimizing FN.

		ACTUAL: FAIR TRANSACTION	ACTUAL: FRAUD TRANSACTION
PREDICTED	FAIR TRANSACTION	TN	FN
	FRAUD TRANSACTION	FP	TP

But, when is the precision more important than recall?

Example: Spam detection

In the detection of spam mail, it is okay if any spam mail remains undetected (false negative), but what if we miss any critical mail because it is classified as spam (false positive). **In this situation, False Positive should be as low as possible. Here, precision is more vital as compared to recall.**



F1-Score

It is the harmonic mean of precision and recall. It takes both false positive and false negatives into account. Therefore, it performs well on an imbalanced dataset.

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$