ML PRIMER FOR SOFTWARE SPECIALISTS

30.11.2023

VeeMG x adnexo

Martin Stypinski

- Partner @ VeeMG GmbH
  - Software & AI Consulting
- Dean of Studies :
  - CAS Machine Learning 4 Software Engineers
- Hobbies: Stupid Ideas & Cycling

Christian Fässler

- Co-Founder @ adnexo GmbH
  - Software & IoT Consulting
  - Sensor / Timeseries Data
  - Pythonista
- Teacher :
  - CAS Machine Learning 4 Software Engineers
- Hobbies: Sports & Music

Let’s connect

Martin Stypinski

https://veemg.com

Christian Fässler

www.adnexo.ch

AI Projects / Consulting

- Identifying Potential AI Projects
- Scoping Effective PoCs
- Evaluating AI Business Cases

- Training:
- https://aikurs.ch

Time Table - Morning

- 09:00 – 11:30: Introduction into ML
  - What is AI, ML, DL
  - Understanding Features, Bias Variance Trade-off, IID assumption

- 11:00 – 12:30: Classical Methods with SciKit Learn
  - Trees & Clustering
  - Lab Session

Time Table - Afternoon

- 13:30 – 15:00: Deep Learning
  - Introduction into Deep Learning
  - Deep Learning Lab

- 15:30 – 17:00: MLOps & Deployment
  - 15:30 – 16:15: MLOps in general & Deployment
  - 16:15 – 17:00 MLOps Lab

- Goal for today: **You train and deploy your first model!**

Goal for Today

1. Understand Supervised & Un-Supervised Learning
2. What is needed for a successful ML Project
3. Get our hands dirty! Doing!
4. Fun!

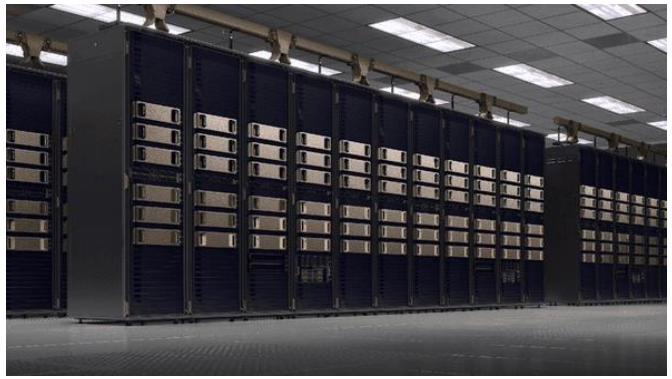
Outline

- «Big Data» as a Phenomenon

- AI, ML, DL – 3 terms, but what is the ”deeper” meaning.
- Some examples of modern Machine Learning Projects
  - Industry Drivers

Data Mining

- Automatic extraction of valuable knowledge from large datasets.
- Support human decision-making.
  - Hmm, GPT?



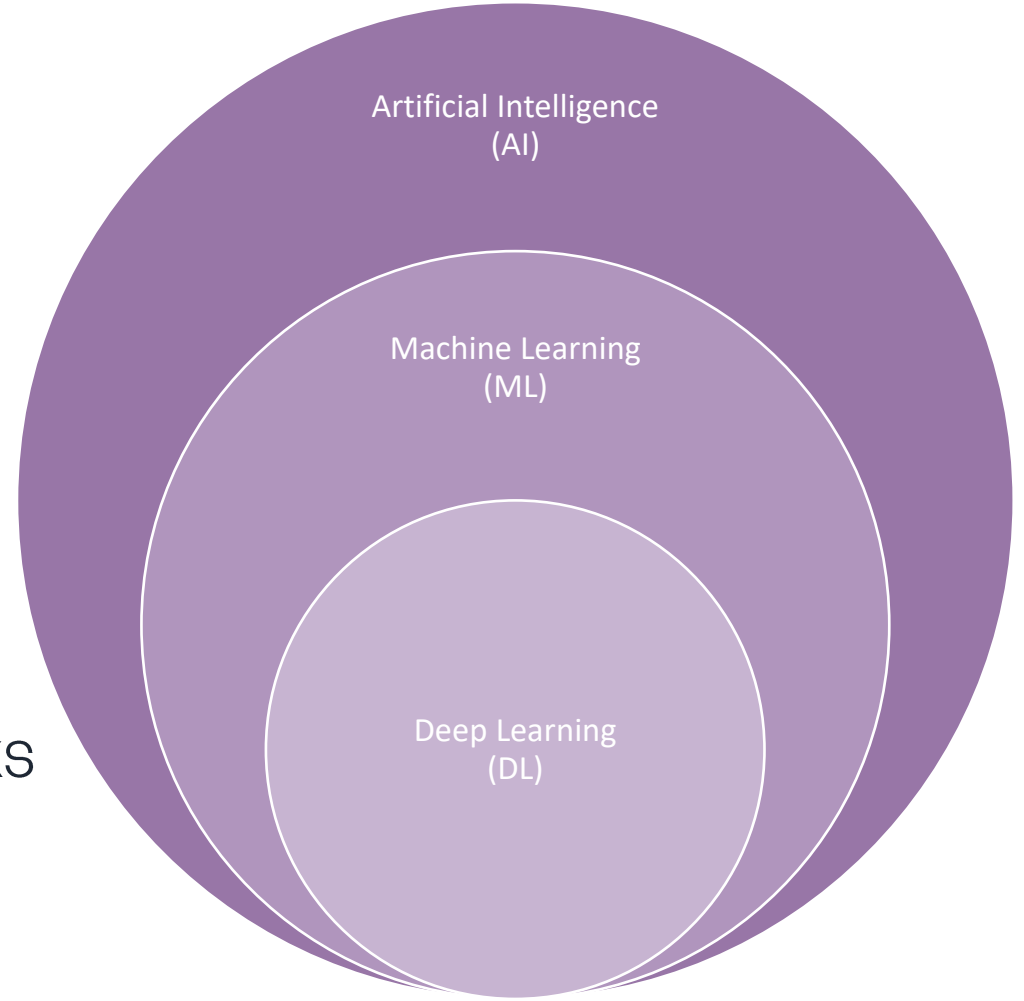
Machine Learning

- Use computers to recognize **patterns** in data and make **predictions** or decisions **automatically**.

- Useful in case of:
  - Automation of human processes.
  - «Beyond Human Scope» - Ever looked through 1 TB of data? 😳

AI, ML & DL?

- Artificial Intelligence (AI)
  - Intelligent Agents
  - Deterministic!
- Machine Learning (ML)
  - Collection of Mathematical Methods to find **Patterns** in Data.
- Deep Learning (DL)
  - Machine Learning with Neuronal Networks
  - Still Deterministic; but *Fuzzy*



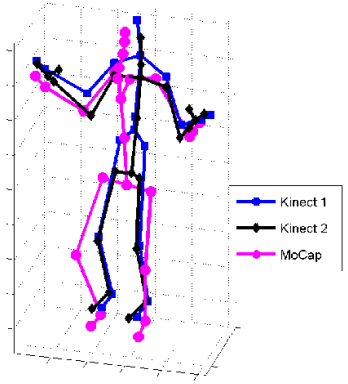
Beispiele für AI, ML, DL

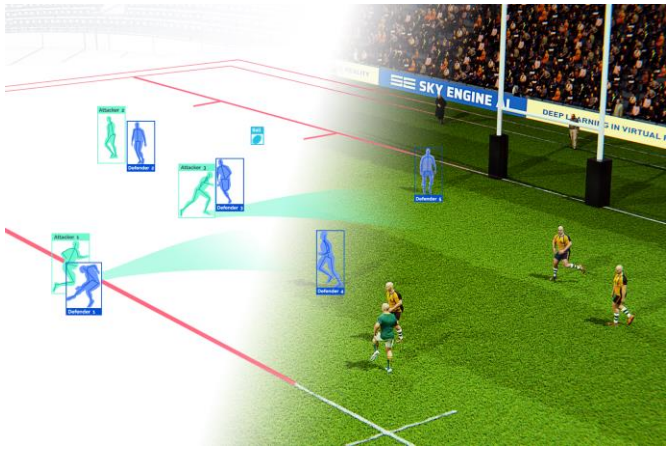
- Use Cases for AI:
  - Agent Systems (Google Maps, Board Games)
  - CSP Problems (Stundenplan)
- Use Cases for ML:
  - Regression Methods (Forecasting)
  - Random Forest (Loan, Morgage Prediction)
- Use Cases for DL:
  - Semantic Image Recognition
  - Language Understanding, Translation
  - Reinforcement Learning (Computer Games)
  - ChatGPT (GPT-4 & Falcon-44B)

Muffin or Chihuahua?

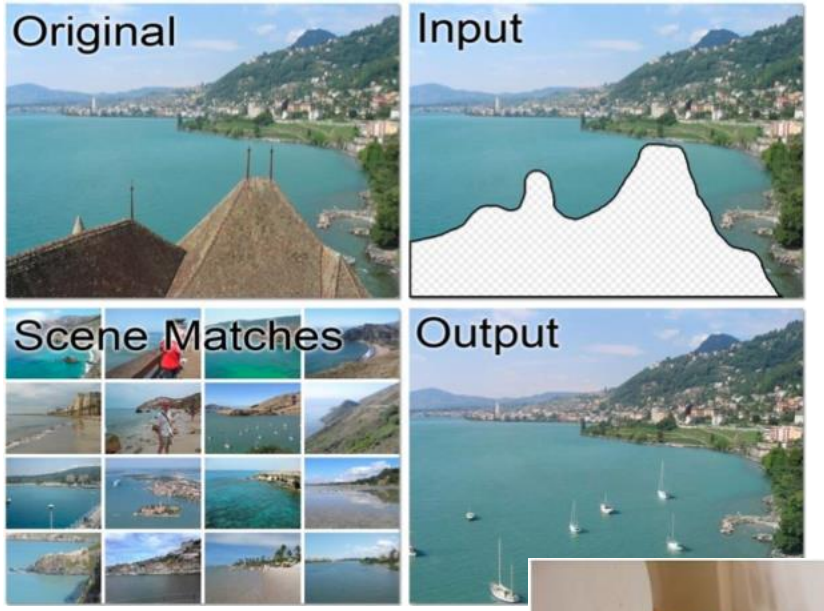
More Examples for ML Use-Cases

- Motion Capture
- Quality Control (Video)
- Movement Analysis (Hockey, Soccer)





More Examples




a cat is sitting on a toilet seat
logprob: -7.79

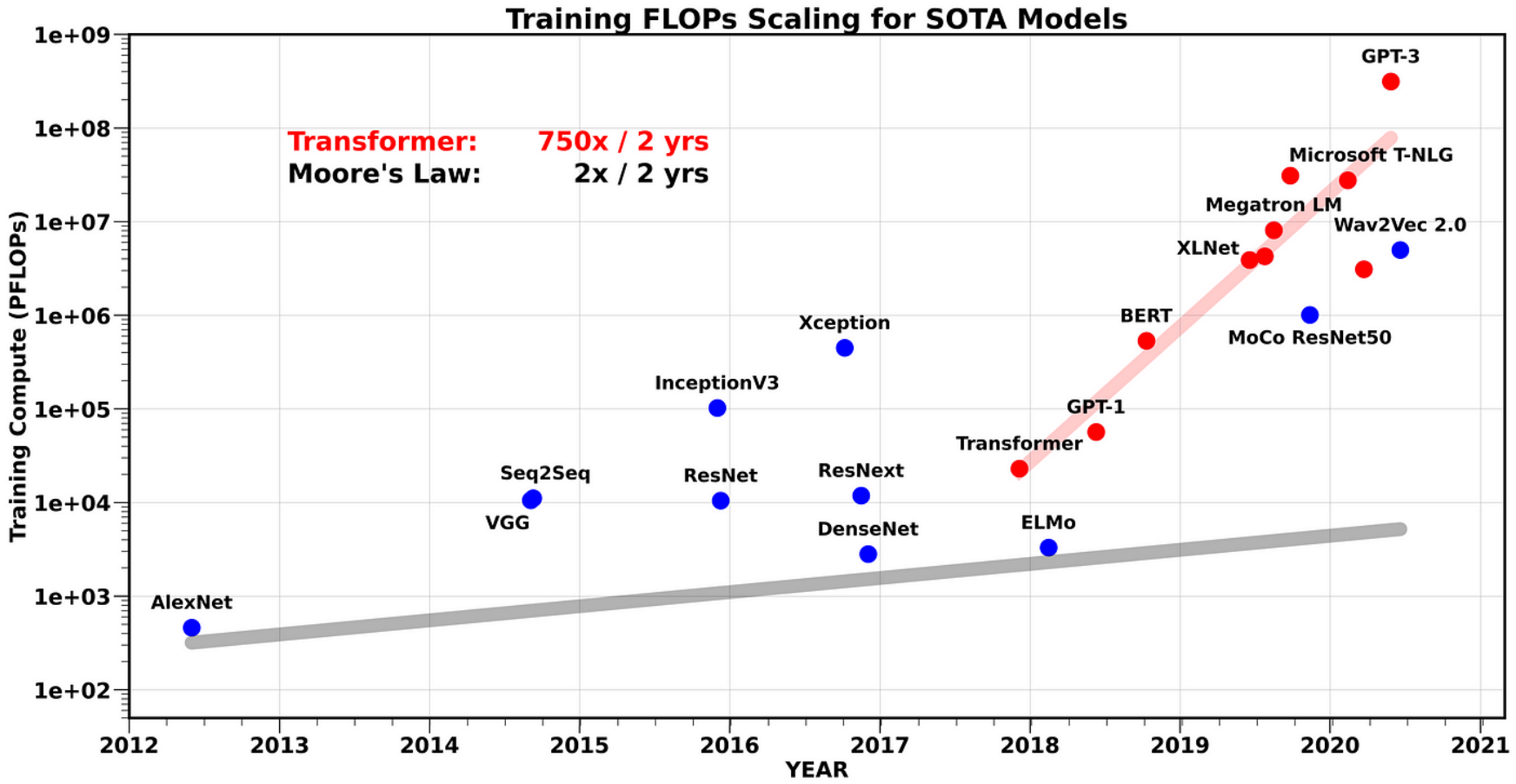

a display case filled with lots of different types of donuts
logprob: -7.78

Driver of the Deep Learning Movement

- 2006: IBM Blue Gene L ≈300TFLOPS

- 2022: Nvidia RTX4090 > 100 TFLOPS

Model Size & Computing Power



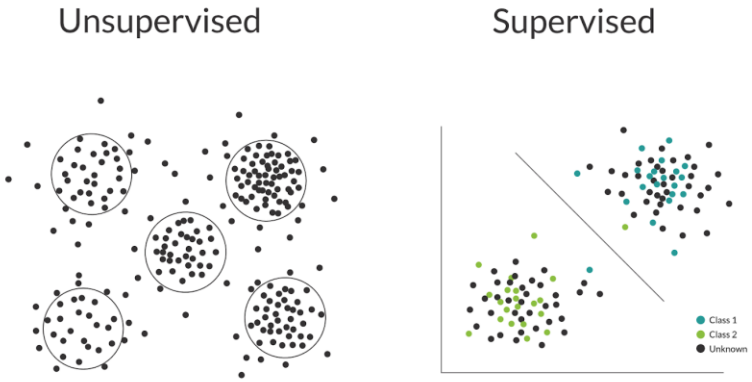
Content – Block II

- Supervised, Unsupervised Learning
- Loss, Quality, Accuracy
- What are Labels?
- Training, Validation, Testing

- Datasets
- Bias – Variance Trade-off
- IID – Independent, Identically Distributed (IID)

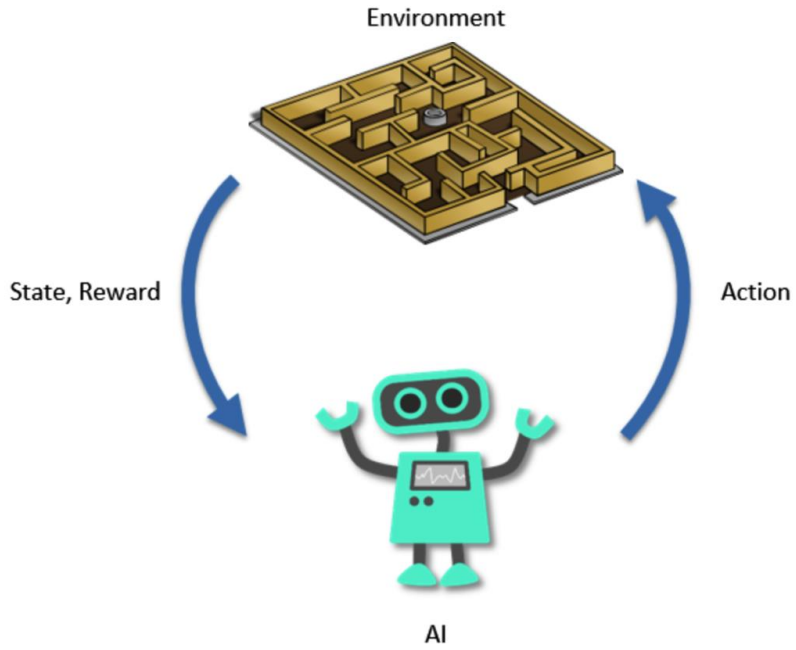
Supervised vs Unsupervised Learning

- **Supervised Learning:**
  - Making predictions based on data.
  - Training data with corresponding labels.
  - Model accuracy can be directly determined.
- **Unsupervised Learning:**
  - Understanding of data.
  - Searching for structure or patterns, correlations.
  - Not specific to a label (unsupervised).
  - Does not necessarily require labels.
  - Evaluation is often indirect or only qualitative.



Reinforcement Learning

- **Reinforcement Learning:**
  - Targeted Learning
  - Reward if the Agent gets better.
  - Games, Model Predictive Control



- Example: Gran Turismo Sophy

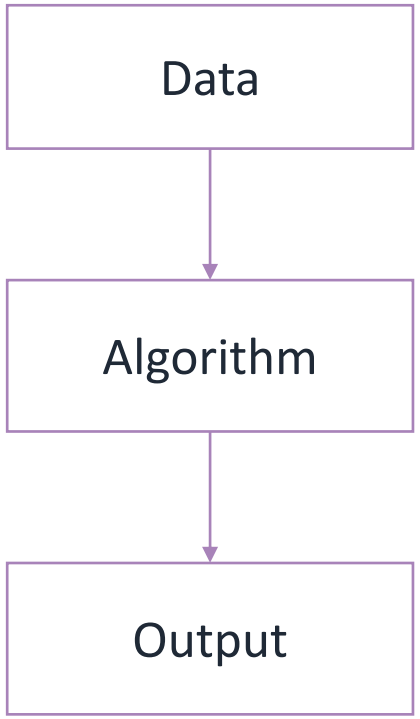
https://www.youtube.com/watch?v=AmtzhWilq0I

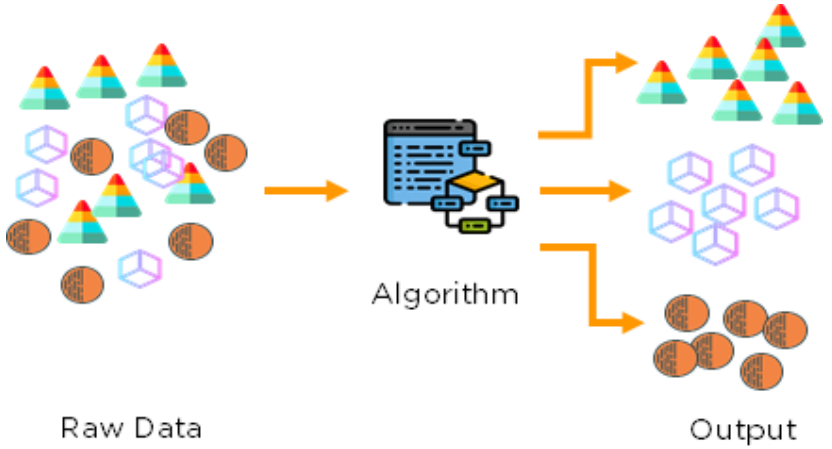
Supervised and Unsupervised, that’s it?

- Semi-Supervised:
  - Often used to improve supervised algorithms.
  - Reality: Many data points, but few have labels.
  - Example: Network Intrusion Detection.

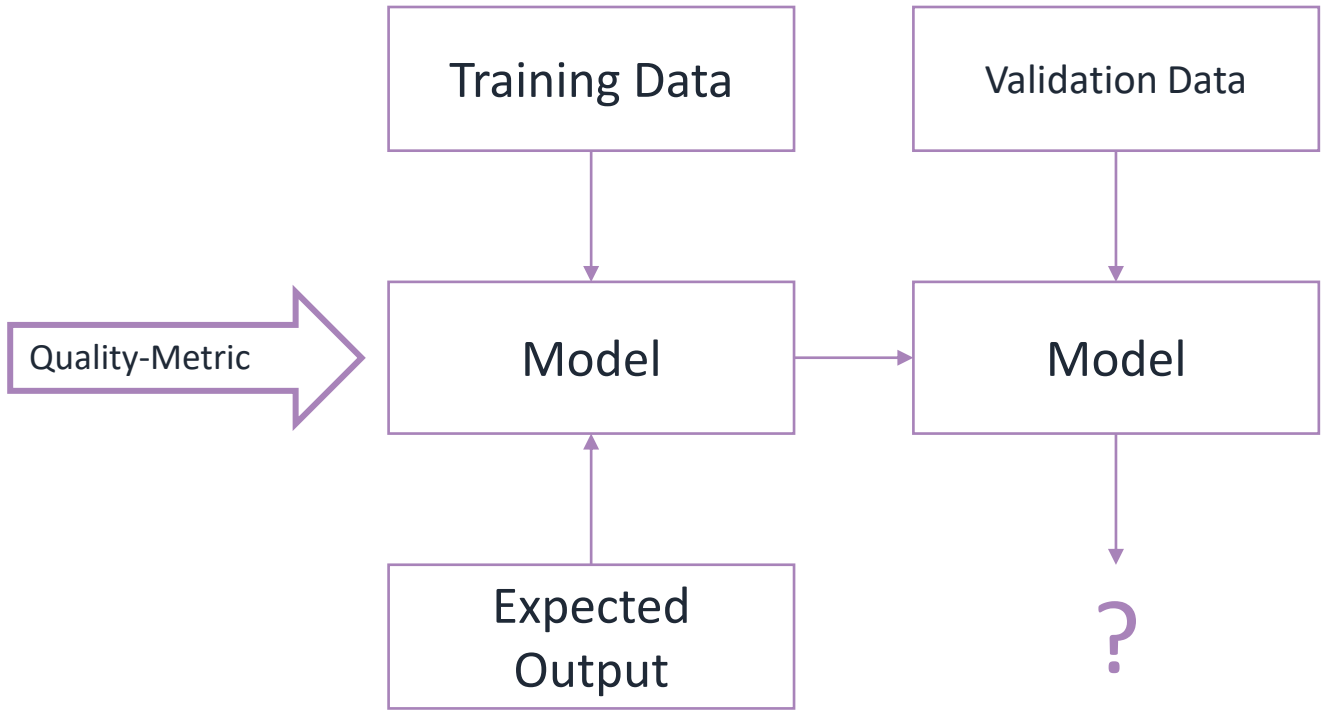
- Advantages:
  - Requires fewer experts for dataset creation.
  - Offers significantly more possibilities than unsupervised learning.

Unsupervised Models

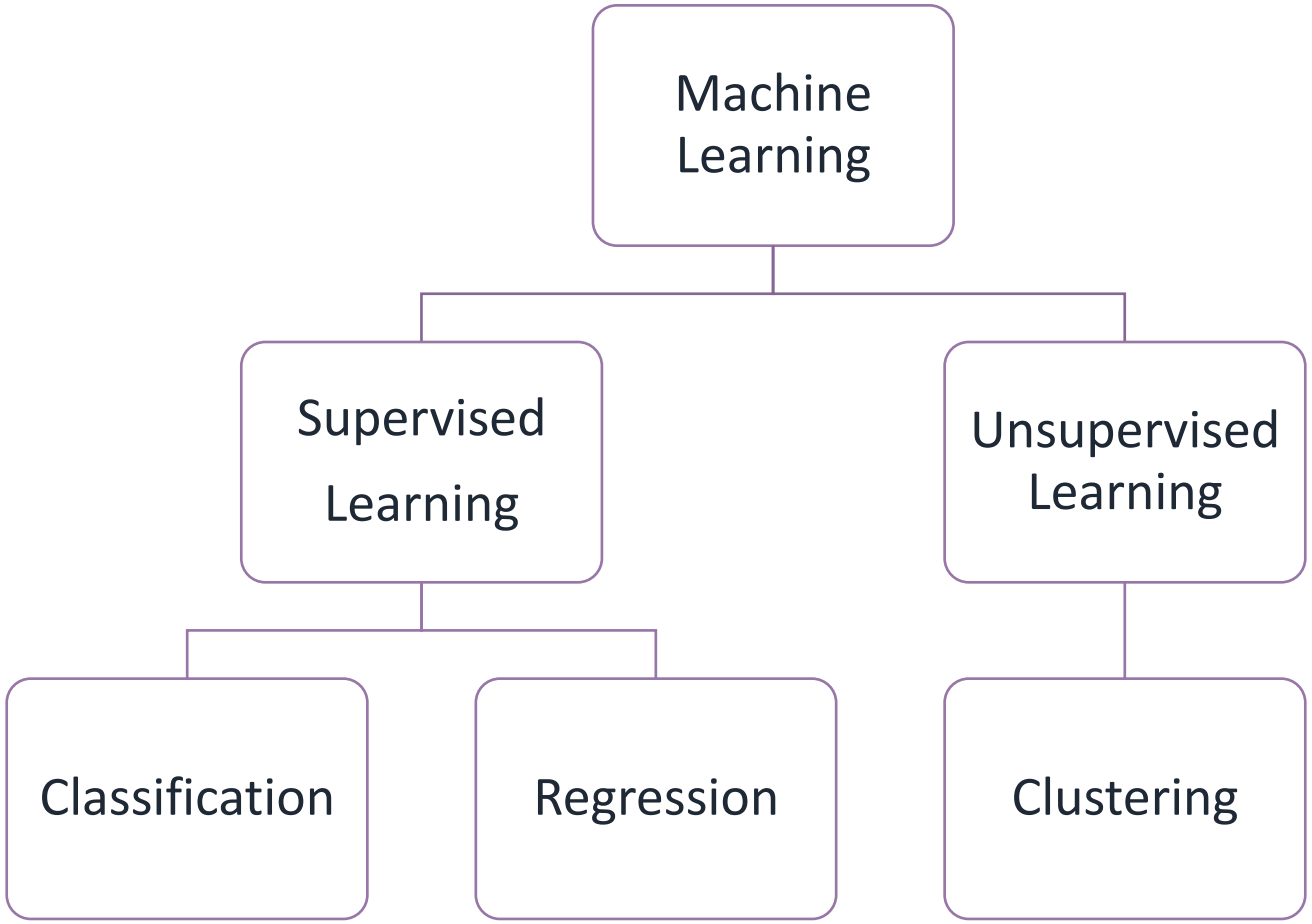




Supervised Models



Methods and Propierties



Overview on Methods

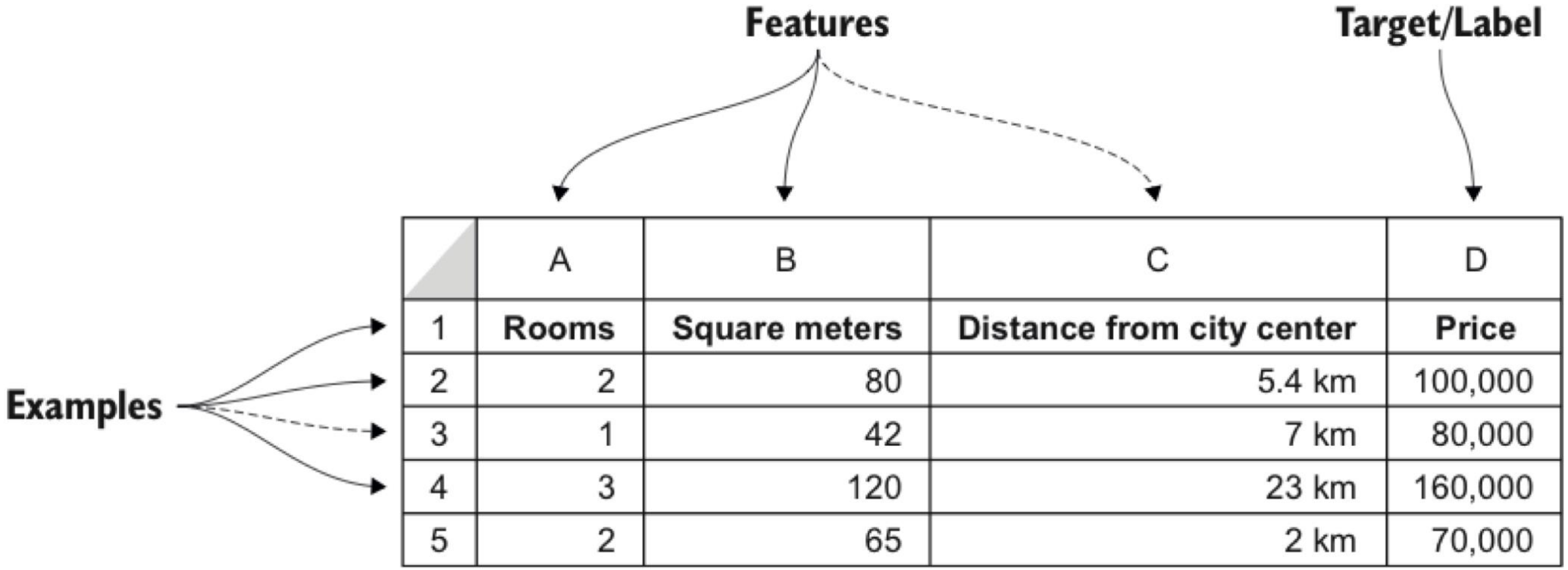
- Supervised:
  - Regression (Really?)
  - SVM (Support Vector Machine)
  - Decision Trees
  - Neuronal Networks: Problem -> Label
- Unsupervised:
  - Clustering
  - Neuronal Networks Problem -> Problem (Transformer)
- Semi-Supervised:
  - Neuronal Networks: Problem -> Problem & Problem -> Label

Let’s talk about Features?

- Ein Feature ist quasi ein ‘einzelner Datenpunkt’: **X, y**
  - Oft wird in Python X und y angetroffen, das kommt aus der mathematischen Beschreibung

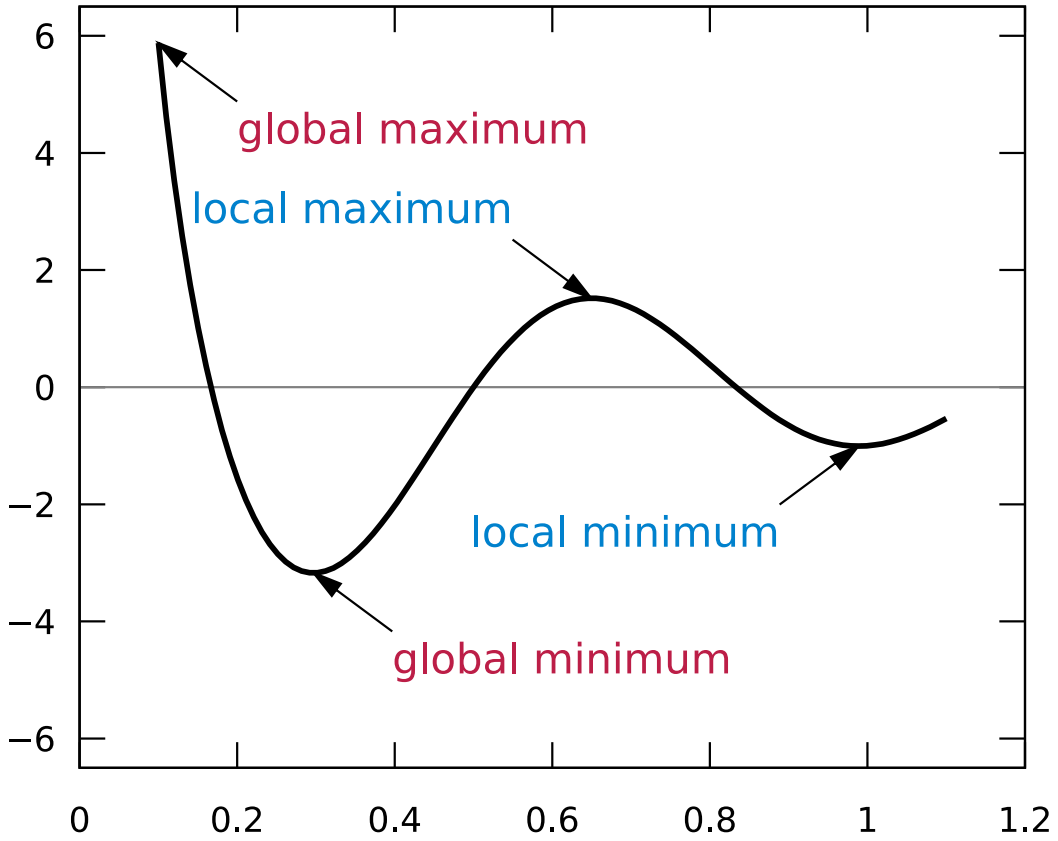
- Numerical Values can stay numerical: 27 is 27!
- Categorical Values can be handled like Enums:
  - Car: 1, Bike 2
- Sometimes abstract representations are necessary:
  - Cat is at home
  - Dog is at home
  - Represented by 2 Vectors

Features, Examples, Labels

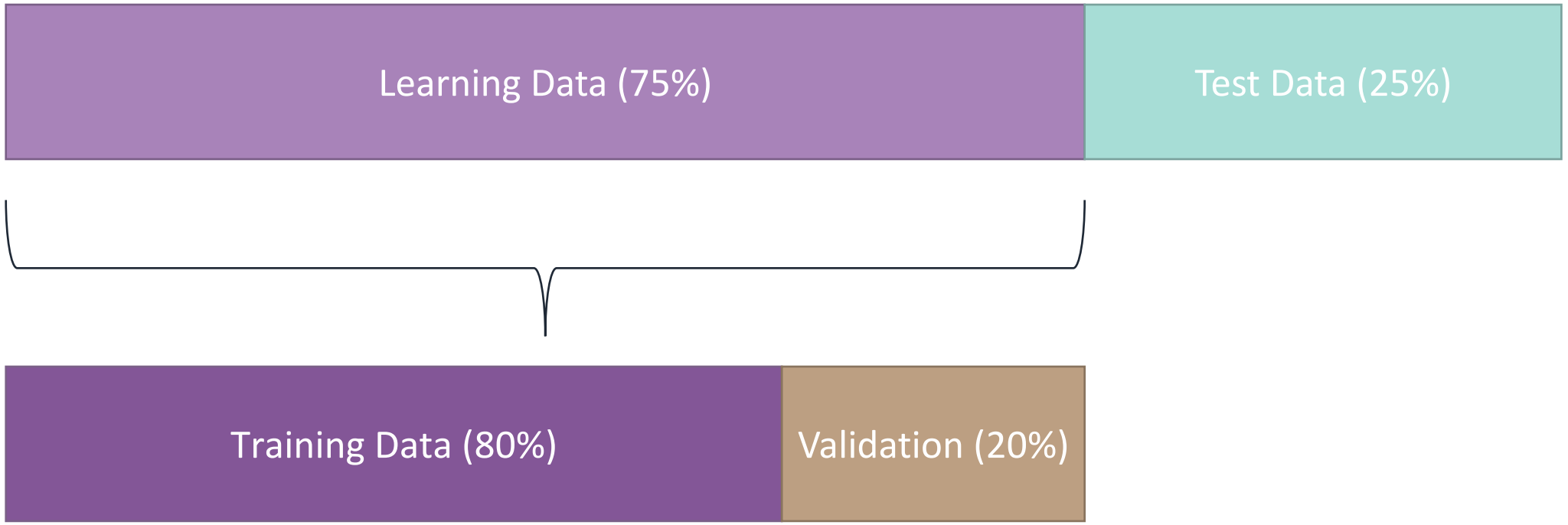


	A	B	C	D
1	**Rooms**	**Square meters**	**Distance from city center**	**Price**
2	2	80	5.4 km	100,000
3	1	42	7 km	80,000
4	3	120	23 km	160,000
5	2	65	2 km	70,000

Supervised Learning: Optimization Problem



Data Distribution



Training, Validation & Testing

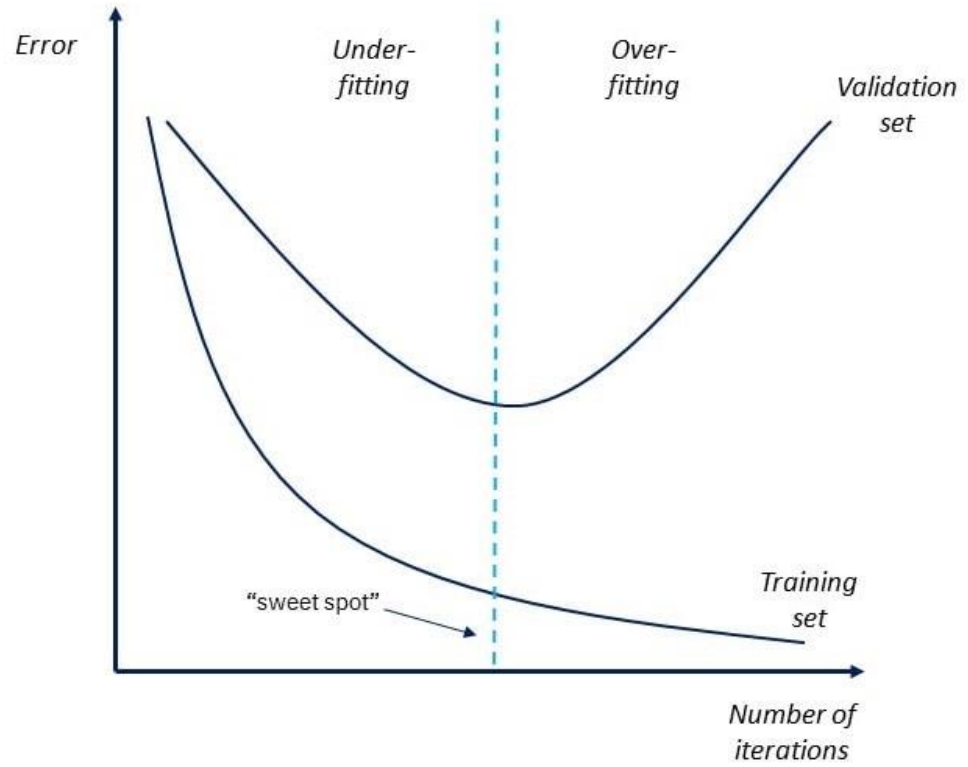
- Training Set:
  - Used to train the model.
  - These data can be manipulated as needed.
- Validation Set:
  - Validation of the trained model.
  - Checking the hyperparameters.
  - How well has the learner (model) performed?
- Test Set:
  - Independent dataset used solely for quality control.
  - Data distribution must be analogous to the training set and validation set (e.g., 10 fruits, 3 apples).

Hyperparameter?

- Hyperparameters are control settings that influence the model.
  - Decisions like which model to train (regression, forest, SVM, etc.) are made through hyperparameters.
  - Hyperparameters are not learned during training; they determine the model's architecture.
  - In function fitting, considerations include the degree (2nd, 3rd, 4th, etc.) such as x^2, x^3, x^4.

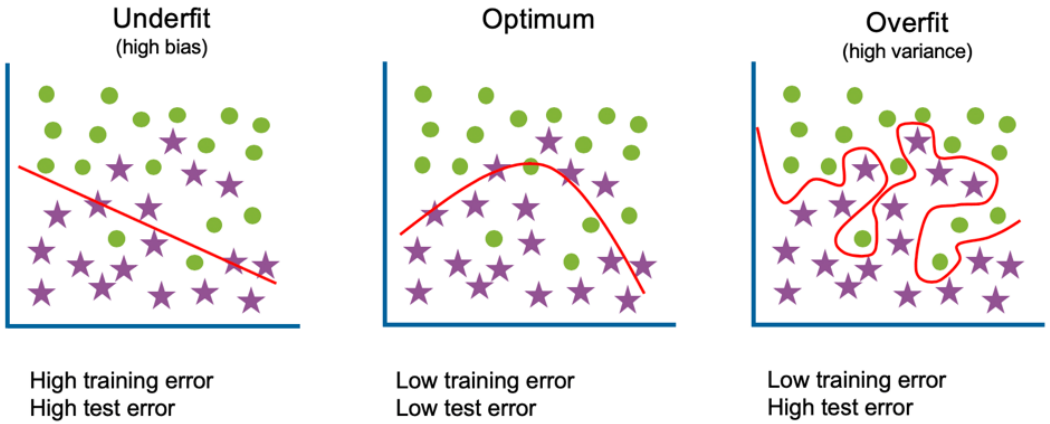
- Hyperparameter tuning is often performed on the training and validation sets.
  - Grid Walk
  - **Bias**: "He who looks at the validation set too often knows too much."

Overfitting & Underfitting



- Underfitting:
  - Training and validation 'in-sync' but performing poorly!
  - The problem may be too 'difficult' for the model.
  - Requires a more sophisticated model.
  - More 'training time' may be necessary.
  - Consider incorporating more features or gathering more data.
- Overfitting:
  - Training great – validation bad
  - More Data
  - Model Complexity to high

Bias Variance – Trade-off



- Underfit
  - To “simple”
- Optimum:
  - That’s what we need!
- Overfit:
  - Bad generalization

Question

Why does the training error decrease monotonically with increasing model complexity?

Question

Why does the training error decrease monotonically with increasing model complexity?

- The larger model allows for a larger search space.
- More parameters enable 'more learning

Question

Why is the Test Error decreasing with increasing model complexity?

Question

Why is the Test Error decreasing with increasing model complexity?

- As long as we ‘under-fit’, more information can be ‘learned’. Training and Test errors decrease.
- Higher model-complexity enables more «insight»

Question

Explain what is happening after the optimum, and why?

Question

Explain what is happening after the optimum, and why?

- Over-fitting
- We learn »noise» instead of «signal».

Question

Why is the Bias-Variance trade-off so important?

Question

Why is the Bias-Variance trade-off so important?

- It's the fundamental trade-off of too much, and too little.
- We know: If it looks too good, something is fishy.

Recap: Supervised Learning Notation

- X: Training Data
- y: Training Label
- $\hat{y}$: Predicted Label

- $\tilde{X}$: Test Data
- $\tilde{y}$: Test Labels

Recap: Supervised Learning Notation

X

Eier	Milch	Fisch	Soja	Nüsse	Gluten
0	0.7	0	0.3	0	0
0.3	0.7	0	0.6	0	0.01
0	0	0	0.8	0	0
0	0.7	1.2	0	0.1	0.01
0.3	0	1.2	0.3	0.1	0.01

Krank
1
1
0
1
1

y

- Feature Matrix X: Spalte ist Feature, Zeile ist Datensample für Kategorie
  - X_ij entspricht Menge an Essen j an Tag I
  - X_i entpricht der Nahrung an Tag I
- Label y: Enthält ob “Krank” oder ”nicht Krank”

Können wir überhaupt lernen?

- Sagt unser Training Error etwas über den Test Error aus?
  - NEIN: Testdaten können möglicherweise nichts mit Trainingsdaten zutun haben
  - Einnahme von Medikamenten wurde nicht dokumentiert (Laktose-Tabletten)

X

Eier	Milch	Fisch
0	0.7	0
0.3	0.7	0
0	0	0

Krank
1
1
0

y

$\tilde{X}$

Eier	Milch	Fisch
0	0.7	0
0.3	0.7	0
0	0	0

Krank
0
0
0

$\tilde{y}$

- Um zu lernen, müssen wir eine Annahme treffen:
  - Trainingsdaten und Testdaten müssen im Zusammenhang stehen
  - Die einfachste Annahme: **Independent and identically distributed (IID)**

Dataset Splitting:

- Grundsätzlich muss Training-Set, Validation-Set, Test-Set gleiche **Verteilung** haben!
- Aus 100 Features sind: x Training, y Validation, z Test
  - x, y, z sind ‘in sich’ gleich!

- Gleichzeitig bedenken wir aber:
  - Haben wir ‘ungleichmässige’ Verteilungen?
  - Sind wir zeitliche gebunden? (Börsenkurse, Fabrik)

IID Assumption

- **Independent and identically distributed (IID)**:
  - Unabhängig und identisch verteilte Zufallsvariabeln

- IID Assumption ist erfüllt wenn:
  - Alle Beispiele haben / kommen von der gleichen Verteilung
  - Die Beispiele sind unabhänig (Reihenfolge spielt keine Rolle)

- Beispiel:
  - Trainingset X und Testset $\tilde{X}$ enthält blaue und rote Kugeln
  - Wahrscheinlichkeit P_blau ist in Menge X und $\tilde{X}$: **gleich**

IID Assumption: Kartenspiel

- Französisches Blatt:
  - 4 Farben (Kreuz, Pik, Herz, Karo)
  - 9 Wertigkeiten (6-10, Bube, Dame, König, Ass)

- Ist IID Assumption erfüllt?
  - Ziehen einer Karte (oberste), zurücklegen, mischen, wiederholen
  - Ziehen einer Karte, zurücklegen, wiederholen
  - Ziehen einer Karte, *nicht* zurücklegen, mischen, wiederholen

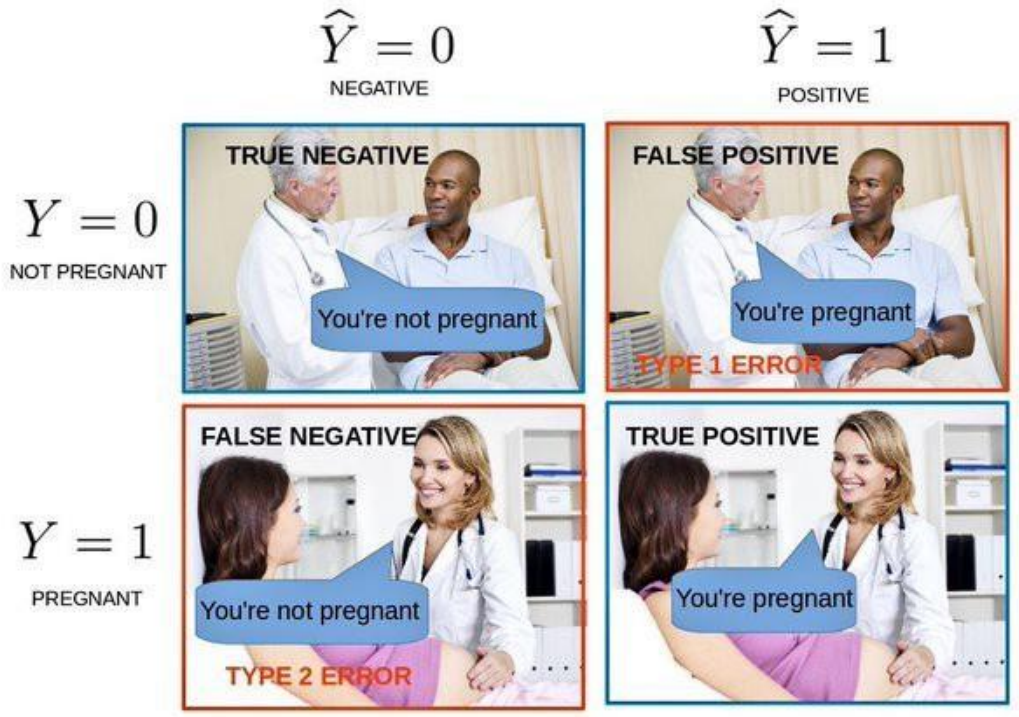
Optimisation Bias

- Nehmen wir einen multipe-choice Test (a,b,c,d) mit 10 Fragen:
  - Zufälllig ausgefüllt: 25% richtig
  - Zufällig 2x ausgefüllt und den Besseren gezogen: 33% richtig
- Skalieren wir das Experiment auf 10 Tests: 47%
  - Bei 100: 62%
  - Bei 1000: 73%
- Wenn man viele Versuche hat, dann hat man die Möglichkeit es gut zu tun.
- Bester Score mit *Random.seed(42)*!

Metriken

- Confusion Matrix für 2-Klassen Klassifikation
- Metriken: Accuracy, Recall, Precision, F1-Score
- Unterschied zwischen Recall und Precision
- Decision Threshold: ROC Kurve und AOC Fläche

Confusion Matrix

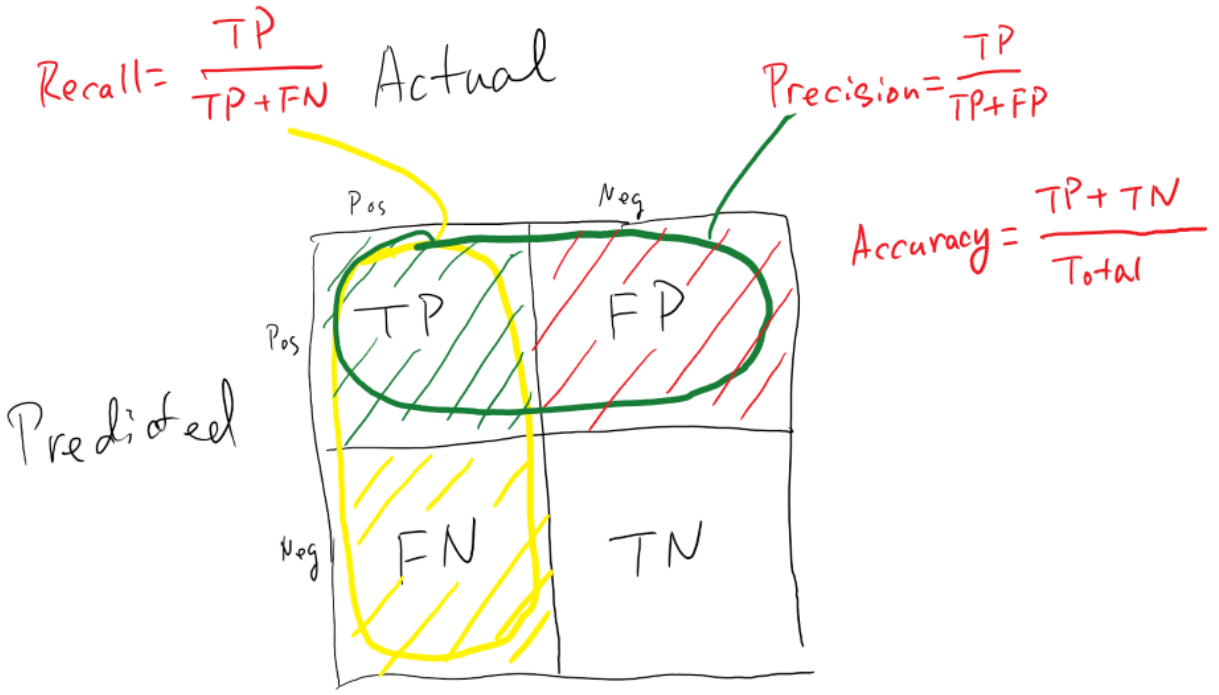


		Echte Werte (Actual Values)	
		1	0
Vorhergesagte Werte (Predicted Values)	1	True Positive (TP)	False Positive (FP): Type 1 Error
	0	False Negatives (FN): Type 2 Error	True Negatives (TN)

Metriken

- $Accuracy = \frac{Correct}{Total} = \frac{TP+TN}{TP+TN+FP+FN}$
- Recall: ‘Wieviele relevante Daten habe ich erwischt?’
  - $Recall = \frac{TP}{TP+FN}$
- Precision: ‘Wieviele ausgewählte Daten sind relevant?’
  - $Precision = \frac{TP}{TP+FP}$
- $F1 - Score = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}} = 2 * \frac{Precision*Recall}{Precision+Recall}$

Precision und Recall



VeeMG

adnexo

Case 1: Covid

- Kosten für FN > Kosten für FP

- Labels:
  - Covid-Positive: 1
  - Gesund: 0

		Echte Werte (Actual Values)	
		1 (hat Covid)	0 (gesund)
Vorhergesagte Werte (Predicted Values)	1	True Positive (TP)	False Positive (FP): Type 1 Error
	0	False Negatives (FN): Type 2 Error	True Negatives (TN)

Hat Covid, aber Diagnose sagt gesund!

Case 1: Covid

- TP sind Covid Patienten mit Covid Diagnose
- TN sind gesunde Patienten mit gesunder Diagnose

- FP: Schade 10 Tage Quarantäne für nichts!
- FN: Potentielles *Risiko*, da Ansteckungsmöglichkeiten!

- FN ist für das System teurer als FP: ***Recall***

Case 2: Spam

- Kosten für FN < Kosten für FP

- Labels:
  - Spam: 1
  - Kein Spam: 0

		Echte Werte (Actual Values)	
		1 (Spam)	0 (not Spam)
Vorhergesagte Werte (Predicted Values)	1	True Positive (TP)	False Positive (FP): Type 1 Error
	0	False Negatives (FN): Type 2 Error	True Negatives (TN)

Das Jobangebot von Google

Case 2: Spam

- TP sind Spam Mails, welche als Spam Mails erkennt werden
- TN sind legitime Mails, welche als solche erkennt warden

- FP: Das Job Angebot von Google
- FN: Der Persische-Prinz und sein Milionen-Erbe

- FP hat weittragendere Konsequenzen als FN: *Precision*

Übung: Kreditvergabe

- Zeit 10min
- Überlegen Sie mit ihrem Partner das Szenario Bankkredit
  - Label 1: Schlechter Kredit (wird nicht bezahlt)
  - Label 0: Guter Kredit (wird bezahlt)

Nachtrag: F1-Score vs Accuracy

- Accuracy: Fokus auf TP und TN
- F1: Fokus auf FP and FN

- Accuracy ist ‘intuitiv’, F1 ist etwas ‘künstlicher’
- Accuracy funktioniert gut, wenn Klassen in balance
- F1-Score funktioniert aber auch, wenn Klassen disbalance besteht

- In der Realität ist F1-score nützlicher als Accuracy.
  - *Ich oute mich hier: primäres Augenmerk auf Accuracy und auf Confusion Matrix.*

Label Prediction: Probabilistisch

- Labels warden in allen stochastischen ML Modellen probabilistisch vorhergesagt:
  - Die Abbildung kann somit [0;1] umfassen
  - Je näher an 0 desto eher ist es 0.
  - Je näher an 1 desto eher ist es 1.

- **Gutes Model**: Scharfe Trennung
- Threshold-Tuning möglich

Warum Neuronale Netzwerke?

- Popularität in Hype-Wellen:
  - Zur Zeit: Eines der heissten Themen in der Wissenschaft

- Aktuelles Hoch getrieben durch unbeschreibliche Performance in:
  - Computer Vision Tasks (Bild Erkennung, Semantische Segmentierung)
  - Language Recognition (Speech 2 Text)
  - Language Models (Sprach Verständnis, Interpretation)

Warum Neuronale Netzwerke II?

- Haupt-Treiber für den Erfolg:
  - Grosse qualitativ hochwertige Datensets: 100TB+
  - Sehr tiefe Netze (Deep Learning)
  - Unglaubliche Computerleistung zu ‘keinem’ Geld.

- Einige Tweaks an der Struktur der Netzwerke
  - CNN (Convolution)
  - LSTM (Long—Short Term Memory)

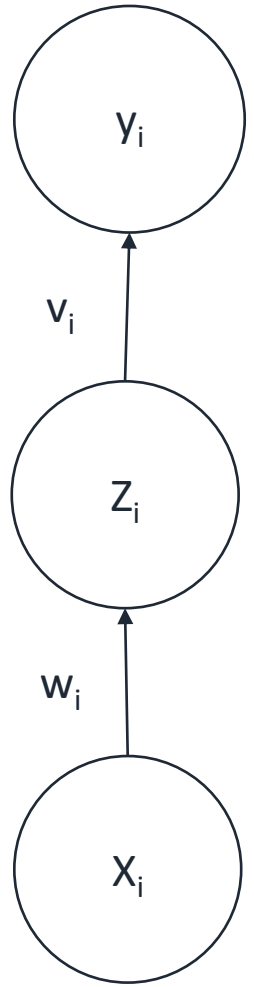
- Mathematik praktisch unverändert seit 1960...

Neural Networks: Motivation

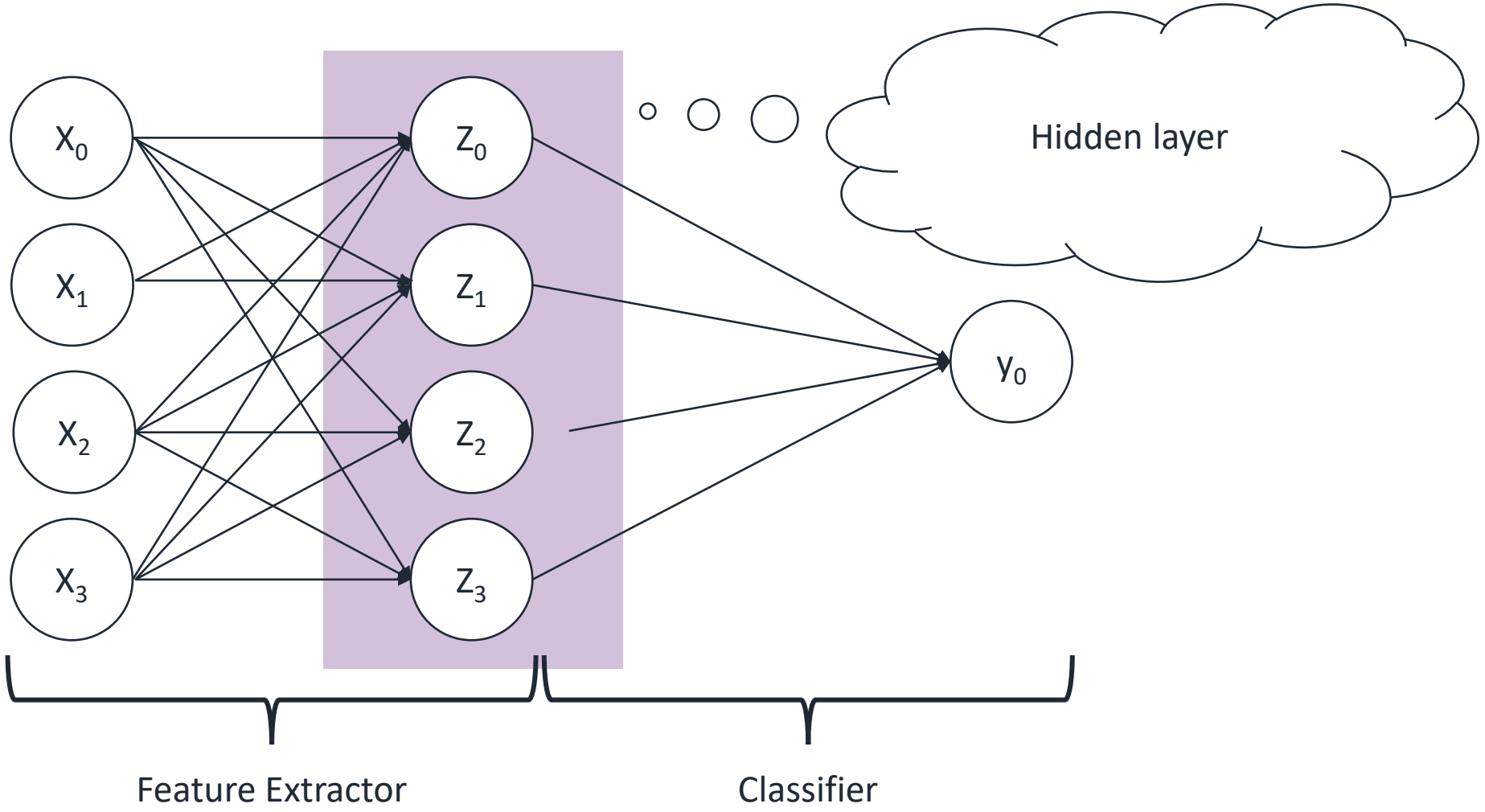
- Viele Domänen brauchen eine nicht lineare Transformation der Features
  - Aber manchmal ist es offensichtlich
    - Bsp. SciKit learn
- Neuronale Netzwerke versuchen diese Transformation zu lernen
  - Optimierung der ‘Feature’ für bestes Resultat
- Wir fangen mit einem hidden-layer an und gehen dann Deep…

Supervised Learning Roadmap

- **Level 1**: «Direct» Supervised Learning
  - Wir lernen Parameter *w* basierend auf x_i und y_i.
- **Level 2**: Basis Transformation
  - Wir lernen Parameter *v* basierend auf Basis Z_i und y_i. (x_i -> z_i existiert)
- **Level 3**: PCA, t-SNE, Latent-Factor Models (Manifolds)
  - Wir können Parameter *w* von Basis Z_i aufgrund von X_i erlernen
  - Wir können Parameter *v* für Z_i -> y_i lernen. (Classification)
- **Level 5**: Neuronale Netzwerke
  - *w* und *v* werden direkt basierend auf X_i und y_i gelernt
  - Basis Z_i wird impliziert gelernt, welche gut geeignet für Supervised Learning Classification ist.

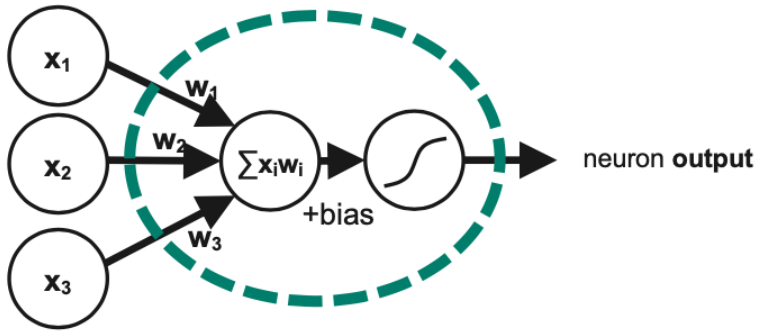


Feature Extractor & Classifier

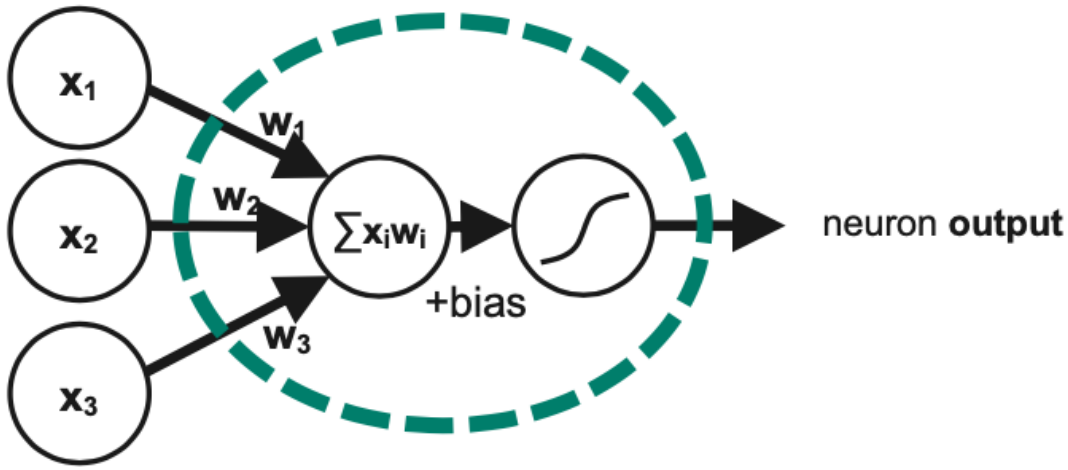


Neuronales Netz als Funktion

- Das Neuron erhält Input [x_1, x_2, …]
- Jedes Neuron besitzt eigene Eingangsgewichte *W*, einen Bias *b*.
- Output entspricht der Summe der gewichteten Inputs



Neuronales Netz als Funktion II

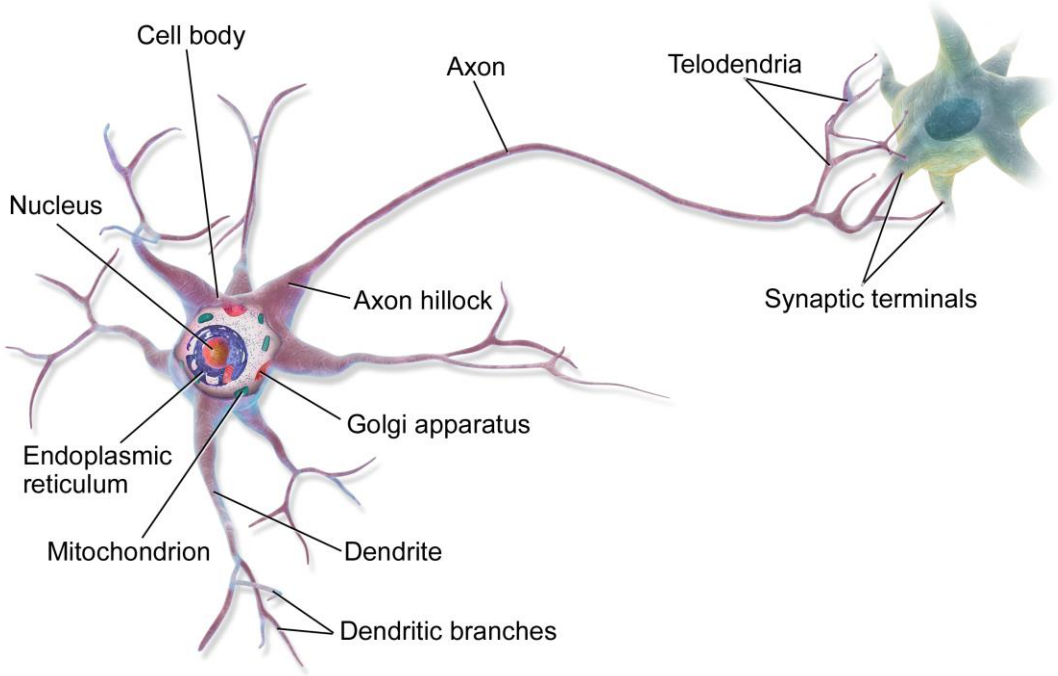


$$\sigma\left(\sum_i w_i f_i + b\right)$$

Aktivierungsfunktion?

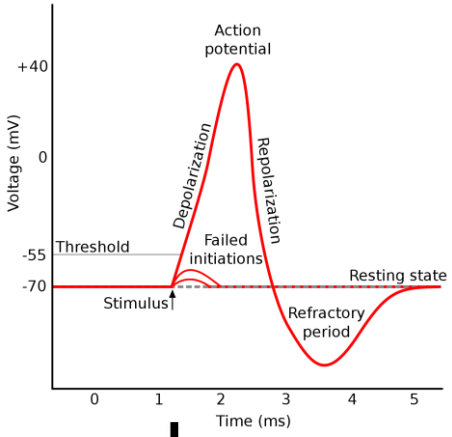
- Avoid Linear Combinations:
  - A*x + b*x = (a+b)*x

Biologische Sicht

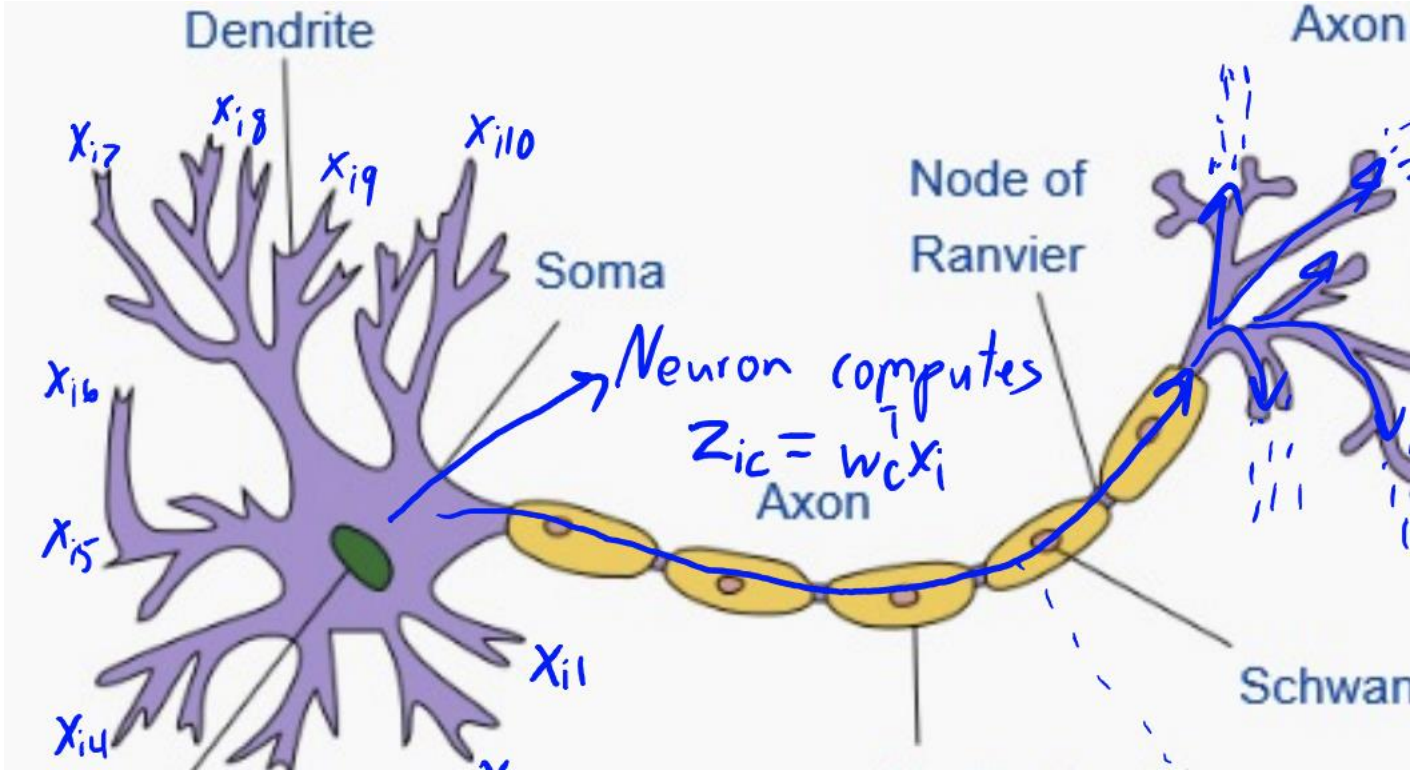


https://en.wikipedia.org/wiki/Neuron

- Neuron hat ‘Dendriten’ als Inputs
- Neuron hat ‘Axon’ als einzelnen Output
- Mit dem richtigen Input, wird ein Output getriggert

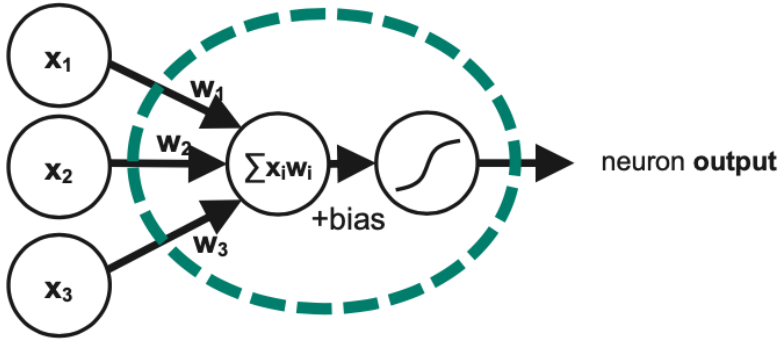


Biologische Sicht II



Single Layer Perceptron

- Fähigkeit linear separierbare Muster zu lernen
- Hohe Input Dimensionalität -> lineare Separierbarkeit, aber im n-dimensionalen Raum
- Terminologie:
  - Perceptron ist ein ‘Einzelnes Neuron’
  - Perceptron ist aber auch der Algorithmus zum trainieren eines binären Klassifizierers (Classifier-part)



- Layer können geschichtet (stacked)werden. Diese Architektur nennt man Multi Layer Perceptron MLP.
- Lasagne war eines der ersteren Deep Learning Frameworks! ☺

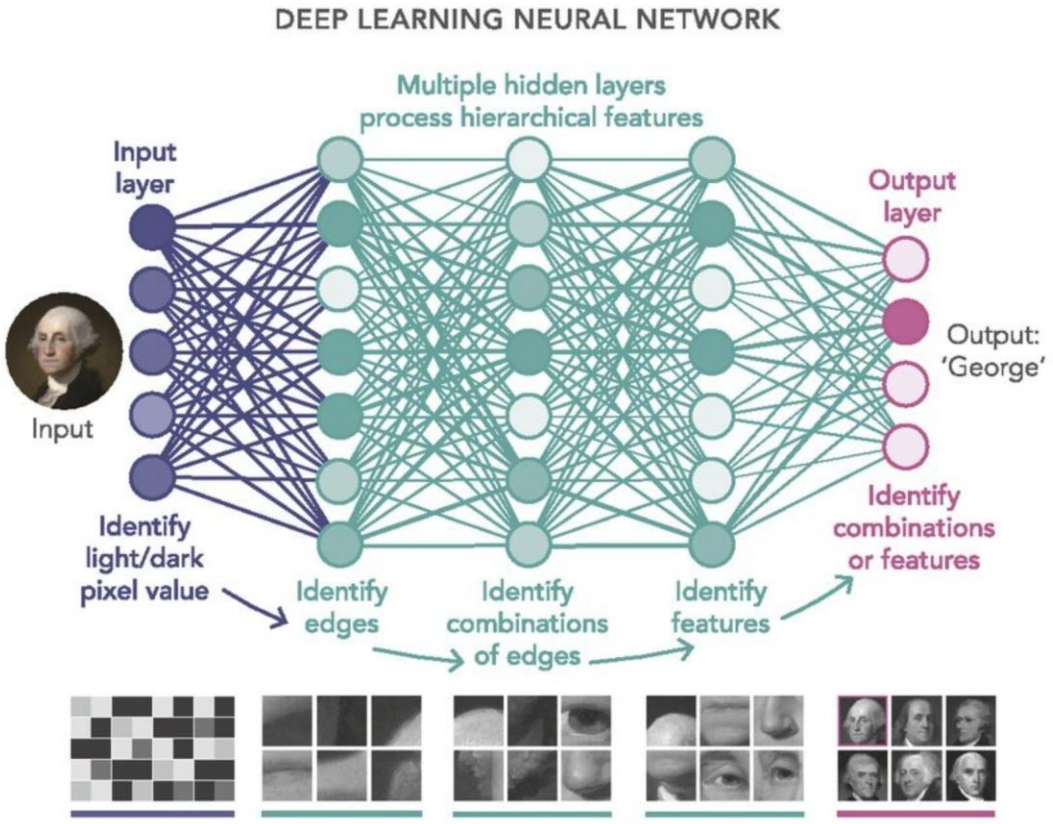
Stacking Layers -> Deep Learning

- Der Output des jeweiligen Layers, entspricht dem gewichteten Input des nächsten.
  - Komplexe Features kombinieren und aggregieren Informationen durch die Netztiefe
  - Alle Parameter werden mithilfe des Backpropagation Algorithmus erlernt. (Vorlesung Oliver am Donnerstag).
- Aktuelle KI ist angetrieben durch tiefe Modelle, mit Miliarden von Parametern (auch GPTv4!)
- Die freien Parameter werden durch **UNMENGEN** an Daten ‘trainiert’.

Visualisierung: Tiefe Neuronale Netzwerke

VeeMG

- Tiefe neuronale Netzwerke werden auch Deep Neural Networks genannt
- ANNs lernen Features zu erkennen und kombinieren in tieferen Schichten diese zu komplexeren Mustern
- Das Resultat ist nichts anderes als eine komplexe mathematische Funktion welche X_i auf y_i abbildet, bestehend aus vielen einfachen Komponenten.



https://www.positronic.ai/consulting/deep-learning/

Kleines Gedankenexperiment

- Schauen Sie sich die vorherige Folie an:
  - Wie würden Sie das Netz implementieren?
  - Besprechen Sie die Konzepte mit der Person neben ihnen.

- Implementieren Sie nicht!

Künstliche NN vs Reale NN

- Künstliche NN:
  - X_i ist Feature Darstellung der Welt
  - Z_i ist interne Darstellung der Welt
  - Y_i ist Output / Klassifikation o. Regression
- Reale NN:
  - Timing / Zeit-diskrete Aktion sind wichtig (Kontinuierliches Signal)
  - Hirn ist ‘höchst organisiert’ (Bsp. Strukturen für diverse Tasks)
  - Verbindungen -> Struktur Änderungen
  - Unterschiedliche Neurotransmitter (Elektrochemisch, Hormonell, etc.)

Agenda

- Intro classical ML Methods
- What is it
- Terminology
  - Models, Data, Cost Function
- Popular Examples
  - Decision Tree
  - KNN
- Lab: Get your hands dirty

What is a model

- Function approximation and optimization

- Assumption:
  - There exists a function / relationship / set of rues
  - To Interpret data in a certain way
  - g(x) is the «given» function

- Approximation
  - f(x^, p)

Model vs Algorithm

- An **Algorithm** is used to train a **Model**
- We can think of the **Algorithm** as a function
  - We give the algorithm data and it produces a model
  - **Model** = **Algorithm**(Data)

- Models are the specific representations learned from data
  - Example
    - f(x) = ax+b is an **Algorithm**
    - f(x) = 4x + 2 is a **Model**

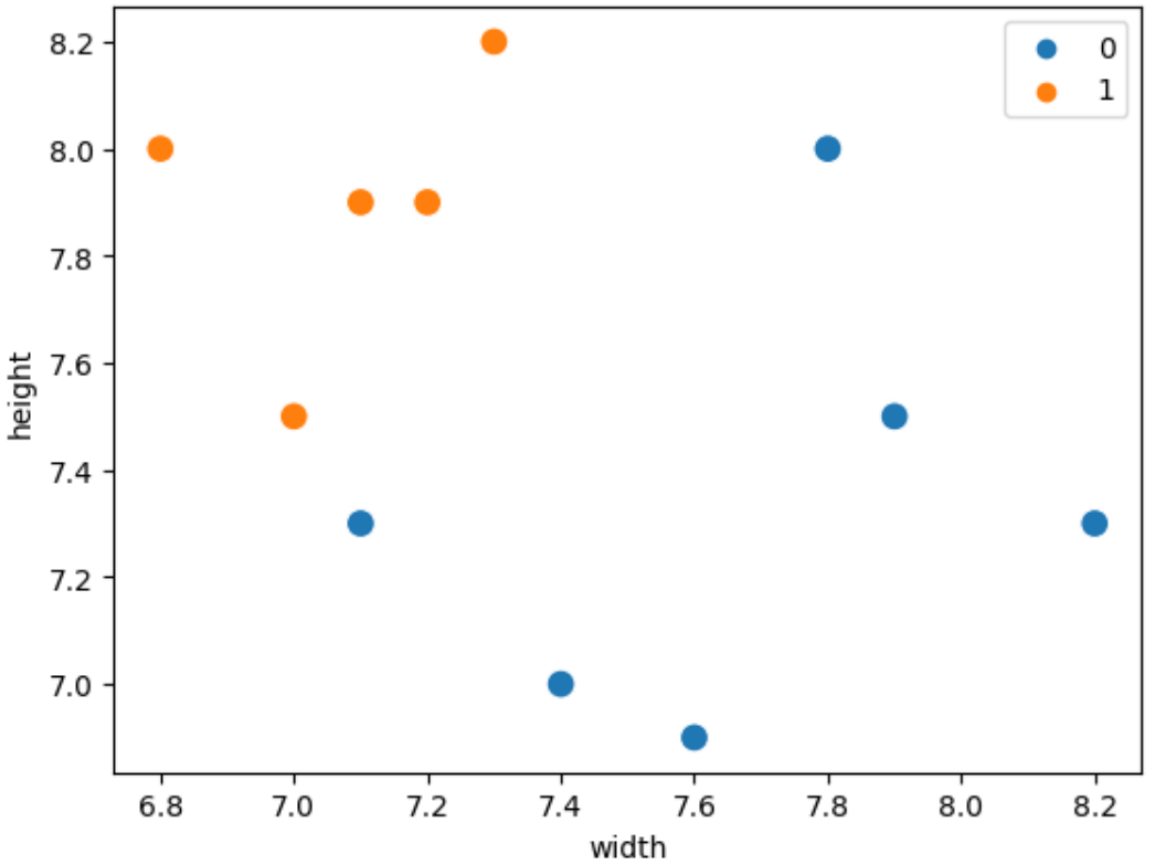
Apple & Pear Classification

- Let's design an algorithm to differentiate between apples and pears given their width and height
- Question:
  Given width and height what is the correct fruit class
- What would be your approach as a programmer?

Width	Height	Fruit
7.1	7.3	Apple
7.9	7.5	Apple
7.4	7.0	Apple
8.2	7.3	Apple
7.6	6.9	Apple
7.8	8.0	Apple
7.0	7.5	Pear
7.1	7.9	Pear
6.8	8.0	Pear
6.6	7.7	Pear
7.3	8.2	Pear
7.2	7.9	Pear

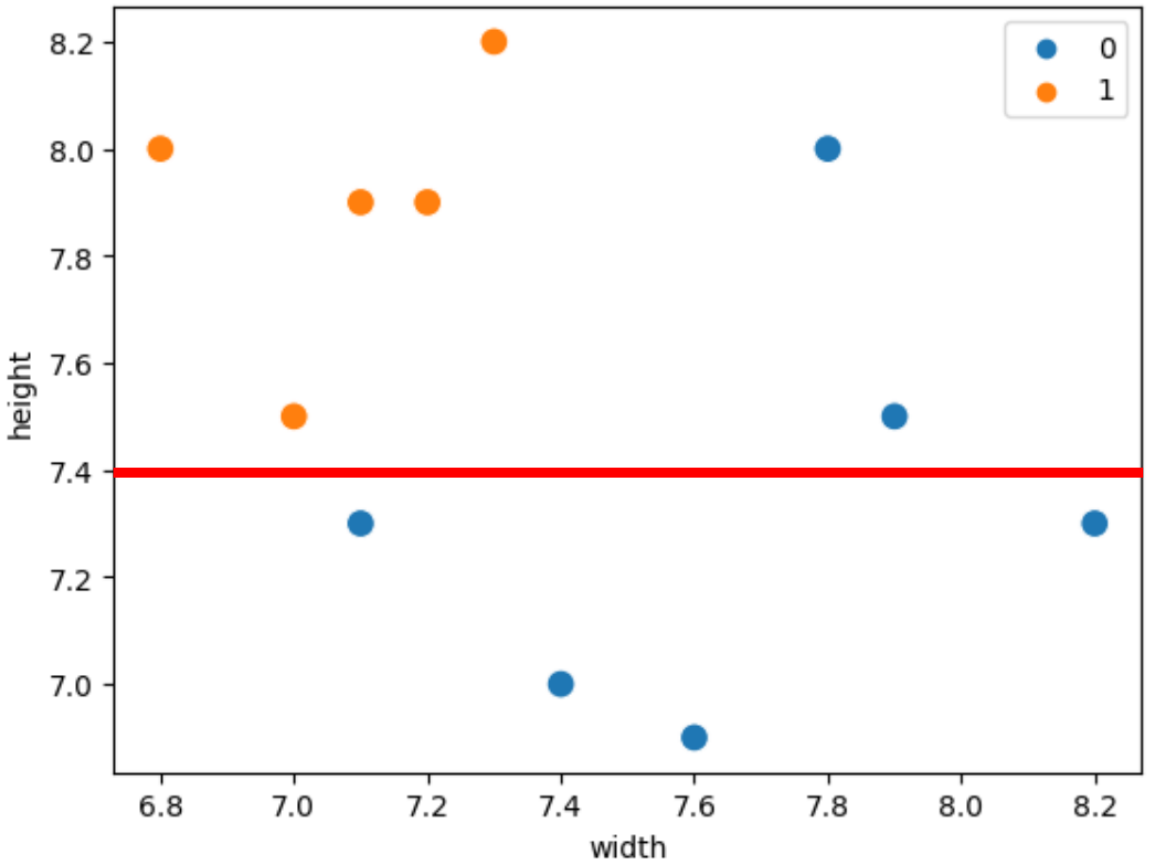
Decision Tree Idea

- Create **splits** by recursively dividing regions into subsplits
- Finish if split is **homogeneous**
- A **homogeneous split**: only contains datapoints from same class



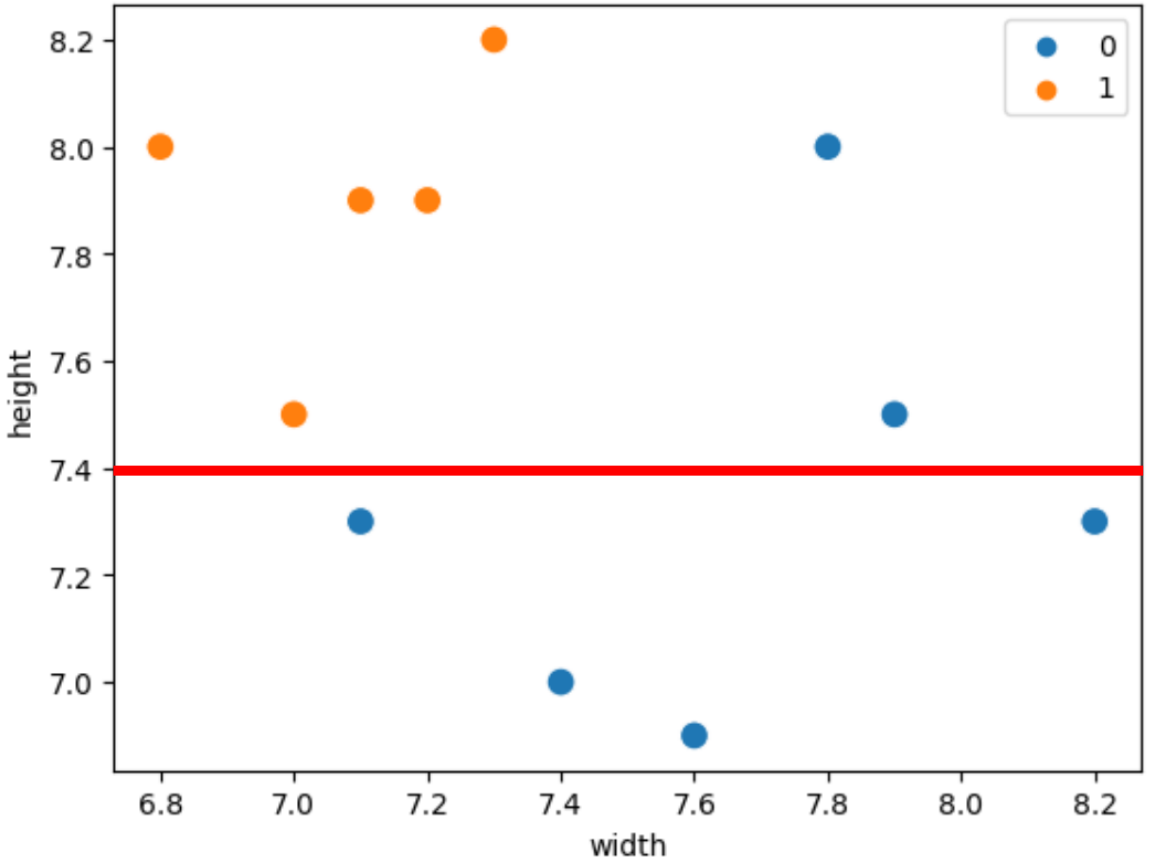
Decision Tree

- Only use vertical or horizontal lines to create homogeneous splits
- How does a programmer find that line?



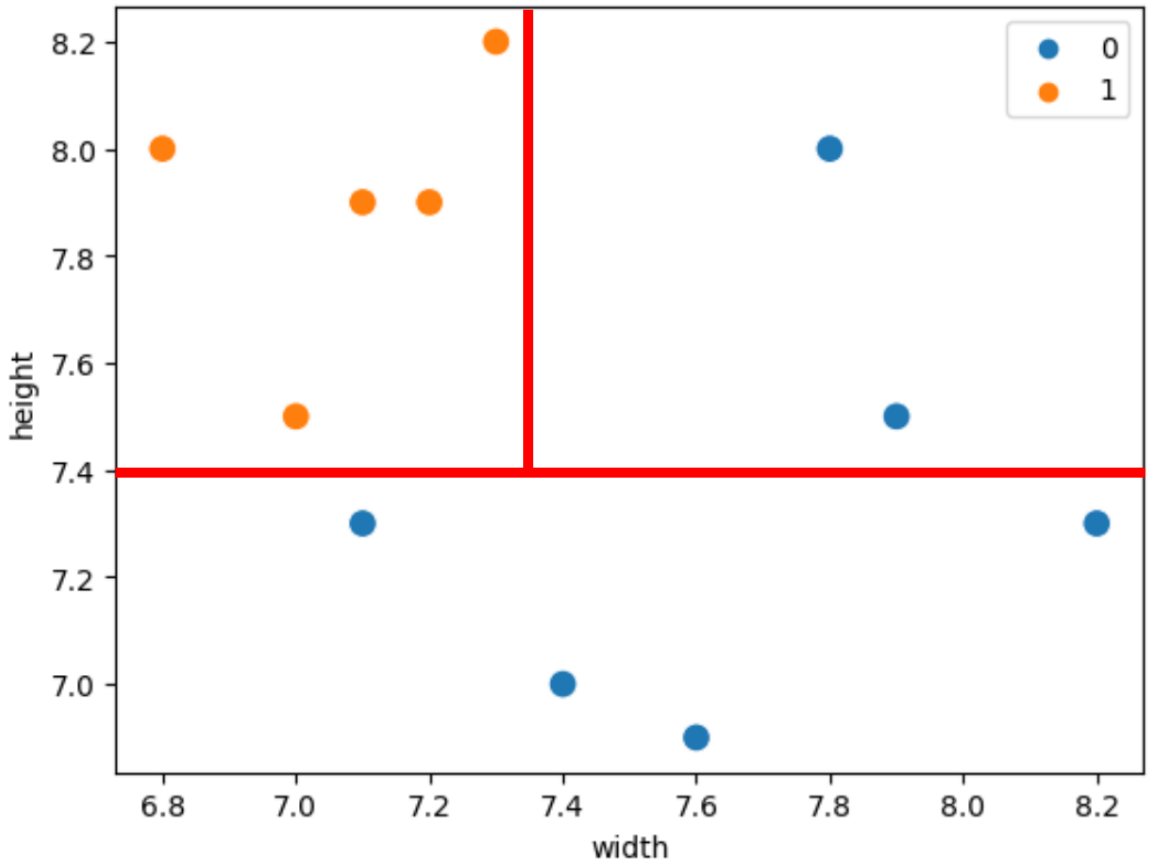
Decision Tree

- Wait what is a good split?
- Define quality for split
  - Ideas?
  - GINI
  - Entropy



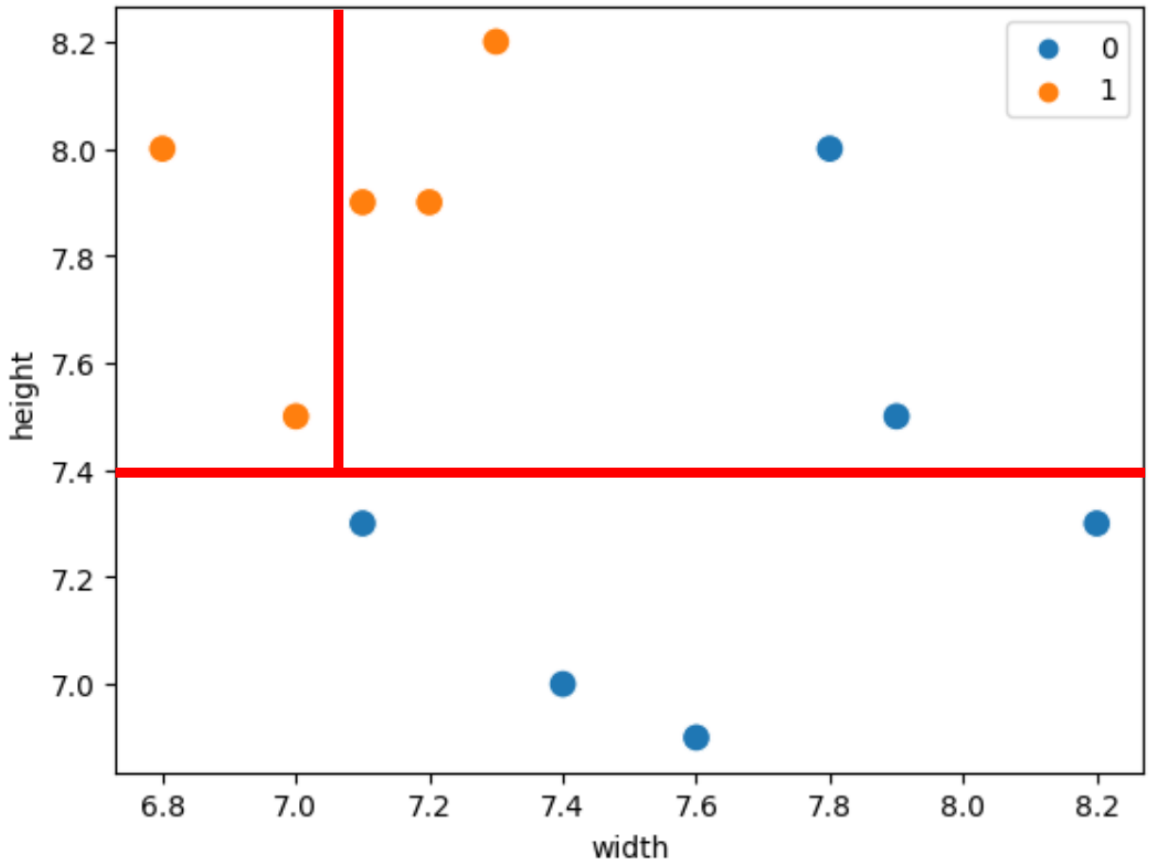
Decision Tree

- Divide splits further recursively



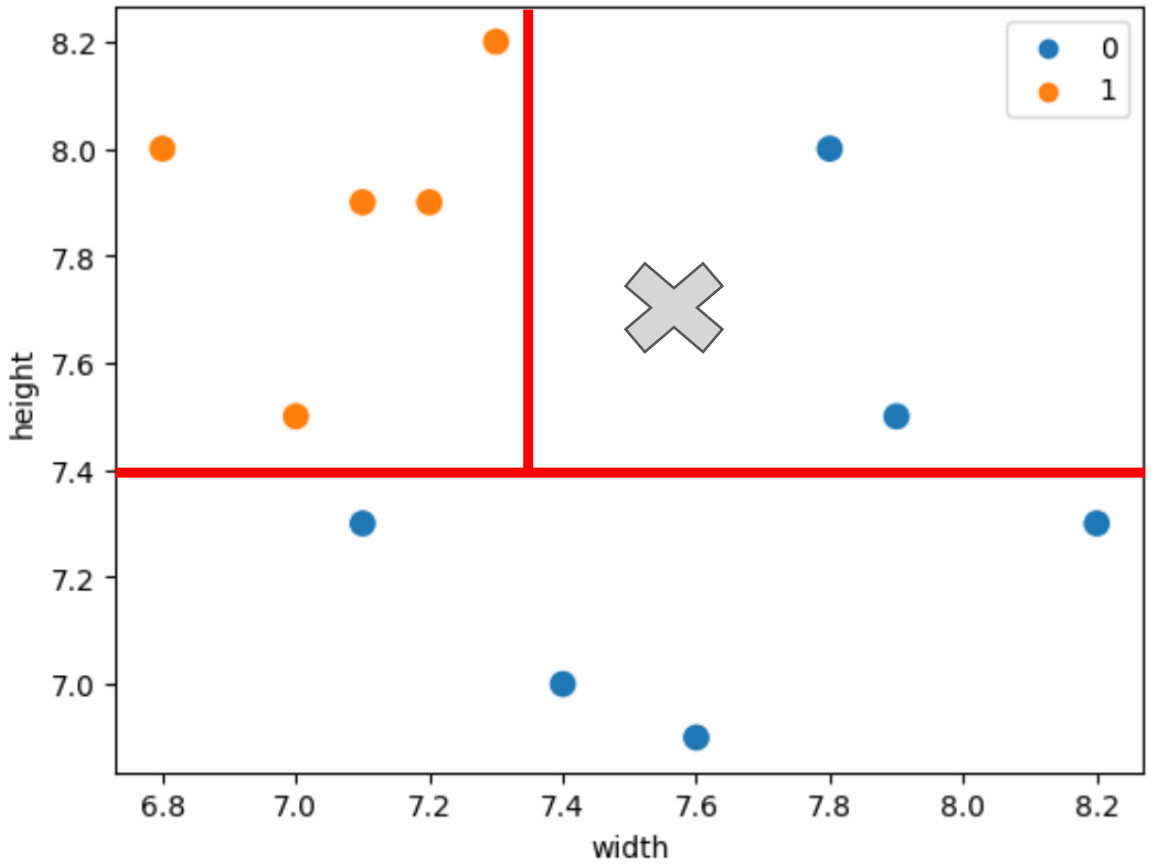
Decision Tree

- Bad Split



Decision Tree

- How can we predict class of new datapoints?



Decision Tree

VeeMG

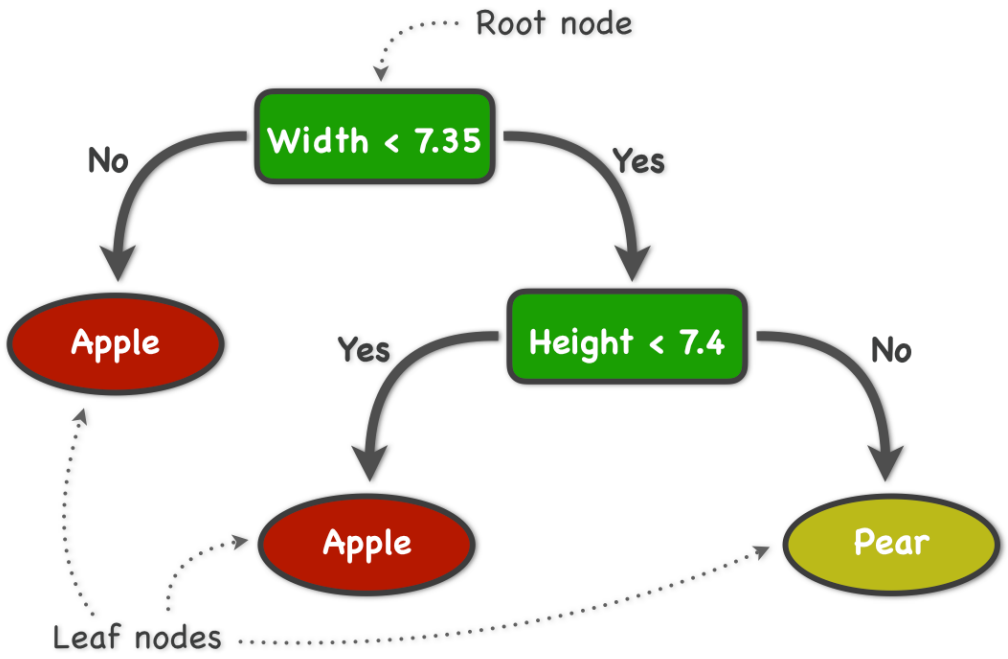
- Simple if else structure

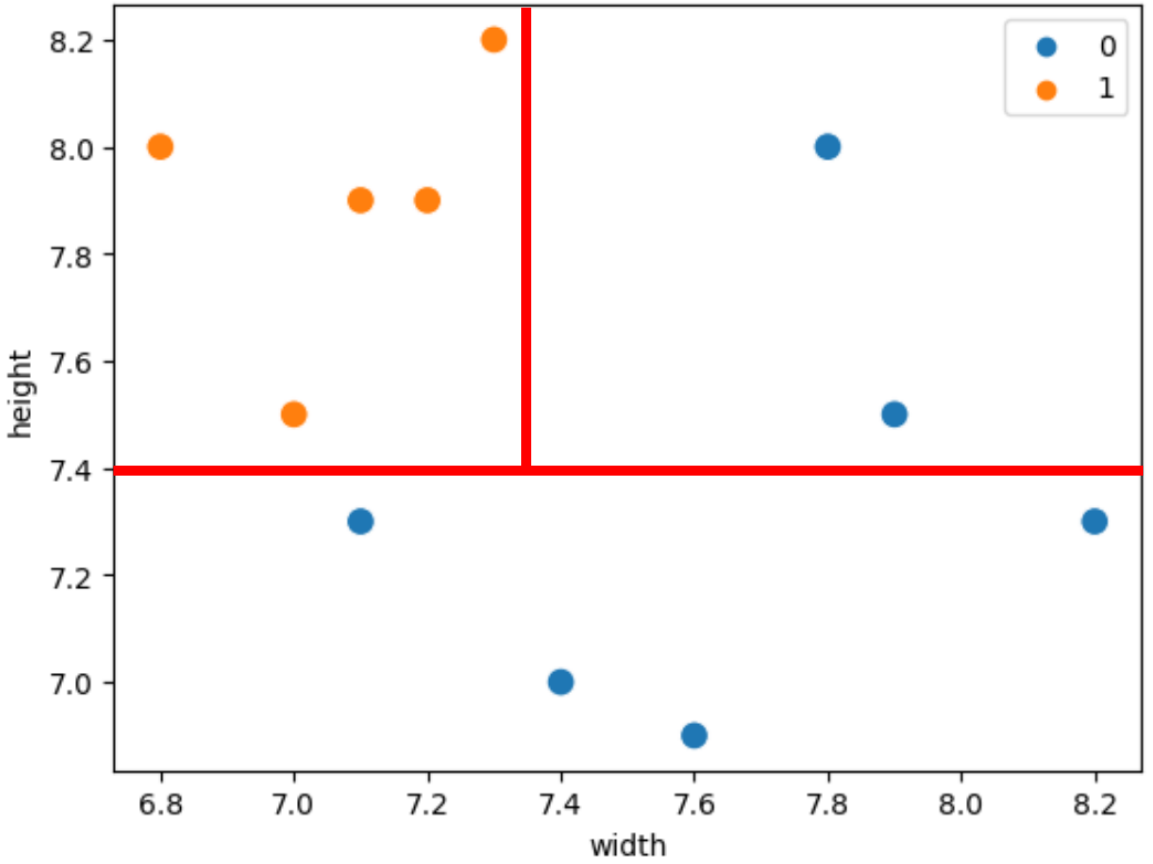
```
def get_fruit_class(width, height):
    if width < 7.35:
        if height < 7.4:
            return 'Apple'
        return 'Pear'
    return 'Apple'
```

Width	Height	Fruit
7.1	7.3	Apple
7.9	7.5	Apple
7.4	7.0	Apple
8.2	7.3	Apple
7.6	6.9	Apple
7.8	8.0	Apple
7.0	7.5	Pear
7.1	7.9	Pear
6.8	8.0	Pear
6.6	7.7	Pear
7.3	8.2	Pear
7.2	7.9	Pear

This is a tree!

- Allows reasoning





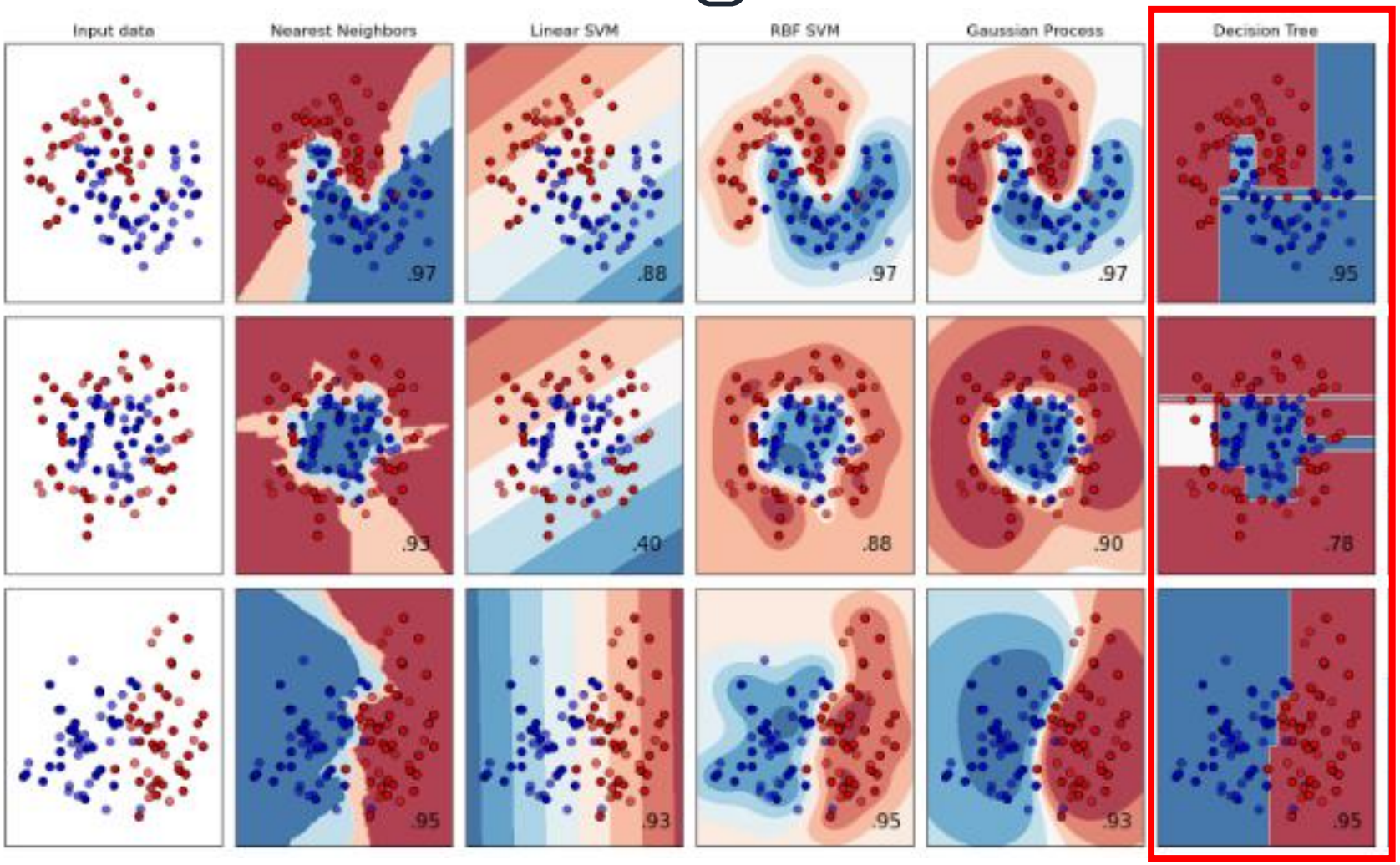
What do we have so far

- **Algorithm** Decision Tree
- **Training Data** applied on Algorithm gives us a Model
  - *That parametrized instance of the Algorithm has wisdom of training data*
- A **Model** we can use for predictions
  - *For a new unseen datapoint infer the class*

How good is our model?

- Score with (unseen) Test Data
  - Test with data we know the correct answer for
  - BUT the model hasn’t been trained with
- Accuracy = % of correct predictions
- Not as simple as that
  - False positives vs False negatives → Business impact

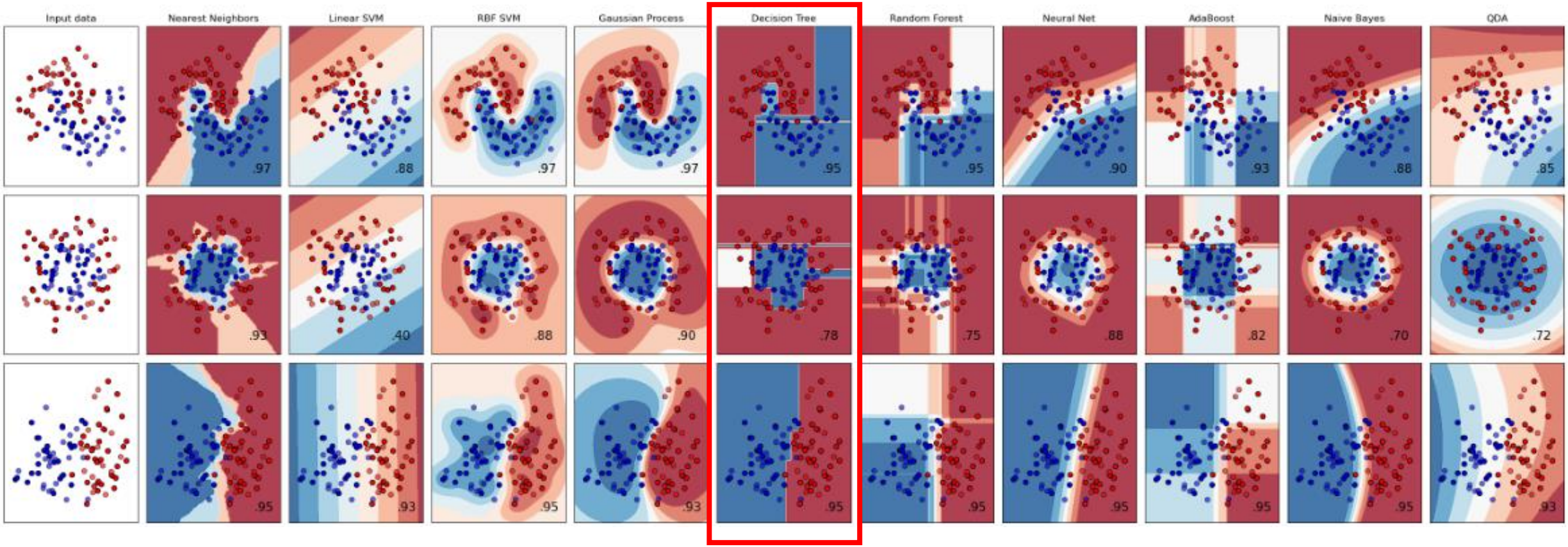
Classical ML Algorithms



VeeMG

adnexo

Classical ML Algorithms

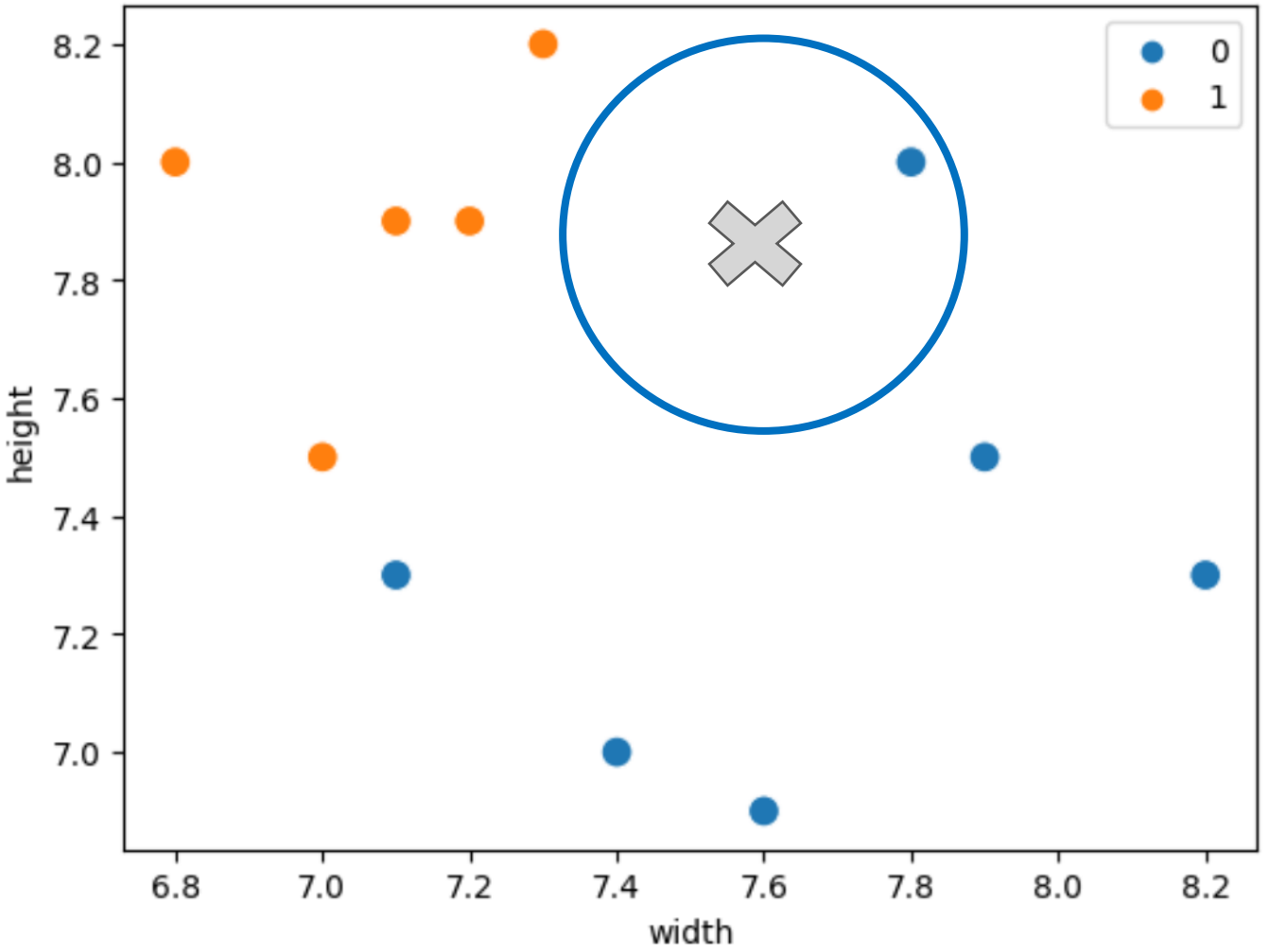


- https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

KNN

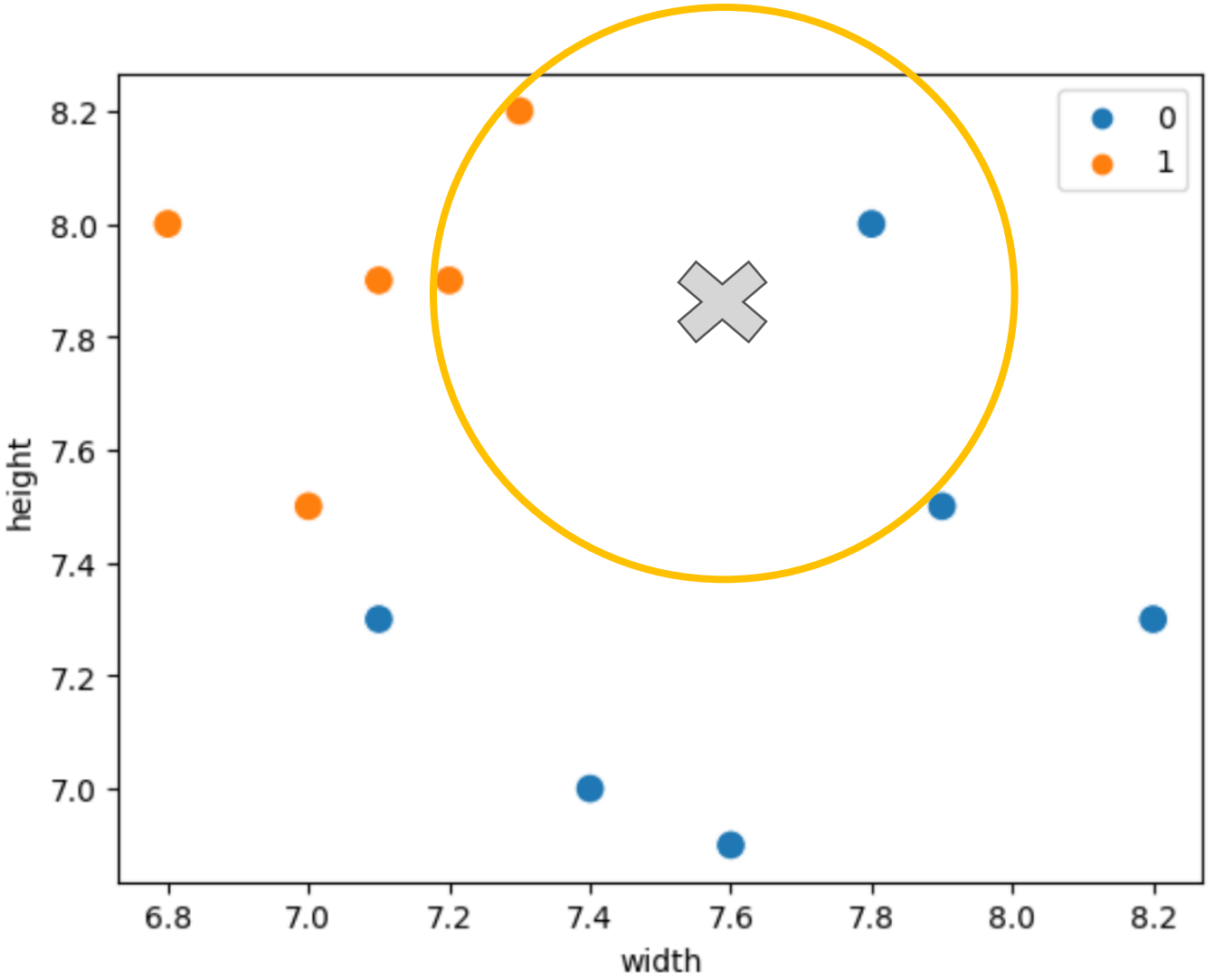
- K Nearest Neighbours
- Look which class the N neighbors have

- K=1 Its an Apple
- K is **Hyperparameter**



KNN

- K Nearest Neighbours
  - K=3 Its a Pear
- K = Hyperparameter
- We need to find best



Test / Train Split DEMO

- Code für splitting

Demo

- Today it's like building with LEGO
  - Yeah... almost
- Tooling
  - Jupyter Notebooks
  - Scikit-Learn

LAB

- Setup Jupyter Notebooks
  - https://jupyter.org/install
- OR use Google Colab
  - https://colab.research.google.com/
- Download Notebook and let it run
- Some simple tasks are in the Notebook

MLOps

- Deployment
  - How do we **deploy** and what do we **deploy**?
  - What are the moving parts?
- Lifecycle
  - How does it evolve?
- Integration
  - How to integrate in our Software Solutions
- Monitoring
  - Model Performance

Deployment

- Trained Models can be serialized

```
from joblib import dump, load
# Serialize
dump(model, 'trained-model.joblib')
# Deserialize
loaded_model = load('trained-model.joblib')
output = loaded_model.predict([[50, 50, 100]])
```

Create an API

- FastAPI with guvicorn

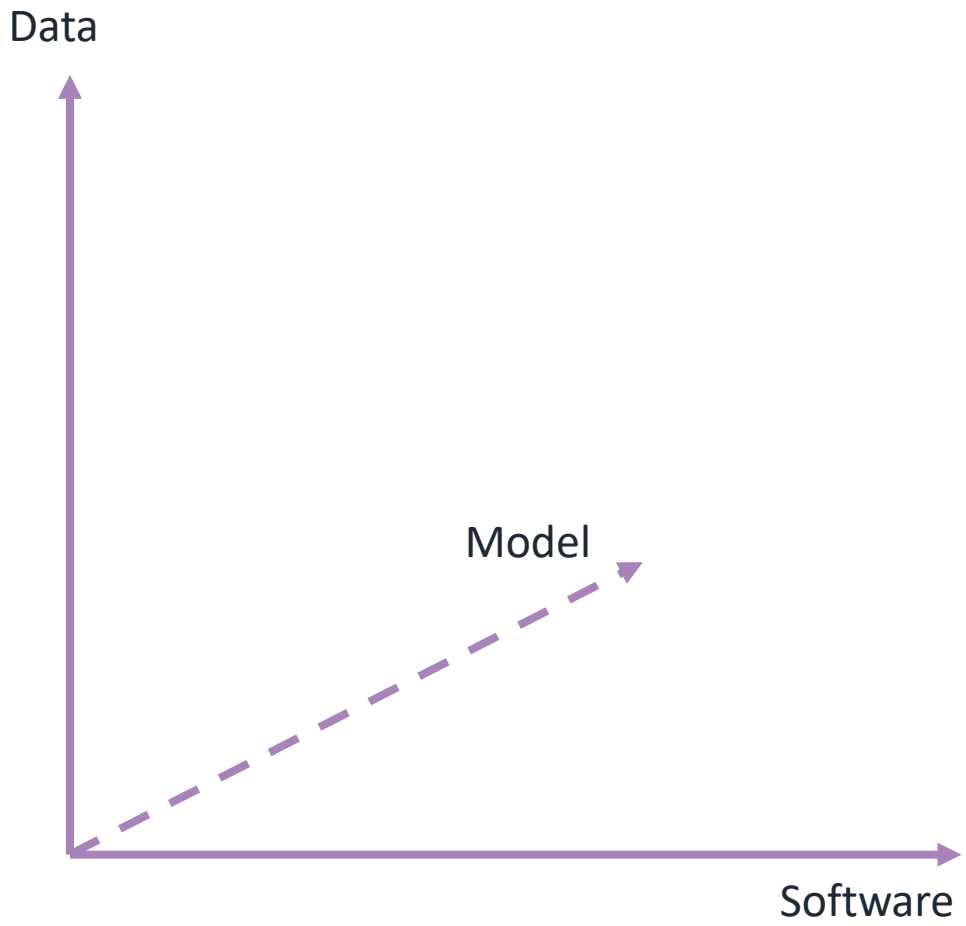
```python
from fastapi import FastAPI
from joblib import load
from sklearn.tree import DecisionTreeClassifier

app = FastAPI()
loaded_model = load('./trained-apple-pears.joblib')


@app.get("/predict/")
def read_item(width: float, height: float):
    output = loaded_model.predict([[width, height]])
    if output[0] == 0:
        return "Apple"
    else:
        return "Pear"
```

What do we need to track

- Code Base of our Software
  - Source Code
  - Dependencies
- Data
  - RAW Data
  - Preprocessed Data
- Models
  - Architecture / trained Parameters
  - Hyperparameters
  - Experiments



VeeMG

adnexo

Experiment Tracking

- Keeping track of what we have tried

- Systematic procedure of discovery and validation of a model
  - Training
  - Evaluation
  - Changing hyperparameters and observe behavirour

- Measure «quality and performance»
- Reproducability

DEMO

- How does experimenting look like

Useful Tools

- Weights and Biases https://wandb.ai/site
- MLFlow https://mlflow.org