

Grundlagen des maschinellen Lernens

Lineare Verfahren

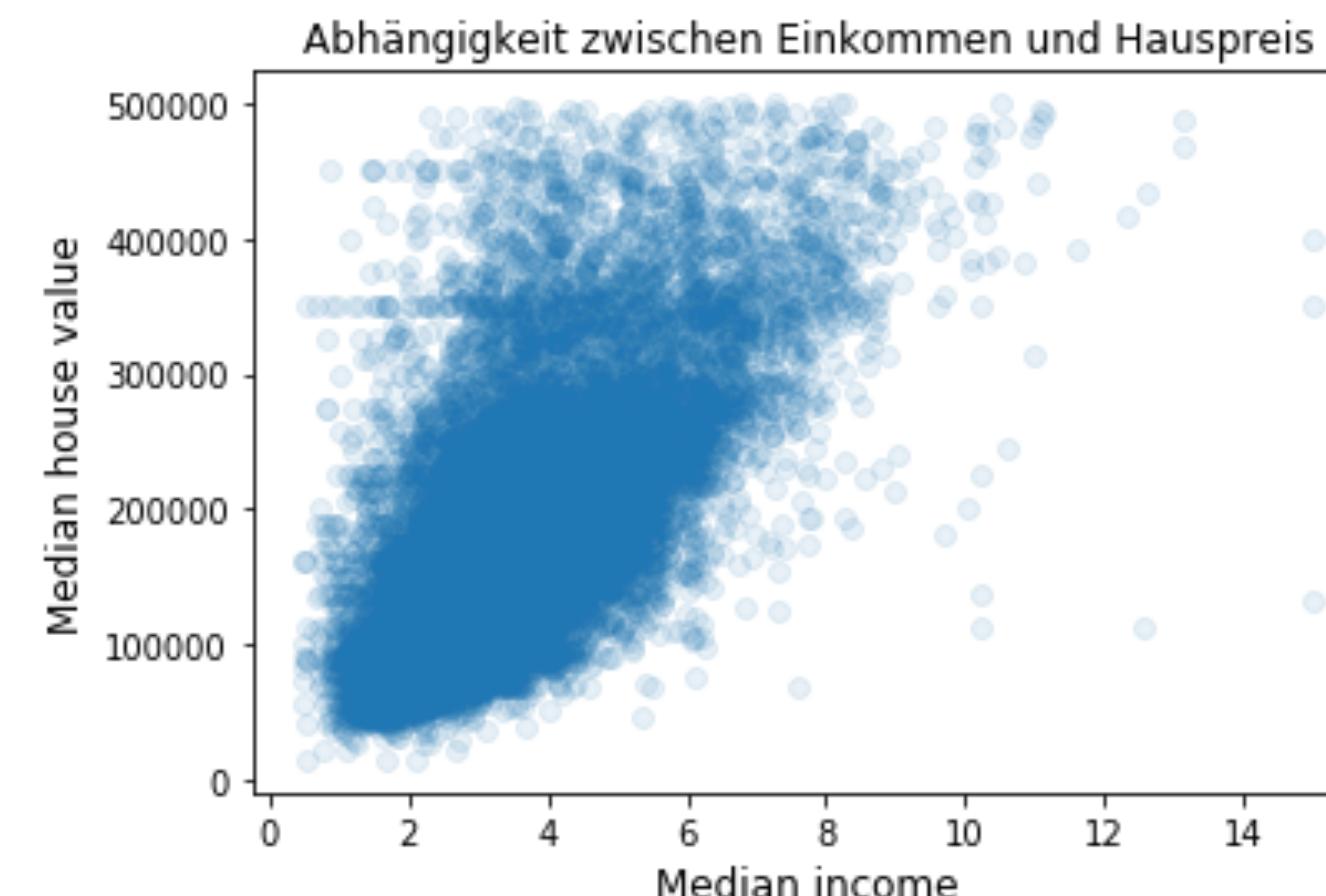
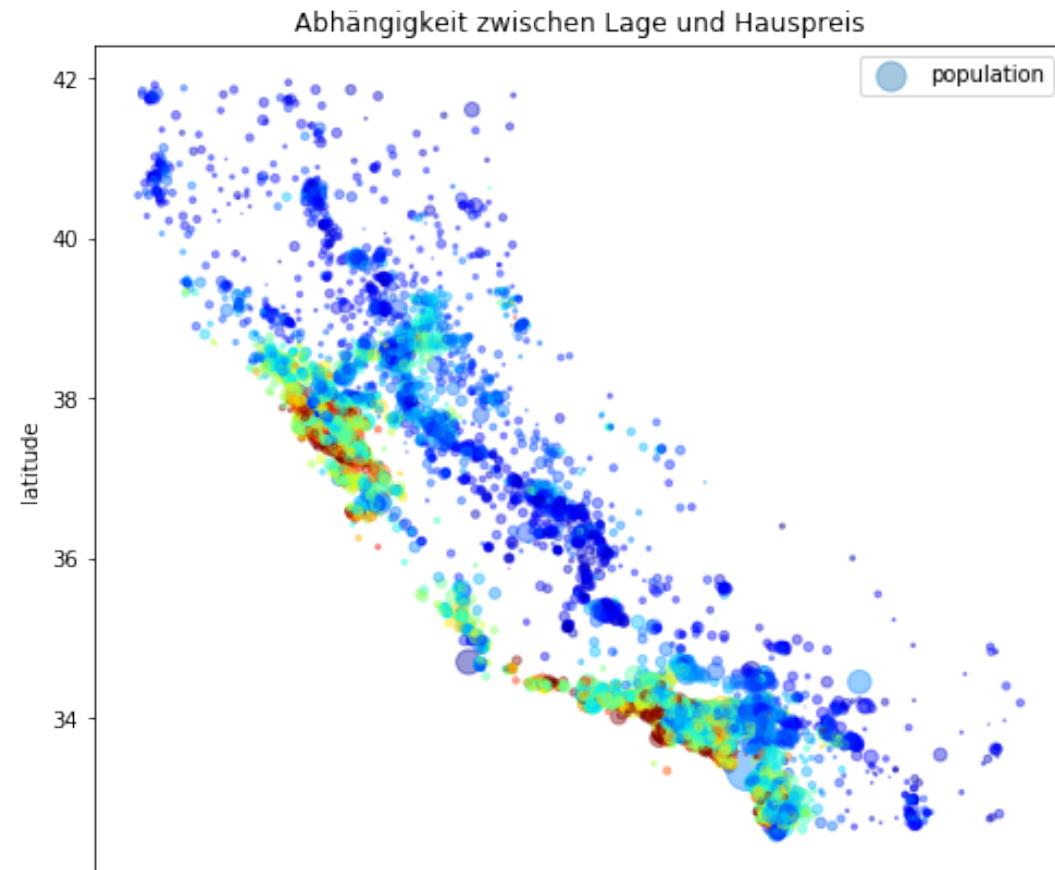
Volker Gruhn



1. Lineare Regression
2. Klassifikation (Logistische Regression)
3. Support Vector Machines

Lineare Regression

- Beispiel: Vorhersage von Hauspreisen
- Vorgehen: Betrachtung verschiedener Merkmale einer Immobilie, um eine Prognose für den Preis abzugeben



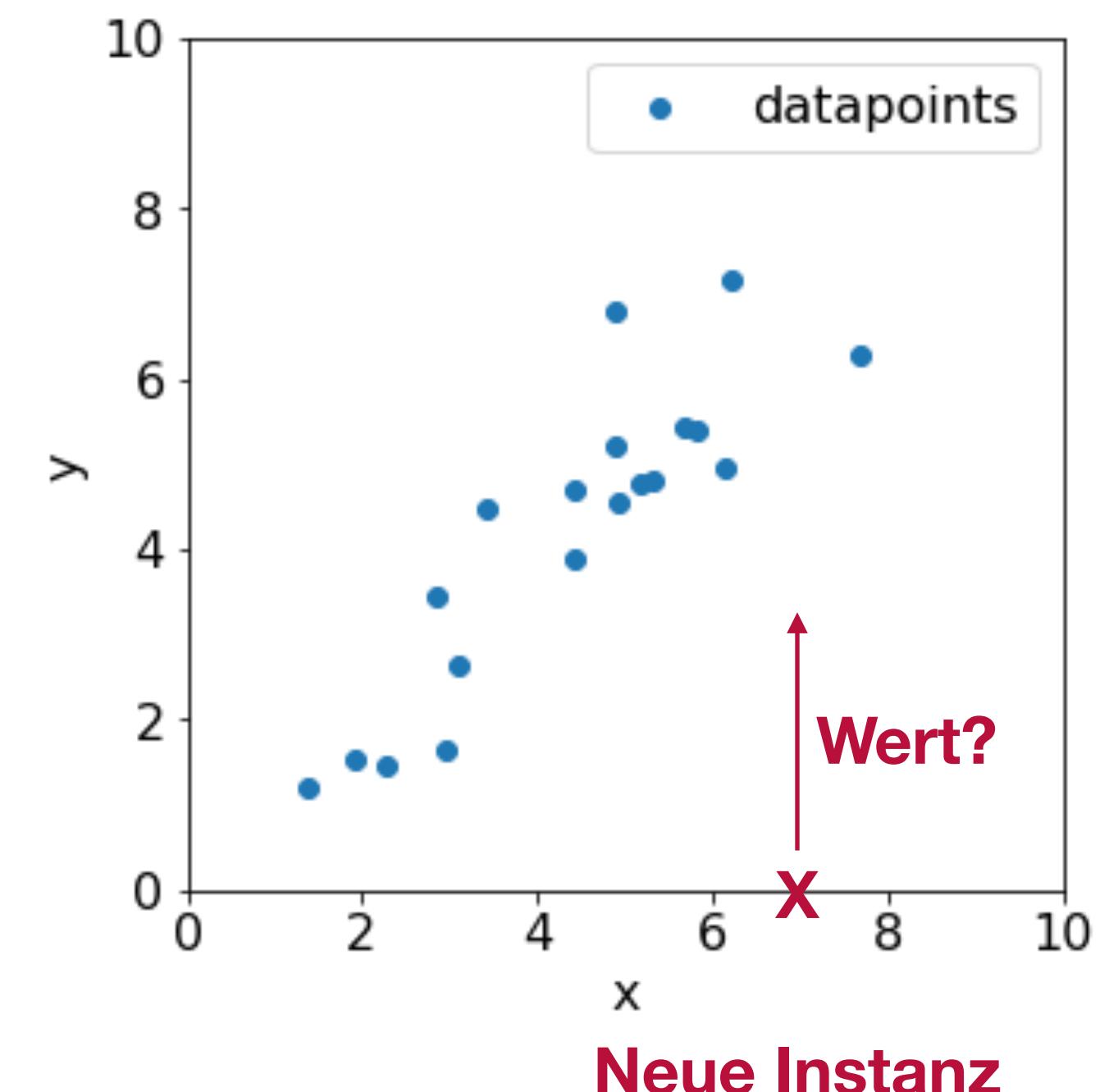
- Anmerkung: Zwischen betrachteten Merkmalen und dem Preis muss kein **kausaler** Zusammenhang bestehen, trotzdem kann das Merkmal zur Vorhersage des Preises geeignet sein, wenn es fachlich(!) Sinn macht.

- Menschen mit hohem Einkommen wohnen häufig in teuren Häusern, aber deshalb ist das Haus nicht teuer.
→ kein kausaler Zusammenhang
- Häuser, die in einer bestimmten Gegend liegen, sind häufig teuer, eben weil sie genau dort stehen.
→ kausaler Zusammenhang
- Häuser, in denen es spukt, sind billiger.
→ vermutlich Unsinn

- Merkmale zur Bewertung eines Hauses könnten zum Beispiel sein:
 - die Lage (Geo-Koordinaten)
 - Größe (in m²)
 - Einfamilienhaus: Anzahl der Schlafzimmer
 - Mehrfamilienhaus: Anzahl der Wohnungen
 - das durchschnittliche Einkommen der Bewohner
 - ...
- Wie kann ein Algorithmus einen Zusammenhang zwischen diesen Merkmalen lernen, um daraus auf den Preis zu schließen?
- Wie können wir feststellen, wie genau unsere Vorhersage ist?
- Lösungsansatz: Lineare Regression

Definition

- Regression beschreibt die Bestimmung einer abhängigen Variablen (auch *Regressor* oder *Kriterium*) durch mehrere unabhängige Variablen (auch *Regressanden* oder *Prädiktoren*)
 - *Lineare Regression* bedeutet, dass zwischen unabhängigen und abhängigen Variablen ein linearer Zusammenhang besteht: Unabhängige und abhängige Variable verändern sich in einem festen Verhältnis zueinander
 - Die abhängige Variable muss kardinalskaliert sein
 - Unabhängige Variablen können nominell oder höher skaliert sein
- Im Kontext des Machine Learning nennen wir die abhängige Variable *Label* und die unabhängigen Variablen *Feature*

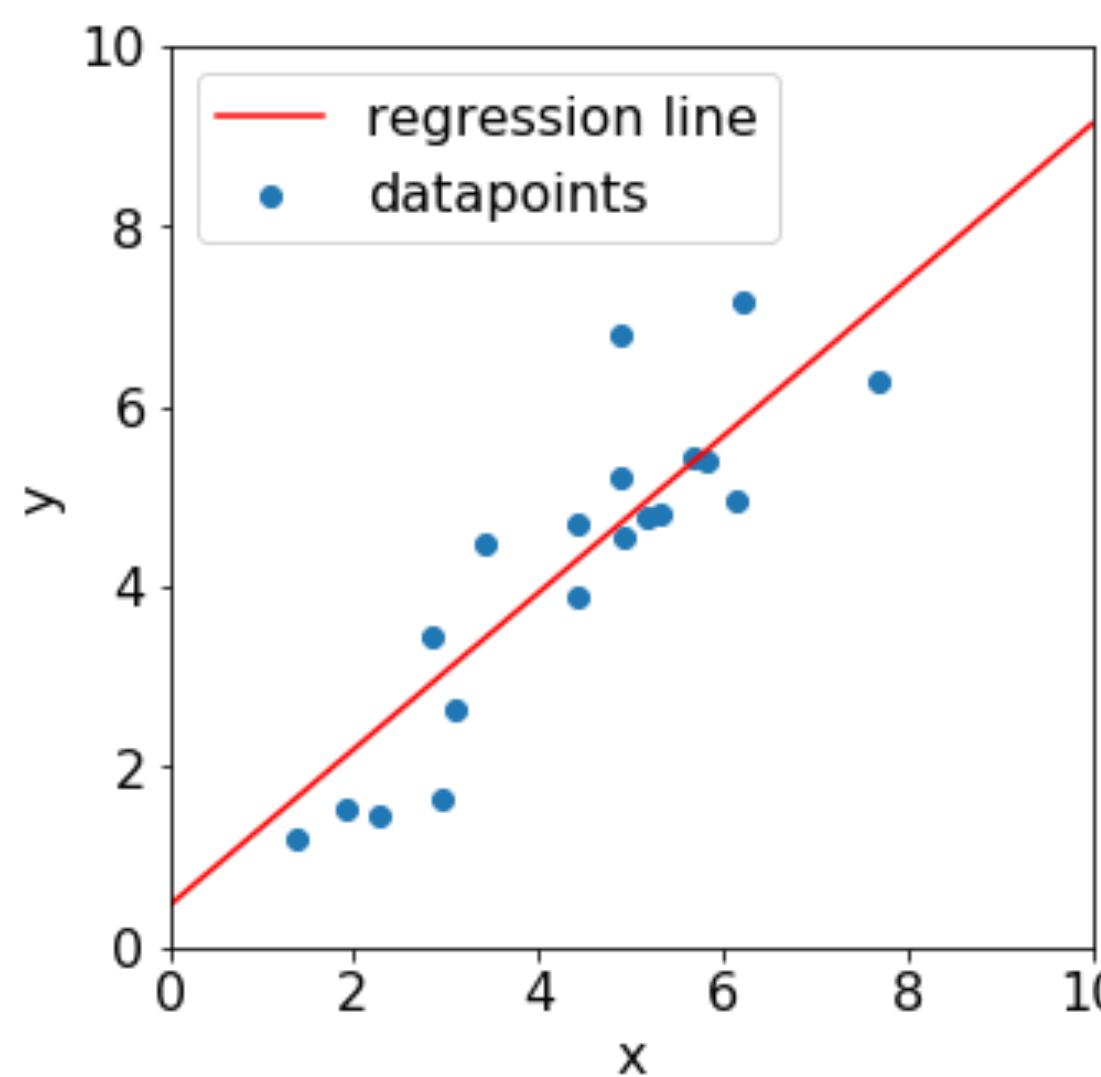


Linear

- Geradengleichung:
 $f(x) = \omega x + b$, mit $b, x, \omega \in \mathbb{R}$
- Unabhängige und abhängige Variable verändern sich in einem festen Verhältnis zueinander
(Steigung ω der Geraden, zzgl. Schnittpunkt der y-Achse b (häufig *bias term*; deutsch y-Achsenabschnitt))

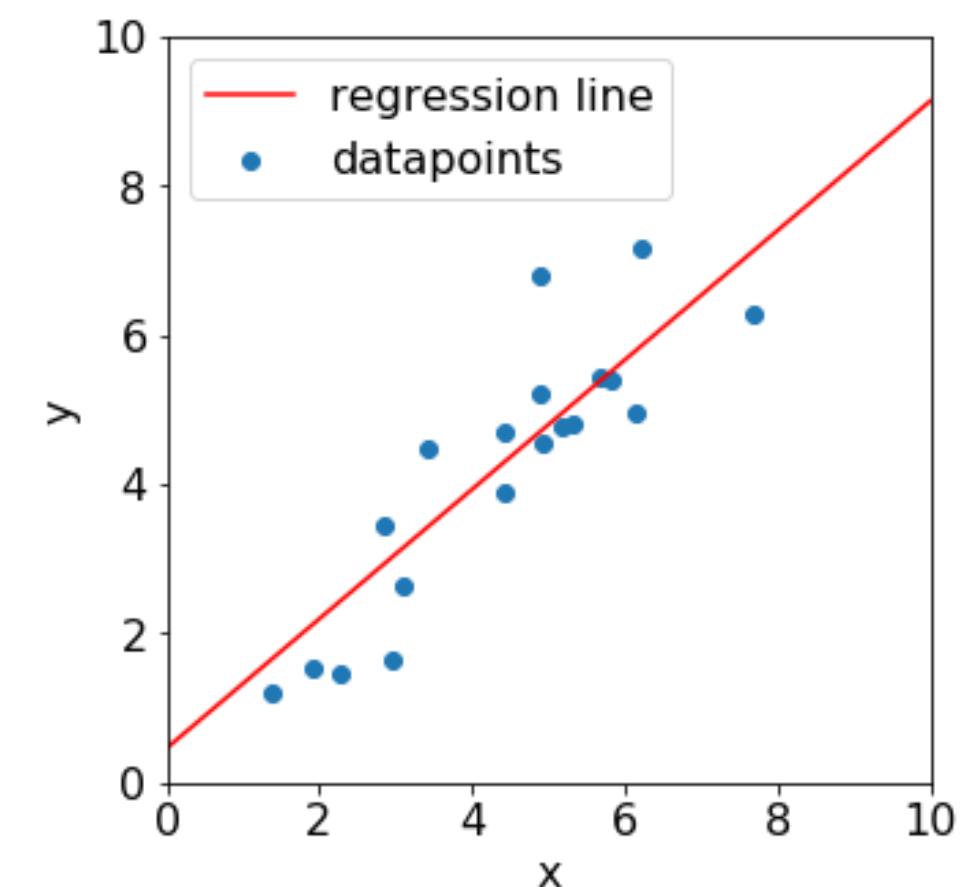
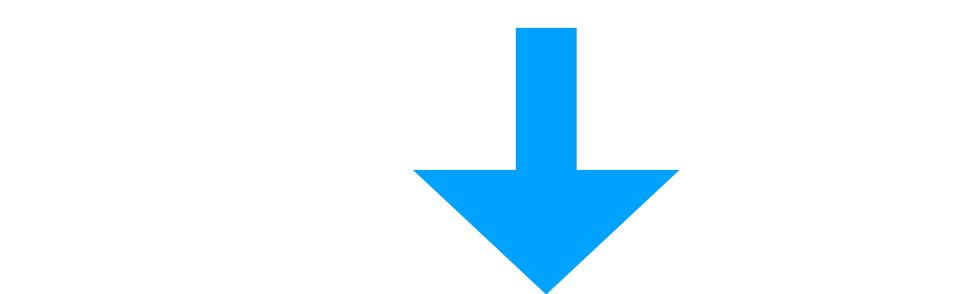
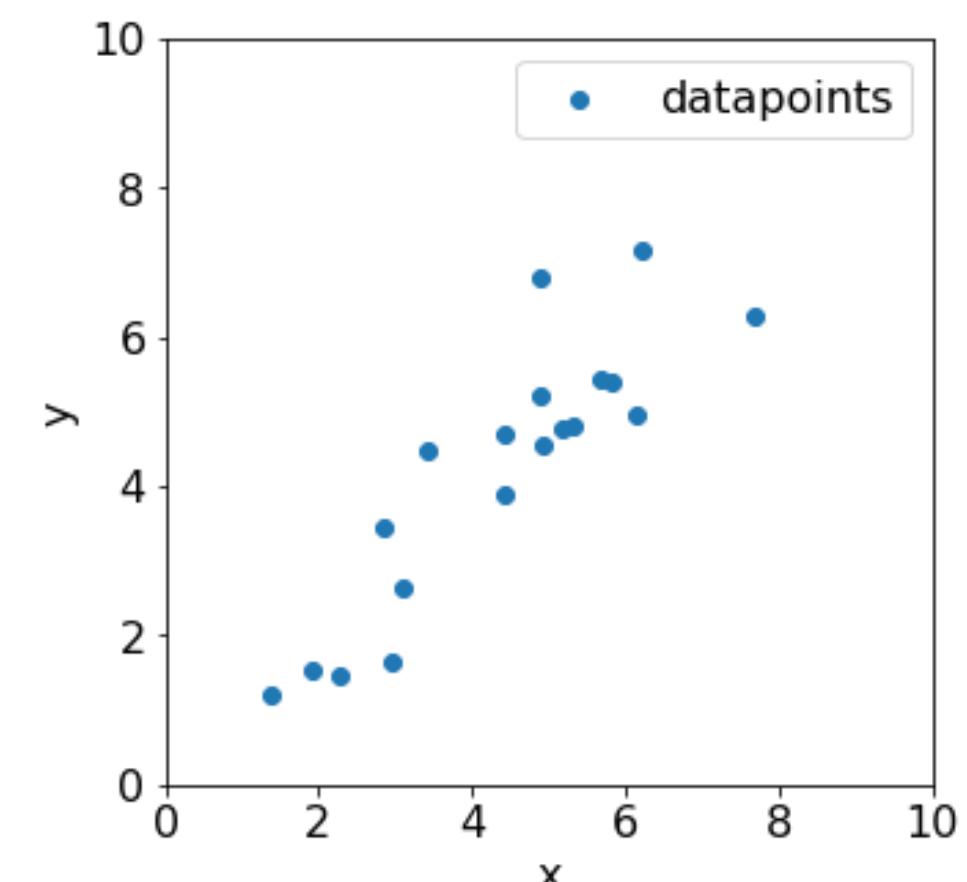
Nichtlinear

- Alle Funktionen, die sich nicht in der links genannten Form beschreiben lassen
- Streng genommen muss die Definition auf der linken Seite \mathbb{R}^{n-1} lauten. Alle Funktionen in \mathbb{R}^n sind nichtlinear.



Formale Definition

- $f(x) = y$, mit $x \in \mathbb{R}^n$, $y \in \mathbb{R}$, wobei x eine unabhängige Variable (Feature) und y eine abhängige Variable (Label) ist.
- Unter Berücksichtigung der Linearität bleibt:
$$y = \omega x + b, \text{ mit } b, x, \omega \in \mathbb{R}$$
- Wir suchen die Regressionsgerade, die mit o.g. Formel umgesetzt wird, um das unbekannte Label y aus einem bekannten Feature x abzuleiten
- Was hast das mit Machine Learning zu tun? Ein Algorithmus soll die optimalen Parameter ω und b mithilfe unserer Trainingsdaten lernen und die bestmögliche Regressionsgerade bestimmen!
- Hinweis: Wir fassen im Folgenden ω und b zusammen: $\theta = \begin{pmatrix} b \\ \omega \end{pmatrix}$

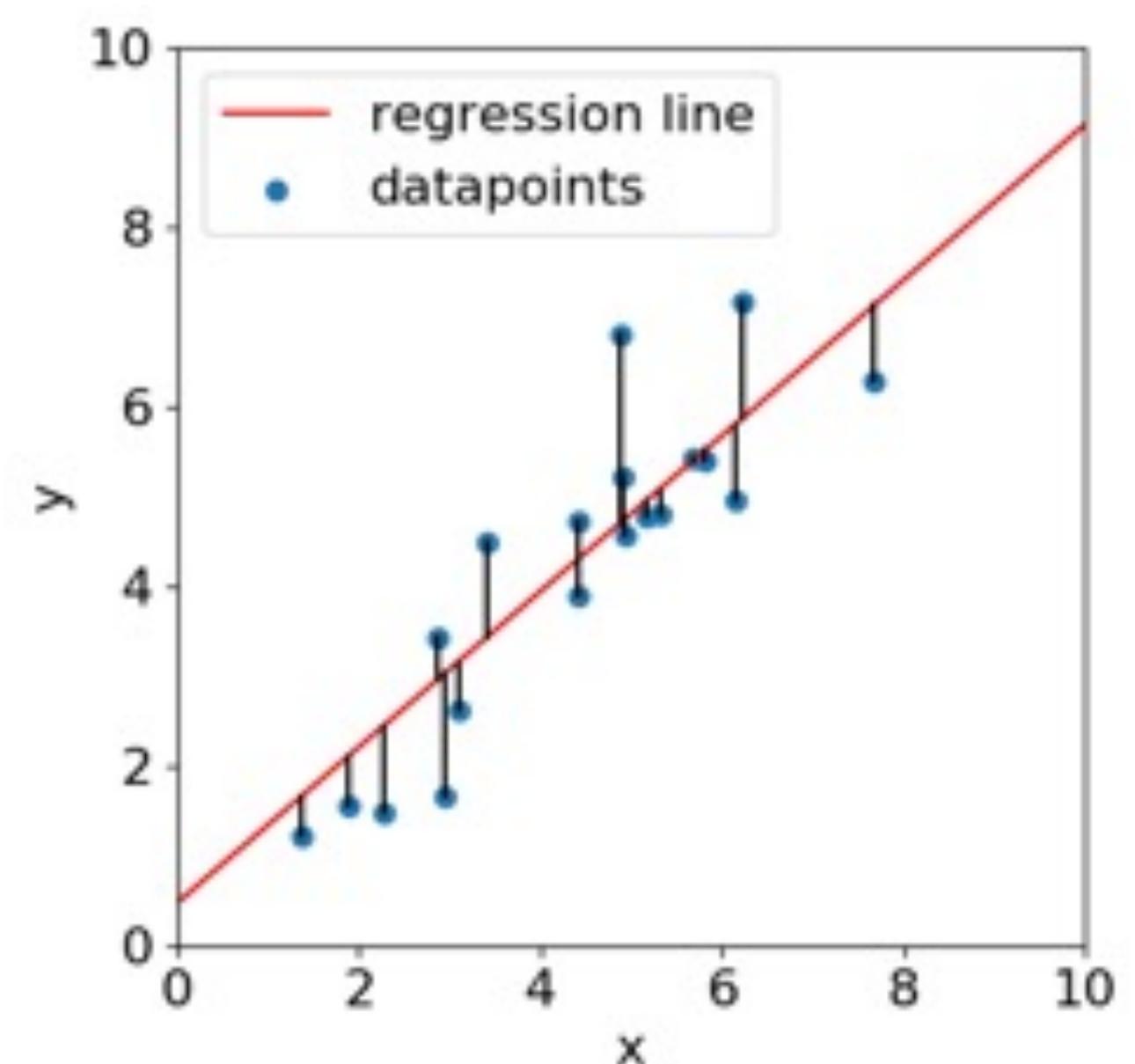


Loss Function

- Um die optimale Regressionsgerade zu finden, müssen wir zunächst eine *Loss Function* (Bewertungsfunktion, Kostenfunktion) aufstellen, die die Güte der möglichen Regressionsgeraden bewertet
- Wir nutzen dafür den *Mean Squared Error (MSE)*, den Durchschnitt der quadrierten Abstände zwischen allen Punkten der Trainingsdaten und der Regressionsgeraden:

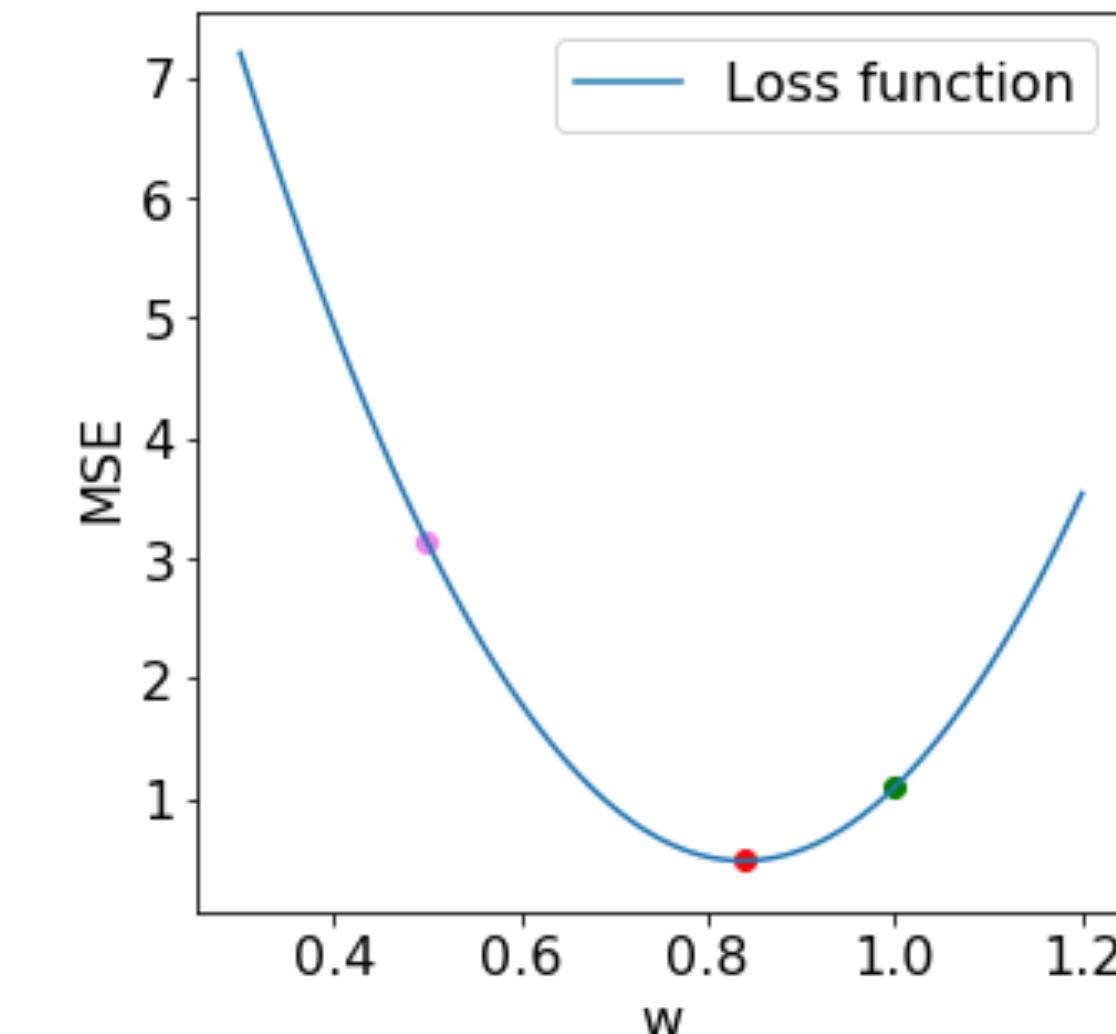
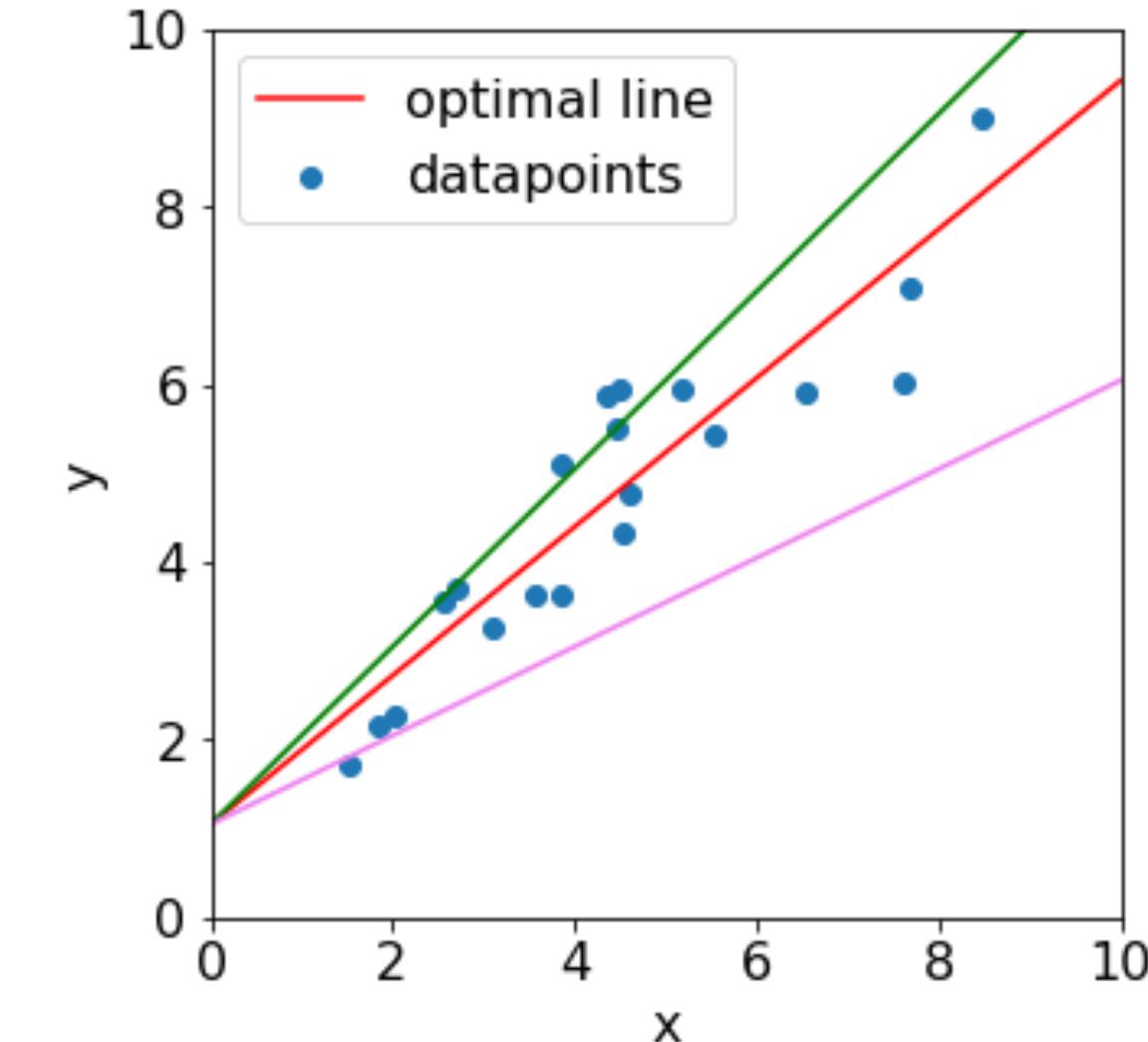
$$MSE = \frac{\sum_{i=1}^m (f(x_i) - y_i)^2}{m}, \text{ mit } m = \text{Anzahl der Punkte}$$

- Alternativen sind z. B. *Mean Absolute Error (MAE)* oder *Root Mean Square Error (RMSE)*
- Wie finden wir die optimalen Parameter für unser Modell?
 - Die optimale Regressionsgerade minimiert unsere Loss Function, d. h. der MSE ist möglichst niedrig
 - Unser Machine-Learning-Algorithmus passt θ so an, dass unsere Loss Function ein Minimum erreicht



Loss Function

- Unsere Kostenfunktion MSE ist strikt konvex: Es existiert genau ein Minimum mit der optimalen Belegung von θ
- Jede andere Belegung von θ führt zu höheren Kosten und somit zu einer schlechteren Regressionsgeraden
- Eine Funktion f hat an der Stelle x ein Minimum, wenn gilt
 - $f'(x) = 0$
 - $f''(x) > 0$ (Hinweis: hier nicht nötig, weil Funktion streng konvex ist)



Mehrere Features

- Aber: x ist in 2D nur ein Feature. Was machen wir, wenn wir mehrere Features zur Definition des Modells verwenden wollen?
- Zur Erinnerung: Unsere Datenbasis zur Ermittlung von Hauspreisen enthält viele Informationen und wir glauben, dass ein Feature alleine keine guten Ergebnisse liefern wird.
- Unser Modell muss eine lineare Regression über einen Vektor lernen

ID	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
...										
20635	-121,09	39,48	25,0	1665,0	374,0	845,0	330,0	1,5603	78100,0	INLAND
20636	-121,21	39,49	18,0	697,0	150,0	356,0	114,0	2,5568	77100,0	INLAND
20637	-121,22	39,43	17,0	2254,0	485,0	1007,0	433,0	1,7000	92300,0	NEAR OCEAN
20638	-121,32	39,43	18,0	1860,0	409,0	741,0	349,0	1,8672	84700,0	<1H OCEAN
20639	-121,24	39,37	16,0	2785,0	616,0	1387,0	530,0	2,3886	89400,0	INLAND
...										

- Das Modell lernt die gewichtete Summe (zzgl. des *Bias Term*) über alle für die Vorhersage verwendeten Features:

$$f(x) = y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- Dabei ist
 - y das vorhergesagte Label
 - n die Anzahl der verwendeten Features
 - x ist der Vektor aller Features, wobei x_j das j -te Feature ist (Hinweis: $x_0 = 1$)
 - θ ist der Vektor der Modellparameter, wobei θ_j der j -te Modellparameter ist (mit θ_0 als Bias Term und $\theta_1, \theta_2, \dots, \theta_n$ der Gewichtung der einzelnen Features)

Mehrere Features: Alternative Notation

- Verwendung von Vektoren mit mehreren Features erlaubt kürzere Schreibweise als Skalarprodukt:

$$y = h_{\theta}(x) = \theta \bullet x$$

- Eine oft verwendete Alternative ist die Multiplikation mit einem transponierten Vektor für die Parameter:

$$y = \theta^T x$$

- Dabei ist

- y das vorhergesagte Label
- θ ein Spaltenvektor mit allen Parametern, inkl. des Bias Term θ_0 und den Gewichtungen der einzelnen Features $(\theta_1, \theta_2, \dots, \theta_n)$
- θ^T der entsprechend transponierte Zeilenvektor
- x ist der Spaltenvektor aller Features der zu prüfenden Instanz (die Eingabedaten), wobei $x_0 = 1$ gilt und x_1, x_2, \dots, x_n die Daten enthalten

- Zur Erinnerung: Multiplikation von Vektoren

- Sei $a = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, dann gilt $a^T = (1, 2, 3)$

- Sei $b = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$, dann gilt

$$a^T \cdot b = (1, 2, 3) \cdot \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} = (1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6) = (32)$$

- Übertragen auf unsere Definition:

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}, x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \text{ dann gilt}$$

$\theta \bullet x = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$, was unserer Formel auf der vorigen Folie entspricht (bitte berücksichtigen, dass weiterhin $x_0 = 1$ gilt!)

- Unsere Loss Function muss entsprechend angepasst werden, um die nun verwendeten Vektoren zu enthalten und lautet dann

$$MSE(X, h_{\theta}) = \frac{\sum_{i=0}^m (\theta^T x^{(i)} - y_i)^2}{m}$$

- Dabei ist
 - $X = (x^{(1)}, \dots, x^{(m)})^T$ die Matrix aller Trainingsdaten (Samples)
 - $x^{(i)}$ der Vektor des i .ten Samples
 - Rest wie gehabt

Lösung: Normal Equation (Normalengleichung)

- Ideale Werte für θ , die die Loss Function minimieren, können bei Verwendung von MSE berechnet werden:

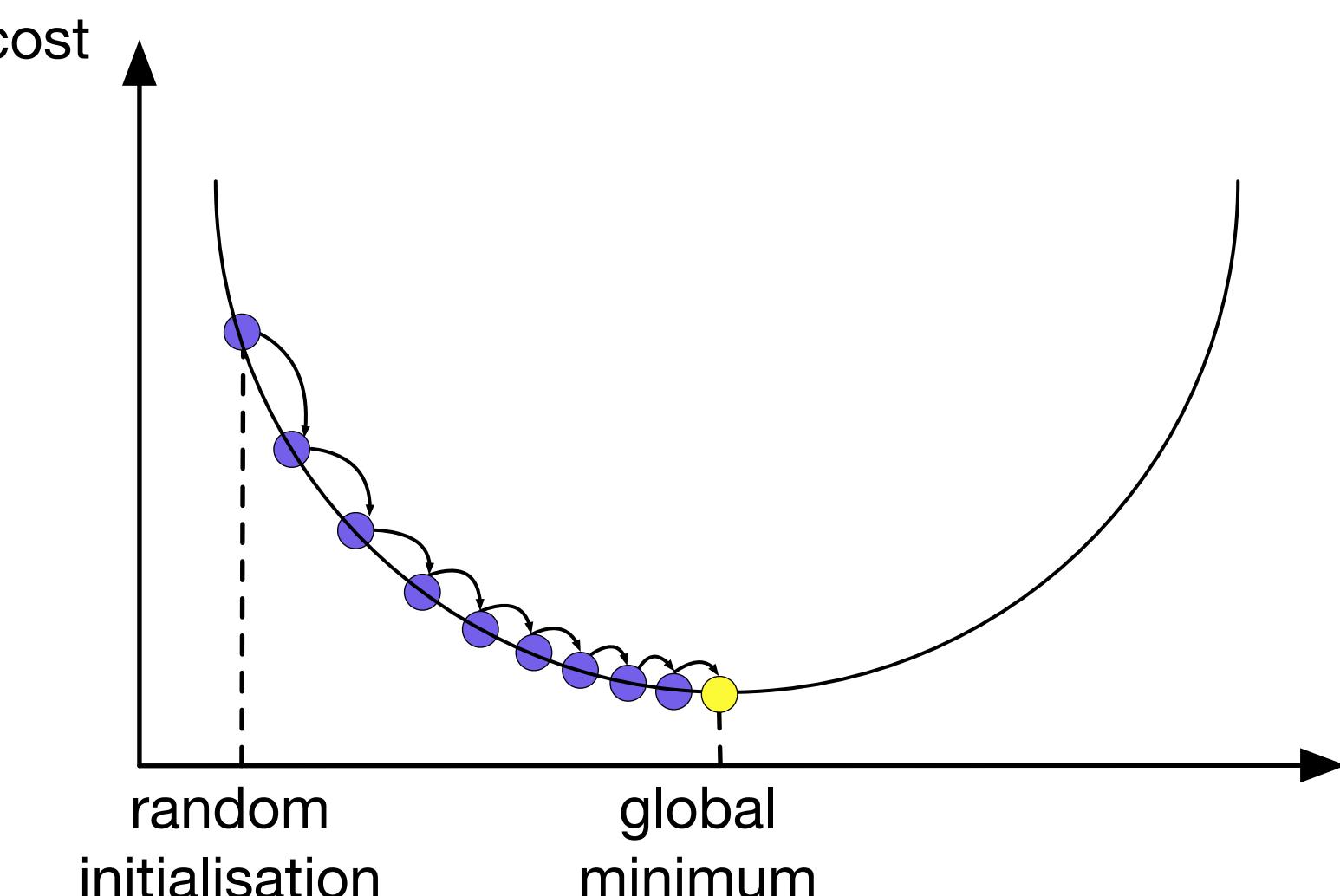
$$\theta = (X^T X)^{-1} X^T y$$

- Hinweis: Die Herleitung dieser Formel führt an dieser Stelle zu weit und ist im Anhang zu finden.
- Dabei ist
 - θ der optimale Parametervektor für die Kostenfunktion
 - X die Matrix aller Samples
 - y die vorhandenen Labels
- Damit haben wir die passende Regressionsgerade gefunden!

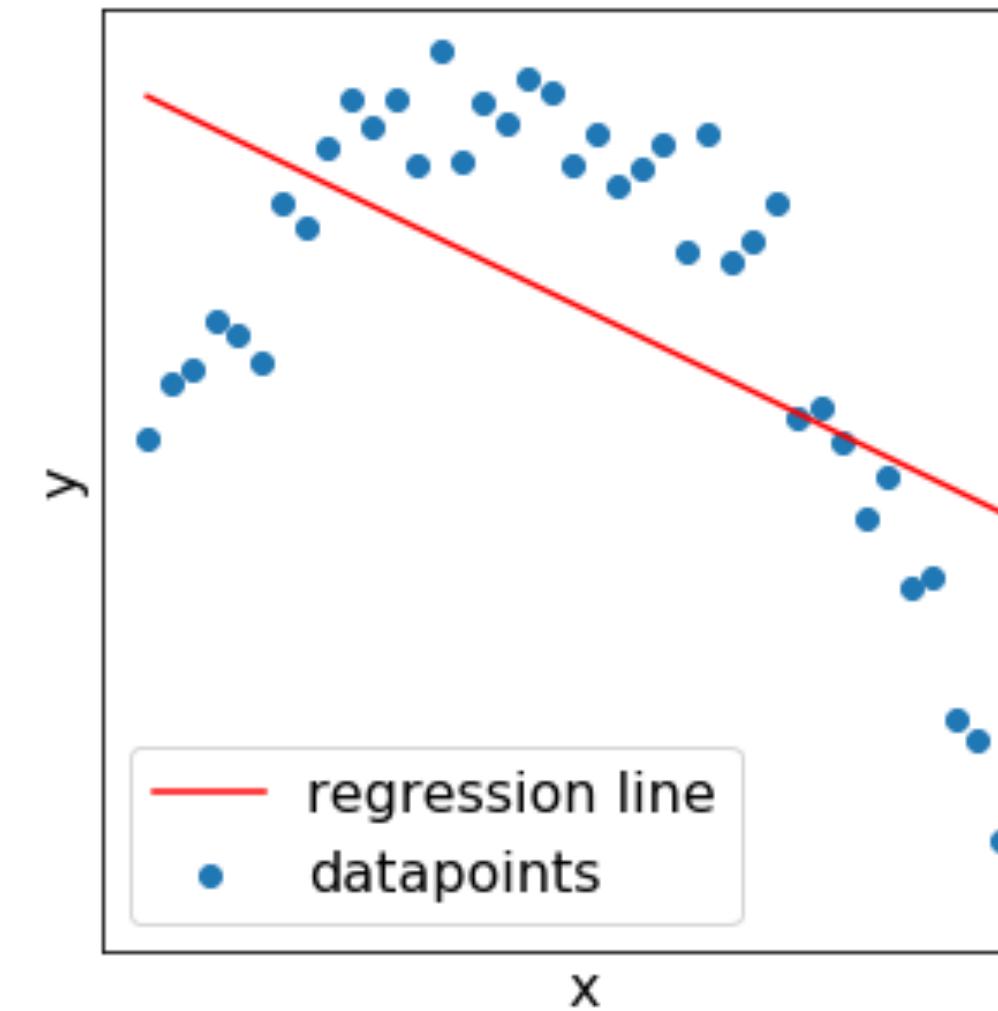
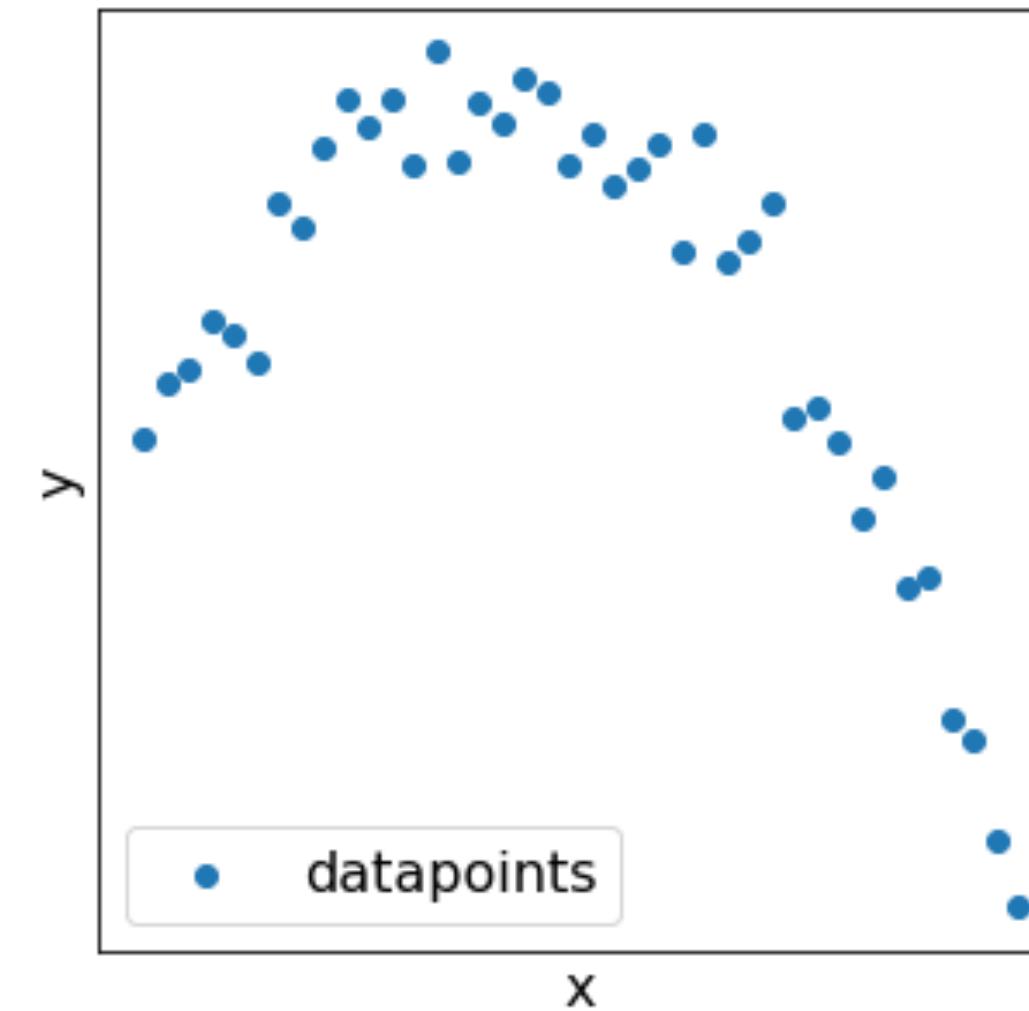
- Bei sehr vielen Samples wird die Berechnung der optimalen Lösung sehr aufwendig. Können wir die ideale Kostenfunktion approximieren? Ja!

- Ein mögliches Verfahren: *Gradient Descent* (Gradientenverfahren)

- Vereinfacht: Starte mit Zufallswert für θ und folge dem fallenden Gradienten bis zur nächsten Steigung
- Detailliert in VL *Neuronale Netze*



- Nicht alle Daten sind linear verteilt
- Eine einfache Regressionsgerade kann die Daten nicht ausreichend modellieren. Das Modell wird etwas lernen, aber die Ergebnisse werden voral. sehr ungenau und daher unbrauchbar sein.
- Was tun? Es gibt zwei Optionen:
 - Wir wählen ein anderes Modell, um die Verteilung der Daten besser abzubilden.
 - Wir erweitern unsere Daten so, dass sie linear abbildbar sind. Dafür nutzen wir den *Polynomtrick*.



Nichtlineare Daten: Polynomtrick

- Ein Polynom ist eine Linearkombination der Potenzen einer Variable:

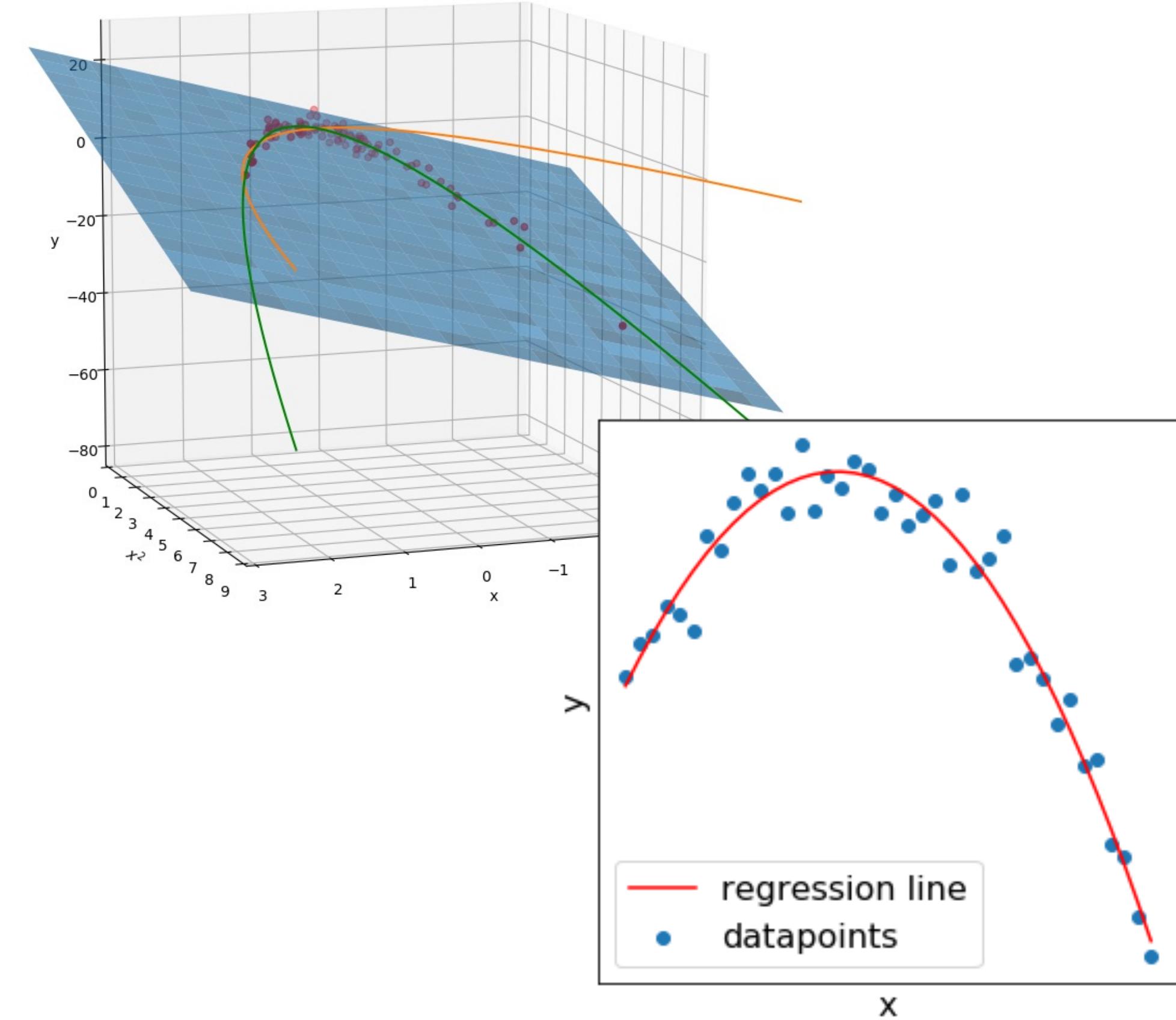
$$P(x) = \sum_{i=0}^n a_i x^i + a_1 x + a_2 x^2 + \dots + a_n x^n, n \geq 0$$

- Statt unserer ursprünglichen Funktion $f(x) = \omega x + b$ mit $b, x, \omega \in \mathbb{R}$ suchen wir eine Lösung für die Parameter des Polynoms $f(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ mit $\theta \in \mathbb{R}^3$

- Zur Erinnerung: Wir hatten $\theta = \begin{pmatrix} b \\ \omega \end{pmatrix}$ definiert

- Das bedeutet, dass wir zu dem Merkmal x sein Quadrat x^2 als ein neues Merkmal hinzufügen
- Da wir x aus dem Trainings-Set kennen, können wir x^2 einfach berechnen

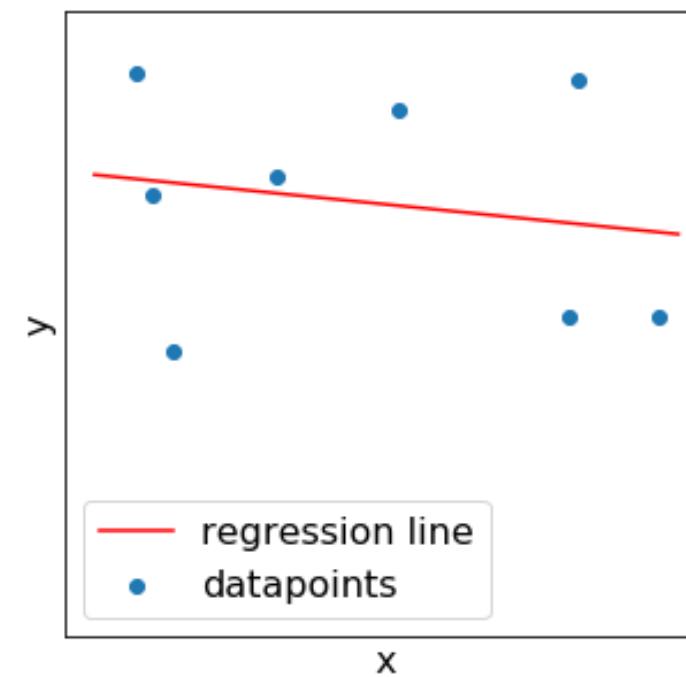
- Somit existiert weiterhin eine Gerade entlang jeder Achse und das Modell ist linear in Bezug auf seine Parameter θ



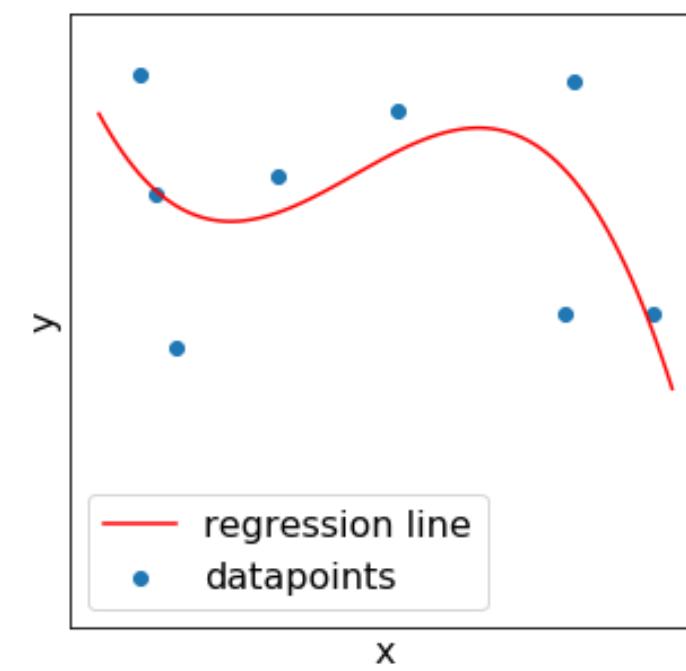
Nichtlineare Daten: Polynomtrick

- Was passiert, wenn wir die Koeffizienten eines immer komplexeren Polynoms lernen, um die Verteilung der Daten noch zu treffen?
- Das Polynom von Grad 9 trifft alle Punkte im Trainings-Set und erreicht einen Fehler von 0 in der Kostenfunktion!
- Ist das ein gutes Modell?

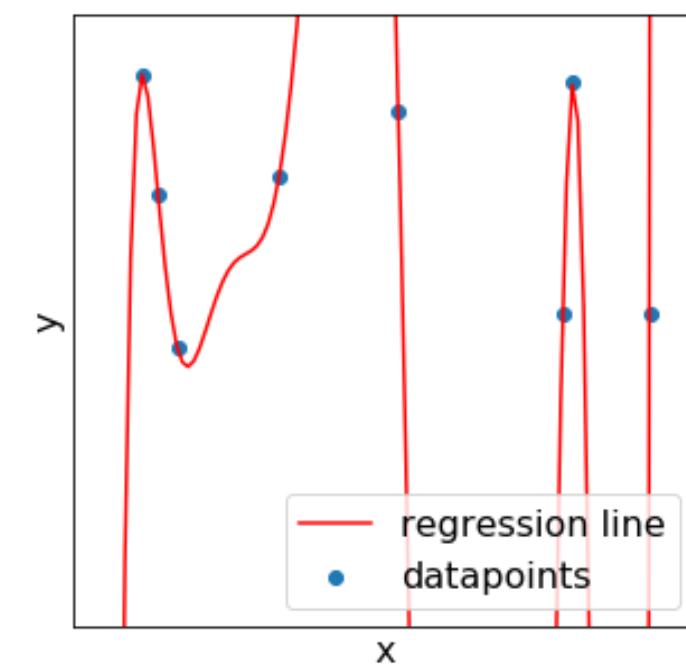
- Polynom Grad 1



- Polynom Grad 3



- Polynom Grad 9

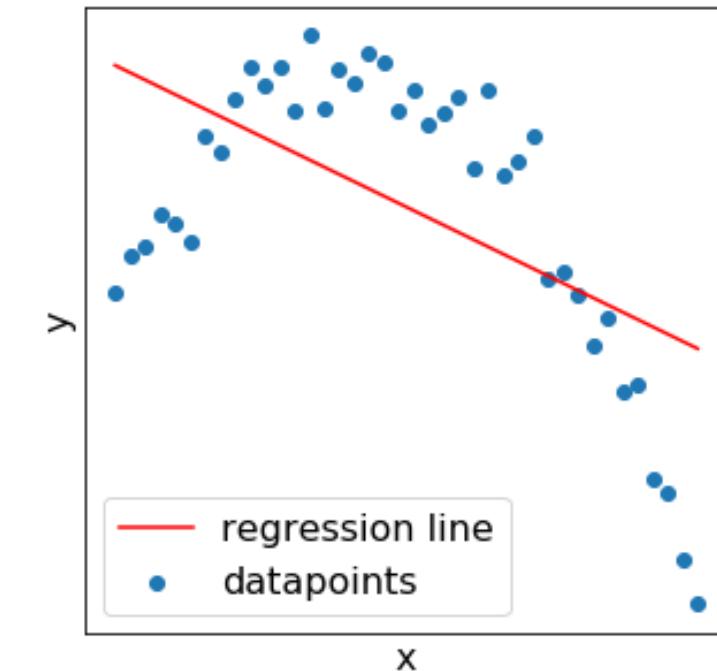


Underfitting und Overfitting

- ***Underfitting***

- Die Komplexität des gewählten Modells reicht nicht aus, um die Verteilung der Daten ausreichend genau zu lernen
- Das Modell kann unbekannte Daten nicht genau gut bewerten

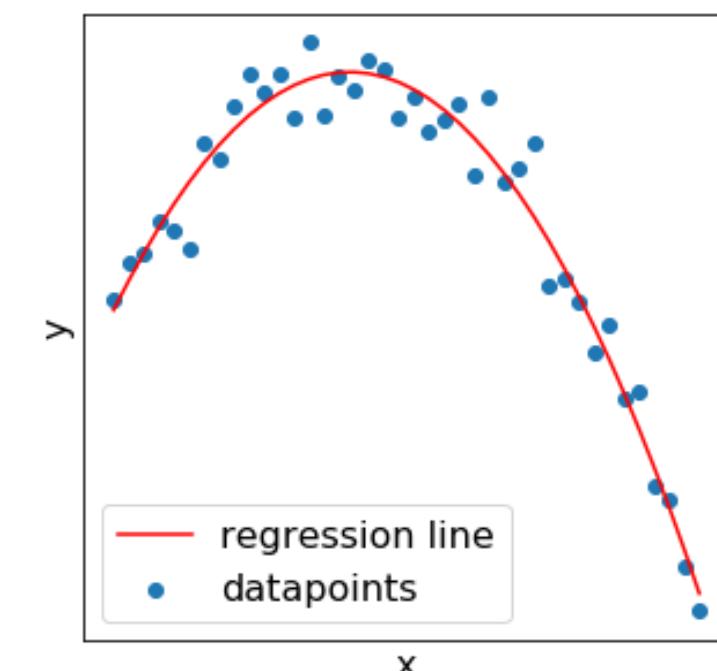
- Polynom Grad 1



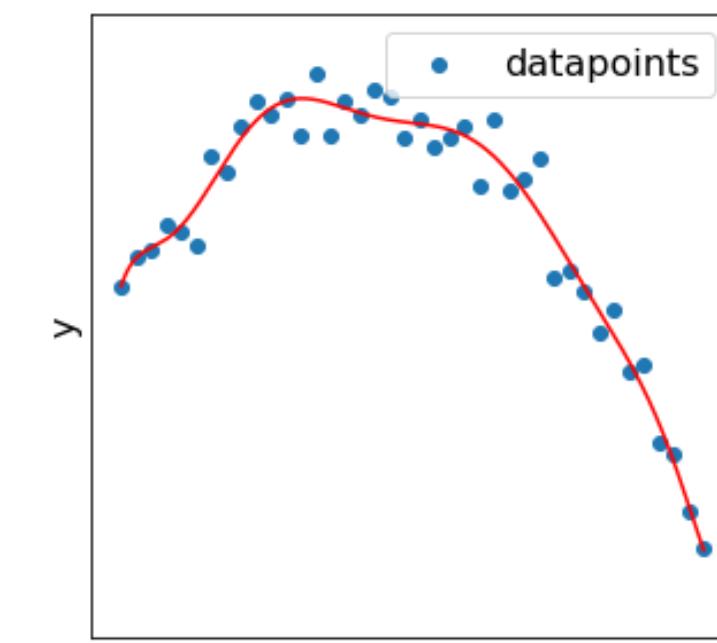
- ***Overfitting***

- Die Modellkomplexität ist so hoch, dass das Trainings-Set auswendig gelernt wird
- Neben der Verteilung der Daten wird das Rauschen mitgelernt
- Das Modell kann nicht generalisieren und unbekannte Daten gut bewerten
- Macht sich durch extrem kleine Werte der Loss Function für das Trainings-Set bemerkbar

- Polynom Grad 2



- Polynom Grad 9

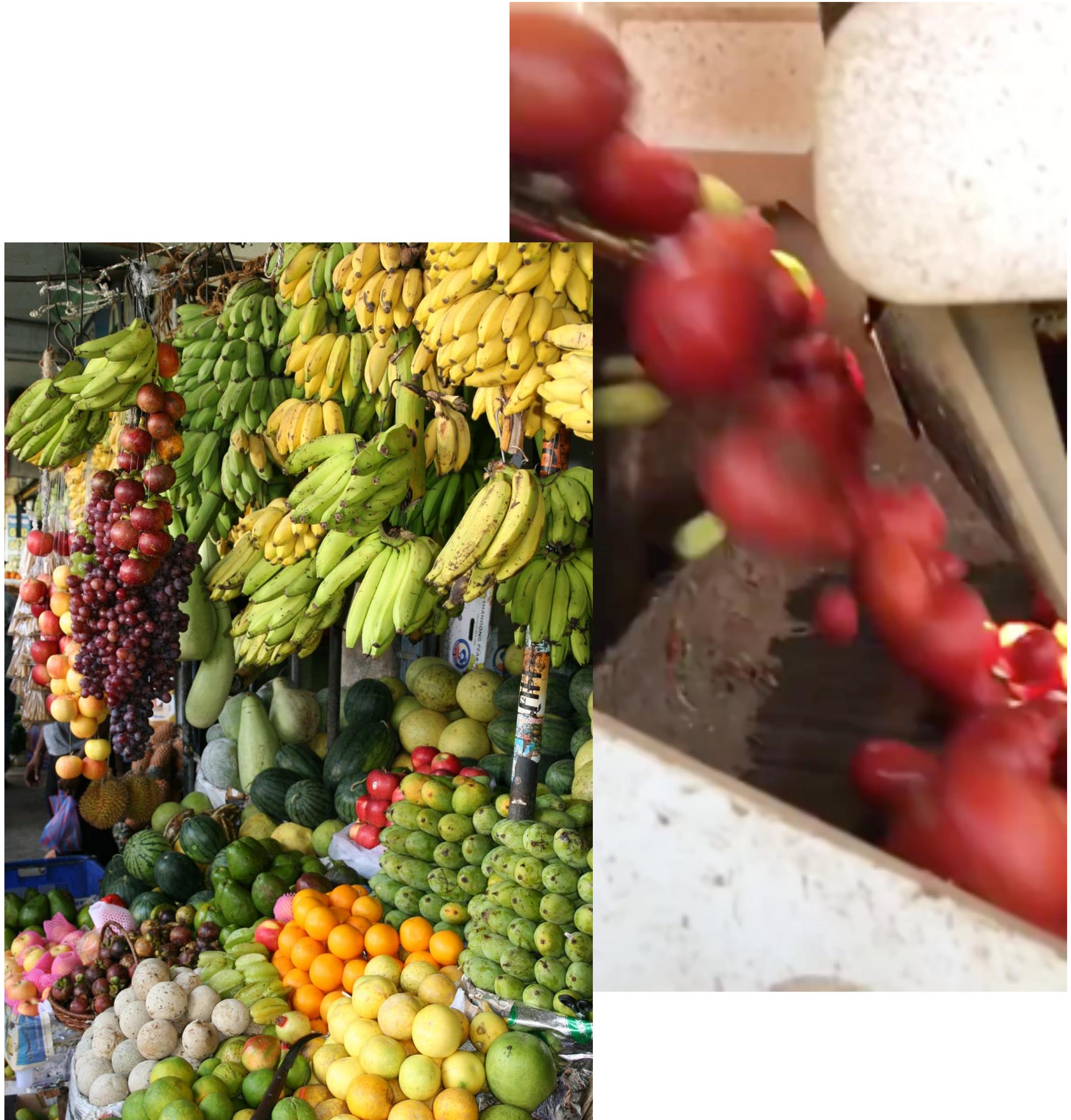


Klassifikation

mit Logistischer Regression

Beispiel

- Szenario: Wir möchten mithilfe einer Maschine nach der Ernte unreife Früchte aussortieren
- Die Maschine erhält verschiedene Informationen über eine Frucht und bewertet diese
 - Durchmesser
 - Gewicht
 - Farbe
 - ...
- Können wir so ein Problem mithilfe der linearen Regression lösen? Welche Schwierigkeiten treten dabei auf?



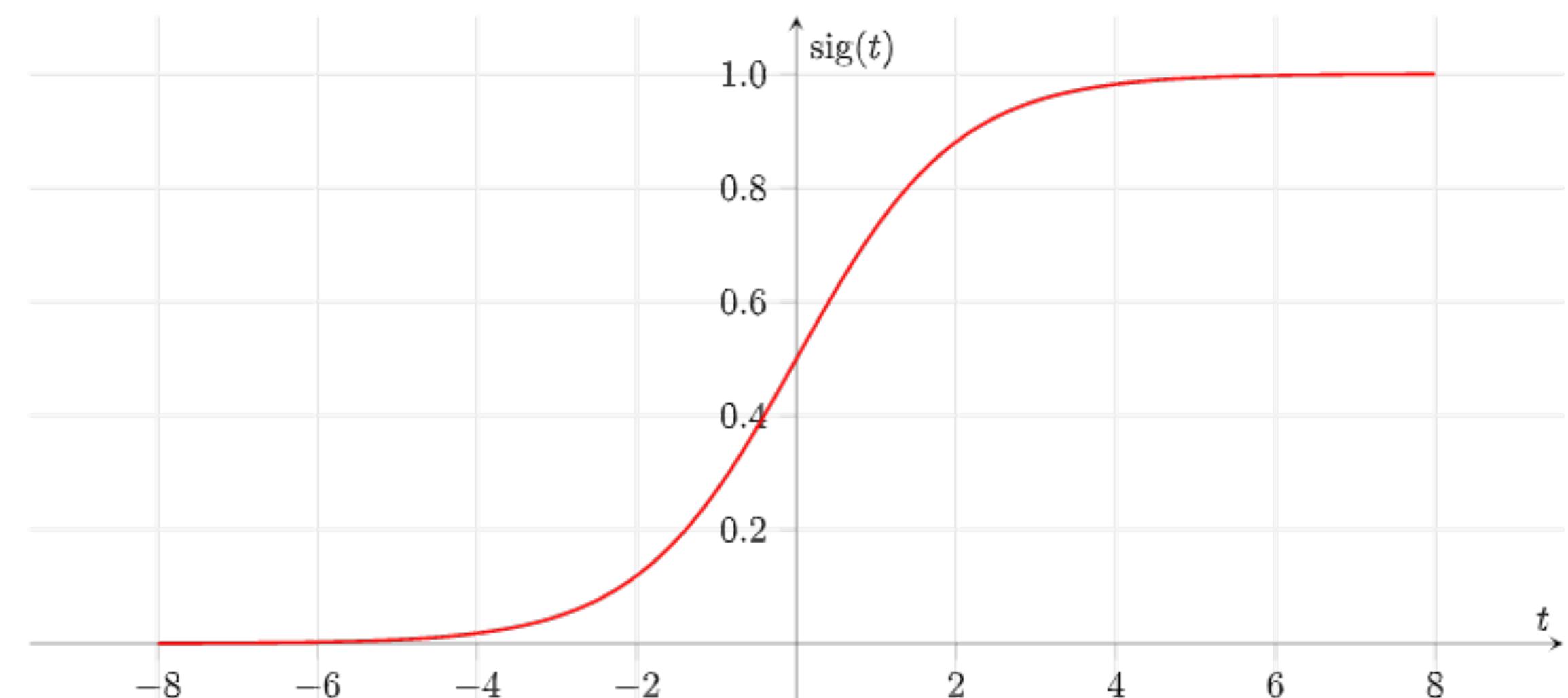
- Wir könnten eine Regressionsgerade durch unsere Daten legen, um die Werte 1 (reif) und 0 (unreif) vorherzusagen
- Dabei können uns Probleme begegnen
 - Was bedeutet 0,42?
 - Was bedeuteten Werte < 0 und > 1 ?
 - Wie viel wahrscheinlicher ist eine Frucht mit den Werten 2/100/-3 reifer als eine Frucht mit den Werten 4/80/7?
- Wir brauchen ein Modell, das eine Wahrscheinlichkeit p_{reif} für den Reifegrad einer Frucht ermittelt!
 - Erinnerung: Wahrscheinlichkeiten werden im Intervall $[0,1]$ definiert
 - Einen Wahrscheinlichkeit für die Unreife einer Frucht ist $p_{unreif} = 1 - p_{reif}$
- Klassifikation beschreibt die Zuordnung von Datenpunkten in unterschiedliche *Klassen*, d. h. das ML-Modell kategorisiert neue Daten nach Mustern in den Features der Trainingsdaten.
- Klassifikation ist ein weit verbreiteter Anwendungsfall
 - Erkennung von Verkehrsschildern
 - Spamfilter
 - und viele mehr ...
- Hinweis: Wir betrachten hier die binäre Klassifikation, d. h. die Unterscheidung in zwei Klassen (*binary classification*). Wir betrachten später noch die Unterscheidung in mehrere Klassen (*multi-class classification*).

Logistische Regression

- Idee: Wir quetschen die Werte der linearen Regression in das Intervall $[0,1]$ und bewerten die Ergebnisse als Wahrscheinlichkeit, dass eine Frucht der *Klasse* "reif" angehört

- Wir nutzen dafür die *Logistic Function* (Logistische Funktion) σ
 - stetige, eindimensionale Wahrscheinlichkeitsverteilung
 - σ ist eine *Sigmoid Function*, charakterisiert durch ihre S-Form

$$\bullet \sigma(x) = \frac{1}{1 + e^{-x}}$$



- Bei der linearen Regression haben wir eine Lösung für den Parameter θ der Funktion $f(x) = \theta^T x$ gesucht
 - Erinnerung: θ_0 war unser Bias Term und wir haben $x_0 = 1$ gesetzt
- Bei der logistischen Regression ergänzen wir diesen Term um die Logistic Function:

$$f(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-(\theta^T x)}}$$

- Außerdem haben wir in der Linearen Regression *Mean Squared Error* (MSE) zur Bewertung der Güte unserer Vorhersage verwendet
 - Wie sieht die Metrik jetzt aus?
 - Wann ist die Klassifikation “gut”?

Kostenfunktion Log Loss

- Wir streben weiterhin an, die optimalen Modellparameter θ während des Trainings zu finden. Dafür suche wir eine Belegung, die

- hohe Wahrscheinlichkeiten für positive Instanzen ($y = 1$) generiert
- niedrige Wahrscheinlichkeiten für negative Instanzen ($y = 0$) generiert

- Für ein einzelnes(!) Sample x ist

$$c(\theta) = \begin{cases} -\log(p) & \text{wenn } y = 1 \\ -\log(1 - p) & \text{wenn } y = 0 \end{cases}$$

denkbar, weil $-\log(p)$ bei kleinen p sehr groß wird, wodurch die Kosten für falsche Zuordnungen sehr hoch werden.

- Für das gesamte Trainingsset wird einfach der Durchschnitt über die Kosten der einzelnen Samples berechnet

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)})]$$

- Die schlechte Nachricht: Für Log Loss existiert keine Lösung, die mithilfe der Normalengleichung die optimale Belegung von θ berechnet werden kann.
- Die gute Nachricht: Log Loss ist auch eine konvexe Funktion, d. h. eine Annäherung, z. B. Gradient Descent, wird ein globales Minimum finden.

- Es gibt verschiedene Möglichkeiten, die Güte der Klassifikation zu prüfen.
- Beispiel: Unser Fruchtsortierer unterscheidet zwei Klassen *Reif* (Positiv) und *Unreif* (Negativ)
 - *True Positives* (TP): Das Modell ordnet die Eingabe korrekt der positiven Klasse zu
→ Reife Frucht wird durchgelassen
 - *False Positives* (FP): Das Modell ordnet die Eingabe fälschlicherweise der positiven Klasse zu
→ Unreife Frucht wird durchgelassen
 - *True Negatives* (TN): Das Modell ordnet die Eingabe korrekt der negativen Klasse zu
→ Unreife Frucht wird aussortiert
 - *False Negatives* (FN): Das Modell ordnet die Eingabe fälschlicherweise der negativen Klasse zu
→ Reife Frucht wird aussortiert

- Die einfachste Möglichkeit ist, die *Accuracy* (Genauigkeit) des Modells zu bestimmen:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- Accuracy ist aber nur sinnvoll, wenn die zu lernenden Klassen gleichverteilt sind

Klassifikation bewerten

- Eine alternative Möglichkeit ist die *Confusion Matrix*, die alle Informationen (TP, FP, TN und FN) aufbereitet anzeigt.

- Eine etwas prägnantere Metrik ist die *Precision* (Präzision) des Classifiers, der die Genauigkeit der Klassifikation zeigt:

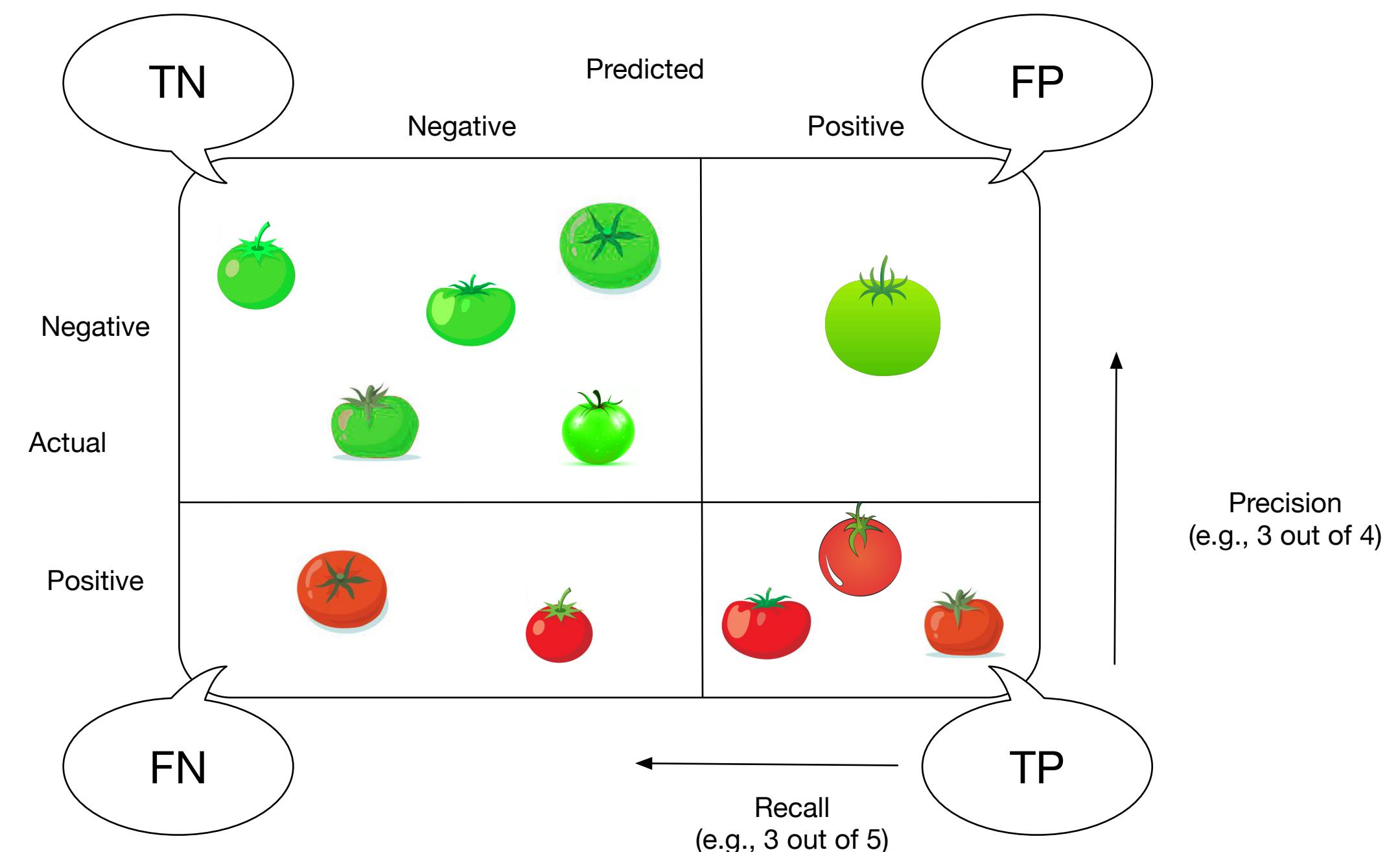
$$precision = \frac{TP}{TP + FP}.$$

Dabei schaut man auf alle einer Klasse zugeordneten Eingaben.

- Die Bedeutung von Precision ist im Kontext des Anwendungsfalles zu sehen!
- Precision alleine ist nicht aussagekräftig. Es braucht eine weitere Metrik: Recall.

- *Recall* ("Trefferquote", manchmal auch *Sensitivity* oder *True Positive Rate (TPR)*) beschreibt das Verhältnis aller klassifizierten Eingaben einer Klasse.

$$recall = \frac{TP}{TP + FN}$$



Klassifikation bewerten

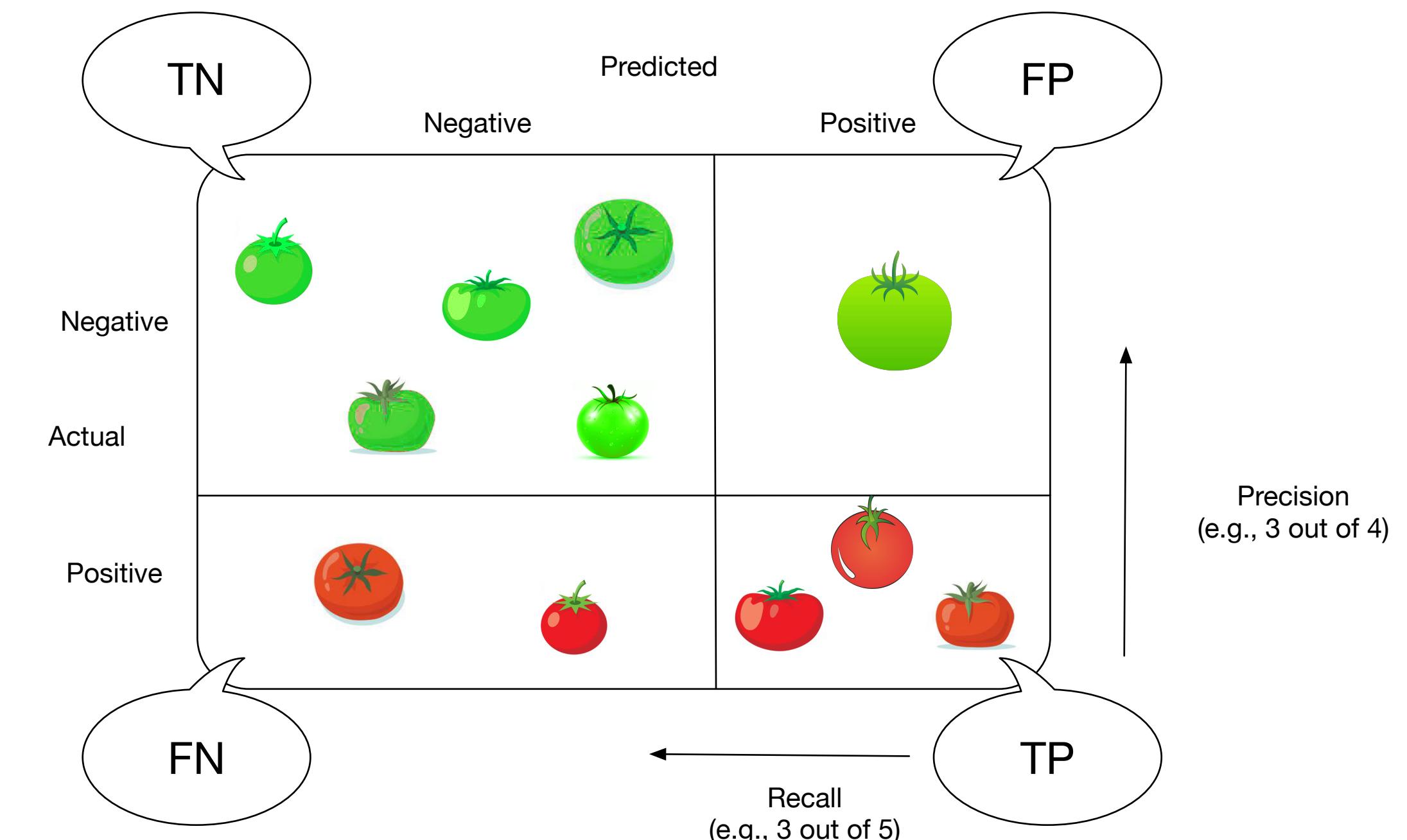
- Precision und Recall werden oft als F_1 -Score kombiniert:

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

- Der F_1 -Score begünstigt Classifier mit ähnlichen Precision und Recall, aber das ist nicht immer wünschenswert.

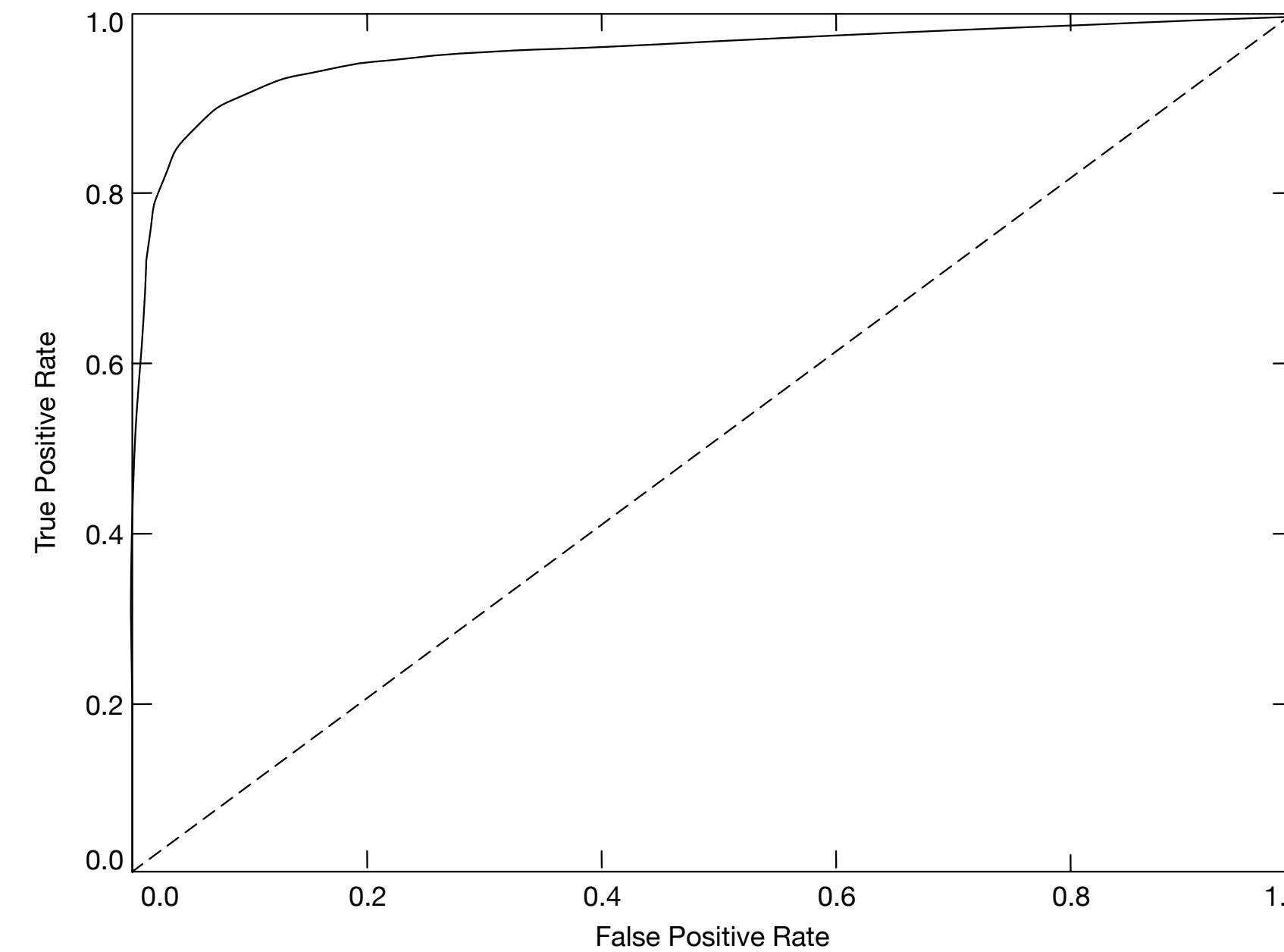
- Beispiel 1: Der Classifier bewertet, ob ein Video jugendfrei ist oder nicht. Es ist besser, ein jugendfreies Video falsch zu klassifizieren, als ein nichtjugendfreies (high precision/low recall)
- Beispiel 2: Der Classifier sucht Ladendiebe auf Bildern von Überwachungskameras. Fehlalarme sind besser, als Diebe unentdeckt entwischen lassen (low precision/high recall)

- Precision und Recall stehen also oftmals in einem Verhältnis zueinander und man muss abwägen, was wichtiger ist
 - Weniger FPs = High Precision
 - Weniger FNs = High Recall



Klassifikation bewerten

- Eine weitere Metrik ist die *ROC Curve*
 - *Receiver Operating Characteristic (ROC)*
 - Vergleicht die *True Positive Rate (TPR, Recall)* mit der *False Positive Rate (FPR)*.
- Die gestrichelte Linie ist ein Classifier, der das Ergebnis einer binären Klassifikation zufällig bestimmt, folglich möchte man mit der ROC Curve möglichst weit davon entfernt sein.
- Classifier können mithilfe des *ROC AUC Score* verglichen werden (*AUC = Area Under the Curve*)
- Ein *perfekter* Classifier hat einen ROC AUC Score von 1 und ein Classifier, der das Ergebnis zufällig bestimmt, einen ROC AUC Score von 0,5

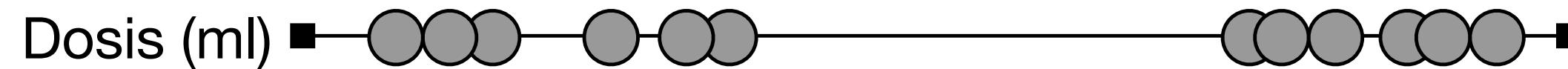


Support Vector Machines (SVM)

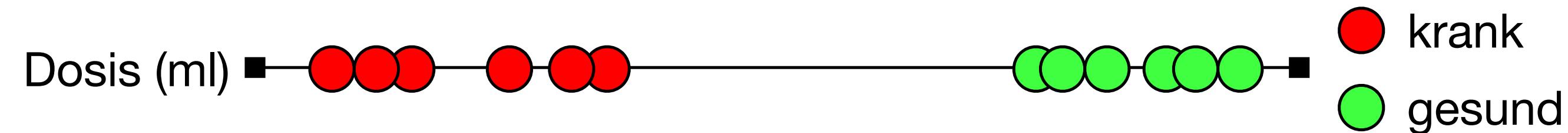
Stützvektormethode

Beispiel

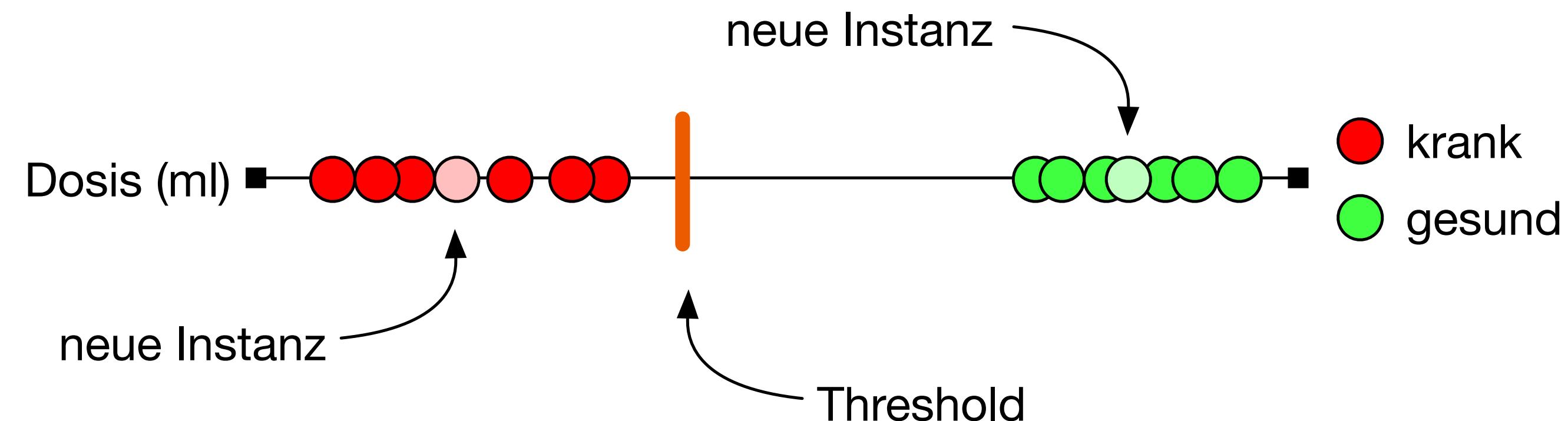
- Wir betrachten einen Datensatz mit Patienten, die ein Medikament in unterschiedlichen Dosen erhalten



- In Abhangigkeit der Dosis konnen Patienten genesen oder auch nicht

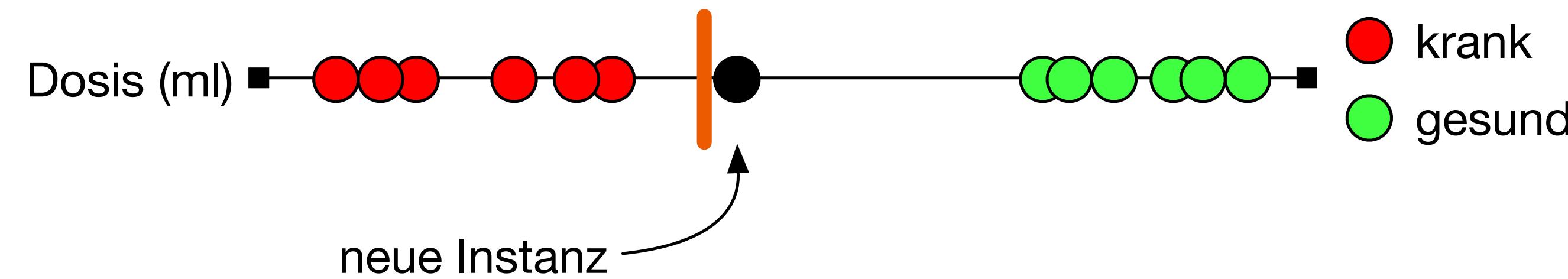


- In Abhangigkeit von diesen Beobachtungen konnen wir einen *Threshold* (Schwellwert) definieren, mit dessen Hilfe wir neue Patienten in *krank* und *gesund* klassifizieren

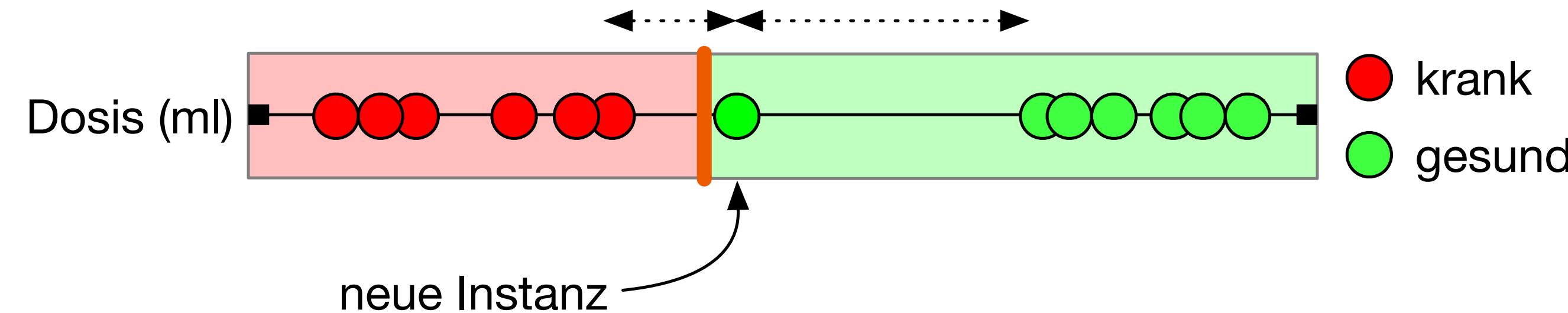


Beispiel

- Aber: Was passiert mit neuen Beobachtung nahe am Schwellwert?

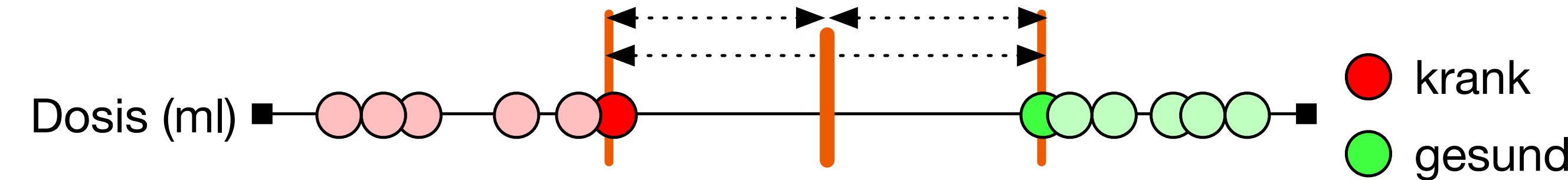


- Wir würden die Beobachtung als *gesund* klassifizieren, tatsächlich ist sie aber viel näher an den *kranken* Patienten dran als an den *gesunden*. Wie können wir diese Situation verhindern und unser Modell optimieren?

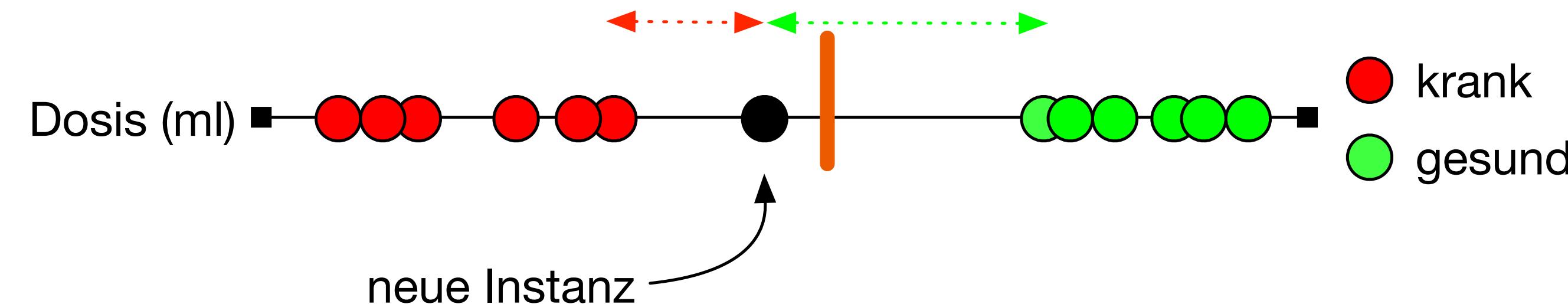


Hard Margin Classifier

- Lösung: Wir betrachten die Ränder der Gruppen und ermitteln daraus einen Mittelwert, den wir als Threshold verwenden.

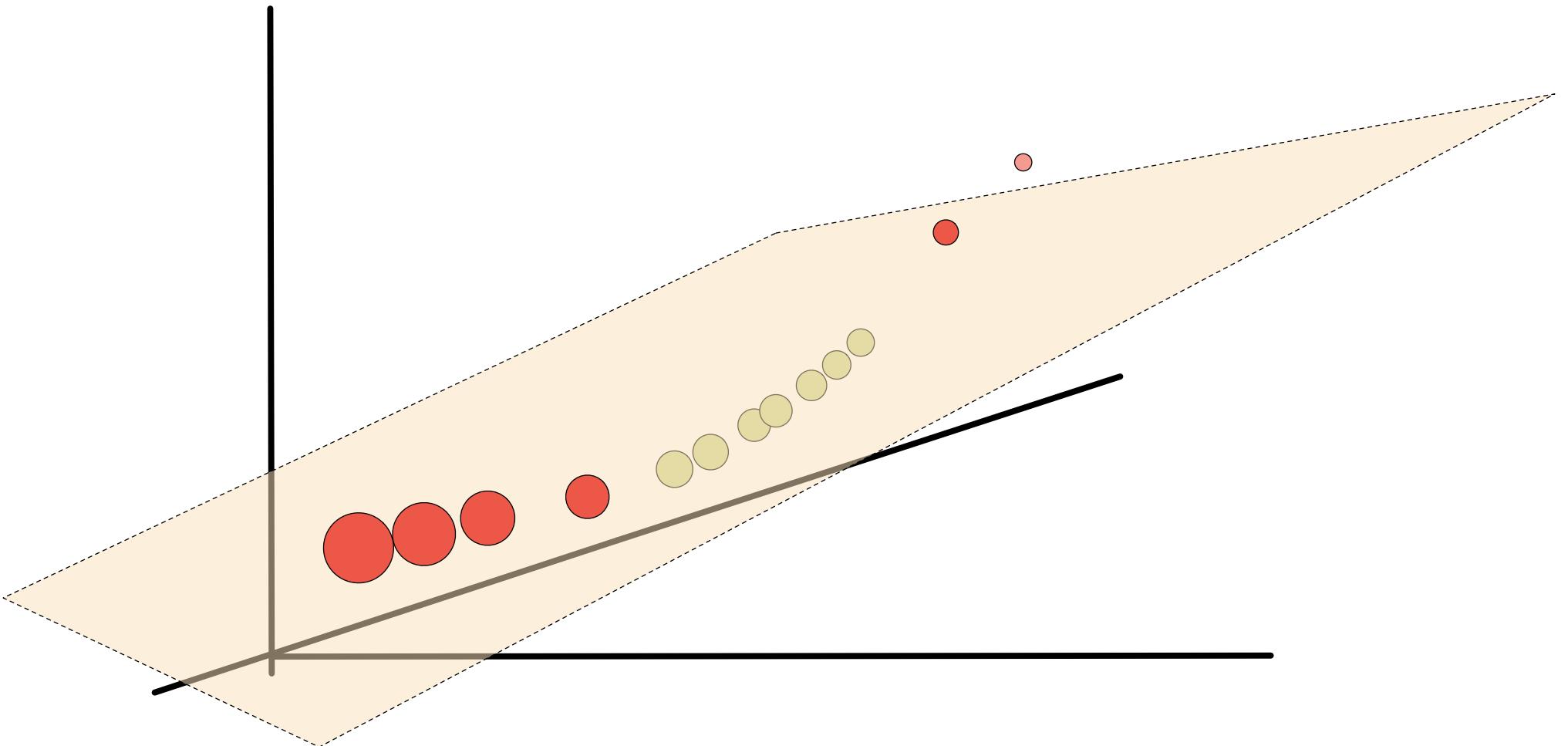
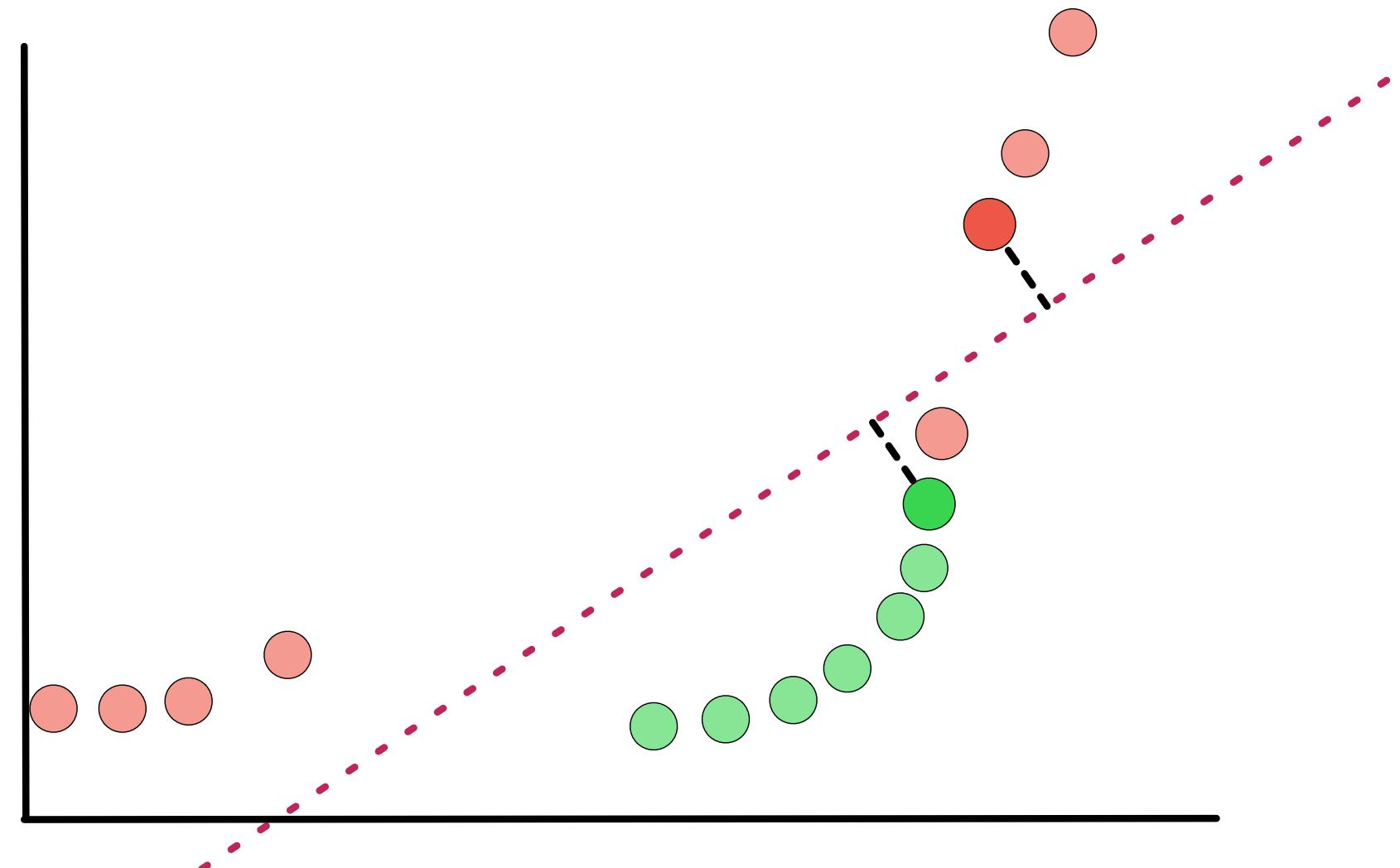


- Wenn es bis zu den äußersten *Margins* (Rändern) keine Instanzen in den Trainingsdaten gibt, dann nennen wir das *Hard Margin Classification*.
- Die Datensätze, die die Margins definieren, nennen wir *Support Vectors* (Stützvektoren), daher sprechen wir hier auch von *Support Vector Classifiers* (Stützvektorklassifizierer).
- Die Folge ist, dass neue Beobachtungen jetzt näher an bestehenden Datensätzen ihrer jeweiligen Klasse sind.



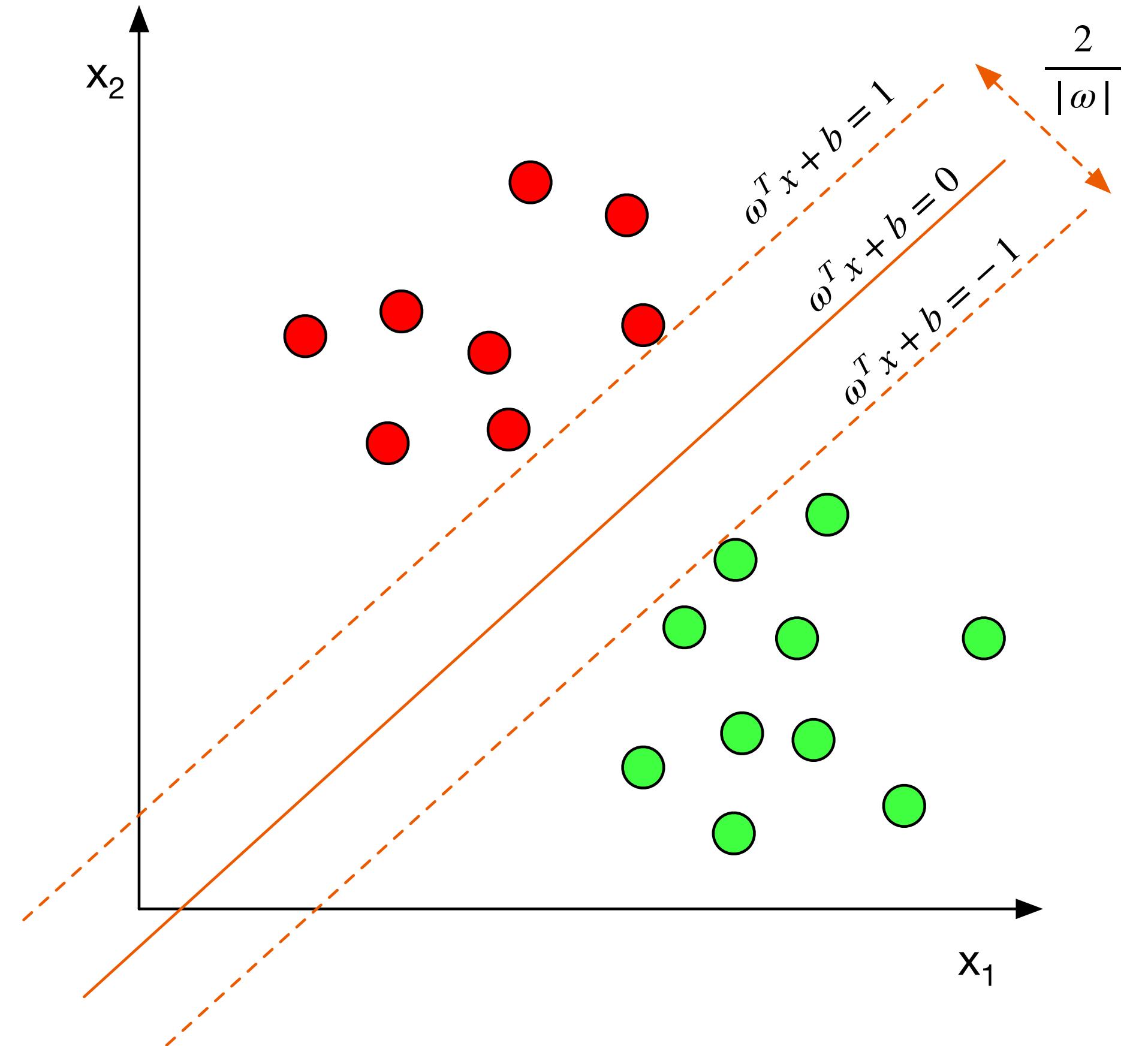
Hard Margin Classifier

- Grundsätzlich suchen wir eine *Hyperplane* (Hyperebene), die die Daten optimal trennt
 - Als Hyperplane bezeichnen wir einen $(n - 1)$ -dimensionalen Subraum eines n -dimensionalen Raumes
 - Es sind immer unendlich viele Hyperplanes zur Trennung der Daten denkbar
 - *Optimal trennen* bedeutet, dass die Entfernung zwischen Datenpunkten und Hyperplane maximiert wird
- Die Hyperplane ist
 - ein Punkt bei eindimensionalen Daten
 - eine Linie bei zweidimensionalen Daten
 - eine Ebene bei dreidimensionalen Daten
 - etwas nicht mehr sinnvoll darstellbares ab vierdimensionalen Daten
- Was hat das mit Machine Learning zu tun? Unser Modell soll die optimale Hyperplane lernen!



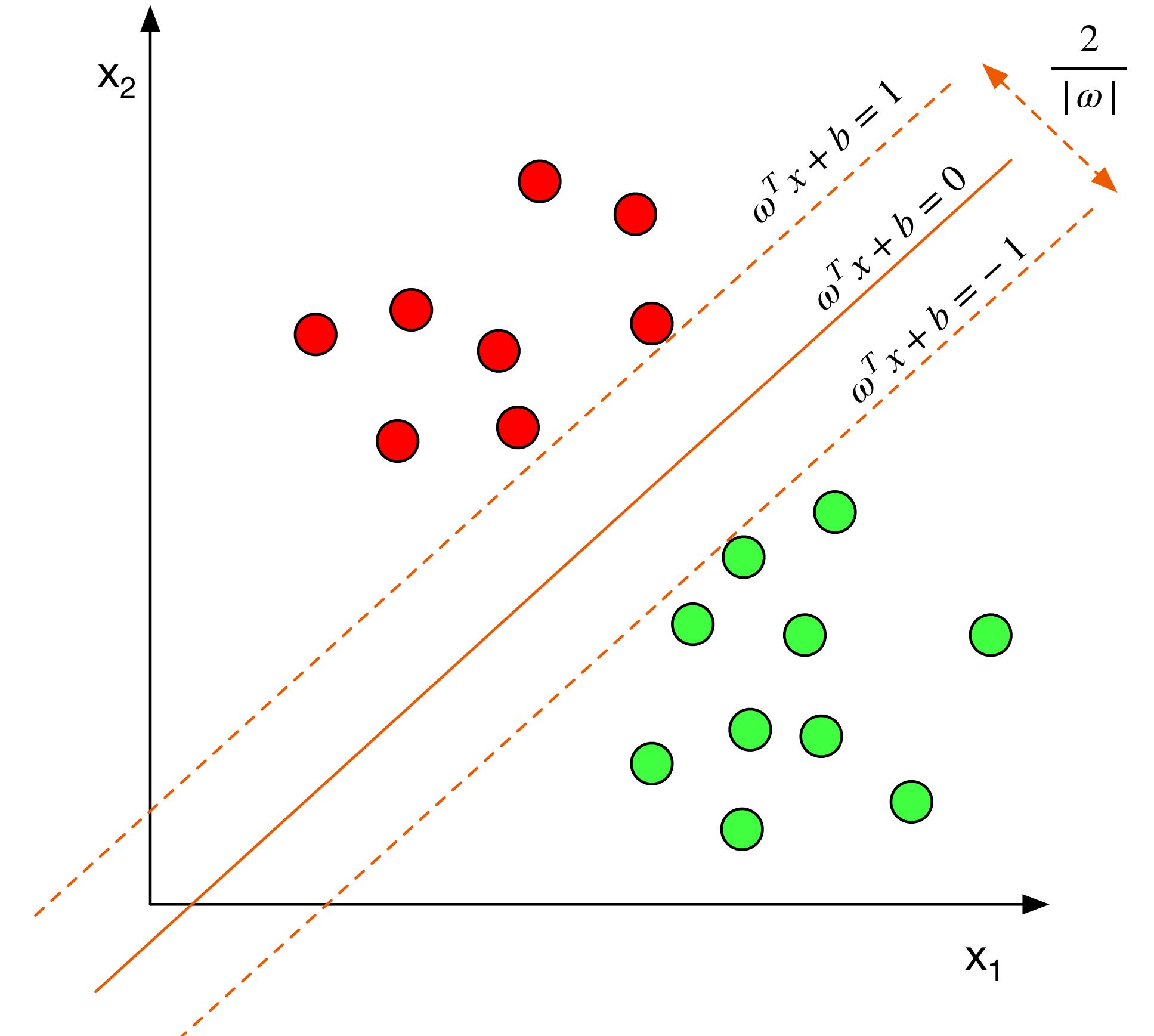
Hard Margin Classifier

- Eine Hyperplane wird definiert über $\omega^T x + b = 0$, dabei ist
 - x ein Vektor mit Trainingsdaten
 - ω ein rechtwinklig zur Hyperplane verlaufender Vektor (auch *Normal Vector* oder *Weight Vector* genannt)
 - b der Bias Term
- Uns interessieren darüber hinaus zwei parallele Hyperplanes, um die Margins abzubilden. Diese werden beschrieben mit $\omega^T x + b = 1$ und $\omega^T x + b = -1$
- Die Klassifizierung erfolgt über $f(x) = \text{sign}(\omega^T x + b)$, woraus folgt, dass das Ergebnis binär einer der zwei Klassen 1 und -1 zugeordnet wird



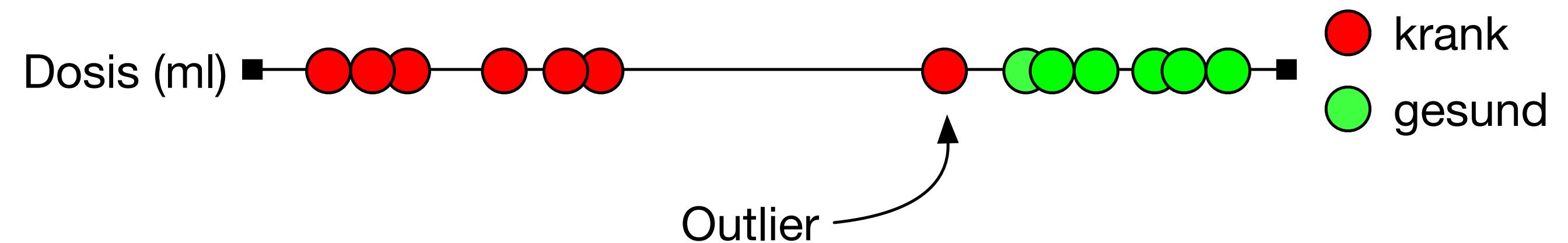
Hard Margin Classifier

- Der optimale (maximierte) Abstand zwischen den Hyperplanes ist $\frac{2}{|\omega|}$, also wollen wir $|\omega|$ minimieren.
- Umgewandelt in ein Minimierungsproblem folgt, dass wir ω und b suchen, für die folgendes gilt, wobei $t^{(i)}$ eine Umsetzung des Ergebnis der Klassifikation ist (also -1 oder 1)
 - $\frac{\omega^T \omega}{2}$ (oder $\frac{||\omega||^2}{2}$)
 - $t^{(i)}(\omega^T x_i + b) \geq 1$
- Daraus folgt ein Problem, dass sich mit *Quadratischer Optimierung* lösen lässt, aber an dieser Stelle zu weit führt.
- Hinweis: Wir gehen hier davon aus, dass die Daten passend skaliert wurden. Die Skalierung ist notwendig, um bessere Ergebnisse zu erreichen. Details dazu folgen in den nächsten Veranstaltungen.

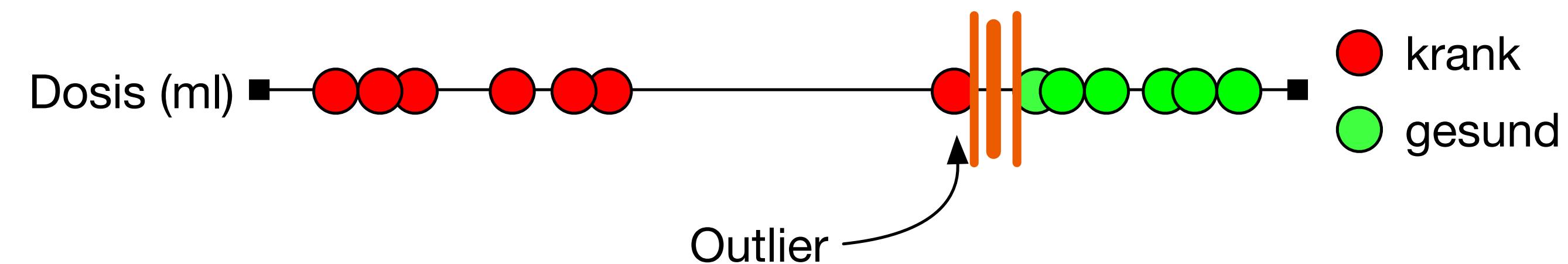


Hard Margin Classifier

- Problem: Outlier (Ausreißer) können aus vielen Gründen entstehen
 - Natürlich vorkommende Outlier
 - Messfehler
 - Fehler im Labeling



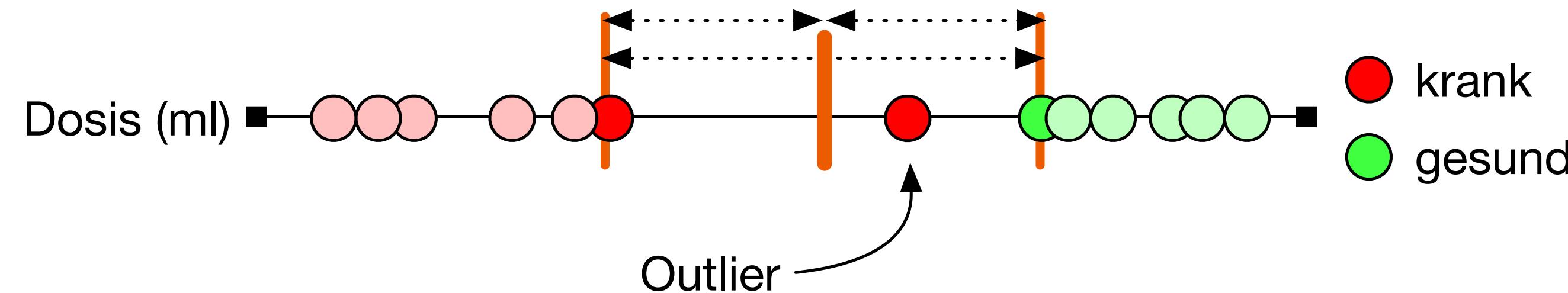
- Outlier beeinflussen unser Modell ggf. erheblich, wenn Sie wie normale Datensätze behandelt werden. Im konkreten Fall würden sie dafür sorgen, dass unsere Margins und der Threshold sehr enge Korridore bilden.



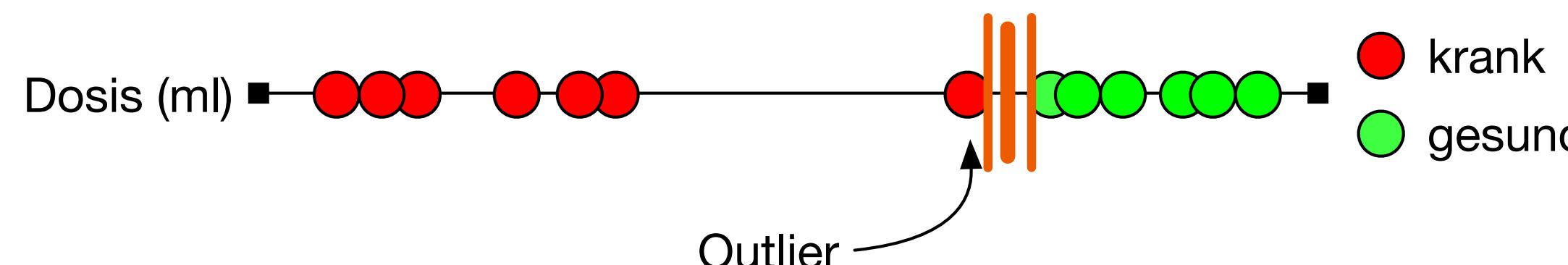
- Hard Margin Classifiers sind folglich sehr empfindlich gegenüber Outliern. Wie können wir das optimieren?

Soft Margin Classifier

- Eine denkbare Optimierung ist, vorhandene Outlier in den Trainingsdaten zu identifizieren und als *Misclassification* (falsche Klassifizierung) zu akzeptieren. Wir nennen dieses einen *Soft Margin Classifier*.



- Dieses Vorgehen ist ein Beispiel für den Kompromiss zwischen *Bias* und *Variance*, der beim Einsatz von Machine Learning immer getroffen werden muss.
- Unser ursprünglicher Classifier war sehr gut auf den Trainingsdaten (low bias), aber eher schlecht auf unbekannten Daten (high variance). Unser neuer Classifier hingegen erlaubt Misclassification und ist daher schlechter auf den Trainingsdaten (higher bias), dafür besser auf unbekannten Daten (low variance).



Soft Margin Classifier

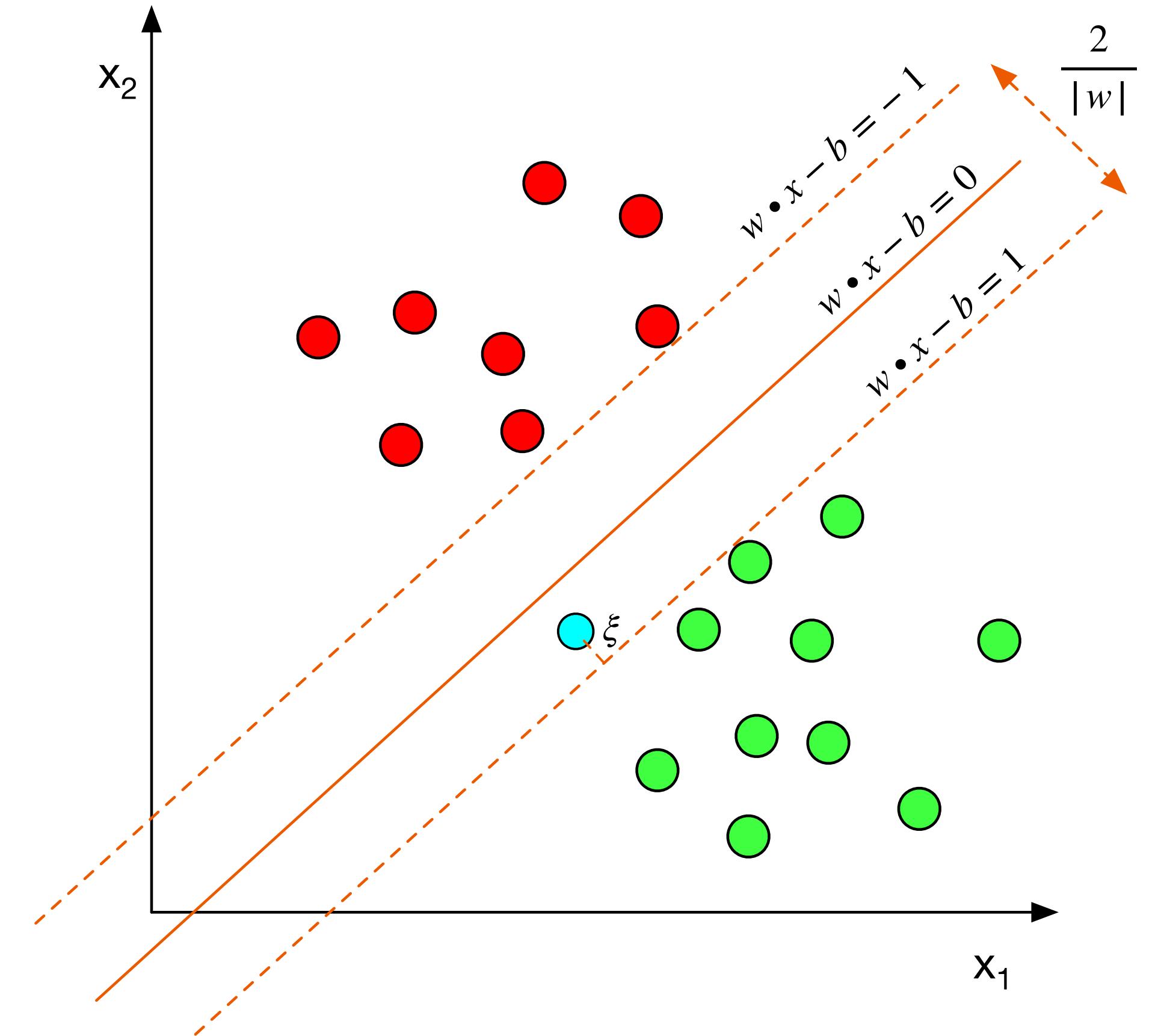
- Zur Integration der Soft Margin und der Berücksichtigung von Outliern, also wenn die zuvor genannten Hyperplanes nicht eindeutig bestimmt werden können, wird eine *Slack Variable* (Schlupfvariable) ξ (ksi) für jedes Instanz im Trainingsset eingeführt, um den erlaubten Grad an Misclassification zu definieren.

- Dadurch erweitert sich unser ursprüngliches Problem zu einer Suche nach einem ω , b und $\xi_i \geq 0$, für die gilt, dass

- $\frac{\omega^T \omega}{2} + \lambda \sum_i \xi_i$ minimiert wird

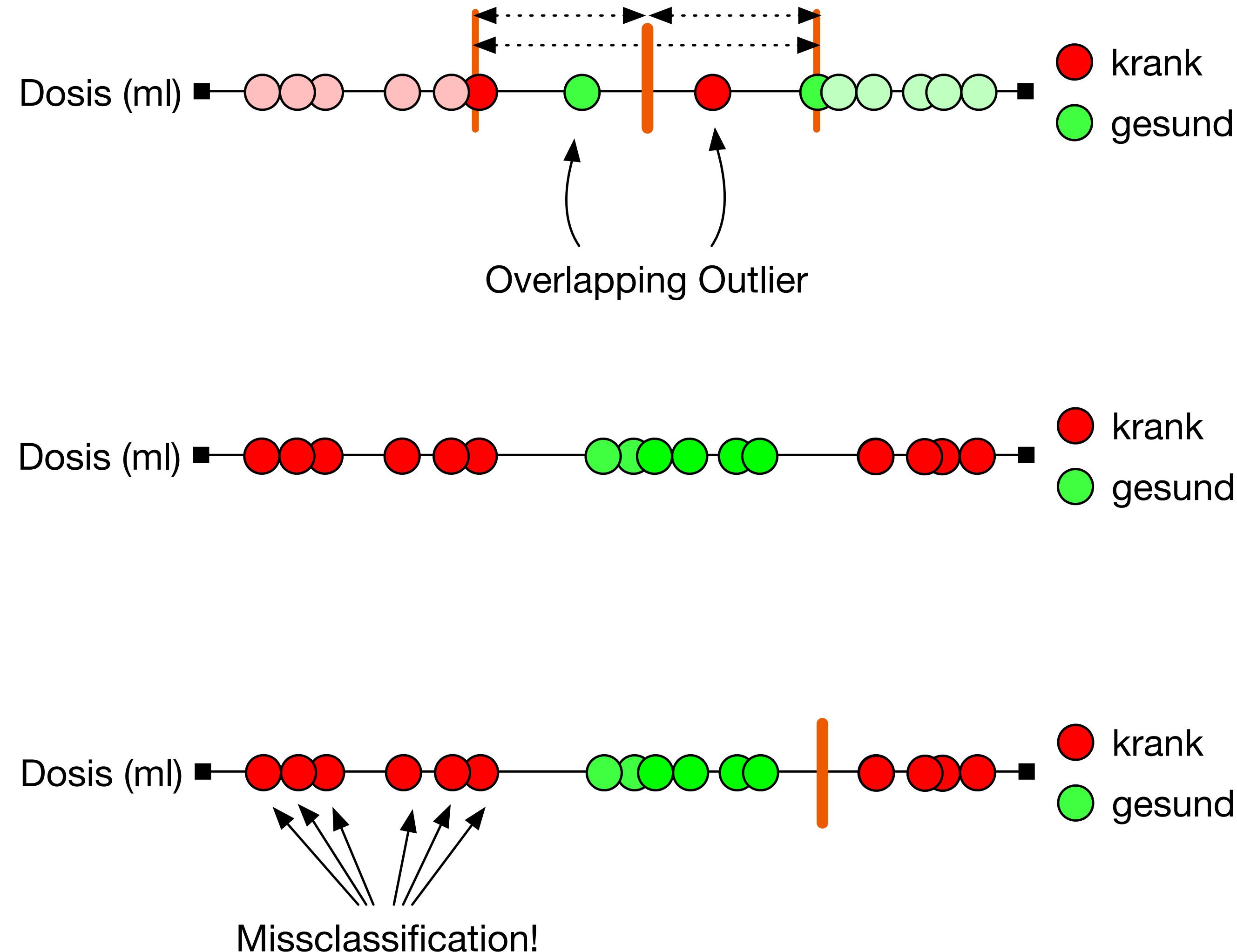
- $t^{(i)}(\omega^T x_i + b) \geq 1 - \xi$

- Daraus folgen zwei widersprüchliche Ziele
 - Minimierung von ξ , um Verletzung des Margins zu verhindern
 - Minimierung von $\frac{\omega^T \omega}{2}$, um den Margin zu vergrößern
- λ ist ein *Hyperparameter*, der bei der Entwicklung eingestellt werden muss, und steuert das Verhältnis dieser Werte steuert



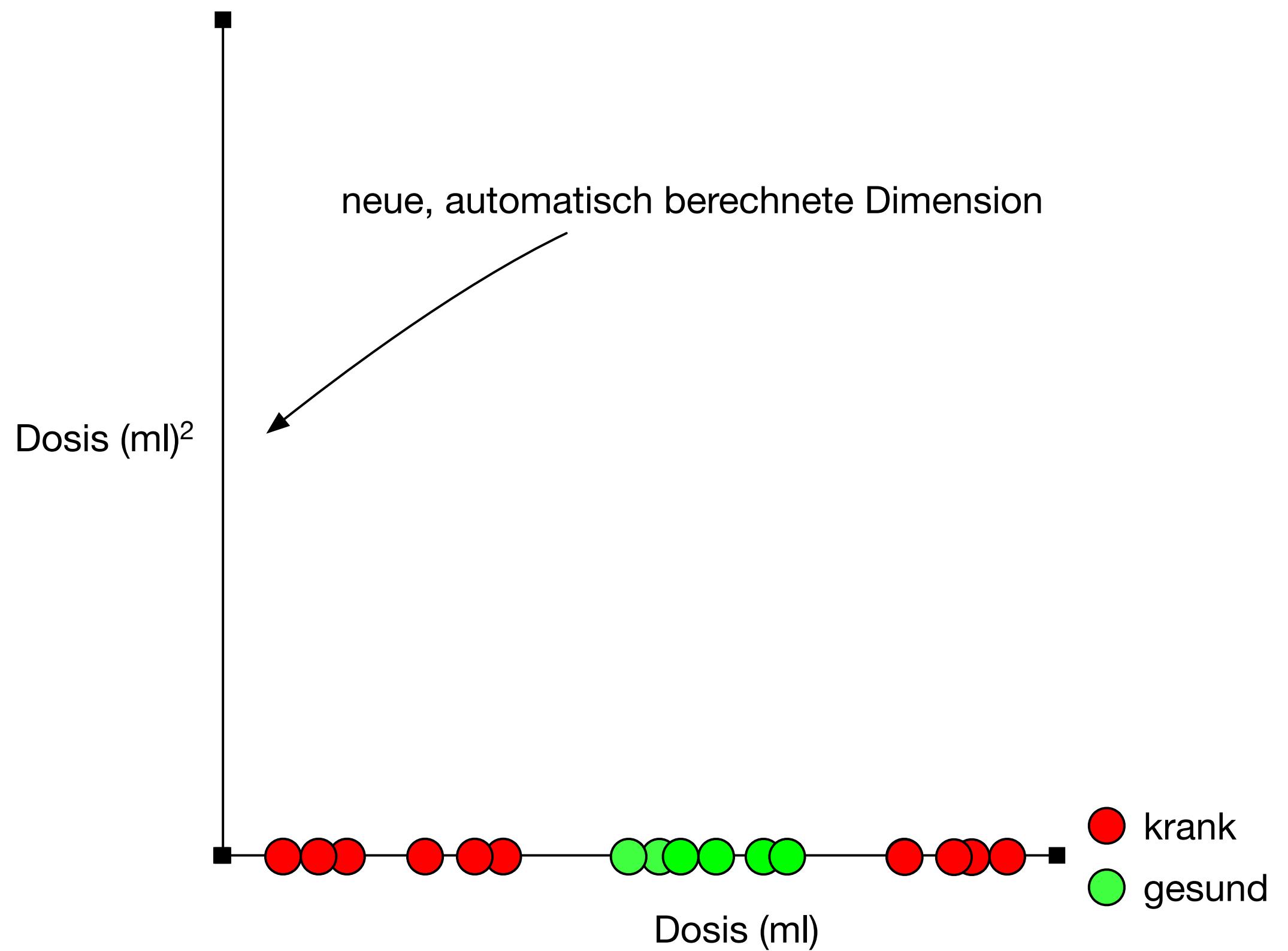
Soft Margin Classifier

- Soft Margin Classifier sind gut, denn
 - sie erkennen Outlier
 - sie können auch überlappende Daten handhaben, denn sie erlauben Misclassifications.
- Aber was, wenn die Daten sehr stark überlappen? Beispiel: Es werden nur Patienten gesund, die die richtige Dosis des Medikaments bekommen. Sowohl zu niedrige als auch zu hohe Dosen führen nicht zur Genesung.
- Weder Hard noch Soft Margin können mit diesen Daten umgehen, weil hier kein sinnvoller Threshold gesetzt werden kann.



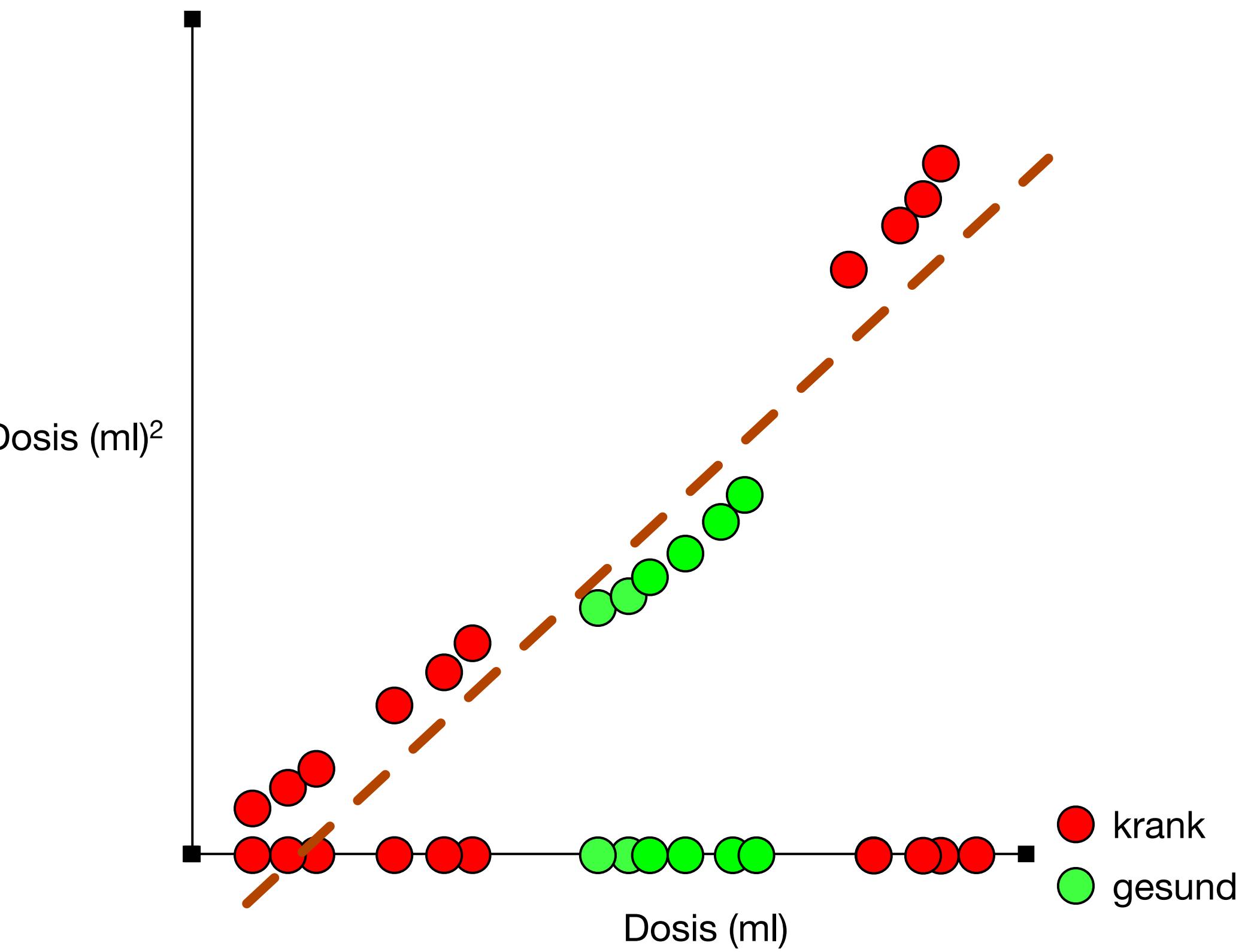
Support Vector Machines

- Wir wandeln daher unseren bestehenden Support Vector Classifier zu einer *Support Vector Machine* (SVM) um.
- Dazu fügen wir unseren Trainingsdaten eine weitere, automatisch berechnete(!) Dimension hinzu, in diesem Falle das Quadrat der Dosis (Dosis (ml)²).
- Die x-Achse bleibt unverändert!
- Die Werte der y-Achse berechnen wir anhand der Werte der x-Achse.



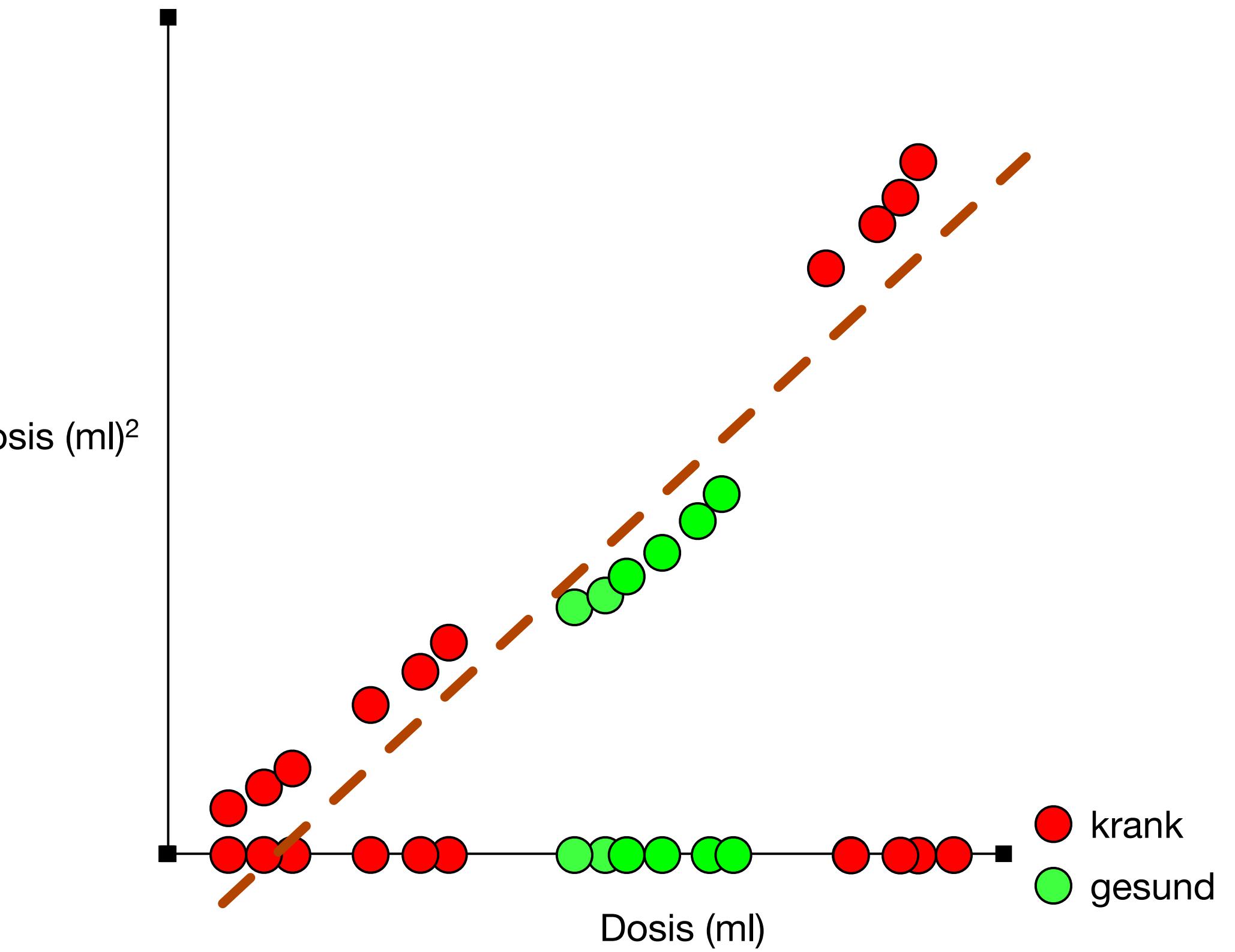
Support Vector Machines

- Im neuen zweidimensionalen Raum können wir wieder eine Hyperplane als eindeutigen Threshold definieren, jetzt als Linie.
- Für diesen neuen Raum können wir wieder einen Support Vector Classifier trainieren, der die Unterscheidung vornehmen kann.
- Die grundlegende Idee einer SVM ist
 - Starte mit den Trainingsdaten in einer tiefen Dimension
 - Transformiere die Daten in eine höhere Dimension
 - Finde einen Support Vector Classifier, der die Daten in der höheren Dimension klassifizieren kann



Support Vector Machines

- Frage: Wie finden wir die passende Transformation für die Daten?
- Lösung: SVMs verwenden einen *Kernel*, mit dem die passenden Support Vector Classifiers in höheren Dimensionen verwendet werden können. Details zum Kernel-Trick folgen in VL zu nichtlinearen Verfahren.
- Hinweis: Bitte berücksichtigen Sie, dass wir zur Veranschaulichung ein stark vereinfachtes Beispiel mit wenig Dimensionen verwenden. In der Realität wird ein derartiges Verfahren auf einem großen Datensatz mit vielen Features trainiert, sodass ein für den Menschen unüberschaubarer, mehrdimensionaler Raum entsteht!



- SVMs werden insbesondere dann verwendet, wenn
 - eine lineare (vor allem binäre) oder nichtlineare Klassifikation umgesetzt werden soll
 - eine Regression umgesetzt werden soll
 - Ausreißer erkannt werden sollen (*Outlier Detection*)
- SVMs eignen sich insbesondere für komplexe, aber eher kleine bis mittelgroße Datenmengen, die viele Features bieten
- SVMs werden für die *binäre Klassifikation*, also die Unterteilung in zwei Klassen, eingesetzt
 - *Multi-Klassen-Klassifikation (multiclass classification)* kann mit SVMs ebenfalls umgesetzt werden. Es gibt zwei mögliche Strategien:
 - *One-vs-One*: Das Problem wird in mehrere binäre Klassifikationen unterteilt, wobei immer zwei Pärchen der möglichen Klassen gebildet werden
 - *One-vs-Rest*: Das Problem wird in mehrere binäre Klassifikationen unterteilt, wobei für jede Klasse eine Klassifikation in *Klasse* oder *Rest* vorgenommen wird

- Im Gegensatz zu vielen anderen Methoden des Machine Learning bieten SVMs keinen probabilistischen Output, sondern erzeugen ein *hartes Label*, das entweder 1 oder –1 als Ergebnis hat.
- Daraus ergeben sich einige Probleme, insbesondere bei Multiclass Classification, z. B. ist es erstrebenswert, dem Ergebnis eine Konfidenz für das zugeordnete Label hinzuzufügen.
- Eine denkbare Erweiterung ist das *log-odds ratio*
- Die Ausgabe einer SVM in eine Wahrscheinlichkeit erfolgt dann mit $p(y = 1 | x, \theta) = \sigma(af(x) + b)$, wobei a und b auf einem separaten *Validation Set* bestimmt werden müssen, weil eine Verwendung des Training Sets zu extremen Overfitting führt.
- Problematisch bleibt aber, dass diese Informationen im Nachgang ermittelt werden und im Trainingsprozess der SVM keine Rolle spielen, wodurch die Ergebnisse nur bedingt sinnvoll eingesetzt werden können

$$\log\left(\frac{p(y = 1 | x)}{p(y = 0 | x)}\right)$$

Anhang

Herleitung der Lösung der linearen Regression unter Verwendung von MSE als Kostenfunktion

$$MSE(X, h_{\theta}) = \frac{1}{m} \sum_{i=0}^m (\theta^T x^{(i)} - y_i)^2$$

- Dabei ist
 - $X = (x^{(1)}, \dots, x^{(m)})^T$ die Matrix aller Trainingsdaten (Samples)
 - $x^{(i)} = (x_1^{(i)}, \dots, x_n^{(i)})$ der Vektor des i -ten Samples, $x_j^{(i)}$ das j -te Feature des i -ten Samples, x_j bezeichne das j -te Feature
- Um das θ zu finden, das $MSE(X, h_{\theta})$ minimiert, wird die erste Ableitung (hier wegen $\theta = (\theta_0, \dots, \theta_n)^T \in \mathbb{R}^n$ der Gradient) gebildet und auf 0 gesetzt

$$\nabla_{\theta} MSE(h_{\theta}, x) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} MSE(h_{\theta}, x) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(h_{\theta}, x) \end{pmatrix} = \begin{pmatrix} \frac{2}{m} \sum_{i=1}^m (\theta^T x^{(i)} - y_i) x_1^{(i)} \\ \vdots \\ \frac{2}{m} \sum_{i=1}^m (\theta^T x^{(i)} - y_i) x_n^{(i)} \end{pmatrix} = \frac{2}{m} \begin{pmatrix} x_1^T (X\theta - y) \\ \vdots \\ x_n^T (X\theta - y) \end{pmatrix} = \frac{2}{m} X^T (X\theta - y) = \frac{2}{m} (X^T X\theta - X^T y)$$

$$\nabla_{\theta} MSE(h_{\theta}, x) \stackrel{!}{=} 0 \iff X^T X\theta - X^T y = 0 * \frac{m}{2} \iff X^T X\theta = X^T y \iff \theta = (X^T X)^{-1} X^T y$$

Oder eine viel schnellere „alternative“ Herleitung

- $MSE(X, h_\theta) = \frac{1}{m} \sum_{i=0}^m (\theta^T x^{(i)} - y_i)^2 = \frac{1}{m} \|X\theta - y\|^2$
- $\nabla_\theta MSE(h_\theta, x) = \frac{2}{m} X^T(X\theta - y) = \frac{2}{m}(X^T X\theta - X^T y)$
- $\nabla_\theta MSE(h_\theta, x) \stackrel{!}{=} 0 \iff X^T X\theta - X^T y = 0 * \frac{m}{2} \iff X^T X\theta = X^T y \iff \theta = (X^T X)^{-1} X^T y$

Hinweis:

- Da die Matrix $X^T X$ nicht immer invertierter ist, ist die richtige Schreibweise $(X^T X)^+$ statt $(X^T X)^{-1}$.
- $(X^T X)^+$ ist die sogenannte Pseudoinverse (verallgemeinerte Inverse) von $X^T X$.
- Ist $X^T X$ invertierter, so gilt $(X^T X)^+ = (X^T X)^{-1}$.