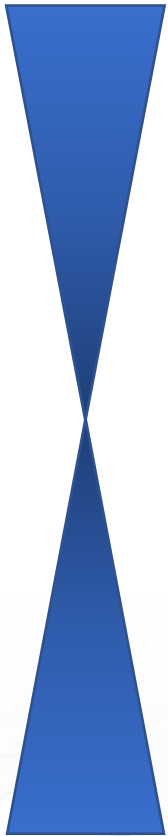




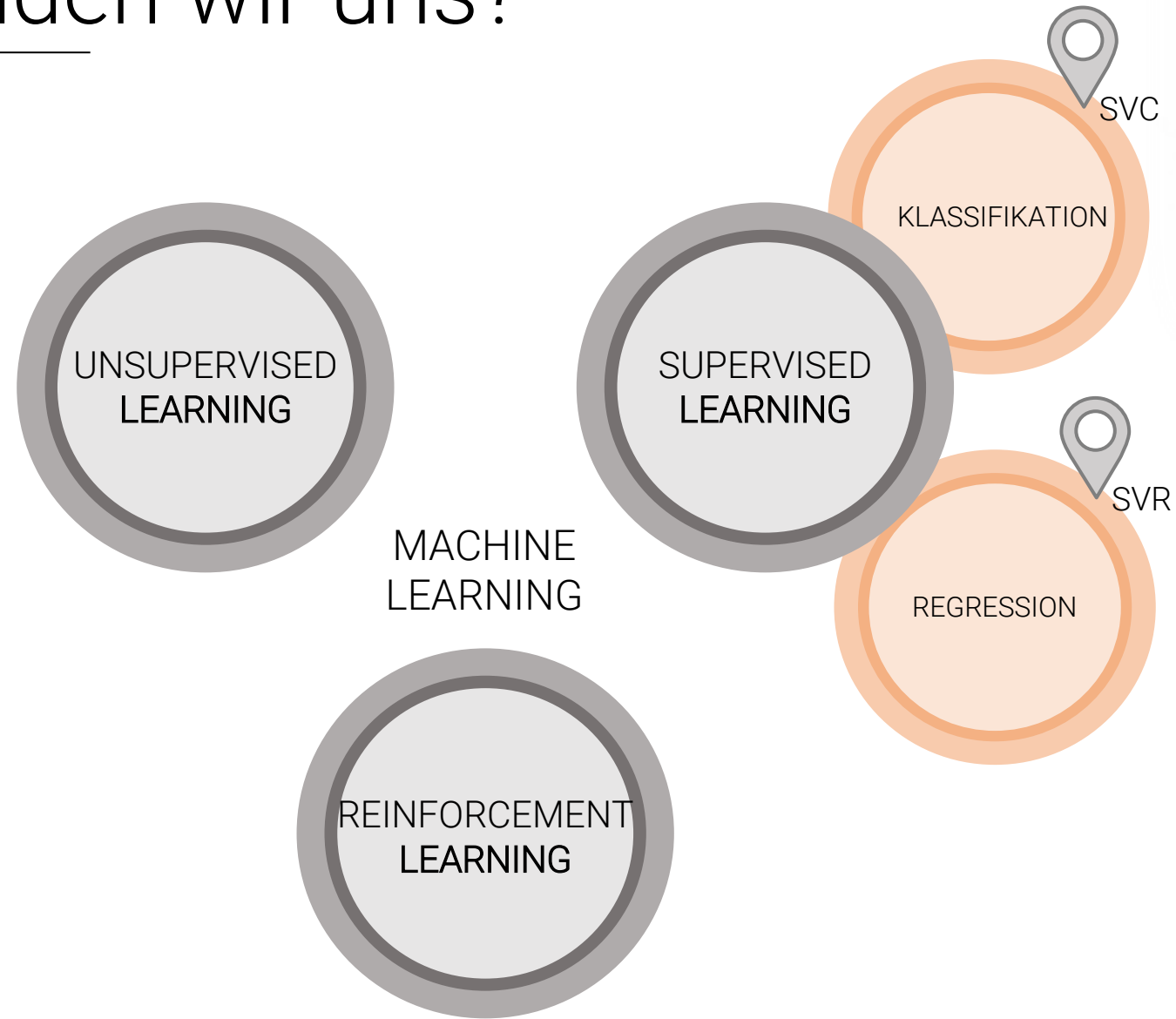
Modeling

Supervised Learning: Support Vector Machines

Agenda

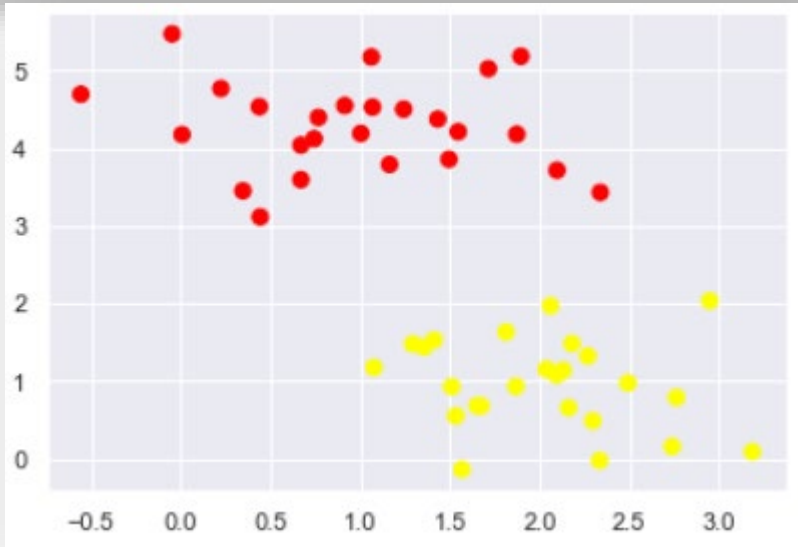
- 
1. SVM: Margin Maximierung und Support Vectors
 2. Mathematischer Hintergrund zu SVMs
 3. Soft-Margin SVM und Optimierung
 4. Nicht-lineare Probleme und Kernel-Trick

SVM: wo befinden wir uns?



SVM: allgemein

```
1 from sklearn.datasets.samples_generator import make_blobs
2 X, y = make_blobs(n_samples=50, centers=2,
3                   random_state=0, cluster_std=0.60)
4 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



Unsere Ausgangssituation:

Zwei Klassen in einem 2D-Feature-Raum, die wir mit einem Klassifikator zu trennen versuchen



Was denken Sie?
Welches Problem
sehen wir hier sofort?

- Support Vector Machines (**SVM**) sind eine der bekanntesten Modelle im Machine Learning
- Vor allem für **klein- und mittelgroße** Datensätze geeignet
- Sie sind flexibel und können für Klassifikations- und Regressionsprobleme verwendet werden
- SVMs stellen im Kontext der Klassifikation *diskriminative* Klassifikatoren dar



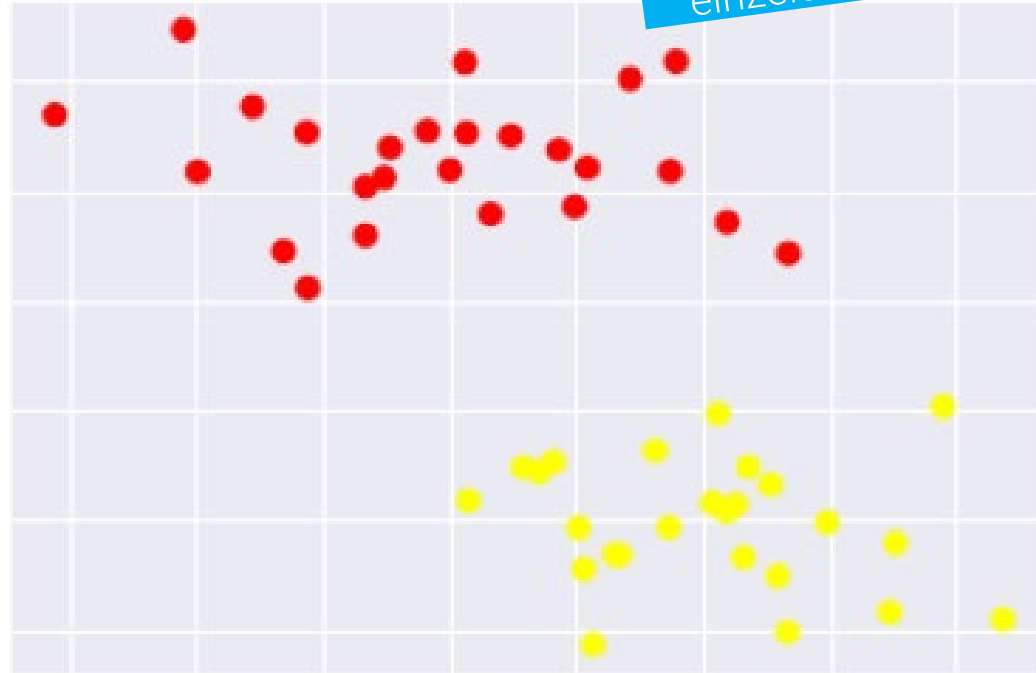
Was denken Sie?

Was heißt
diskriminativ?

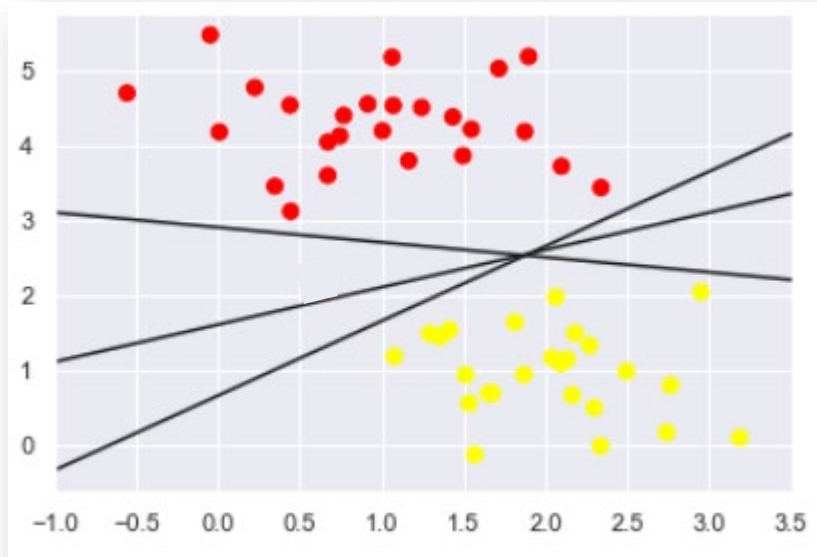
Wir als Klassifikatoren



Was denken Sie?
Wie würden Sie – als Klassifikator
– eine optimale Trennlinie
einzeichnen? (Zoom: zeichnen)



SVM: allgemein



- Wir können also beliebig viele Geraden einzeichnen, die unsere Klassen trennen
- Alle drei Geraden separieren unsere Klassen quasi ideal
- Wie sieht es nun mit **neuen Datenpunkten** aus? Je nachdem welche Gerade wir in unsere Ausgangssituation legen, können neue Datenpunkte unterschiedlich klassifiziert werden

```
1 xfit = np.linspace(-1, 3.5)
2 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
3 plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)
4
5 for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
6     plt.plot(xfit, m * xfit + b, '-k')
7
8 plt.xlim(-1, 3.5);
```



Was denken Sie?
Wie würden Sie intuitiv
das „x“ zuordnen – je
nach Gerade?



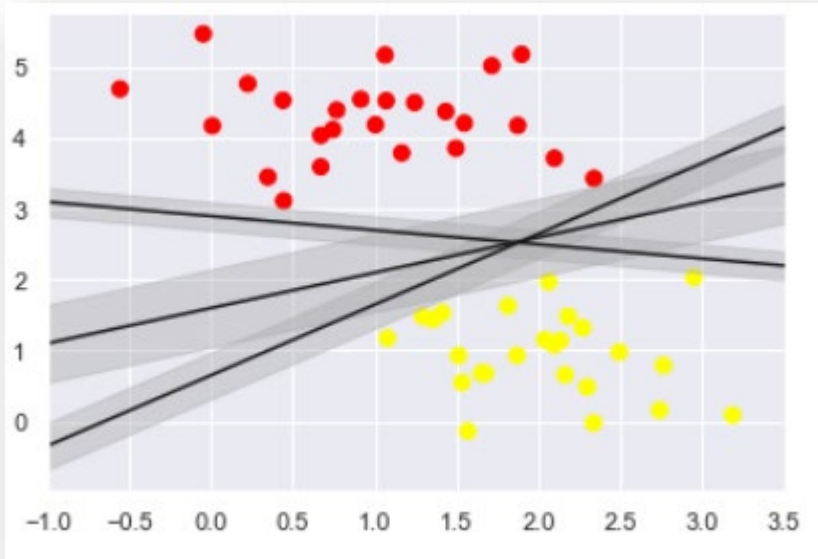
Was denken Sie?
Wie würden Sie intuitiv
entscheiden, welche
Gerade die beste ist?

SVM: Margin Maximierung



Was denken Sie?

Was könnte man noch tun – außer nur eine Gerade zu zeichnen?



```
1 xfit = np.linspace(-1, 3.5)
2 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
3
4 for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
5     yfit = m * xfit + b
6     plt.plot(xfit, yfit, '-k')
7     plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
8                     color='AAAAAA', alpha=0.4)
9
10 plt.xlim(-1, 3.5);
```

- Anstatt nur eine „nackte“ Gerade zur Trennung der Klassen zu nutzen, wird ein sog. *Margin* (dt. Rand, Spanne) um die Gerade gelegt
- Dieser Margin wird bis zu den nächstliegenden Punkten der Klassen erweitert
- Die optimale Gerade ist dann die mit dem größten Margin

→ SVMs sind sog. *Maximum Margin Estimators*



Was denken Sie?

Welche Gerade wird wohl die beste sein – und warum?

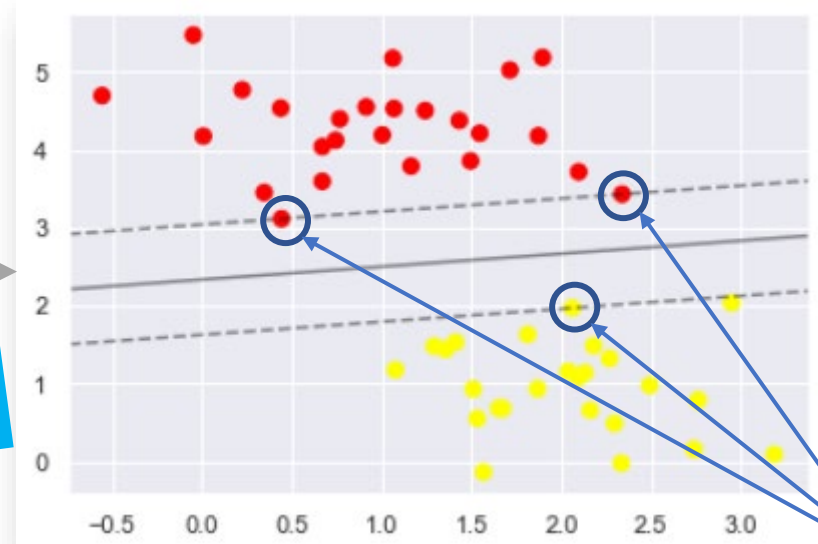
SVM: Fitting/Training

Modell

```
1 from sklearn.svm import SVC # "Support Vector Classifier"
2 model = SVC(kernel='linear', C=1E10)
3 model.fit(X, y)
```

Was denken Sie?

Können Sie sich vorstellen
woher das „Support Vector“
in SVM kommt?



Support Vectors

- Wie wir es bei der Einführung der `sklearn`-Syntax gelernt haben instanziiieren wir das SVM Modell

```
SVC(kernel=<kernel_type>, C=<regularization_parameter_value>)
```

Was `kernel` und `C` repräsentieren lernen wir später

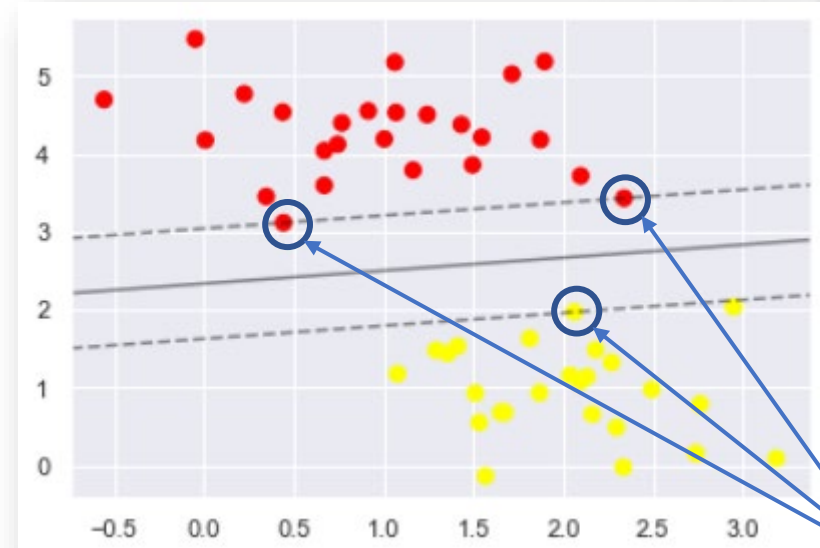
- Dann können wir die `.fit()`-Methode verwenden, um das Modell auf unsere Daten in `x` (Features) und `y` (Labels) anzupassen/fitten/trainieren
- Dadurch erhalten wir die Trennlinie mit dem *maximalen Margin*, die unsere beiden Klassen am besten trennt
- `SVC` bedeutet *Support Vector Classifier* – SVMs können also auch regressiv verwendet werden

SVM: Support Vectors

```
1 model.support_vectors_  
  
array([[ 0.44359863,  3.11530945],  
       [ 2.33812285,  3.43116792],  
       [ 2.06156753,  1.96918596]])
```



Was denken Sie?
Welchen? Von was
hängt das Modell ab?



Support Vectors

- Diese Support Vectors stellen einen entscheidenden Aspekt dieses ML-Modells dar
- Der Fit des Modells hängt **nur** von diesen Datenpunkten ab!
- Wir können über das Attribut `.support_vectors_` auf diese zugreifen

0

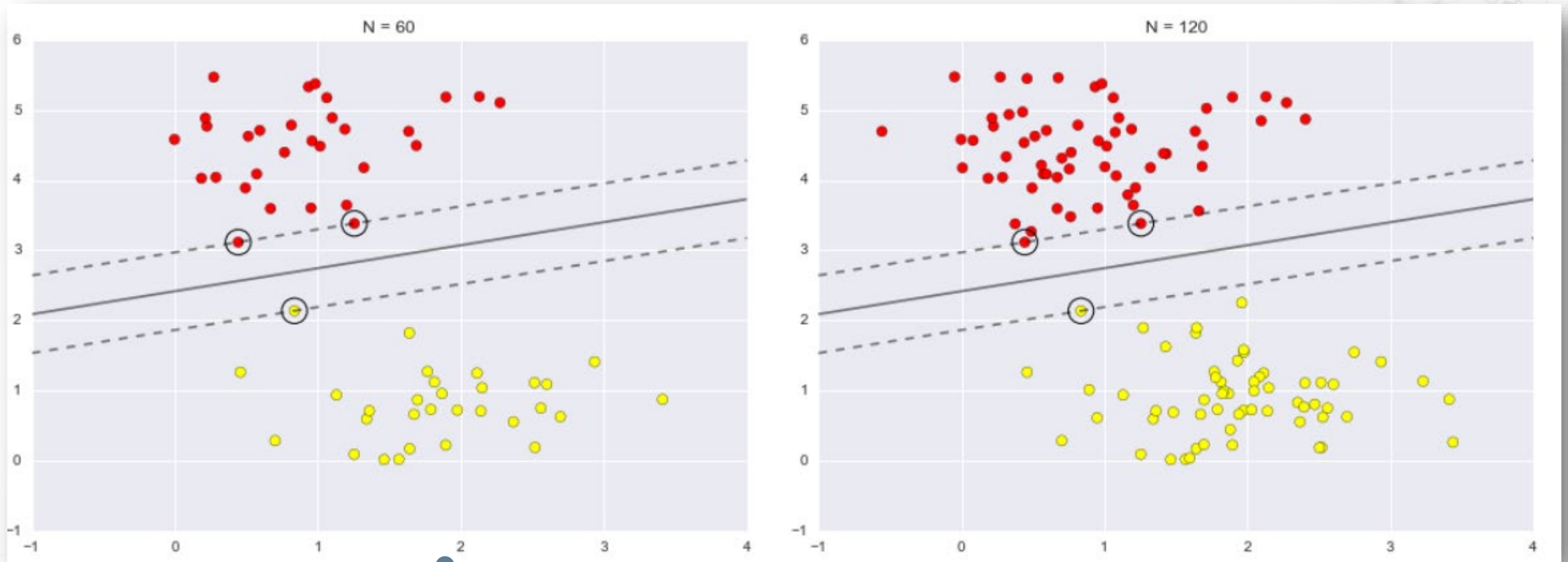
So what?

In höheren Dimensionen spricht man nicht mehr von Geraden, sondern Hyperebenen, die unsere Klassen trennen



Was denken Sie?
Wie sieht unser Modell in
höheren Dimensionen aus?

SVM: Support Vectors



0

So what?

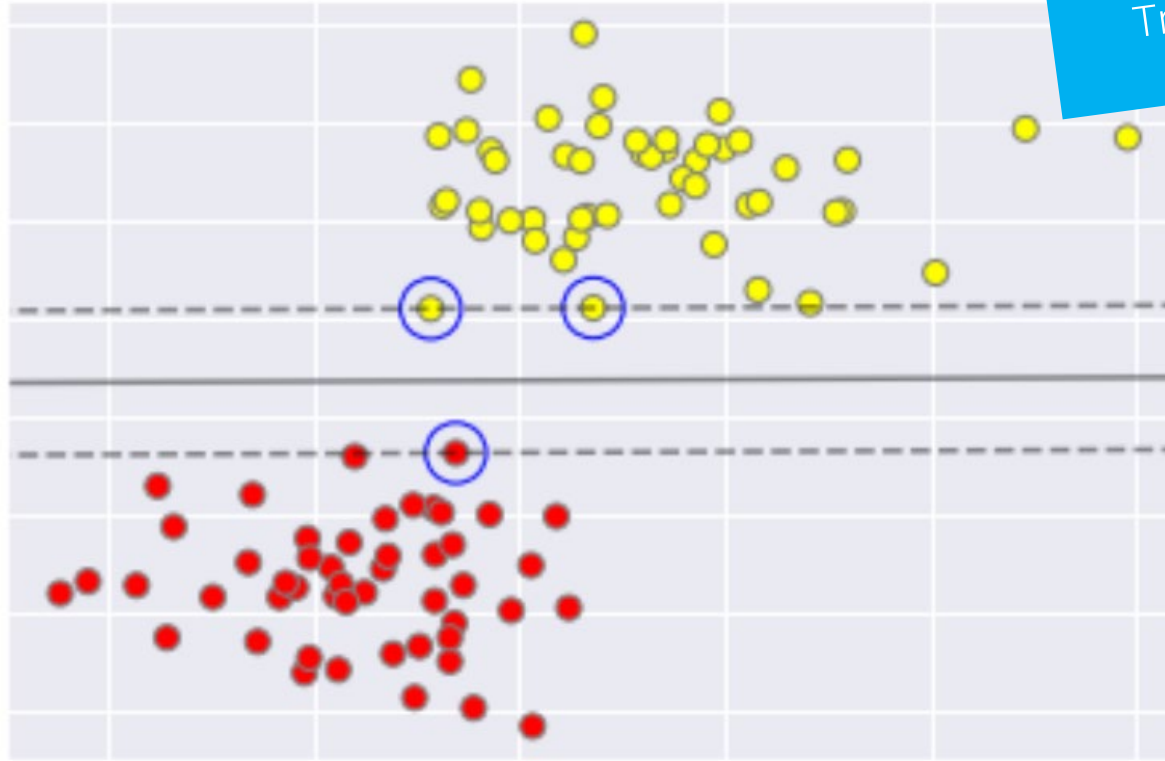
Diese Unempfindlichkeit des Modells auf entfernt liegende Punkt ist eine der Stärken dieser Art ML-Modelle

Wir als Klassifikatoren 2.0



Was denken Sie?

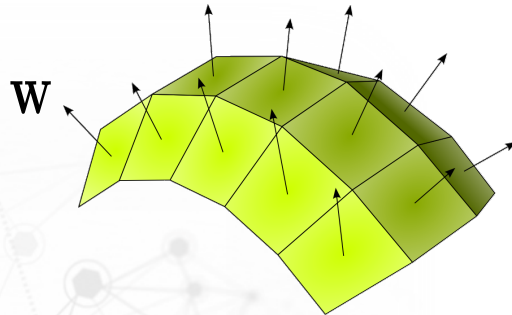
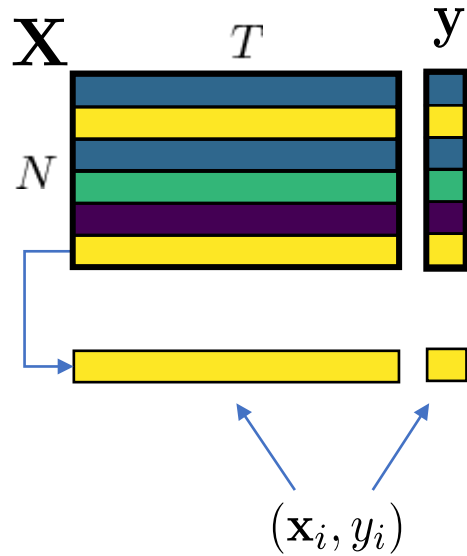
Wie würden Sie nun – als Klassifikator – eine optimale Trennlinie einzeichnen? (Zoom: zeichnen)



Was denken Sie?

Was sind die Support Vektoren? (Zoom: zeichnen)

SVM: Mathematik



Viele Normalenvektoren

Uns liegt ein historischer Datensatz bzw. Trainingsdatensatz vor

$$\{(\mathbf{x}_i, y_i) | i = 1, \dots, N, y_i \in \{-1, 1\}\}$$

Anhand dessen wird unsere Hyperebene gelernt. Wir beschreiben unsere Hyperebene mittels des Normalenvektors \mathbf{W} durch den Ursprung

und des Abstandes vom Ursprung b , den man *Bias* nennt

Mit diesen beiden Informationen haben wir eindeutig eine Hyperebene im Raum definiert



SVM: Mathematik

Für die Datenpunkte auf der Hyperebene gilt

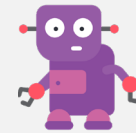
$$\langle \mathbf{w}, \mathbf{x}_i \rangle + b = 0$$

Für Punkte, die nicht auf der Hyperebene liegen ist diese Gleichung nicht 0, sondern positiv (Richtung \mathbf{w}) bzw. negativ (entgegengesetzt zu \mathbf{w})

Wenn also eine Hyperebene alle Punkte unserer Klassen voneinander trennt, dann weisen die Punkte einer Klasse alle entweder positive oder negative Vorzeichen auf

Für unsere Klassen gilt dann also

$$y_i = \text{sgn}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$$
$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 0$$



Unser Modell!

Aus den unendlich vielen Hyperebenen wählt man nun die, die den größten *Margin* zwischen den beiden Klassen aufweist



Was denken Sie?
Wie drückt man das
in einer Formel aus?



Was denken Sie?
Was gilt für Punkte,
die nicht auf der
Hyperebene liegen?



Was denken Sie?
Wie könnten wir das
mit einer Ungleichung
ausdrücken?



Was denken Sie?
Warum soll der Margin
maximiert werden?



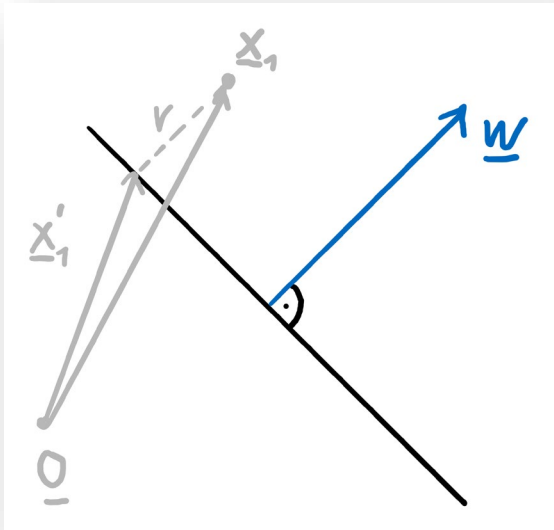
Was denken Sie?
Wie berechnet man also die
Klassen/Labels neuer Daten?



Was denken Sie?
Was sucht also der
Trainingsprozess?

SVM: Mathematik

$$\mathbf{x}_1 = \mathbf{x}'_1 + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$$



Was bedeutet ein **maximaler Margin** mathematisch für uns?

Der Margin wird durch den Abstand zu den nächsten Punkten der Klassen zur Hyperebene definiert – wir brauchen also einen Abstand

Mit diesem Abstand r können wir nun unsere **Bedingung**, auf den jeweiligen Seiten der Hyperebene nur Datenpunkte je einer Klasse vorzufinden formulieren

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq r$$

Wir können nun ein Optimierungsproblem formulieren

$$\max_{\mathbf{w}, b} r$$

unter den Bedingungen

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq r \quad \|\mathbf{w}\| = 1 \quad r > 0$$

SVM: Mathematik

Der Normalenvektor sei nun **nicht mehr normalisiert**. Um das Optimierungsproblem *berechenbar* zu machen, müssen wir eine kleine Umformung machen: durch eine **Skalierung** der Daten auf die Bedingung

$$\langle \mathbf{w}, \mathbf{x}_a \rangle + b = 1$$

mit \mathbf{x}_a als der nächste Punkt zur Hyperebene (Support Vector) und den vorher definierten Abstand kann gezeigt werden, dass

$$r = \frac{1}{\|\mathbf{w}\|}$$

und für unsere Bedingung gilt dann

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

Damit reduziert sich unser Problem auf

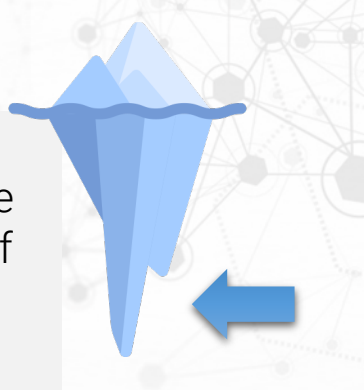
$$\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \quad \text{bzw.} \quad \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

unter der Bedingung

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

?

Was denken Sie?
Wieso heißen diese
Bedingungen Hard Margin?

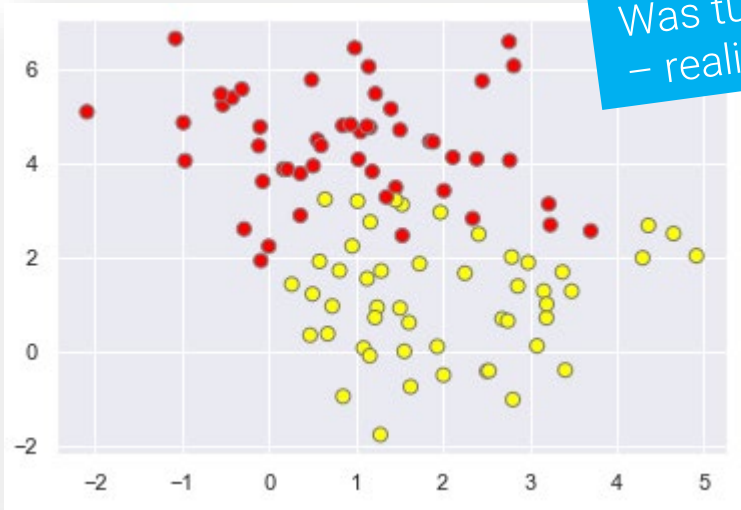


SVM: Hard- und Soft-Margin



Was denken Sie?

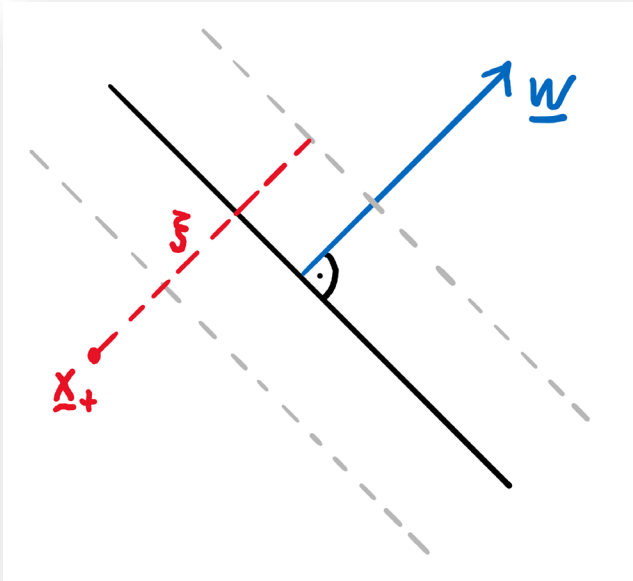
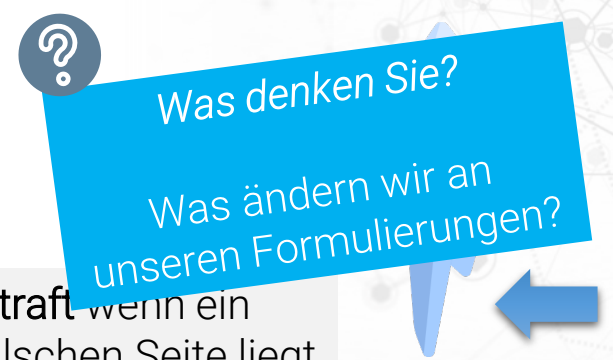
Was tun Sie in so einem
– realistischeren – Fall?



```
1 X, y = make_blobs(n_samples=100, centers=2,  
2                   random_state=0, cluster_std=1.2)  
3 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn', edgecolors='gray');
```

- Bei SVMs hat man die Möglichkeit die Margins entweder starr bzw. *hard* oder weich bzw. *soft* zu definieren
- In anderen Worten: lässt man zu, dass Datenpunkte innerhalb des Margins auftauchen dürfen – Leitfrage: wie viele bzw. zu welchem Ausmaß?
- Hierzu gibt es einen Hyperparameter, der getuned werden muss – in `sklearn` ist es das Argument `C`
- Große Werte entsprechen einem Hard-, kleine Werte einem Soft-Margin
- Der optimale Wert von `C` hängt also vom Datensatz ab und muss via z.B. Cross-Validation bestimmt werden
→ Zuerst schauen wir, wo in der Mathematik dieses `C` zu finden ist

SVM: Soft-Margin: Mathematik



Wir führen eine sog. *Slack Variable* ein, die **bestraft** wenn ein Datenpunkt innerhalb des Margins bzw. auf der falschen Seite liegt – dies ermöglicht uns im Umkehrschluss: Verletzungen des Margins werden zugelassen!

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i$$

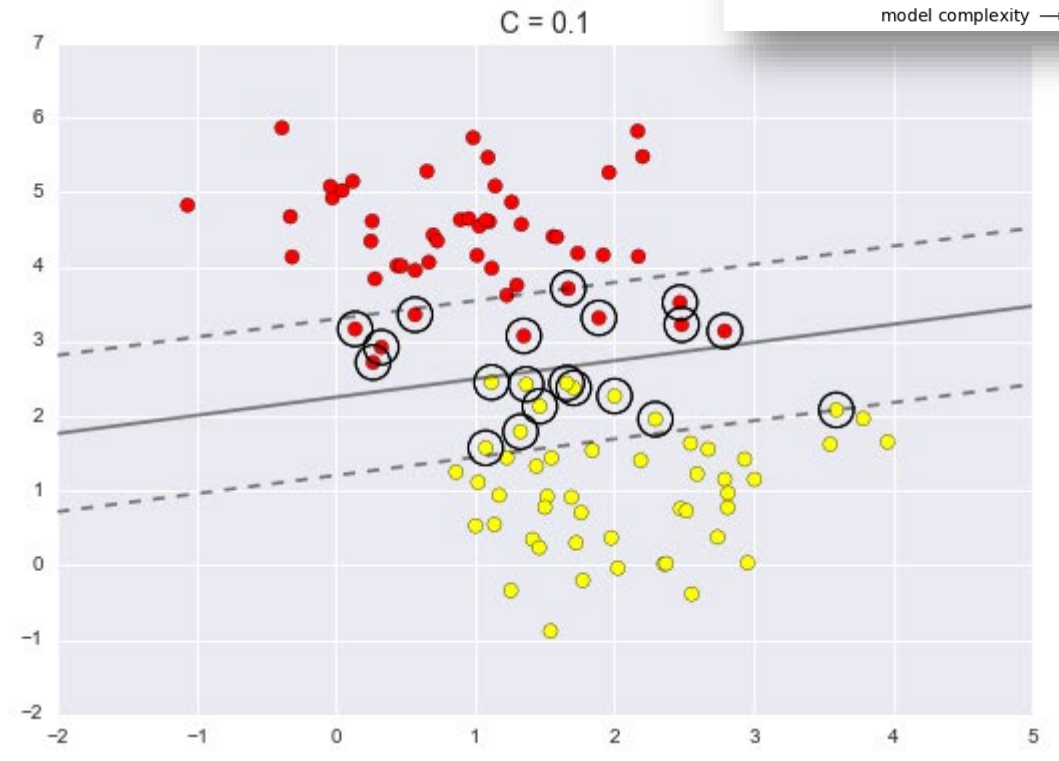
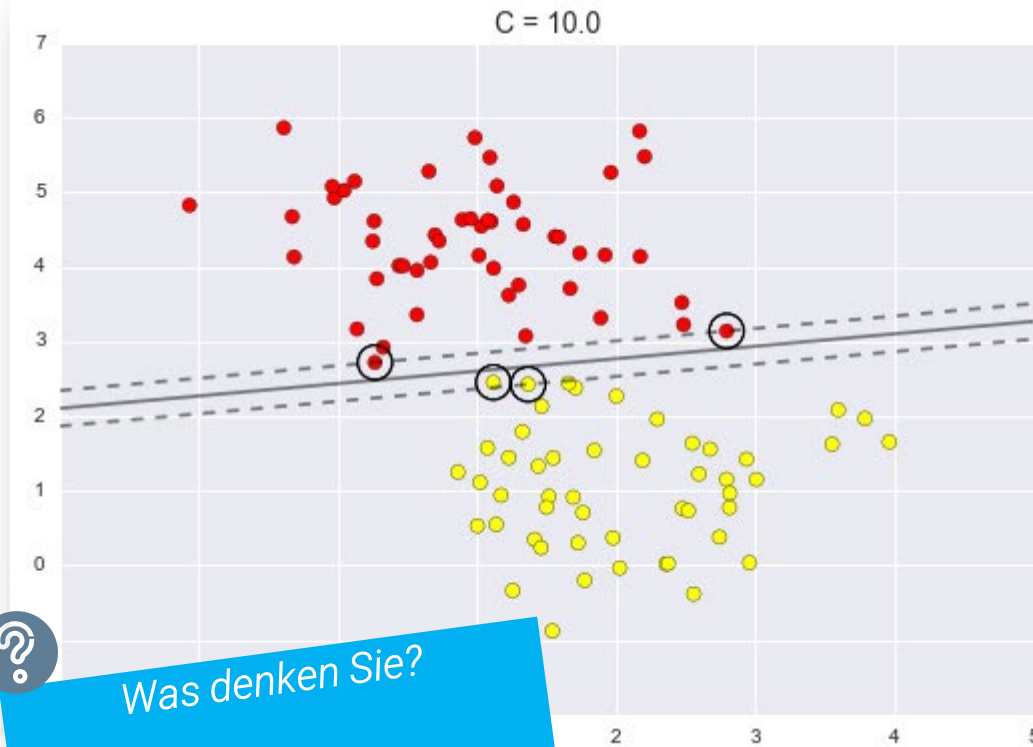
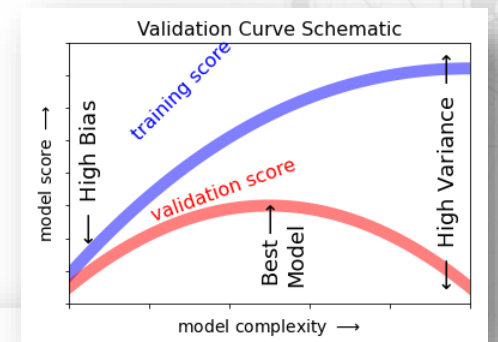
mit den Bedingungen

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{und} \quad \xi_i \geq 0$$

Der Parameter C versucht also eine Balance bzw. einen Trade-Off zwischen „Größe des Margins“ und „Verletzung dessen“ herzustellen

Zu diesem Parameter sagt man **Regularisierungsparameter**

SVM: Hard- und Soft-Margin



Was denken Sie?

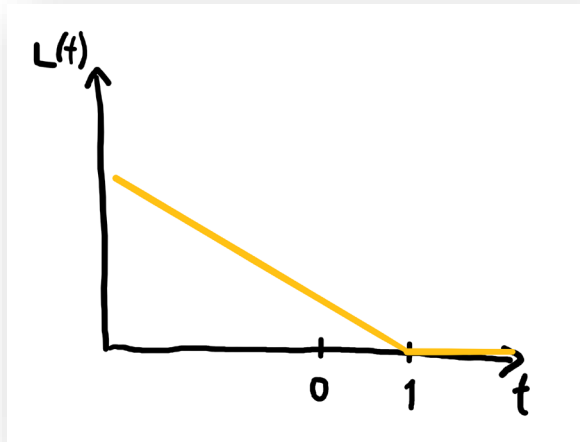
Was entspricht einer hohen, was einer niedrigen Modellkomplexität?

SVM: Optimierung mittels Loss Function



Was denken Sie?

Wie würden Sie die Loss Function aufstellen?



Die naheliegendste Formulierung einer Loss Function wäre die Accuracy – also: wie gut decken sich Vorhersage und Labels

Diese ist jedoch ungeeignet, da es sich hierbei um ein sog. *kombinatorisches Optimierungsproblem* handelt

Eine *kontinuierliche Loss Function* ist also anzustreben – mögliche Formulierung: **Hinge Loss**

$$L(t) = \max\{0, 1 - t\} \quad \text{mit} \quad t = yf(\mathbf{x}) = y(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

Damit ergibt sich unser letztendlich optimierbares Problem zu

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\}$$

Hierzu sagt man auch **Regularisierung** – Margin Maximierung ist also Regularisierung

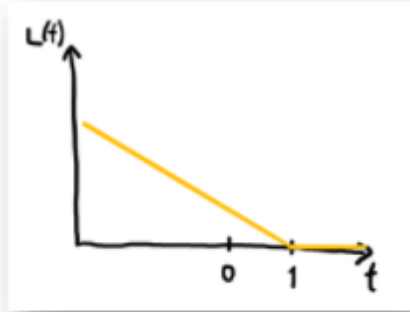
Anmerkung: dies ist auch gleichbedeutend mit der Minimierung der eingeführten Slack-Variablen



Was denken Sie?

Wie schaut die math. Formulierung hierzu aus?

Beispiel: SVM from Scratch: Loss Function



Wir haben gesehen, dass man durch ein paar mathematische Umformungen zu einem übersichtlichen Optimierungsproblem bzw. einer relativ überschaubaren Kostenfunktion gelangt. Wir wollen uns nun in diesem Beispiel damit beschäftigen die Loss Function

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\}$$

als Funktion in Python abzubilden.



Was denken Sie?

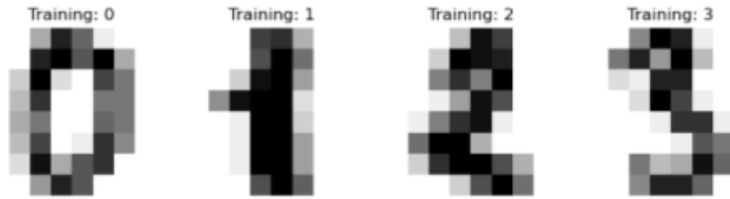
Ohne es hier schon behandelt zu haben: haben Sie eine Vermutung wie man damit nun die optimalen Modellparameter findet?



So what?

Nun wissen wir wie die SVM als mathematisches Modell aussieht und welche Kostenfunktion minimiert werden soll. Das Werkzeug für die Minimierung – z.B. den sog. *Gradientenabstieg* – sehen wir uns in einer späteren Veranstaltung an.

Beispiel: SVM zur Handschrifterkennung



Stellen Sie sich vor Sie müssten einem Modell aus handschriftlichen Notizen beibringen geschriebene Zahlen zu erkennen. Dieser Use Case kann Ihnen an vielen Stellen begegnen - z.B.

- wenn Sie als Post von Briefen die geschriebene Postleitzahl und Hausnummer digitalisieren müssten und so die Weiterverarbeitung von Briefen beschleunigen würden
- wenn in hochautomatisierten Produktionsanlagen in Bauteilen eingestanzte Seriennummern erkannt und digitalisiert werden müssen
- wenn Sie im Rahmen einer Absatzprognose Notizen von Mitarbeitern erhalten, auf denen noch Absatzzahlen handschriftlich festgehalten werden. In diesem Beispiel wollen wir eine SVM nutzen, um genau diesen Use Case zu lösen. Hierzu nutzen wir das sog. *Digits-Dataset* aus `sklearn`, in dem eine Vielzahl handschriftlicher Zahlen in `(8, 8)` großen Arrays abgespeichert sind. Unser Ziel ist es, dass die SVM so trainiert wird, dass sie **ungesehene** handschriftliche Zahlen in die richtige Klasse (die zugrundeliegende Zahl) einordnen kann. Bemerkung: wir kommen hier mit einer *Multi-Class SVM* in Berührung.

0

So what?

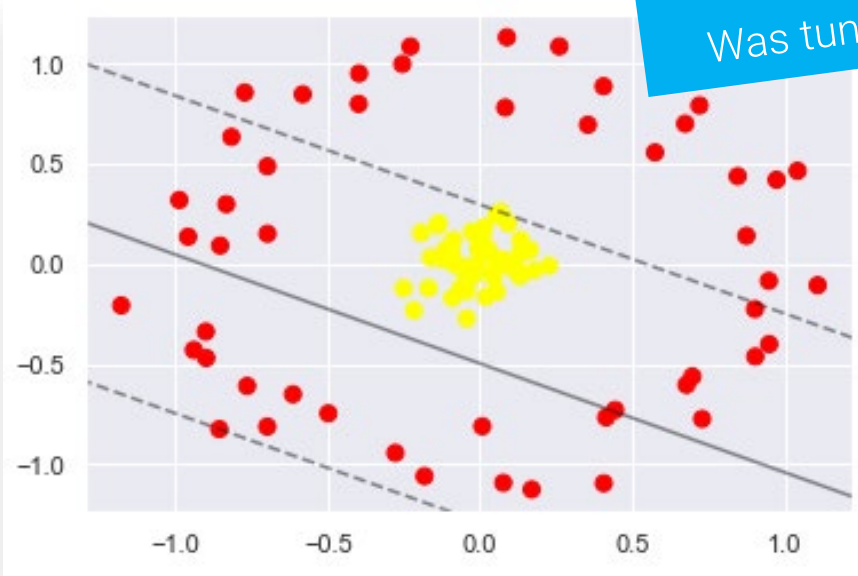
Hier haben wir etwas neues kennengelernt: SVMs können auch für Multi-Class-Probleme verwendet werden (sprechen wir später kurz an)

SVM: Nicht-lineare Probleme



Was denken Sie?

Was tun Sie hier?

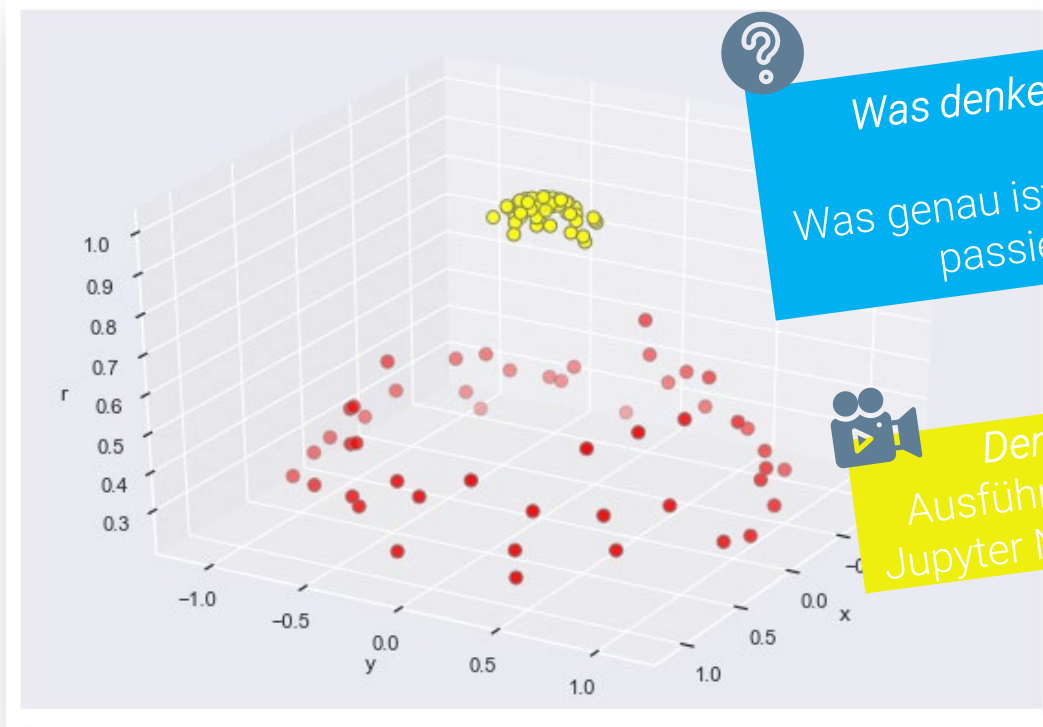


```
1 from sklearn.datasets.samples_generator import make_circles
2 X, y = make_circles(100, factor=.1, noise=.1)
3
4 clf = SVC(kernel='linear').fit(X, y)
5
6 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
7 plot_svc_decision_function(clf, plot_support=False);
```

- Bei unserem Digits-Beispiel haben wir implizit angenommen, dass wir die Klassen *linear* – also mit einer geraden Linie – trennen können. Was passiert wenn dem nicht so ist?
- Keine lineare Hyperebene wird die beiden Klassen jemals trennen können
→ Es liegt also ein **nicht-linear separierbares** Problem vor
- SVMs sind in der Lage auch solche Probleme zu lösen

Lösung: Projektion der Daten in einen höherdimensionalen Raum!

SVM: Nicht-lineare Probleme



- Eine Möglichkeit eine neue Dimension an unseren Raum hinzuzufügen ist z.B. sogenannte *Radial Basis Functions* zu verwenden
- RBFs sind skalare Funktionen $\phi(r)$, die nur vom Abstand zum Ursprung abhängen
- Dadurch machen wir also aus unserem 2D- einen 3D-Datensatz

→ Das Problem wird in einer höheren Dimension linear separierbar

```
1 def plot_3D(elev=30, azim=30, X=X, y=y):
2     ax = plt.subplot(projection='3d')
3     ax.scatter(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn', edgecolors='gray')
4     ax.view_init(elev=elev, azim=azim)
5     ax.set_xlabel('x')
6     ax.set_ylabel('y')
7     ax.set_zlabel('r')
8
9 plt.figure(figsize=(10, 7))
10 plot_3D()
```

Was denken Sie?
Welches Problem sehen Sie hier generell?

```
1 np.concatenate((X, r.reshape(-1, 1)), axis=1)

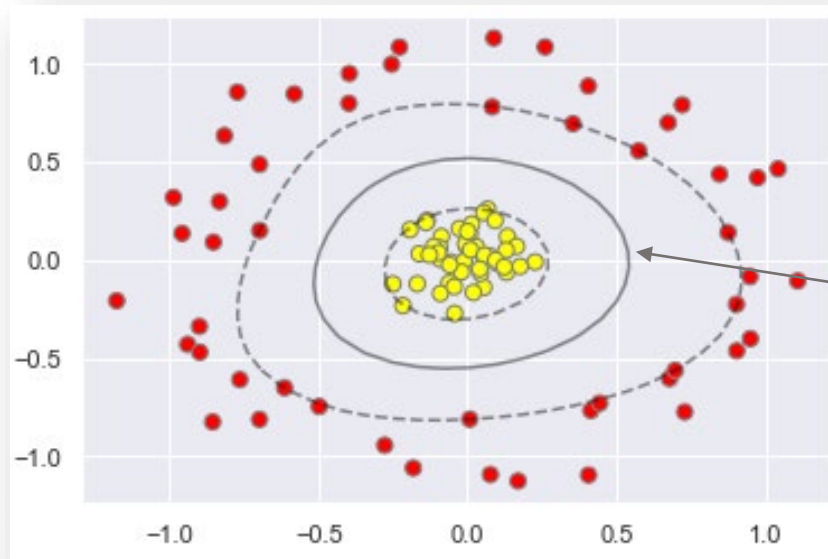
array([[ -8.98674223e-01, -3.38601143e-01,  3.97616982e-01],
       [ 1.04061333e+00,  4.64382873e-01,  2.72933376e-01],
       [ 6.75066947e-02,  2.58705545e-01,  9.31009579e-01],
       [-3.21867918e-03,  7.00834554e-02,  9.95090043e-01],
       [ 7.27800725e-01, -1.153768e-01,  3.23203690e-01])
```

Was denken Sie?
Wieso hängt r hier nur vom Ursprung ab?

```
1 r = np.exp(-(X ** 2).sum(1))
```

SVM: Kernel-Trick

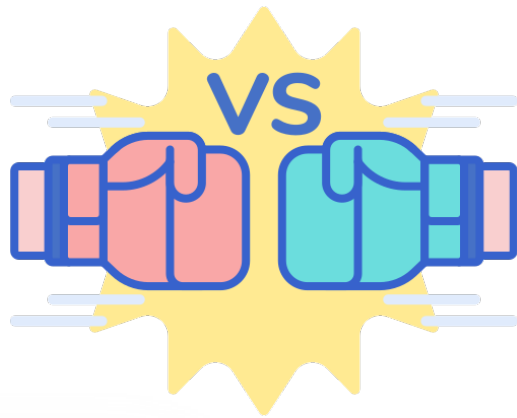
```
1 clf = SVC(kernel='rbf', C=1E6)
2 clf.fit(X, y)
```



- Diese Projektion in höhere Dimensionen wird durch den sog. *Kernel-Trick* auf die Spitze getrieben
- Man kann **implizit** an jeden der N Datenpunkt eine zusätzliche Dimension anfügen – ohne diese Projektion wirklich durchzuführen
- In diesem N -dimensionalen Raum wird dann eine linear separierende Hyperebene gesucht
- Diese lineare Hyperebene im N -dimensionalen Raum wird im ursprünglichen, niedrigdimensionalen Raum zu einer nicht-linearen Hyperebene
- In sklearn wird das mittels des Hyperparameters `kernel='rbf'` realisiert

```
1 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn', edgecolors='gray')
2 plot_svc_decision_function(clf)
3 plt.scatter(clf.support_vectors[:, 0], clf.support_vectors[:, 1],
4             s=300, lw=1, facecolors='none');
```


Multi-Class SVMs



- Liegen mehr als zwei Klassen vor, so lässt sich obiger Ansatz nicht eins-zu-eins anwenden
- Eine Möglichkeit mehrere Klassen zu unterscheiden ist: für **jede Klassenpaarung** einen Klassifikator bzw. eine SVM zu trainieren
- Diesen Ansatz nennt man „one-versus-one“ – im Gegensatz zu „one-versus-rest“
- Für N Klassen müssen dann
$$\frac{N(N-1)}{2}$$
SVMs trainiert werden
- Die Entscheidungsgrenzen ergeben sich dann aus der Kombination aller Klassifikatoren



Was denken Sie?

Sie kennen jetzt eine SVM, die zwei Klassen unterscheiden kann: wie funktioniert das bei mehreren Klassen?



Was denken Sie?

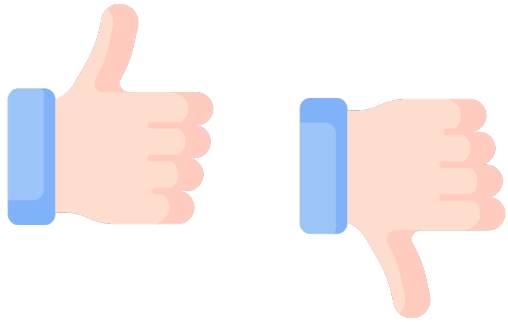
Was könnte „one-versus-rest“ hier bedeuten?



Was denken Sie?

Wie viele SVMs müssen dann erzeugt werden?

Vorteile, Nachteile, Besonderheiten: SVM



Pros:

- SVMs sind sehr effektiv bei höherdimensionalen Datensätzen
- SVMs helfen in Fällen, in denen die Anzahl der Dimensionen größer ist als die Anzahl der Datenpunkte
- SVMs sind komputativ effizient
- Pädagogisch: SVM lassen geometrische Interpretationen zu

Cons:

- SVMs haben Probleme mit sehr großen Datensätzen
- SVMs haben Probleme bei stark überlappenden Klassen bzw. viel Rauschen in den Daten
- Probabilistische Interpretation nicht möglich

Beispiel: SVM zur Face Recognition



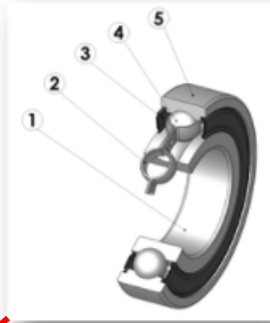
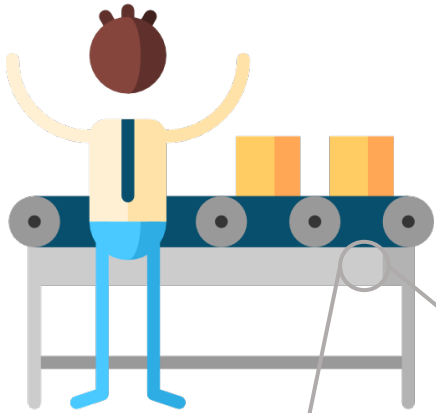
Wir wollen uns in diesem Beispiel gleich einem interessanten Anwendungsfall im maschinellen Lernen widmen: der Gesichtserkennung. Hierzu laden wir uns mittels eines sog. *Fetchers* aus `sklearn.datasets` einen Datensatz, der Bilder von Gesichtern berühmter Personen beinhaltet. Unser Ziel ist es eine SVM zu trainieren, die diese Gesichter erkennen - also **klassifizieren** - kann. Zu jeder Person gibt es eine Vielzahl verschiedener Aufnahmen - jede Person stellt also eine **Klasse** dar. Die Bilder haben die Dimensionalität `(62,47)` und somit ca. 3000 Pixel. Wenn jeder Pixel einer Dimension in einem Raum entsprechen würde, dann wäre ein Gesicht ein Datenpunkt in diesem Raum. Dieser Raum ist uns zu hochdimensional. Daher *projizieren* wir die Gesichter in einen niedrigdimensionaleren Raum (wir greifen hier den Themenblock PCA vorweg).

0

So what?

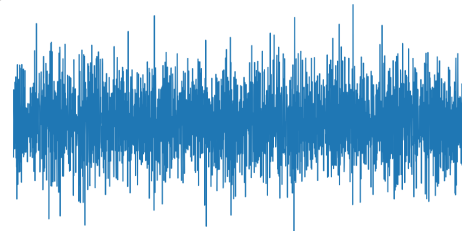
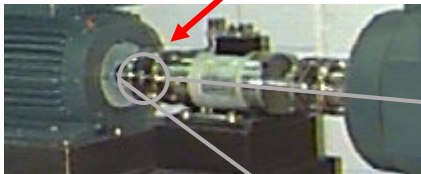
Wir haben unser erstes, *quantitatives* Gütemaß kennengelernt: den `accuracy_score` aus `sklearn.metrics`
→ Entspricht der gewöhnlichen Genauigkeit: wie viel Prozent der Labels im Testdatensatz wurden korrekt vorhergesagt

Beispiel: Fault Classification of Bearing Faults



(Bildquelle: Wikipedia)

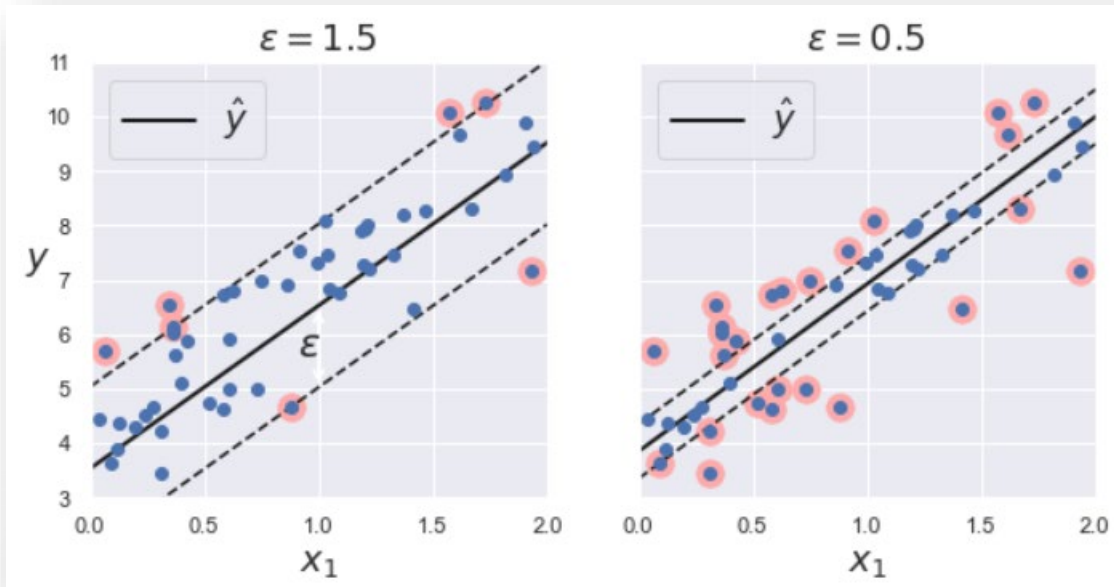
In diesem Beispiel wollen wir uns dem Aspekt der *Fault Classification* annehmen. Hierzu nutzen wir Vibrationsdaten von Kugellagern, die verschiedene Schadensklassen aufweisen. Diese Kugellager sind in einem Servomotorverbaut. Neben den "Gesunddaten" liegen uns Daten von Beschädigungen an einer Kugel (B021), des Innenrings (IR021) und des Außenrings (OR021@6) vor. Wir wollen diese Zeitserien in *Fenster* aufteilen, aus diesen dann bestimmte einfache Features extrahieren und anhand derer dann eine SVM trainieren, die uns ungesehene Daten dann klassifizieren soll.



SVR: Support Vector Regression



Was denken Sie?
Wieso benötigen wir
nun ein Epsilon?



Was denken Sie?
Wie könnte nun unsere
Kostenfunktion aussehen?



Was denken Sie?
Was ist der clevere Trick, der
uns von einer Klassifikation
zu einer Regression bringt?

- SVMs können auch zur Regression verwendet werden – hierzu sagt man dann *Support Vector Regression*
- Hierzu kehrt man unsere Optimierungsbedingung um – wir suchen nun nach einer Hyperebene plus Margin, **innerhalb** dessen sich **so viele Datenpunkte wie möglich** befinden
- Auch die Bestrafung muss hierzu umgekehrt werden: Datenpunkte **außerhalb** des Margins führen zu Penalties!
- Uns liegen in dieser Situation keine wirklichen Support Vectors zwischen verschiedenen Klassen mehr vor → Ein weiterer Hyperparameter ε nötig, der variiert werden muss
- Die Kostenfunktion der SVR ist dann:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \max\{0, |y_i - (\langle \mathbf{w}, \mathbf{x}_i \rangle + b)| - \varepsilon\}$$

SVR: Support Vector Regression: sklearn

```
1 # Some data
2 np.random.seed(42)
3 m = 100
4 X = 2 * np.random.rand(m, 1) - 1
5 y = (0.2 + 0.1 * X + 0.5 * X**2 + np.random.randn(m, 1)/10).ravel()
6
7 # Show
8 plt.scatter(X, y);
```



```
1 from sklearn.svm import SVR
2
3 svm_reg1 = SVR(kernel="rbf", C=100, epsilon=0.1)
4 svm_reg2 = SVR(kernel="rbf", C=0.01, epsilon=0.1)
5 svm_reg1.fit(X, y)
6 svm_reg2.fit(X, y)
```

- In sklearn haben wir eine entsprechende Klasse vorliegen

`SVR(kernel=<kernel_type>,
C=<regularization_parameter_value>,
epsilon=<margin_width>)`

- Auch hier gilt wieder: wir können sowohl lineare, als auch nicht-lineare Probleme lösen → der Kernel-Trick findet auch hier Anwendung

