Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
2 Klassifikation
2.3 Mathematisch

Werner Dähler 2024

2 Klassifikation - AGENDA

21. Instanzbasierte Modelle
22. Regelbasierte Modelle
23. Mathematische Modelle
    231. LinearDiscriminantAnalysis
    232. SVC
    233. GaussianNB
    234. LogisticRegression
24. Neuronale Netze
25. Multiklass Klassifikation

2.3 Klassifikation - Mathematisch

Unterschiede gegenüber Regelbasierten Modellen:

- z.T. Methoden der klassischen Statistik, deren Anfänge ins 19. Jahrhundert zurückreichen (Diskriminanzanalyse, Logistische Regression)

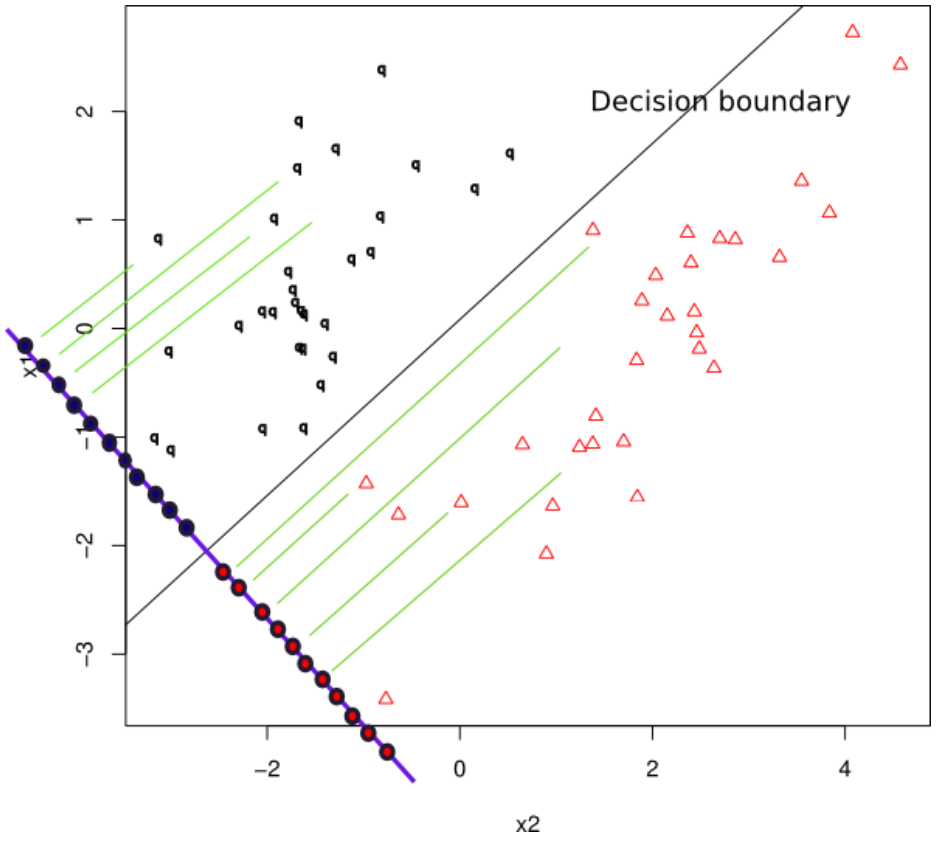
- basieren z.B. auf Methoden der Linearen Algebra (Matrix-Kalkül) und sind daher entsprechend schnell,
  aufgrund deren mathematischer Komplexität wird aber hier nicht vertieft darauf eingegangen

- andere Methoden (SVC, NaiveBayes, Neuronale Netze) sind dagegen jüngeren Datums, basieren aber ebenfalls auf aufwändigen mathematischen Verfahren, welche dank dem Einsatz von Informatikmitteln aber erst in der zweiten Hälfte des 20. Jahrhunderts praktikabel wurden
- dabei handelt es sich zumeist um iterative Näherungsmethoden, welche entsprechend ressourcenhungrig sind

2.3.1 Klassifikation - Mathematisch - LinearDiscriminantAnalysis

2.3.1.1 Theorie

- ein geometrisches Trennverfahren
- basierend auf Arbeiten von Ronald Fisher aus den 1930-er Jahren
- gesucht ist die Hyperebene, welche zwei (oder mehr) Gruppen möglichst gut linear voneinander trennt
- Diskriminanzachse: Achse, auf welcher Projektionen der beiden Gruppen möglichst gut separiert sind
- Trenngerade: steht senkrecht auf Diskriminanzachse und kennzeichnet die gefundene Grenze zwischen den Gruppen
- das Verfahren kann selbstverständlich auf mehrdimensionale Fragestellungen angewendet werden, ist dann aber nicht mehr visualisierbar



2.3.1 Klassifikation - Mathematisch - LinearDiscriminantAnalysis

2.3.1.2 Praxis

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model = LinearDiscriminantAnalysis()
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

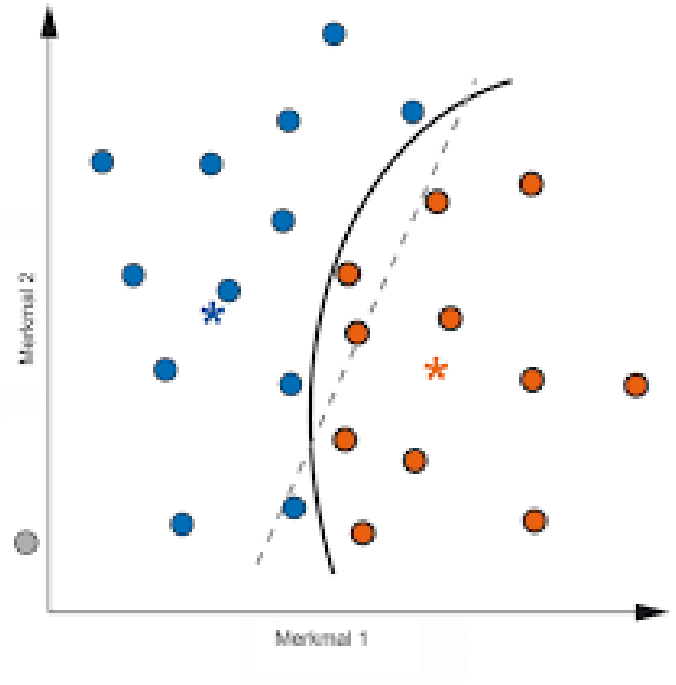
```
0.8487982963188317
```

- Lineare Diskriminantzanalyse ist ein klassischer Klassifikator mit einer Linearen Trennebene, basierend auf Linearer Algebra
- ist daher attraktiv (schnell), da er auf geschlossenen Lösungen basiert und ausserdem inhärent multiklass tauglich ist
- verfügt über keine Tuning Parameter

2.3.1 Klassifikation - Mathematisch - LinearDiscriminantAnalysis

2.3.1.3 QuadraticDiscriminantAnalysis

- eine Variante von LinearDiscriminantAnalyses
- im Gegensatz zu ersterer LDA wird hier nicht eine lineare Trennebene gesucht, sondern eine quadratische (binomial)
- ansonsten bleibt das Verfahren gleich




```
from sklearn.discriminant_analysis \
    import QuadraticDiscriminantAnalysis
model = QuadraticDiscriminantAnalysis()
:
```

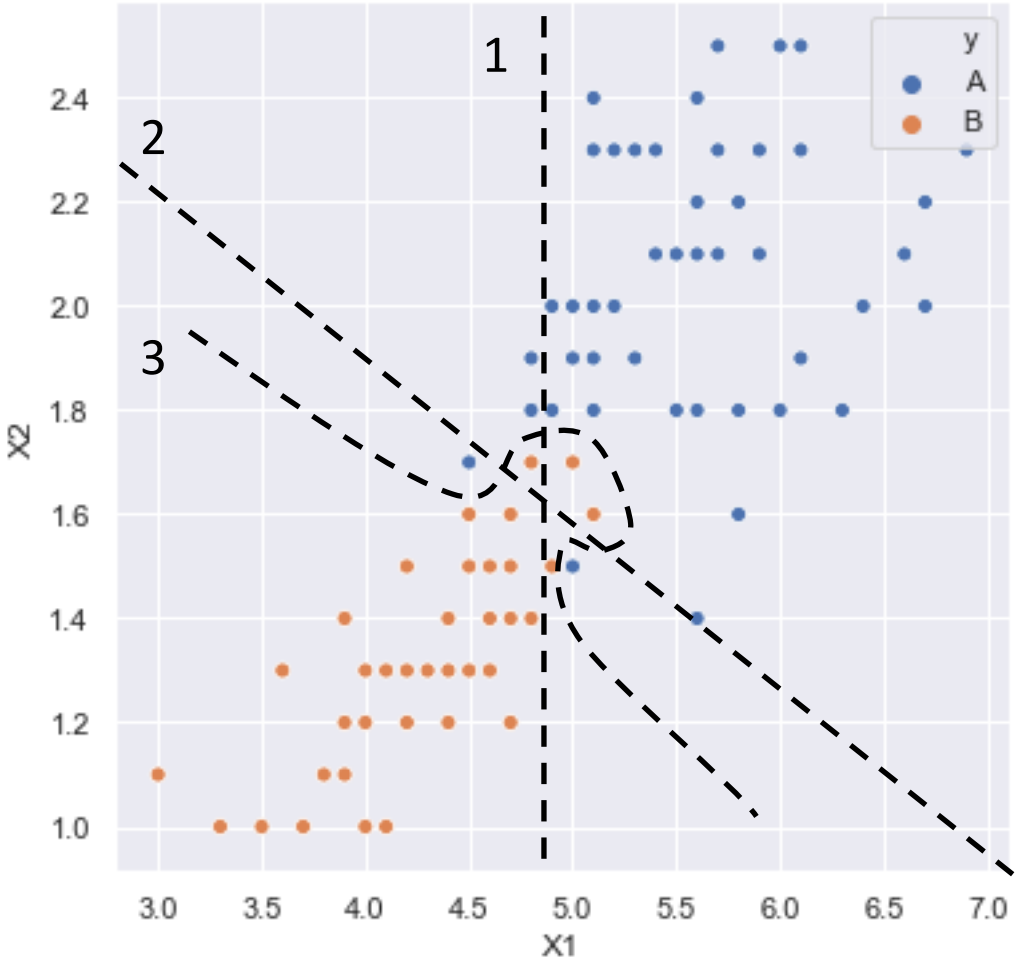
```
0.7246729540614543
```

- auch wenn die Performance auf dem Bankkunden Dataset nicht berauschend ist, kann dieses Verfahren für andere Datenlagen durchaus gute Ergebnisse liefern und sollte zumindest für erste Modellvergleiche auch in Erwägung gezogen werden

2.3.2 Klassifikation - Mathematisch - SVC

2.3.2.1 Theorie

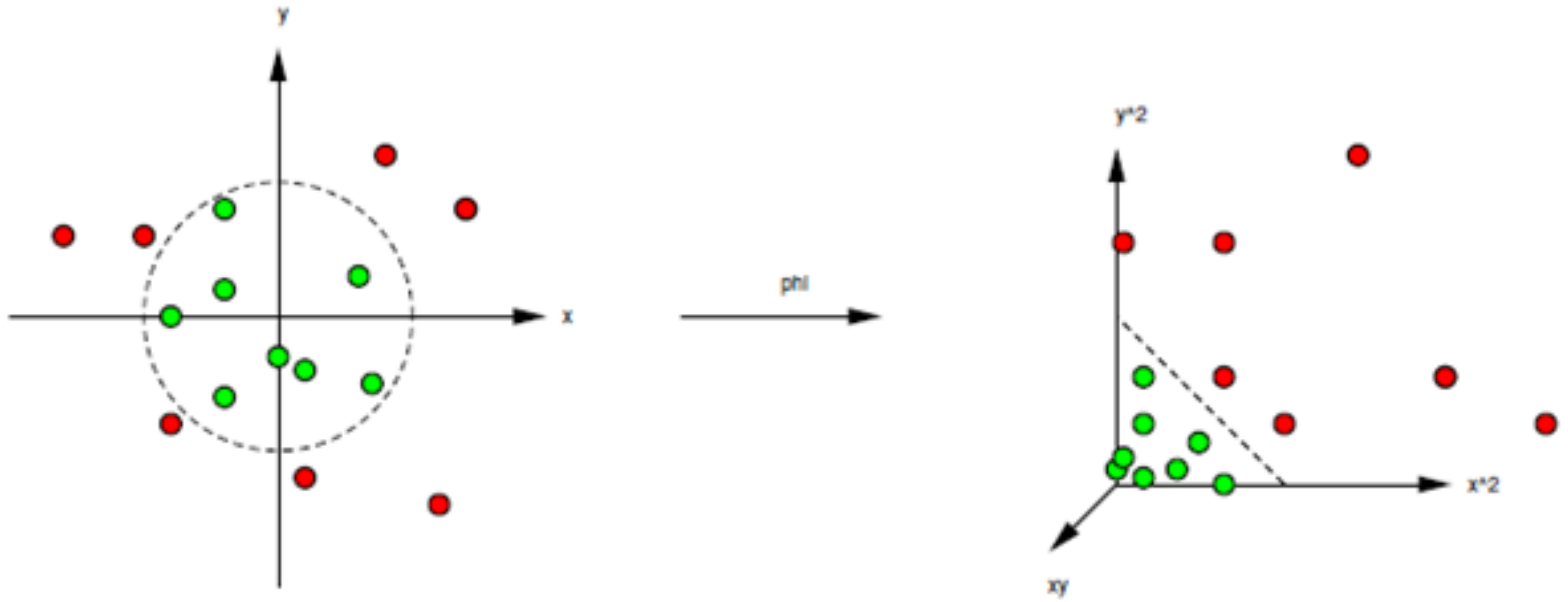
- eigentlich svm.SVC, die Klassifikationsfunktion (Objekt) welche auf **Support Vektor Maschine** beruht [Wikipedia]
- Idee: "Verbiegen" des multidimensionalen Datenraums, so dass lineare Trennungen zwischen den Klassen möglich werden
- am Beispiel des Demodataset:
  1. Entscheidungsbaum sucht beste Trennregel aufgrund einzelner Merkmale, abtrennen von y aufgrund einer einzigen Regel (1) auf "X1"
  2. z.B. Diskriminanzanalysen: trennen der Gruppen in y wäre besser aufgrund von Linearkombinationen von X1 und X2
  3. gesucht: eine nichtlineare Funktion, welche die beiden Klassen (möglichst) vollständig trennt



2.3.2 Klassifikation - Mathematisch - SVC

2.3.2.1 Theorie

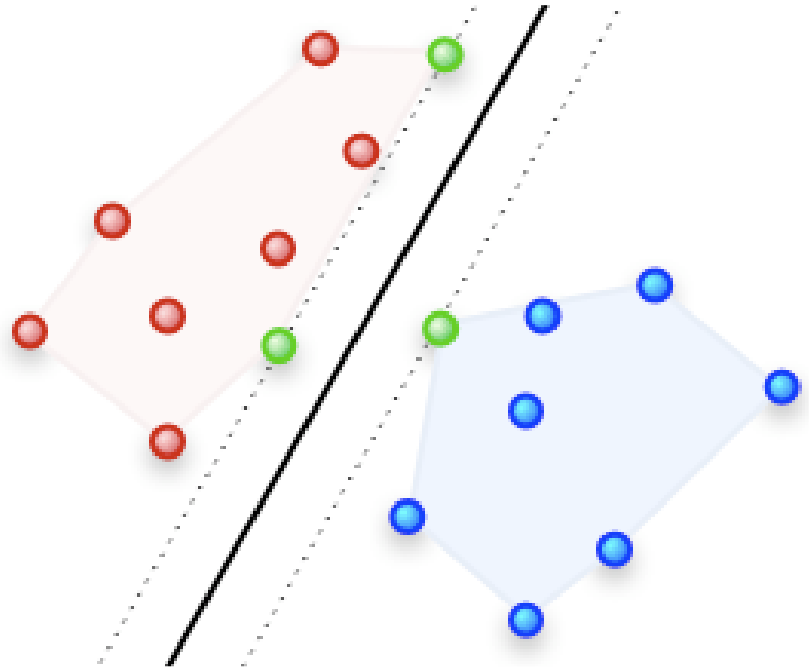
- Support Vektor Maschinen "verbiegen" den Raum durch Hinzufügen weiterer latenter Dimensionen, bis die verlangten Gruppen durch eine Lineare Funktion getrennt werden können
- anschliessend wird die gefundene Trennfunktion zurück transformiert und erscheint dann im Ausgangsraum als nicht lineare Trennfunktion



2.3.2 Klassifikation - Mathematisch - SVC

2.3.2.1 Theorie

- Support Vektoren: Datenpunkte, die in der unmittelbaren Umgebung der Trennzone liegen (in der Grafik grün dargestellt), daher die Bezeichnung



- weiter davon entfernten Punkte interessieren dagegen nicht
- die Trennzone wird so gelegt, dass sie einen möglichst breiten "Kanal" bildet
- eine direkte Interpretation der gefundenen Hyperräume ist nicht möglich und interessiert hier auch nicht

2.3.2 Klassifikation - Mathematisch - SVC

2.3.2.1 Theorie

- zur Illustration eine Visualisierung der Supportvektoren im demo_data_class Dataset,
  die Support Vektoren sind speziell ausgezeichnet

2.3.2 Klassifikation - Mathematisch - SVC

2.3.2.2 Praxis

- vom Vorgehen her nichts neues

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

```
0.7161545482202616
```

- die Performance ist deutlich schlechter als bei den meisten bisherigen Klassifikatoren
- und der Zeitbedarf grösser ...
- allerdings wäre das Potential von Standardisierung abzuklären (vgl. [ipynb])

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.1 Theorie

- auch Bayes-Klassifikator (oder Naive Bayes Klassifikator) genannt
- diese Methode wurde bereits im einführenden Modul "Einführung in die Denk- und Handlungsweise des ML" von Jürgen Vogel vorgestellt
- daher nur eine knappe Wiederholung sowie Anwendung mit sklearn.naive_bayes. GaussianNB mit metrischen Features

- arbeitet mit mehrdimensionalen Wahrscheinlichkeitsdichten und Klassenhäufigkeiten
- basiert auf dem Bayes Theorem (bedingte Wahrscheinlichkeit) und spielt seine Stärken vor allem bei hochdimensionalen Datensätzen aus

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.1 Theorie

- das Trainieren eines Modells ist denkbar einfach, es werden bloss einige Kennzahlen ermittelt welche
  - als Attribute des Modells gesichtet werden können (auch wenn eine Interpretation derselben kaum zweckdienlich ist)
  - bei der Prediction dann aber mathematisch ausgewertet werden
- ein Beispiel der ermittelten Modellattribute, wenn aus dem demo_data_class Dataset X1 als einziges Feature verwendet wird (vgl. [ipynb])

```
classes_ : ['A' 'B']
class_prior_ : [0.55555556 0.44444444]
theta_ :
 [[5.58666667]
 [4.26666667]]
var_ :
 [[0.31182222]
 [0.23055556]]
```

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.1 Theorie

- dabei bedeuten:
  - .classes_ : die Klassenbezeichnungen
  - .class_prior_ : die relativen Klassenmächtigkeiten des Targets
  - .model.theta_ : die Mittelwerte aller Features, aufgeteilt pro Klasse
  - .model.var_ : die Varianzen (Quadrate der Standardabweichungen) aller Features, aufgeteilt pro Klasse

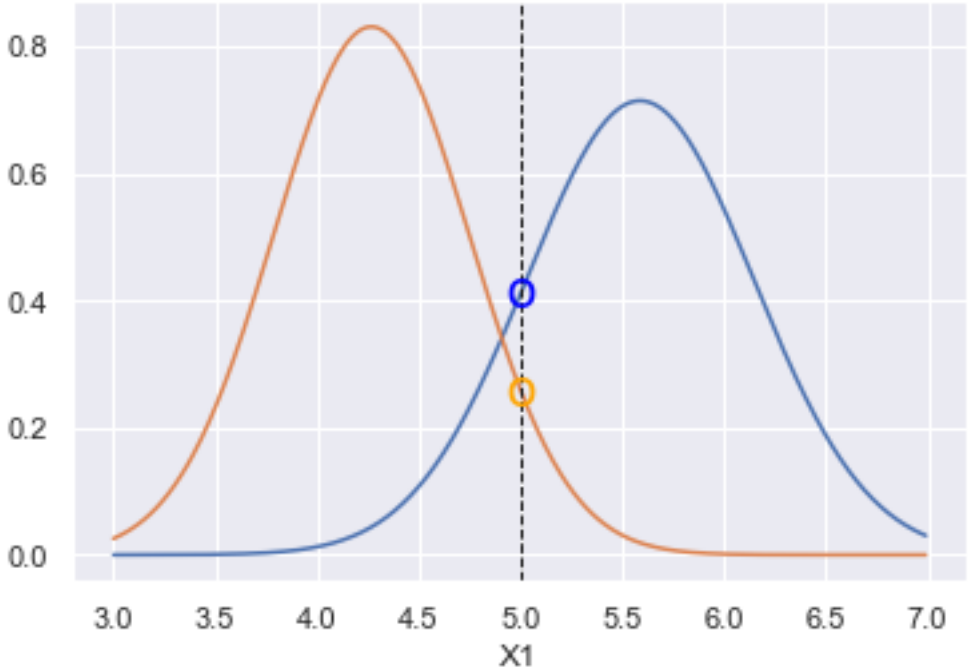
- für die Prediction (Test, neue Daten) werden aus obigen Modellattributen zwei Kennzahlen ermittelt
  - **a priori**: entspricht dem Wert von .class_prior:
  - **Likelihood**: wird aus .model.theta_ und .model.var_ ermittelt

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.1 Theorie

zur Berechnung von Likelihood

- aufgrund von Mittelwert und Standardabweichung aus dem Modell wird für jede Klasse eine (multidimensionale) Normalverteilung modelliert
- (im unten visualisierten Beispiel wird aber aus praktischen Gründen nur ein Feature verwendet)
- zur Erinnerung an Feature Exploration: Mittelwert und Standardabweichung wurden als *parametrische Kennzahlen* bezeichnet, da sie verwendet werden können, um Verteilungsmodelle zu parametrisieren
- bei GaussianNB wird für jede Klasse eine Gauss-Verteilung modelliert (daher der Name)
- für eine neue Beobachtung (z.B. X1 = 5) wird für jede Klasse der entsprechende Dichtewert ermittelt (die beiden Kreise) → Likelihood



2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.1 Theorie

- abschliessend wird für die definitive Prediction für jede Klasse a priori mit Likelihood multipliziert → **a posteriori**
- und die Klasse mit dem höchsten so ermittelten Wert als Voraussage zurückgegeben

Ausblick zum Thema Validierung:

- neben Accuracy gibt es noch eine ganze Menge weiterer Performance-Metriken, von denen unter dem Thema Validierung einige vorgestellt werden (Kap. 4.4.1)
- einige davon benötigen anstelle der Klasse eine Matrix mit Wahrscheinlichkeitswerten aller beteiligten Klassen, deren Summe pro Beobachtung 1 sein muss
- daher wird für die Prediction die Methode .predict_proba() verwendet, welche diese Wahrscheinlichkeitsmatrix zurückgibt, im Falle von GaussianNB werden daher die a posteriori Werte noch entsprechend normiert

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.2 Praxis

- wie gehabt (mit Standard Parametrisierung)

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

```
0.7337998174627319
```

- die verwendeten Parameter:

```
print(model.get_params())
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

- mit priors können die a priori Wahrscheinlichkeiten der einzelnen Klassen überschrieben werden

2.3.3 Klassifikation - Mathematisch - GaussianNB

2.3.3.2 Praxis

andere Varianten von NB:

- naive_bayes.MultinomialNB([alpha, …]):
  Naive Bayes classifier for multinomial models
- naive_bayes.ComplementNB([alpha, fit_prior, …]):
  The Complement Naive Bayes classifier described in Rennie et al.
- naive_bayes.BernoulliNB([alpha, binarize, …]):
  Naive Bayes classifier for multivariate Bernoulli models.
- naive_bayes.CategoricalNB([alpha, …]):
  Naive Bayes classifier for categorical features

2.3.4 Klassifikation - Mathematisch - LogisticRegression

2.3.4.1 Theorie

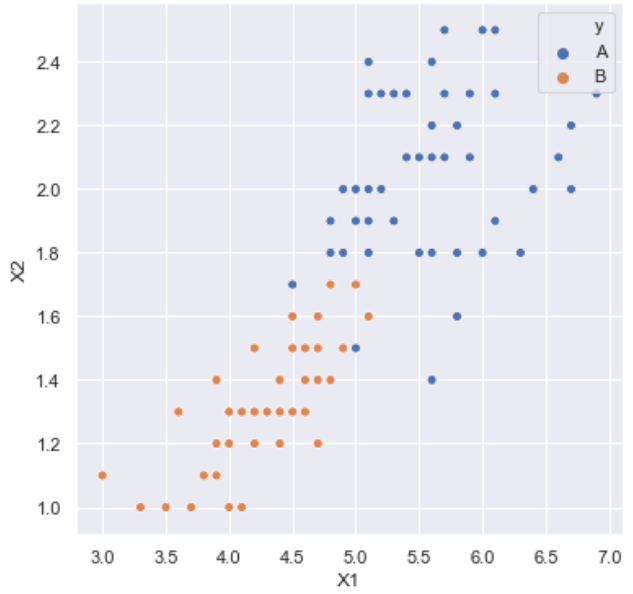
- Logistische Regression ist eigentlich ein "Zwischending" zwischen Regression und Klassifikation
- Regression:
  - von der zugrundeliegenden Mathematik her wird der Einfluss eines oder mehrerer numerischer Features auf ein ebenfalls numerisches Target modelliert
  - das Target weist die Werte 0 oder 1 auf (manchmal auch -1 und +1)
  - die Logistische Regression modelliert also einen Stufenübergang
- Klassifikation:
  - ist besonders gut geeignet, um Klassenübergänge abzubilden

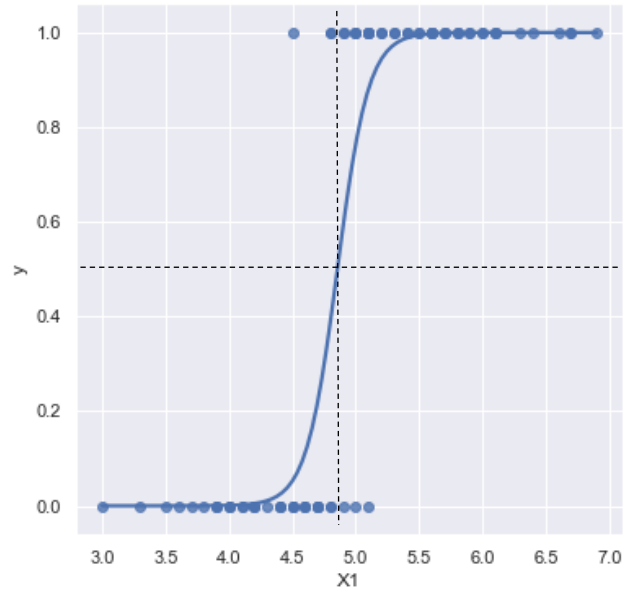
2.3.4 Klassifikation - Mathematisch - LogisticRegression

2.3.4.1 Theorie

- im Demo Dataset wird der Übergang von Gruppe "gelb" zu Gruppe "blau" in Bezug auf das Feature X1 untersucht (oben)
- damit dies numerisch gelöst werden kann, wird das Target (y) numerisch binär wie folgt umcodiert
  - gelb -> 0
  - blau -> 1

  und der Übergang von 0 nach 1 in Bezug auf X1 dargestellt (unten)
- die eingezeichnete sigmoide (s-förmige) Kurve in der unteren Darstellung zeigt die logistische Regression (Anpassung) an die vorliegenden Werte
- die vertikale gestrichelte Linie zeigt die Position auf X1, an welcher Y gerade 0.5 beträgt - das ist normalerweise der gesuchte Schwellwert (threshold)





2.3.4 Klassifikation - Mathematisch - LogisticRegression

2.3.4.1 Theorie

- die logistische Funktion ist wie folgt definiert

$$y = \frac{1}{1 + e^{-\beta_0 - \beta_1 x}}$$

  sie ist stetig (überall differenzierbar) und monoton wachsend und gleicht sich auf der y-Achse asymptotisch an die Werte 0 und 1 an

- β_0 und β_1 die beiden Parameter, welche beim Trainieren ermittelt werden
- um x an der Stelle $\mathrm{y} = 0.5$ zu berechnen, was auch gleich der Wendepunkt ist, kann die Formel umgestellt werden:

$$x = \frac{\ln\left(\frac{y}{1-y}\right) - \beta_0}{\beta_1}$$

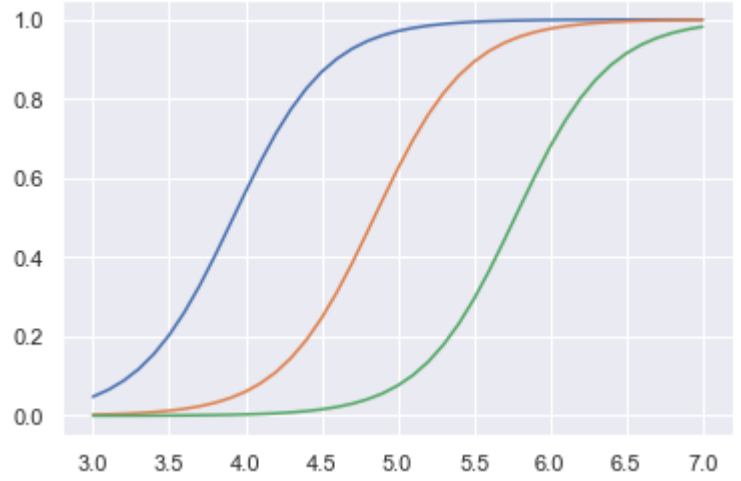
2.3.4 Klassifikation - Mathematisch - LogisticRegression

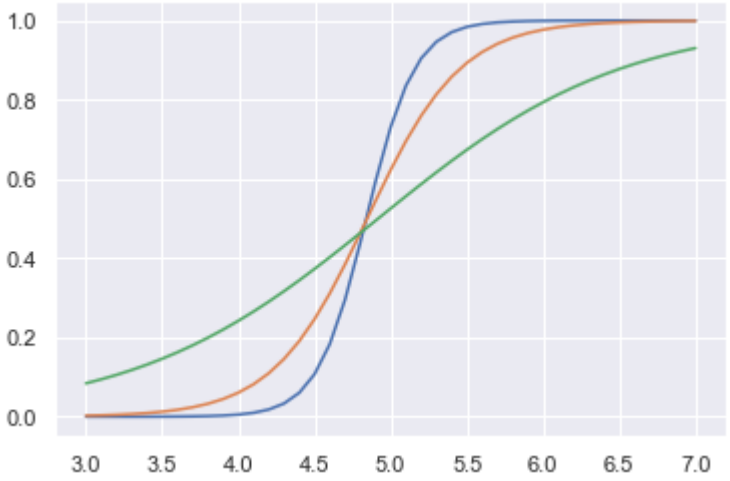
2.3.4.1 Theorie

- je nach Steilheit der Kurve am Wendepunkt (y=0.5), spricht sie für
  - einen Gruppenübergang (steile Kurve): Klassifikation
  - einen kontinuierlichen Übergang (flache Kurve): Regression

- nebenstehend: Beispiele von logistischen Funktionen mit unterschiedlichen Koeffizienten
- logistische Regressionen können auch auf mehrere Features gleichzeitig angewendet werden

- logistische Funktionen werden oft in Neuronalen Netzen als Aktivierungsfunktion eingesetzt (vgl. Kap. 2.3.4)





2.3.4 Klassifikation - Mathematisch - LogisticRegression

2.3.4.2 Praxis

- wie gehabt...

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=4000)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

```
0.8460602372984485
```

- der Parameter max_iter ist per default auf 100 eingestellt, und führt so bei diesem Aufruf zu einer Warnung (probieren Sie es aus)

2.3.4 Klassifikation - Mathematisch - LogisticRegression

2.3.4.2 Praxis

Wichtige Parameter:

- fit_intercept=True: soll ein Intercept (Nullstellendurchgang) mit geschätzt werden, mehr dazu dann in Kap. 3.2 Regression mit OLS
- solver='lbfgs': Optimierungsalgorithmus, hier viele Einstellmöglichkeiten
- max_iter=100: Anzahl Iterationen, muss für unsere Daten erhöht werden
- multi_class='auto': Einstellmöglichkeiten bei Mehrklassen-Problemen
- diverse weitere Parameter für regularisiertes trainieren, mehr dazu in Kap. 3.2.2.1

Wichtige Attribute:

- .intercept_ : Nullstellendurchgang
- .coef_ : geschätzte Koeffizienten für die einzelnen Features