Übungsblatt 3

Abgabe:

bis 17. Dezember 2020 um 23:59 via ecampus

Hinweis: Die Lösungen werden in den Übungsgruppen am Montag–Mittwoch, den 21.–23.12.2020 sowie am Freitag, den 8.1.2021 besprochen.

Aufgabe 3.1 (noch mehr Numerik): Seien eine Matrix $\boldsymbol{X}^\intercal \in \mathbb{R}^{3\times 2}$ und ein Vektor $\boldsymbol{y} \in \mathbb{R}^3$ wie folgt gegeben

$$\boldsymbol{X}^\intercal = \begin{bmatrix} 1.00000 & -1.00000 \\ 0.00000 & 0.00001 \\ 0.00000 & 0.00000 \end{bmatrix}$$

$$\boldsymbol{y} = \begin{bmatrix} 0.00000 \\ 0.00001 \\ 0.00000 \end{bmatrix}$$

Nehmen Sie nun an, Sie sollten einen Vektor $\boldsymbol{w} \in \mathbb{R}^2$ finden, so dass

$$\boldsymbol{X}^\intercal \boldsymbol{w} = \boldsymbol{y}$$

Ohne einen Computer zu benutzen, lässt sich mit etwas Überlegung schnell feststellen, dass die Lösung durch

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

gegeben ist (vergewissern Sie sich, dass das stimmt).

Gleichzeitig ist aber auch klar, dass der Vektor $\boldsymbol{w}$, der $\boldsymbol{X}^\intercal \boldsymbol{w} = \boldsymbol{y}$ löst, auch folgenden Ausdruck minimiert

$$\left\| \boldsymbol{X}^\intercal \boldsymbol{w} - \boldsymbol{y} \right\|^2$$

In der Vorlesung haben wir gesehen, dass die Lösung solcher least squares Probleme durch

$$\boldsymbol{w} = \left(\boldsymbol{X}\boldsymbol{X}^\intercal \right)^{-1} \boldsymbol{X}\,\boldsymbol{y} \tag{1}$$

gegeben ist.

Hier ist Ihre Aufgabe: Implementieren Sie $\boldsymbol{X}^\intercal$ und $\boldsymbol{y}$ als numpy arrays und lassen Sie Ihren Computer den gesuchten Vektor $\boldsymbol{w}$ berechnen.

Nutzen Sie dazu die folgenden 2 Methoden:

1. berechnen Sie $\boldsymbol{w}$ wie in Gleichung (1), d.h. nutzen Sie die numpy Funktion dot und die numpy.linalg Funktion inv, um Gleichung (1) zu implementieren; geben Sie dann Ihr Ergebnis für $\boldsymbol{w}$ aus

2. nutzen Sie die numpy.linalg Funktion lstsq, um $\boldsymbol{w}$ zu berechnen; geben Sie dann Ihr Ergebnis für $\boldsymbol{w}$ aus

Was beobachten Sie? Denken Sie noch einmal darüber nach, ob es einen Unterschied zwischen Mathematik “mit Bleistift und Papier” und den praktischen Berechnungen digitaler Computer gibt …

Aufgabe 3.2 (noch mehr lineare Regression): Die folgenden beiden arrays repräsentieren Datenpunkte (x_j, y_j)

```
vecX = np.array([ 1.0,  1.5,  2.5, 3.0, 4.0, 4.5, 5.0, 5.5])
vecY = np.array([-1.2, -0.8, -0.5, 0.1, 0.2, 0.6, 0.9, 1.4])
```

Nutzen Sie die Methoden der kleinsten Quadrate, um eine lineares Modell

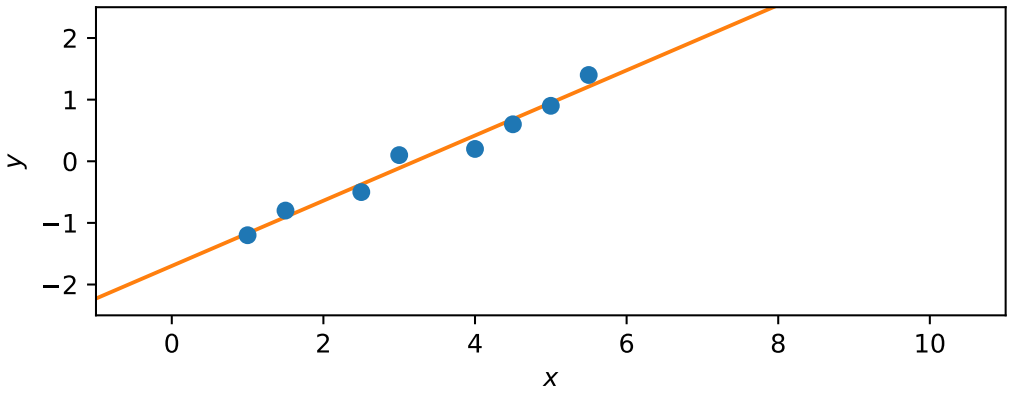
$$y_j = w_0 + w_1 \cdot x_j + \epsilon_j$$

an diese Daten anzugleichen. D.h. nutzen Sie die Methode der kleinsten Quadrate, um optimale Modellparameter $\hat{w}_0$ und $\hat{w}_1$ zu bestimmen.

Plotten Sie nun die Daten zusammen mit der trainierten Modellfunktion

$$f(x) = \hat{w}_0 + \hat{w}_1 \cdot x$$

Ihr Ergebnis könnte / sollte etwa folgendermaßen aussehen:



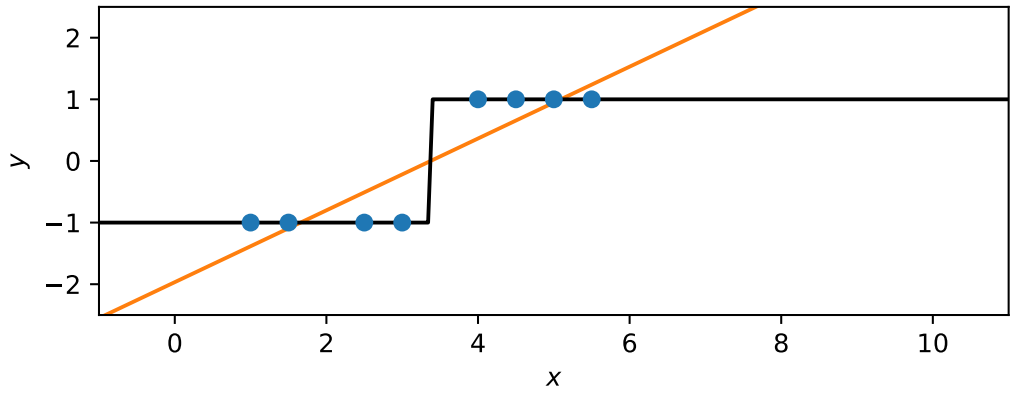
Hier ist eine weiterer Datensatz

```
vecX = np.array([ 1.0,  1.5,  2.5,  3.0, 4.0, 4.5, 5.0, 5.5])
vecY = np.array([-1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0])
```

Gleichen Sie erneut ein lineares Modell an diese Daten an. Plotten Sie anschließend die Daten zusammen mit den Funktionen

$$f(x) = \hat{w}_0 + \hat{w}_1 \cdot x$$
$$g(x) = \text{sign}\big(f(x)\big)$$

Ihr Ergebnis könnte / sollte diesmal etwa folgendermaßen aussehen:



Überlegen Sie, ob es für die Mathematik zur Angleichung linearer Modelle an Daten (x_j, y_j) relevant ist, ob die y_j beliebige reelle Zahlen sind, oder nur aus der Menge $\{-1, +1\}$ stammen?

Überlegen Sie, ob das lineare Modell, dass Sie hier angepasst haben, sinnlos ist, weil die gegebenen Datenpunkte nicht wirklich entlang einer Linie angeordnet sind? Oder liefert es eine zumindest einigermaßen plausible Beschreibung der Daten, weil es einen erkennbaren "globalen Trend" erfasst?

Hier sind drei weitere Datensätze

```
vecX = np.array([ 1.0,  1.5, 2.5, 3.0, 4.0, 4.5, 5.0, 5.5])
vecY = np.array([-1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0])
```

```
vecX = np.array([ 1.0,  1.5, 2.5, 3.0, 4.0, 4.5, 9.5, 9.9])
vecY = np.array([-1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0])
```

```
vecX = np.array([ 1.0,  1.5, 2.5, 3.0, 4.0, 4.5,  9.5,  9.9])
vecY = np.array([-1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0])
```

Verfahren Sie damit jeweils wie zuletzt, d.h. gleichen Sie lineare Modelle an und plotten Sie die Daten zusammen mit den Funktionen $f(x)$ und $g(x)$. Was beobachten Sie? Sind Ihre Ergebnisse plausibel?

Die folgenden vier Aufgaben beziehen sich alle auf die gleichen Daten. In Aufgabe 3.3 geht es darum, diese Daten einzulesen und für Analysen vorzubereiten. In Aufgabe 3.4 implementieren Sie LDA Klassifikatoren, die Sie in Aufgabe 3.5 systematisch testen. In Aufgabe 3.6 lernen Sie, wie die hier betrachteten "hoch-dimensionalen" Daten in 2D visualisiert werden können.

Für all diese Aufgaben bietet etwa scikit-learn special purpose Funktionen. Es ist aber nicht zu empfehlen, diese zu nutzen. Wenn Sie das täten, würden Sie nur lernen, wie die APIs einer speziellen Software zu nutzen sind, aber nicht, wie die zugrunde liegende Mathematik bzw. Gleichungen in Computer Code umgesetzt werden.

In der Tat können alle Aufgaben mit einfachen numpy und matplotlib Funktionen gelöst werden. Das heißt, Sie brauchen nur diese imports

```
import numpy as np
import numpy.linalg as la
import numpy.random as rnd
import matplotlib.pyplot as plt
```

Aufgabe 3.3: Die Datei

iris.csv

enthält den berühmten Iris flower Datensatz, der 1935 von E. Anderson gesammelt wurde, um morphologische Variationen von Iris Blumen zu quantifizieren. Der Datensatz erlangte seine enorme Bekanntheit unter Datenwissenschaftler*innen, weil R.A. Fisher ihn 1936 in der Arbeit, in der er die lineare Diskriminanzanalyse einführte, analysierte.

Die Daten bestehen aus 4 Merkmalswerten (Länge und Breite des Kelchblatts und Länge und Breite des Blütenblatts (in Zentimetern)) zu jeweils 50 Pflanzen der 3 Arten Iris setosa, Iris virginica und Iris versicolor und der Inhalt der Datei iris.csv sieht folgendermaßen aus …

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

```
4.7,3.2,1.3,0.2,Iris-setosa
...
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
...
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
...
```

Sie können diese Datei z.B. folgendermaßen einlesen

```
data = np.loadtxt('iris.csv', delimiter=',', dtype='object')
```

Um die Merkmalswerte in eine Datenmatrix $\boldsymbol{X} \in \mathbb{R}^{4\times 150}$ einzulesen, können Sie so vorgehen

```
matX = data[:,:-1].astype(float).T
```

Um die Artenbezeichner in einen Labelvektor $\boldsymbol{y} \in \{0,1,2\}^{150}$ zu übertragen, können Sie z.B. so vorgehen

```
lmap = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2}
vecY = np.vectorize(lmap.get)(data[:, -1])
```

Fisher fragte sich 1936, ob es die 4 Merkmalswerte zu einer Iris Pflanze erlauben, zu entscheiden, um welche Iris Art es sich handelt. Für die gegebenen Daten ist dies ein 3-Klassen Klassifikationsproblem. Genau wie Fisher werden wir es im Folgenden vereinfachen, indem wir es auf binäre Probleme reduzieren.

Dabei betrachten wir zwei Szenarien

1. Iris setosa (Klasse Ω_1) vs. Iris virginica und Iris versicolor (Klasse Ω_2)
2. Iris virginica (Klasse Ω_1) vs. Iris setosa und Iris versicolor (Klasse Ω_2)

Dazu labeln wir zunächst den Labelvektor $\boldsymbol{y}$ um. Im ersten Szenario können Sie so vorgehen

```
vecY = np.where(vecY==0, +1, -1)
```

Dadurch haben wir nun $\boldsymbol{y} \in \{-1,+1\}^{150}$ und können Matrix $\boldsymbol{X}$ in zwei Teilmatrizen $\boldsymbol{X}_1$ und $\boldsymbol{X}_2$ aufsplitten, die jeweils nur Beispiele aus Klasse Ω_1 und Ω_2 enthalten. Dies geht z.B. so

```
matX1 = matX[:,vecY==+1]
matX2 = matX[:,vecY==-1]
```

Berechnen Sie nun die 4-dimensionalen Mittelwerte

$$\boldsymbol{\mu}_1 = \frac{1}{n_1} \sum_{\boldsymbol{x} \in \Omega_1} \boldsymbol{x}$$
$$\boldsymbol{\mu}_2 = \frac{1}{n_2} \sum_{\boldsymbol{x} \in \Omega_2} \boldsymbol{x}$$

der Beispiele aus Klasse Ω_1 und Ω_2 und geben Sie diese aus. Wiederholen Sie die gesamte bisherige Prozedur für das zweite Szenario.

Aufgabe 3.4: In der Vorlesung hatten wir bereits gesehen, dass Fishers lineare Diskriminanzanalyse einen binären linearen Klassifikator

$$f(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{x}^\intercal \boldsymbol{a} - b)$$

trainiert. Bei gegebenen Trainingsdaten (Datenmatrizen $\boldsymbol{X}_1$ und $\boldsymbol{X}_2$ für Klassen Ω_1 und Ω_2) wird der optimale Projektionsvektor dabei wie folgt bestimmt

$$\hat{\boldsymbol{a}} = \underset{\boldsymbol{a}}{\operatorname{argmax}} \frac{\boldsymbol{a}^\intercal \boldsymbol{S}_B \, \boldsymbol{a}}{\boldsymbol{a}^\intercal \boldsymbol{S}_W \, \boldsymbol{a}}$$

Mit etwas Arbeit lässt sich zeigen, dass dieses Optimierungsproblem eine geschlossene Lösung hat, nämlich

$$\hat{\boldsymbol{a}} = \boldsymbol{S}_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{2}$$

Die Vektoren $\boldsymbol{\mu}_1$ und $\boldsymbol{\mu}_2$ sind die oben berechneten, klassenspezifischen Mittelwertsvektoren. Die Matrix $\boldsymbol{S}_W$ ist wie folgt definiert

$$\boldsymbol{S}_W = \boldsymbol{C}_1 + \boldsymbol{C}_2$$

wobei die Matrizen

$$\boldsymbol{C}_1 = \frac{1}{n_1} \sum_{\boldsymbol{x} \in \Omega_1} (\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^\intercal$$

$$\boldsymbol{C}_2 = \frac{1}{n_2} \sum_{\boldsymbol{x} \in \Omega_2} (\boldsymbol{x} - \boldsymbol{\mu}_2)(\boldsymbol{x} - \boldsymbol{\mu}_2)^\intercal$$

klassenspezifische Kovarianzmatrizen sind, die Sie mit numpy wie folgt berechnen können

```
matC1 = np.cov(matX1)
matC2 = np.cov(matX2)
```

Sobald der optimale Projektionsvektor $\hat{\boldsymbol{a}}$ für einen LDA Klassifikator vorliegt, lässt sich auch der entsprechende, optimale bias Wert $\hat{b}$ ermitteln. Dazu berechnen wir zunächst

$$\mu_1 = \hat{\boldsymbol{a}}^\intercal \boldsymbol{\mu}_1$$
$$\sigma_1^2 = \hat{\boldsymbol{a}}^\intercal \boldsymbol{C}_1 \, \hat{\boldsymbol{a}}$$
$$\mu_2 = \hat{\boldsymbol{a}}^\intercal \boldsymbol{\mu}_2$$
$$\sigma_2^2 = \hat{\boldsymbol{a}}^\intercal \boldsymbol{C}_2 \, \hat{\boldsymbol{a}}$$

und dann die beiden Werte

$$x_{1/2} = -\frac{\sigma_2^2\mu_1 - \sigma_1^2\mu_2}{\sigma_1^2 - \sigma_2^2} \pm \sqrt{\left(\frac{\sigma_2^2\mu_1 - \sigma_1^2\mu_2}{\sigma_1^2 - \sigma_2^2}\right)^2 - \frac{\sigma_1^2\mu_2^2 - \sigma_2^2\mu_1^2 + 2\sigma_1^2\sigma_2^2 \ln\frac{\sigma_2}{\sigma_1}}{\sigma_1^2 - \sigma_2^2}}$$

Der gesuchte bias Wert $\hat{b}$ ist derjenige der beiden Wert x_1 oder x_2, der zwischen den Zahlen μ_1 und μ_2 liegt, d.h.

$$\hat{b} = \begin{cases} x_1 & \text{if } \min(\mu_1, \mu_2) \leq x_1 \leq \max(\mu_1, \mu_2) \\ x_2 & \text{otherwise} \end{cases} \tag{3}$$

Hier ist Ihre Aufgabe: Trainieren Sie einen LDA Klassifikator für das erste Szenario, d.h. berechnen Sie $\hat{\boldsymbol{a}}$ und $\hat{b}$ nach Gleichungen (2) und (3); nutzen Sie alle verfügbaren Daten zum Training und ermitteln Sie anschließend die accuracy Ihres Klassifikators auf den Trainingsdaten.

Trainieren Sie einen LDA Klassifikator für das zweite Szenario; nutzen Sie erneut alle verfügbaren Daten zum Training und ermitteln Sie erneut die accuracy Ihres Klassifikators auf den Trainingsdaten.

Was beobachten Sie? Unterscheiden sich die beiden accuracy Werte?

Aufgabe 3.5: In der Vorlesung hatten wir gesagt, dass die Leistung eines Klassifikators auf unabhängigen Testdaten getestet werden muss.

Teilen Sie daher die gegebenen Daten für das erste und zweite Szenario in Trainings- und Testdaten auf.

D.h. splitten Sie die jeweiligen Matrizen $\boldsymbol{X}_1$ und $\boldsymbol{X}_2$ zufällig zu $^2/_3$ und $^1/_3$ in

$\boldsymbol{X}_{1,trn}$ und $\boldsymbol{X}_{1,tst}$
$\boldsymbol{X}_{2,trn}$ und $\boldsymbol{X}_{2,tst}$

auf und trainieren Sie die LDA Klassifikatoren für beide Szenarien mit den Daten in $\boldsymbol{X}_{1,trn}$ und $\boldsymbol{X}_{2,trn}$. Testen Sie die trainierten Klassifikatoren anschließend mit den Daten $\boldsymbol{X}_{1,tst}$ und $\boldsymbol{X}_{2,tst}$. Testen bedeutet hier: Ermitteln Sie die accuracy auf den Testdaten.

Idealerweise betrachten Sie hier natürlich mehrerer Trainings- und Test-Durchläufe, z.B. 10, und mitteln die gemessenen accuracy Werte.

Was beobachten Sie? Unterscheiden sich die (gemittelten) accuracies der beiden Szenarien? Wenn Sie alles richtig gemacht haben, sollte dies der Fall sein. Überlegen Sie, woran das liegen könnte. An der Methode oder an den Daten?

Aufgabe 3.6: Die Antwort auf die letzte Frage in Aufgabe 3.5 lautet: Es liegt an den Daten. Die beiden Klassen Iris versicolor und Iris virginica sind (unter den hier betrachteten Merkmalen) nicht linear separierbar.

Leider können wir das nicht unmittelbar sehen, denn der 4-dimensionale Raum, in dem die gegebenen Merkmalsvektoren eingebettet sind, ist nicht wirklich visualisierbar. In solchen Fällen werden Daten daher oft in 2- oder 3-dimensionale Räume projiziert, denn solche Projektionen lassen sich visuell inspizieren.

Eine gängige und in einem gewissen Sinne optimale Methode, sinnvolle Projektionen zu berechnen, beruht auf der principal component analysis, die wir hier mittels der singular value decomposition berechnen.

Wie jede andere Matrix auch, kann unsere Datenmatrix $\boldsymbol{X} \in \mathbb{R}^{4\times 150}$ wie folgt durch zwei faktorisiert werden

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$$

wobei $\boldsymbol{U}$ und $\boldsymbol{V}$ orthogonale Matrizen sind und $\boldsymbol{\Sigma}$ eine Diagonalmatrix ist.

In numpy können Sie die Faktoren auf der Rechten Seite wie folgt ermitteln

```
matU, svals, matVt = la.svd(matX)
```

Die Matrix $\boldsymbol{U}$, die wir in unserem Fall als Ergebnis bekommen, ist eine 4×4 Matrix und ihre Spaltenvektoren

$$\boldsymbol{U} = \left[\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{u}_4\right]$$

heißen die left singular vectors von $\boldsymbol{X}$.

Sammeln Sie die ersten beiden left singular vectors von $\boldsymbol{X}$ in einer Matrix

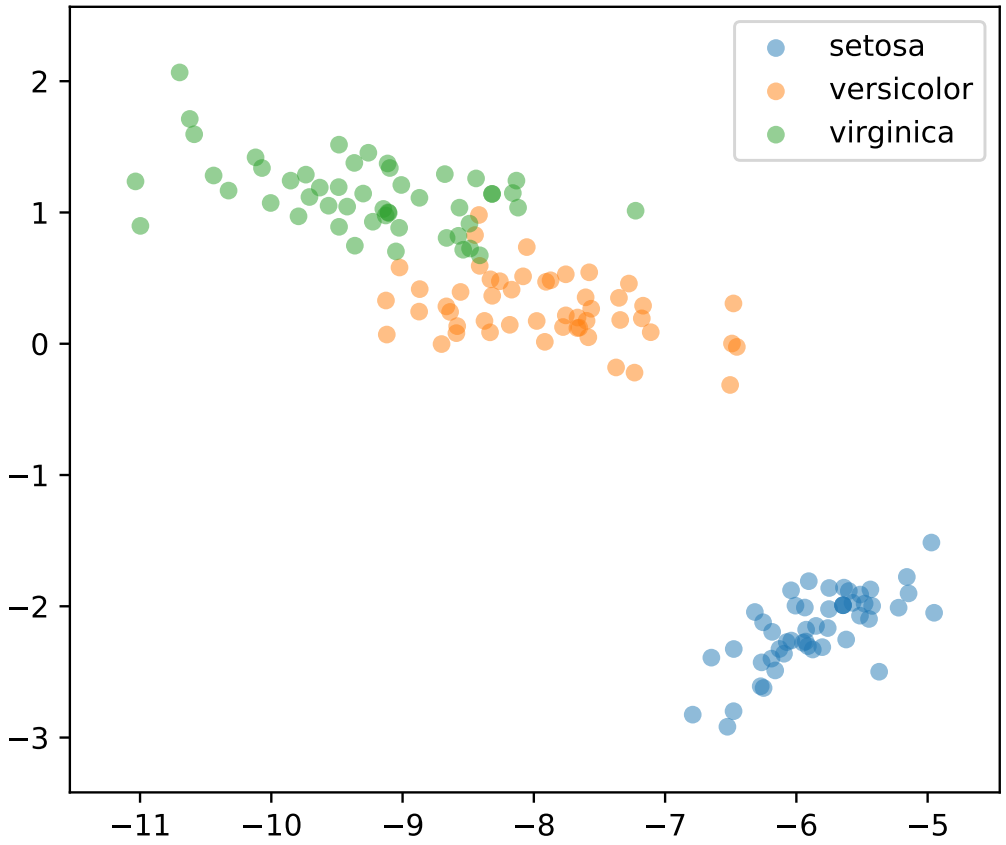
$$\boldsymbol{P} = \left[\boldsymbol{u}_1, \boldsymbol{u}_2\right]$$

und berechnen Sie anschließend die Projektion

$$\boldsymbol{Z} = \boldsymbol{P}^\intercal \boldsymbol{X}$$

Das Ergebnis $\boldsymbol{Z}$ ist eine 2×150 Matrix, deren 2-dimensionale Spaltenvektoren die Merkmalsvektoren in $\boldsymbol{X}$ in einem 2-dimensionalen Raum reprä-sentieren.

Wenn Sie die projizierten Daten in $\boldsymbol{Z}$ mit matplolib plotten und klassenspezifisch einfärben, könnte Ihr Ergebnisse etwa so aussehen …



Wenn Sie mögen, können Sie mit unterschiedlichen Projektionsmatrizen

$$\boldsymbol{P} = \left[\boldsymbol{u}_i, \boldsymbol{u}_j\right]$$

mit $1 \leq i < j \leq 4$ experimentieren.

Was beobachten Sie bei den Ergebnissen, die Sie dadurch bekommen?