Softwarewerkzeuge der Bioinformatik

Prof. Dr. Volkhard Helms
PD Dr. Michael Hutter, Markus Hollander,
Andreas Denger, Marcial Josef Paszkiel

Saarland University
Department of Computational Biology

Winter semester 2022/2023

Tutorial 10

January 19, 2023

Data visualization & Supervised Machine Learning

In dieser Übung werden sie einen weiteren Genexpressions-Datensatz mit Jupyter Notebooks auf Kaggle.com analysieren. Die Daten wurden schon normalisiert und gereinigt. Der Datesatz enthält 39 samples von Leber-Tumor-Gewebe, und 36 samples aus benachbartem gesundem Gewebe. Das Ziel ist es, ein einfaches Machine Learning Modell darauf zu trainieren, vorherzusagen ob eine Gewebeprobe aus Tumor- oder gesundem Gewebe stammt, und anschließend die Performance des Classifiers zu beurteilen.
Erstellen sie eine neue Code-Zelle für jede Unteraufgabe.

Exercise 10.1: Vorbereitung

(a) Erstellen sie ein neue Notebook in ihrem Kaggle account, und laden sie den Datensatz mit dem folgenden Link hoch, so wie sie das in den vorherigen Übungen gemacht haben (**+Add data**):

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Liver/GSE57957/Liver_GSE57957.csv

Geben sie dem Datensatz einen Namen den sie wiedererkennen, wie "*livercancer*".

(b) Lesen sie die Daten in einen Pandas DataFrame:

```
import pandas as pd
file_path = "../input/livercancer/Liver_GSE57957.csv"
df = pd.read_csv(file_path, index_col=0)
```

Spalten sie die Daten in *features* und *labels*:

```
labels = df["type"]
features = df.drop("type", axis=1)
```

(c) Rufen sie die Funktion *labels.value_counts()* auf. Wie viele Tumor und Kontroll-Samples enthält der Datensatz?

(d) Zeigen sie mit der *print*-Funktion die Tabelle *features* an. Wie viele samples und Microarray-Spots enthält der Datensatz?

Exercise 10.2: Preprocessing

In dieser Übung werden die die Daten vorbereiten auf die Machine Learning library die wir benutzen werden (scikit-learn).
Ein Machine Learning Modell wir auf einer Matrix **X** trainiert und Evaluiert, wobei die Reihen den *Samples* entsprechen (in unserem Fall Gewebeproben), und die Spalten *Features* enthalten, (hier: Helligkeitswerte auf dem Microarray).
Die Matrix wird begleitet von einem Vektor **y**, der die Labels für jedes sample enthält (hier: Krebs oder gesund). Der Algorithmus findet dann Muster in den Daten (hier: die Helligkeitswerte der Punkte auf dem Microarray), sodass es lernen kann, zwischen den Labels **0** (normales Gewebe) und **1** (Tumor-Gewebe) zu unterscheiden.

(a) Das Python package *numpy* stellt eine Datenstruktur zur Verfügung, die für Matrizen und Vektoren benutzt werden kann, namens *array*. Pandas Objekt können zu numpy arrays konvertiert werden mit der *to_numpy()* Funktion:

```
X = features.to_numpy()
print(X)
y = labels.to_numpy()
print(y)
```

(b) Die Labels in **y** sind noch Text (also "HCC" und "normal"). Diese Klassen-Labels werden oft zu positiven Zahlen konvertiert. Scikit-learn implementiert ein praktisches Objekt namens *LabelEncoder*, dass unsere Text-labels zu Integer-labels und wieder zurück konvertieren kann:

```
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
y = label_enc.fit_transform(y)
print(y)
print(label_enc.inverse_transform(y))
```

(c) Als nächstes werden wir unseren Datensatz (also **X** und **y**) in ein *training set* (**X_train, y_train**) und ein *test set* (**X_test, y_test**) aufspalten. Der Algorithmus wird dann auf den Trainings-Daten trainiert (**X_train, y_train**), ohne etwas über die Test-Daten zu wissen. Nach dem Training werden wir die Features des Test-Sets (**X_test**) benutzen, um dessen Labels vorherzusagen. Die vorhergesagten Labels(**y_pred**) werden dann verglichen mit dem tatsächlichen labels (**y_test**). Scikit-learn implementiert eine Funktion namens *train_test_split()* um die Daten zu spalten:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=1
)
```

Was ist der Sinn hiner den Parametern *test_size* und *stratify*? *Hinweis: sehen sie sich die offizielle Anleitung an.*

(d) Einige der Algorithmen die wir benutzen werden, genauer gesagt PCA und SVM, funktionieren am besten wenn die Daten standardisiert sind. So können sie sich den Durchschnitt und die Standardabweichung der Features im Trainings-Set angucken:

```
print(X_train.mean(axis=0).round(2))
print(X_train.std(axis=0).round(2))
```

Sind die Daten schon standardisiert?

(e) Scikit-learn stellt ein Objekt bereit namens *StandardScaler*, mit dem wir unsere zwei Datensätze standardisieren können:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Wie auch viele andere objekte in scikit-learn, stellt StandardScaler eine *fit_transform* Funktion, welche auf den Trainings-Daten aufgerufen wird bereit, sowie eine *transform* Funktion, mit denen die Test-Daten bearbeitet werden. Suchen sie nach dem Begriff data leakage im Kontext von Machine Learning. Warum ist es wichtig, den StandardScaler nur auf die Trainings-Daten anzupassen (mit fit), aber nicht auf die Test-Daten?

Exercise 10.3: Dimensionality Reduction

Momentan hat jedes unserer 75 samples Datenpunkte für 47,324 Features. Der Begriff "curse of dimensionality" bezieht sich auf die Nachteile von sehr vielen Dimensionen. Diese umfassen unter anderen viel Rechenaufwand, mehr Redundanz, und mehr Rauschen in den Daten, was schlecht für die Performance von Machine Learning Modellen sein kann.

(a) Wir werden eine *Principal Component Analysis* durchführen um die Anzahl der Dimensionen von 47,324 auf zwei zu reduzieren. Scikit-learn hat eine Klasse dafür:

```
from sklearn.decomposition import PCA

pca_decomp = PCA(n_components=2, random_state=1)

X_train = pca_decomp.fit_transform(X_train)
X_test = pca_decomp.transform(X_test)

print(X_test)
```

(b) Ein weiterer Vorteil von zwei-dimensionalen Daten ist, dass wir schnell einen Scatter-Plot der Samples erstellen können:

```
import matplotlib.pyplot as plt

plt.scatter(
    x=X_train[y_train == 0, 0],
    y=X_train[y_train == 0, 1],
    color="blue",
    label="HCC"
)
plt.scatter(
    x=X_train[y_train == 1, 0],
    y=X_train[y_train == 1, 1],
    color="red",
    label="normal"
)
plt.xlabel("PrincipalComponent1")
plt.ylabel("PrincipalComponent2")
plt.legend()
plt.show()
```

Interpretieren sie den Plot. Können diese Daten klar in zwei Klassen unterteilt werden? Gibt es Ausreißer?

Exercise 10.4: Support Vector Machine

Nun ist es an der Zeit, einen Support Vector Machine Classifier (SVC) zu trainieren und zu evaluieren.

(a) Wir müssen die Daten nochmal standardisieren:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

(b) Hier erstellen wir ein SVC Objekt, trainieren es auf die Trainings-Daten, und sagen die Labels der Test-Daten vorher:

```
from sklearn.svm import LinearSVC
```

```
svc = LinearSVC(C=1, class_weight="balanced", random_state=1)
svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)
print(y_pred)
```

(c) Die vorhergesagten Zahlen, die wir vom Classifier zurück bekommen (**y_pred**), sowie die wahren Labels (**y_test**), können mit dem LabelEncoder aus Übung 10.2 wieder wieder zurück zu Text-Labels konvertiert werden:

```
print(label_enc.inverse_transform(y_pred))
print(label_enc.inverse_transform(y_test))
```

Hat die Klassifizierung gut funktioniert? Wie viele falsche Vorhersagen sehen sie?

(d) Scikit-learn implementiert mehrere Metriken, um die Performance zu messen. Zeigen sie die Genauigkeit des Classifiers an:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_true=y_test, y_pred=y_pred))
```

Interpretieren sie das Resultat von *accuracy_score*.

(e) Das *mlxtend* Python Paket stellt eine Funktion zur Verfügung, mit dem Ihr Classifier nach dem Fitting auf die Trainings-Daten als Plot dargestellt werden kann. Erstellen sie einen Plot der SVM Hyperebene, zusammen mit den Trainingsdaten:

```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X_train, y_train, clf=svc, legend=2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Erstellen sie den selben Plot mit den Testdaten:

```
plot_decision_regions(X_test, y_test, clf=svc, legend=2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Wie viele Ausreißer gibt es? Hat der Algorithmus seine Hyperebene in einer guten Position platziert?

Have fun!