Softwarewerkzeuge der Bioinformatik

Prof. Dr. Volkhard Helms
PD Dr. Michael Hutter, Markus Hollander,
Andreas Denger, Marie Detzler, Larissa Fey

Saarland University
Department of Computational Biology

Wintersemester 2021/2022

Project 3

Abgabe bis: 11. Februar 2022

Machine Learning

In diesem Projekt werden sie die Kenntnisse über Machine Learning in Python, die sie sich in den Übungen angeeignet haben anwenden, um ihre eigene Analyse durchzuführen. Sie werden einen Microarray-Datensatz analysieren, der Genexpressionsdaten von Krebspatienten enthält, sowie von gesunden Menschen als Referenz. Zunächst werden sie die *seaborn* library benutzen um die Daten zu visualisieren, dann die paarweise Korrelation zwischen Samples zu berechnen, und ein hierarchisches Clustering durchführen. In der letzten Übung werden sie mit den Daten eine Machine Learning Pipeline trainieren und optimieren, und sie auf einem unabhängigen Test-Datensatz testen.

Schicken sie das fertige Notebook (die *.ipynb* Datei) an
`andreas.denger@bioinformatik.uni-saarland.de`. Sie können ein Notebook von Kaggle herunterladen indem sie auf *File→Download* klicken.
Das Notebook sollte alles an Code enthalten, den sie zum Lösen der Übungen verwendet haben. Benutzen sie Markdown cells, also Zellen die geschriebenen Text enthalten, um klar zu stellen welche Code cells zu welcher Übung gehören, und um Kommentare, Erklärungen und Beschreibungen für die Schritte und Resultate ihrer Analyse hinzuzufügen.
Sie können an die oben genannte Email Adresse schreiben wenn sie Fragen haben, oder wenn sie ein Meeting auf Microsoft Teams verabreden wollen um Unklarheiten zu klären.

Exercise 3.1: Vorbereitung (5 Punkte)

Im letzten Tutorial haben sie gelernt, wie man kaggle.com benutzen kann um Code zu erstellen und auszuführen in Jupyter Notebooks. Finden sie den Genexpressions-Datensatz für ihre Gruppe auf der Website der Vorlesung, und laden sie den Datensatz in ein neues Notebook.

(a) Rufen sie kaggle.com auf und erstellen sie eine neues Notebook. Geben sie dem Notebook einen Namen den sie wiedererkennen werden, z.B. *SWW Projekt 3 Gruppe X*. Wenn sie das Notebook schließen, können sie es in ihrem Benutzeraccount oben rechts auf der Startseite wiederfinden.

(b) Laden sie den Datensatz der ihrer Gruppe zugewiesen wurde in das Notebook hoch, mit dem +**Add data** Knopf oben rechts. Anschließend sollten sie es in ihrem *input* Ordner unter dem Knopf wiederfinden.

(c) In den Übungen 2, 3, 9 und 10 haben sie schon die Grundlagen von Programmierung in Python gelernt, wie z.B. Datenstrukturen, Loops und Funktionen. Lesen sie sich die Aufgaben noch einmal durch, um ihre Erinnerung aufzufrischen. Außerdem sind die Python standard library Referenz, sowie die Anleitungen zu Numpy arrays, Pandas dataframes, pandas series, seaborn und Scikit-Learn wichtige Ressourcen.

Exercise 3.2: Datenexploration mit pandas & seaborn (30 Punkte)

Das *pandas* Paket stellt die *DataFrame* und *Series* Objekte zur Verfügung. DataFrames enthalten Daten in Tabellenform, ähnlich wie eine Excel-Datei. Series sind Listen-Ähnliche Objekte. Jede Spalte eines DataFrames ist eine Series.
Seaborn ist ein Paket, dass für die Visualisierung von Daten verwendet wird, und kann Plots aus DataFrames erstellen.

(a) Importieren sie das *pandas* Paket, und laden sie ihre Daten in einen DataFrame. Schauen sie sich den Inhalt des DataFrame an, indem sie ihn in die letzte Zeile des Zelle schreiben, so wie im Tutorial. In der oberen Reihe sehen sie die Namen der Spalten, links sind die Namen der Reihen.

(1) Weisen sie die Spalte *type* einer neuen Variable namens *labels* zu. Sie können eine Spalte aus einem DataFrame lesen indem sie eckige Klammern benutzen, die den Namen der Spalte in Anführungszeichen beinhaltet. Dokumentation finden sie hier.

(2) Erstellen sie einen neuen DataFrame mit dem Namen *features*, der alle Spalten der ursprünglichen DataFrame beinhaltet, außer *samples* und *type*. *Hinweis: schauen sie sich die drop() Funktion in der Anleitung zu DataFrames an, und ihren 'axis' Parameter.*

(3) Benutzen sie die *print* Funktion um sie die *labels*-Series und den *features*-DataFrame anzusehen, und festzustellen ob alles funktioniert hat. *features* enthält die Gene als ihre Spalten, und die Patienten als Reihen. *labels* weißt jeder Reihe in *features* ein label zu (Krebs, Gesund).

(4) Benutzen sie die *value_ counts()* Funktion der *labels* Series. Ist ihr Datensatz balanced oder imbalanced? *Balanced* bedeutet, dass beide Labels genauso oft vorkommen. Das wird später noch wichtig.

(5) Für die folgenden Aufgaben werden wir eine transponierte Version des *features* DataFrame brauchen. Benutzen sie die *transpose()* Funktion des DataFrame *features*, und weisen sie das Resultat einer Variable namens *features_ t* zu. Schauen sie sich den Inhalt des transponieren DataFrame an.

(b) Berechnen sie als nächstes die paarweise Korrelationsmatrix für die Patienten-Samples, und visualisieren sie die Matrix mit einer Heatmap.

(1) Die *corr()* Funktion des DataFrame Objects berechnet die paarweise Korrelation zwischen jedem Paar aus Spalten in dem DataFrame. Der transponierte DataFrame textitfeatures_t den sie im letzten Aufgabenteil erstelle haben enthält die Patienten-Samples als Spalten. calculates the pairwise correlation between every pair of columns in a DataFrame. Rufen sie die *corr()* Funktion auf *features_ t* auf, und weisen sie das Resultat einer neuen Variable namens *features_ t_ corr* zu.

(2) Importieren sie die *seaborn* Library. Rufen sie die *heatmap()* Funktion auf *features_ t_ corr* auf, um eine Heatmap der Korrelationsmatrix zu erstellen:

```
seaborn.heatmap(
    data=features_t_corr,
    xticklabels=labels,
    yticklabels=labels
)
```

Interpretieren sie den Plot. Was ist das Verhältnis zwischen den Labels und der Korrelation?

(c) Erstellen sie eine *clustermap* von *features_ t*, so wie sie gerade die heatmap erstellt haben. Benutzen sie *labels* für den *xticklabels* Parameter, and und setzen sie den *yticklabels* Parameter auf to *False*.

(1) Die Datensätze einiger Gruppen enthalten mehr als 50.000 Features. Eine clustermap von einem so großen Datensatz zu erstellen kann mehrere Stunden dauern. Ein möglicher Ausweg ist es, zufällig eine Teilmenge der Gene aus dem DataFrame zu ziehen, und den Plot damit zu erstellen. Sie können die *sample()* Funktion mit dem *n* oder dem *frac* Parameter benutzen, um eine Anzahl oder eine Prozentzahl an Genen zufällig aus den Daten zu ziehen. Der DataFrame *features_t_sampled* enthält 20% der Gene in *features_t*:

```
features_t_sampled = features_t.sample(frac=0.2,random_state=1)
```

Der *random_state* Parameter sichert die Reproduzierbarkeit, indem er jedes mal die selben Samples zieht, abhängig von dem Wert den er bekommt.

(2) Sie können verschiedene Clustering-Methoden mit dem *method* Parameter der *clustermap* ausprobieren. Versuchen sie *, method="ward"* zu dem Funktionsaufruf von *clustermap* hinzuzufügen, und schauen sie ob das zu einem besseren Clustering der Spalten führt.

(3) Schreiben sie eine kurze Interpretation des Plots in eine Markdown cell.

Exercise 3.3: Machine learning mit scikit-learn (40 Punkte)

Im letzten Teil des Projekts werden sie mit scikit-learn (auch *sklearn* genannt) auf den Daten einen Machine Learning Algorithmus trainieren und evaluieren.
Im Tutorium haben sie bereits eine Support Vector Machine (SVM) mit einem *linearen Kernel* benutzt, welche versucht eine Hyperebene, also z.B. eine gerade Linie im Fall eines zweidimensionalen Raumes, zu zeichnen um die zwei Klassen voneinander zu trennen. Dieses mal werden wir einen *Radial Basis Function (RFB) Kernel* (auch *radialer Gauss kernel* genannt) benutzen, welcher die Daten auf nicht-lineare Arten trennen kann.

(a) Die Matrix welche die Features enthält wird in sklearn X genannt, die Liste mit den Labels nennt man y. Letztere enthält ganzzahlige Werte welche die Labels repräsentieren. Sowohl X als auch y sind numpy arrays. Konvertieren sie ihren pandas DataFrame und ihre Series zu numpy arrays, wie sie in Übung 10 gelernt haben:

```
from sklearn.preprocessing import LabelEncoder
X = features.to_numpy()
label_enc = LabelEncoder()
y = label_enc.fit_transform(labels)
```

(b) Im nächsten Schritt werden wir die Daten teilen, in einen Trainings-Datensatz und einen unabhängigen Test-Datensatz:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1,stratify=y)
```

Der Classifier wird auf dem Trainings-Datensatz optimiert, und anschließend auf dem Test-Datensatz evaluiert.

(c) Lesen sie den Getting Started Artikel auf sklearn's website um sich mit den Grundkonzepten und Funktionen bekannt zu machen. Die Website stellt auch einen umfangreichen User Guide bereit, ein Reference Manual (API) für die Funktionen, sowie viele Beispiele aus denen man etwas lernen kann.

(d) Benutzen sie die *make_pipeline* Funktion um eine Pipeline zu erstellen, die einen StandardScaler und ein SVC objekt (sklearn.svm.SVC) enthält. Vergessen sie nicht die Funktionen vorher von ihren jeweiligen Paketen zu importieren. Schauen sie in den *Getting Started* Guide um zu verstehen wie das funktioniert. Wenn sie in der letzten Aufgabe herausgefunden haben dass ihr Datensazu *imbalanced* ist, sollten sie den *class_weight* Parameter der SVM auf *"balanced"* setzen, da das sehr wahrscheinlich die Genauigkeit verbessert. Zeigen sie mit der print Funktion die Pipeline an, um zu sehen ob alles funktioniert hat.

(e) Nun werden sie den Classifier mit dem Trainings-Datensatz trainieren. Dafür werden wir *GridSearchCV* benutzen. Dieses Objekt nimmt einen Estimator, was ein Classifier oder eine Pipeline sein kann, sowie ein Raster aus Parametern. Es wird dann jede Kombination aus Parametern ausprobieren, und die Kombination auswählen die die besten Ergebnisse liefert.

(1) Erstellen sie ein GridSearchCV Object mit ihrer Pipeline als ihren ersten Parameter, und dem folgenden Parameter-Raster als seinen zweiten Parameter:

```
param_grid={
    'svc__gamma':[0.1, 1e−2, 1e−3, 1e−4],
    'svc__C':[1, 10, 100, 1000]
}
```

Dies wird vier verschiedene Werte für *gamma* und vier verschiedene Werte für C für das SVC objekt in der Pipeline ausprobieren, also werden insgesamt 16 Modelle getestet. Setzen sie den *scoring* Parameter von *GridSearchCV* auf "f1". Die Standard-Option ist äccuracy", was einen Bias verursachen kann wenn der Datensatz *imbalanced* ist. Rufen sie die Print-Funktion auf dem GridSearchCV Object auf.

(2) Benutzen sie die *fit()* Funktion des *GridSearchCV* Objektes mit den Trainings-Daten auf (*X_train, y_train*). Geben sie mit print die besten Parameter aus die gefunden wurden, sowie die beste Score.

Hinweis: Sie können einen ähnlichen Ansatz mit RandomizedSearchCV im Getting Started Guide finden. Schauen sie sich außerdem die Attributes in der Dokumentation von GridSearchCV an.

(3) Was sagen die Parameter die sie gefunden haben über ihr Modell aus? *Hinweis: schauen sie sich die Folien von Vorlesung 12 an, sowie die Dokumentation für RBF kernels auf sklearn's Website.*

(f) Das *GridSearchCV* Objekt verhält sich nun wie ein Estimator, und benuitzt automatisch die besten Parameter die es während der Optimierung gefunden hat. Rufen sie die *score()* Funktion auf den Test-Daten auf (*X_test, y_test*) um eine F1 Score für die optimierte Pipeline zu bekommen.

(g) Eine Sache welche die Genauigkeit ihrer Pipeline negativ beeinflussen kann ist die hohe Anzahl an Features im Datensatz (mehr als 20.000 Gene). Benutzen sie *feature selection* um die besten k Features für die Klassifizierung auszuwählen. Der Classifier wird dann nur auf diesen trainiert. Erstellen sie ein *SelectKBest* Objekt, dass nur die besten 20 Features behält:

```
from sklearn.feature_selection import SelectKBest
kbest = SelectKBest(k=20)
```

Erstellen sie eine weitere Pipeline die einen StandardScaler, das *kbest* object, noch einen StandardScaler, und ein SVC Objekt enthält (vergessen sie nicht den class_weights Parameter). Führen sie das Training und die Evaluation mit GridSearchCV erneut aus für diese Pipeline, und schauen sie ob sich die Score verbessert.

(h) Die Wahl des Trainings-und Test-Datensatzes kann einen großen Einfluss auf die finale Score haben. Ein *train_test_split* könnte zu einer Score von 1.00 führen, ein anderer zu einer Score von 0.50. Um diese Fehlerquelle auszuschließen, kann man den Datensatz in 5 Teilmengen aufteilen. Es wird immer eines davon zum Testen benutzt, die anderen vier zusammen zum Training. Am Schluss berechnet man die durchschnittliche Score und die Standardabweichung der 5 Testläufe. Führen sie eine cross validation mit ihrem GridSearchCV Objekt durch, mit *f1* als *scoring* Parameter. Geben sie die durchschnittliche Score nach den fünf Tests, sowie ihre Standardabweichung aus:

```
from sklearn.model_selection import cross_val_score
res = cross_val_score(gsearch, X,y,scoring="f1", cv=5)
print(res)
print(f"{res.mean()}+-{res.std()}")
```

Um die letzten fünf Punkte für das Projekt zu bekommen, müssen sie eine durchschnittliche F1 score von mindestens 0.90 erreichen. Wenn ihr Modell diese Score noch nicht erreicht hat, versuchen sie verschiedene Werte für k, *gamma* und C auszuprobieren, oder lesen sie die Dokumentation von SCV, um eventuell noch andere Parameter zu finden die sich mit GridSearchCV optimieren lassen.

Exercise 3.4: Machine Learning Visualization (15 Punkte)

Zusätzlich zu den allgemeinen Paketen wie *seaborn* gibt es noch spezielle Pakete, um Machine Learning Daten zu veranschaulichen, wie z.B. *yellowbrick*.

(a) Eine Möglichkeit, einen Datensatz mit mehr als zwei Dimensionen darzustellen ist es, die Dimensionen auf zwei zu reduzieren. Zwei häufig verwendete Methoden dafür sind die Principal Component Analysis (PCA) und t-distributed Stochastic Neighbor Embedding (t-SNE). Letztere wird auch oft für Genexpressions-Daten verwendet.

(1) Erstellen sie einen PCA-plot von X, y aus Aufgabe 3 mit Hilfe von yellowbrick. Die Klassen für den parameter *classes* sind noch im *label_enc* aus Aufgabe 3.3 gespeichert:

```
from yellowbrick.features import PCA
visualizer = PCA(scale=True, classes=label_enc.classes_, random_state=1)
visualizer.fit_transform(X, y)
visualizer.show()
```

(2) Erstellen sie von dem selben Datensatz einen t-SNE plot. Welche Vorteile hat t-SNE im Vergleich zu PCA?

(b) Es gibt auch andere Arten von Plots, mit denen mehr als zwei Dimensionen dargestellt werden können, wie z.B. RadViz. Der Übersicht halber werden wir die Anzahl der Dimensionen aber auf 20 beschränken, und deren Namen aus dem DataFrame aus Aufgabe 2 holen:

```
kbest = SelectKBest(k=20)
X_selected = kbest.fit_transform(X,y)
feature_names_selected = features.columns[kbest.get_support()]
```

Erstellen sie einen RadViz plot von $X\_selected$ und y:

```
from yellowbrick.features import RadViz
visualizer = RadViz(classes = label_enc.classes_, features=feature_names_selected)
visualizer.fit_transform(X_selected, y)
visualizer.show()
```

Exercise 3.5: Projektbericht (10 Points)

Machen sie aus dem Notebook dass sie gerade erstellt haben einen Projektbericht. Benutzen sie Markdown cells, um die Schritte zu erklären die sie während ihrer Analyse durchgeführt haben. Beschreiben sie die Plots und die Resultate.

Hier können sie weitere Informationen über ihre Datensätze finden:

- Gruppe 1: GSE41328
- Gruppe 2: GSE60502
- Gruppe 3: GSE57297
- Gruppe 4: GSE22405
- Gruppe 5: GSE12452
- Gruppe 6: GSE7670
- Gruppe 7: GSE16515

Hinweis: Die Seite der verwendeten Microarray *Platform* (GPL...) enthält eine Tabelle mit Annotationen für ihre Feature-Namen aus Aufgabe 3.4b, wie z.B. Gen-Name, oder Gen-Symbol.

Klicken sie schlussendlich auf **Run All** im Notebook, warten sie bis die Berechnungen fertig sind, und laden sie das Notebook herunter (File→Download). Senden sie die Datei an die Email die oben zu finden ist. Vergessen sie nicht ihre Namen in das Notebook zu schreiben.

Have fun!