Hard- und Software

Soft- und Hardware für Data Science

- Grundlage zu Hard- und Software-Anforderungen
- Ersten Schritte. Mehr in fortgeschrittenen Vorlesungen und Veranstaltungen

- Soft- und Hardware
  - Gewöhnliches Notebook / PC ausreichend
  - Später auch Cloud: EDSC (private Cloud DHBW MA), AWS, SAP CP, …
  - Software
    - Python
    - Jupyter
    - Numpy, Scipy
    - Scikit-Learn
    - Tensorflow
    - Empfohlene IDE (Entwicklungsumgebung)
      - Visual Studio Code
      - ggf. Eclipse

Soft- und Hardware für Data Science

Python

- Version 3 oder höher (**nicht 3.7 und nicht 3.8**), **64-bit-Ausführung**
- Installation: https://docs.python.org/3/using/index.html
- Python Packages
  - Nützliche und notwendige Erweiterungen der Standard-Python-Installation
  - Package Installation Tool pip: https://pip.pypa.io/en/stable/ (in Python enthalten)
  - Package Index: https://pypi.org/
- Python Virtual Environments
  - Für das Arbeiten mit separierten Python-Konfigurationen
  - https://docs.python.org/3/tutorial/venv.html

Soft- und Hardware für Data Science

- Installation von Python, pip, Jupyter, etc:
  - Windows (beispielhaft)
    - pip: `%SYS_PATH%\Scripts\pip.exe`
    - python: `%SYS_PATH%\python.exe`
    - Jupyter: `%SYS_PATH%\Scripts\jupyter.exe`

    - `SYS_PATH = C:\Program Files\Python36` (beispielhaft)

- Kommandozeile und Umgebung
  - https://docs.python.org/3/using/cmdline.html

Beachte

- Koexistenz existierender Python-2-Installation möglich. In neuer Python-3-Installation sind die Anwendungen mit Suffix „3“ aufrufbar. Beispiel: `python3`
- Empfehlung: Entferne existierende Python-3-Installation vor Installation einer neuen Python-3-Installation. Alternative: Verwendung virtueller Python-Umgebungen.
- Manche Installationen (z. B. über Anaconda) haben verschiedene Installationspfade
- Ggf. muss der Installationspfad noch nachträglich in die PATH-Umgebung eingetragen werden

Pfadanpassungen

- **Linux / MacOS (per user) –** Beispiel: ba shell (bash)
  - dauerhaft:

    Öffne (i. B. mit Editor nano) die Datei `~/.bash_profile`

    ```
    $ nano ~/.bash_profile
    ```

    Füge folgende Zeile am Ende ein – speichern nicht vergessen:

    ```
    export PATH=$PATH:/dir1/location1:/dir2/location2
    ```

  - transient:

    ```
    PATH=$PATH:/dir1/location1:/dir2/location2
    export PATH
    ```

- **Linux / MacOS (global – empfohlen)**
  1. Lege neue Datei `mynewPathFile1` in `/etc/paths.d/` an
  2. Füge neuen Pfad /`dir1/location1:/dir2/location2` in dieser Datei am Ende ein
  - Kommando:

    ```
    $ sudo -s 'echo "/dir1/location1:/dir2/location2" > /etc/paths.d/mynewPathFile1'
    ```

Pfadanpassungen

Windows

- über Systemsteuerung → System → Erweiterte Systemeinstellungen → Umgebungsvariablen → User Variable / System Variable → `PATH` bearbeiten

  (Semikolon-separiert oder neuer Zeileneintrag)

- über Kommandozeile
  - user path

```
set path "%path%;c:\dir1\location1;d:\dir2\location2"
```

  - system (and user) path

```
pathman
 /as path[;path[;path ...]]  Adds semicolon-separated paths to system path
 /au path[;path[;path ...]]  Adds semicolon-separated paths to user path
 /rs path[;path[;path ...]]  Removes semicolon-separated paths from system path
 /ru path[;path[;path ...]]  Removes semicolon-separated paths from user path
```

Soft- und Hardware für Data Science

Python

- **Wichtige Python Packages**

  ipython
  jupyter
  matplotlib
  notebook
  numpy
  pandas
  pip
  scikit-learn
  scipy
  tensorflow → **Stand 12.2020: Inkompatibilität mit Python 3.7 und 3.8, noch nicht in 3.9**
  virtualenv
  …

Soft- und Hardware für Data Science

Python

- Python Packages Management (mit Tool pip) – vorab: `pip install –upgrade pip`
  - Installation
    - Neuinstallation
      - `pip install --user [package name]`
    - Installation einer spezifischen Version
      - `pip install --user [package name]==[version number]`
    - Installation ab und unter spezifischen Versionen
      - `pip install --user [package name]=[version 1],< [version 2]`
  - Upgrade eines existierenden Pakets
      - `pip install --upgrade --user <package name>`
  - Liste aller installierten Pakete
      - `pip list`
  - Deinstallation eines existierenden Pakets
      - `pip uninstall --user <package name>`
  - Weitere Infos
      - https://docs.python.org/3/tutorial/venv.html#managing-packages-with-pip

Python

Entwicklungsumgebung Visual Studio Code (VSC)

- Installation
  - Setup VSC
    - https://code.visualstudio.com/docs/setup/setup-overview
  - Python Extension für VSC
    - https://marketplace.visualstudio.com/items?itemName=ms-python.python
- Python-auf-VSC-Tutorium
  - https://code.visualstudio.com/docs/python/python-tutorial

Soft- und Hardware für Data Science

Jupyter und Jupyter Notebooks

- Web-Anwendung / Web-Dokument: Lokal oder auf Server
- Erlaubt interaktive Erstellung und Freigabe von Inhalten und Nutzung von Python
- Notebook besteht aus Zellen vom Typ
  - *Code*: Jederzeit bearbeitbarer und ausführbarer Python Code
  - *Markdown*: Formatierter Text zur Beschreibung – einschließlich LaTeX
  - *Raw*: Unformatierter Text
  - *Heading*: Formatierbare Überschriften zur Strukturierung des Notebooks
- Visualisierung
  - eingebettete Grafiken
- Anwendungsgebiete:
  - Python-Programmierung, numerische Simulationen, statistische Modellierungen, Datenvisualisierung, Data Science, maschinelles Lernen, …

Jupyter und Jupyter Notebooks

- Installation: http://jupyter.org/install.html
  - `pip install --user jupyter`
- Lokale Konfiguration des Jupyter-Notebook-Verzeichnisses
  - `jupyter notebook --generate-config`
    - Erstellt Verzeichnis `.jupyter` im Home-Verzeichnis
      - Windows: `C:\Users\<username>\.jupyter`
      - Mac OS: `/Users/<username>/.jupyter`
    - Erstellt darin die Datei `jupyter_notebook_config.py`
    - In `jupyter_notebook_config.py`: De-Kommentierung von Property `c.NotebookApp.notebook_dir`
    - Setzen des Wertes dieser Property auf neuen Standardpfad für Jupyter Notebooks.
- Aufruf Jupyter
  - `jupyter notebook`

Zentrale Pakete

- numpy
- scipy
- matplotlib

Pakete für Data Science und Machine Learning

- scikit-learn
- tensorflow

`numpy`

- grundlegendes Paket für wissenschaftliches Rechnen mit Python
- Enthält u. a. :
  - flexiblen und mächtigen Datentyp für mehrdimensionale Arrays
  - mehrdimensionale Container für generische Daten
    - Integration mit diversen Datenbanken
  - Werkzeuge zur Integration von C/C++ and Fortran
    - viele Funktionen und Module in maschinen-nahen Sprachen
  - Funktionen und Methoden für lineare Algebra, Fourier-Transformationen, Zufallszahlen, …
  - Vektorisierte Operationen auf Objekten von Datensätzen
- Mehr unter: http://docs.scipy.org/doc → Numpy Reference Guide

`scipy`

- verwendet `numpy`
- Sammlung vielfältiger mathematischer Operationen und numerischer Algorithmen
- Domänen-spezifische Werkzeugen zur Signalverarbeitung, Optimierung, Statistik, …
- Mehr unter
  - https://www.scipy.org
  - http://docs.scipy.org/doc → Scipy Reference Guide

`matplotlib`

- zur grafischen 2D-Visualisierung und Plotting von Daten und Zusammenhängen
- erzeugt hochwertige Grafiken
- nutzbar in Python-Programmen, Python Shell, Jupyter Notebook, Web-Anwendungen, …
- Mehr unter
  - https://matplotlib.org/
  - http://matplotlib.sourceforge.net

VORSICHT!

Stand 11.2019: Inkompatibilität von `matplotlib` mit Python 3.8

`scikit-learn`

- basierend auf `numpy`, `scipy`, `matplotlib`
- für Datenanalyse, Data Mining, maschinelles Lernen
  - Klassifikation, Regression, Clustering, …
    - Entscheidungsbäume
    - Ensemble Learning und Random Forests
    - Gradientenverfahren
    - Support-Vektor-Maschinen
    - k-means
    - Dimensionsreduktion

`scikit-learn`

Allgemeine Prinzipien

- **Zentrale Entitäten:** Python-Objekte
- **Einheitliche API:** Alle Objekte bauen auf einheitlichen Interfaces (API) auf
  - Wichtige Interfaces
    - `Estimator` API zum Lernen und Bau von Modellen
    - `Predictor` API für die Anwendung von Vorhersagen
    - `Transformer` API zur Datentransformation
    - `Pipeline` API zur Kopplung mehrerer `Estimator`-Instanzen
- **Inspection:** Konstruktor-Parameter und Parameter-Werte der Modelle sind öffentliche Attribute
- **Abgrenzung der Klassen:** Nur Lernalgorithmen werden durch die Klassen bereitgestellt. Datensätze und Parameter sind Standard-Python-Konstrukte.
  - Vorteil: Scikit-Learn einfach zu bedienen und kombinierbar mit anderen Bibliotheken
- **Verknüpfung:** Sequenzen oder Kombinationen von ML-Aufgaben

`scikit-learn`

Datenrepäsentation

- Basierend auf `numpy` und `scipy`
  - Datensätze für dichte Daten: Multidimensionale `numpy arrays`
  - Datensätze für dünnbesetzte Daten: `scipy sparse matrices`

`scikit-learn`

Estimator

- Zentrales Interface
  - Mit `fit`-Methode für das Lernen und die Modellgenerierung
    - Die Estimator-Instanz (self) wird als Modell zurückgegeben
    - Dieses Modell wird auf neuen Daten getestet bzw. angewendet
- Klassen, die `Estimator` implementieren:
  - alle Lernalgorithmen
  - Feature-Extraktion, Feature-Selektion, Dimensionsreduktion

`scikit-learn`

Predictor

- erweitert das `Estimator`-Interface
- Zusätzliche Methoden
  - `predict` – Vorhersage / Bestimmung von Zielwerten von Datenobjekten
  - `score` – Bestimmung der Güte des Modells unter Verwendung von Testdaten

`scikit-learn`

Transformer

- erweitert `Estimator`-Interface
- Zusätzliche Methoden
  - `transform` – transformiert Eingabedaten auf ein benötigtes Format
  - Anwendungsbeispiele
    - Datenvorbverarbeitung, Merkmalselektion/-extraktion, Dimensionsreduktion
- Beispiel: Standardisiere Input auf 0-Mittelwert mit `StandardScaler`

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_dataset)
X_adjusted_dataset = scaler.transform(X_dataset)
```

`scikit-learn`

Pipeline

- erweitert `Estimator`-Interface
- Kettet mehrere (n) Estimator-Instanzen zu einem Estimator zusammen
  - Erste n-1 Komponenten sind Transformer, letzte ist Estimator
- Use Case: Mehrere Lernalgorithmen arbeiten zusammen

Feature Union

- Erweitert `Transformer`-Interface
- Mehrere `Transformers` werden zu einem `Transformer` kombiniert
- Ausgaben werden als Eingaben des nächsten `Transformer` verwendet bzw. am Ende als Transformationsergebnis ausgegeben

scikit-learn

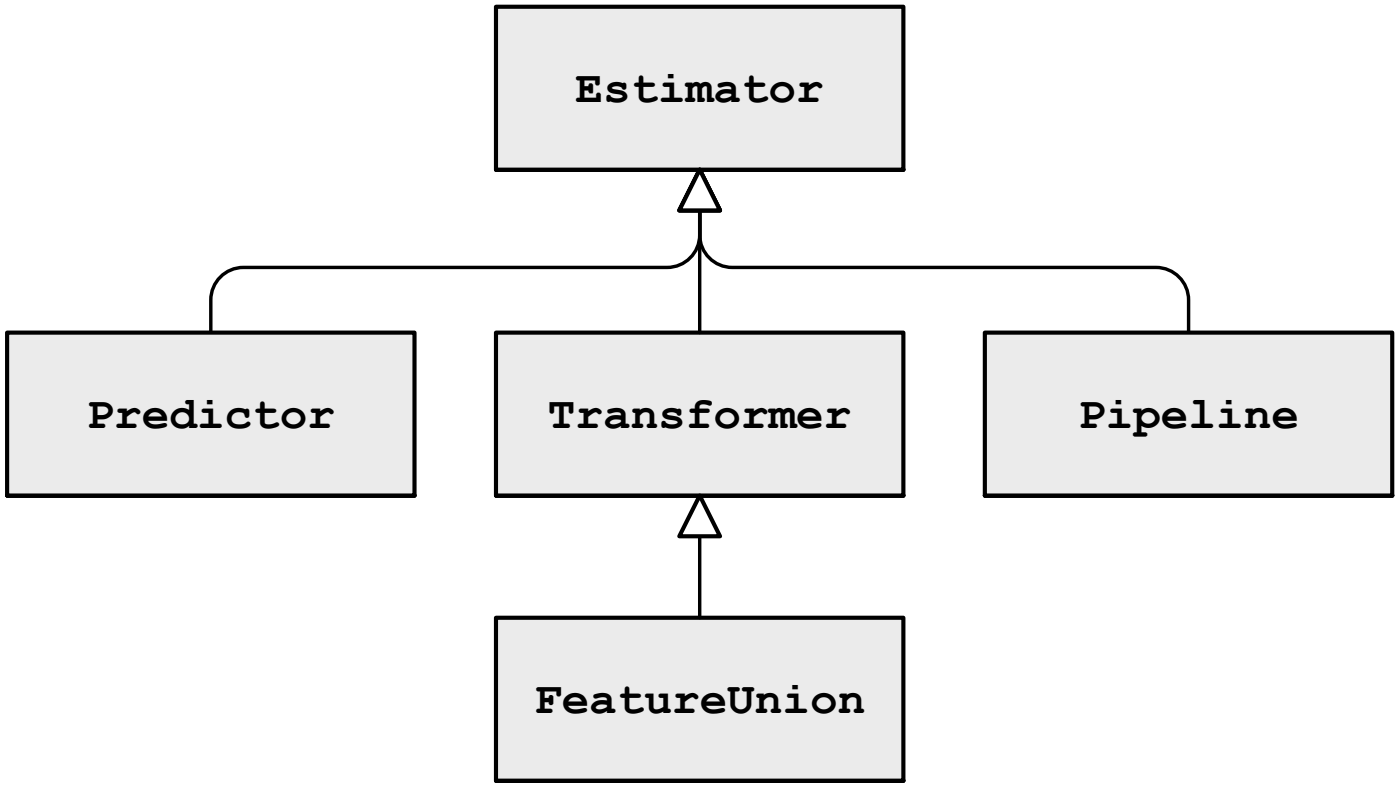
Pipeline / Feature Union

- Beispiel:

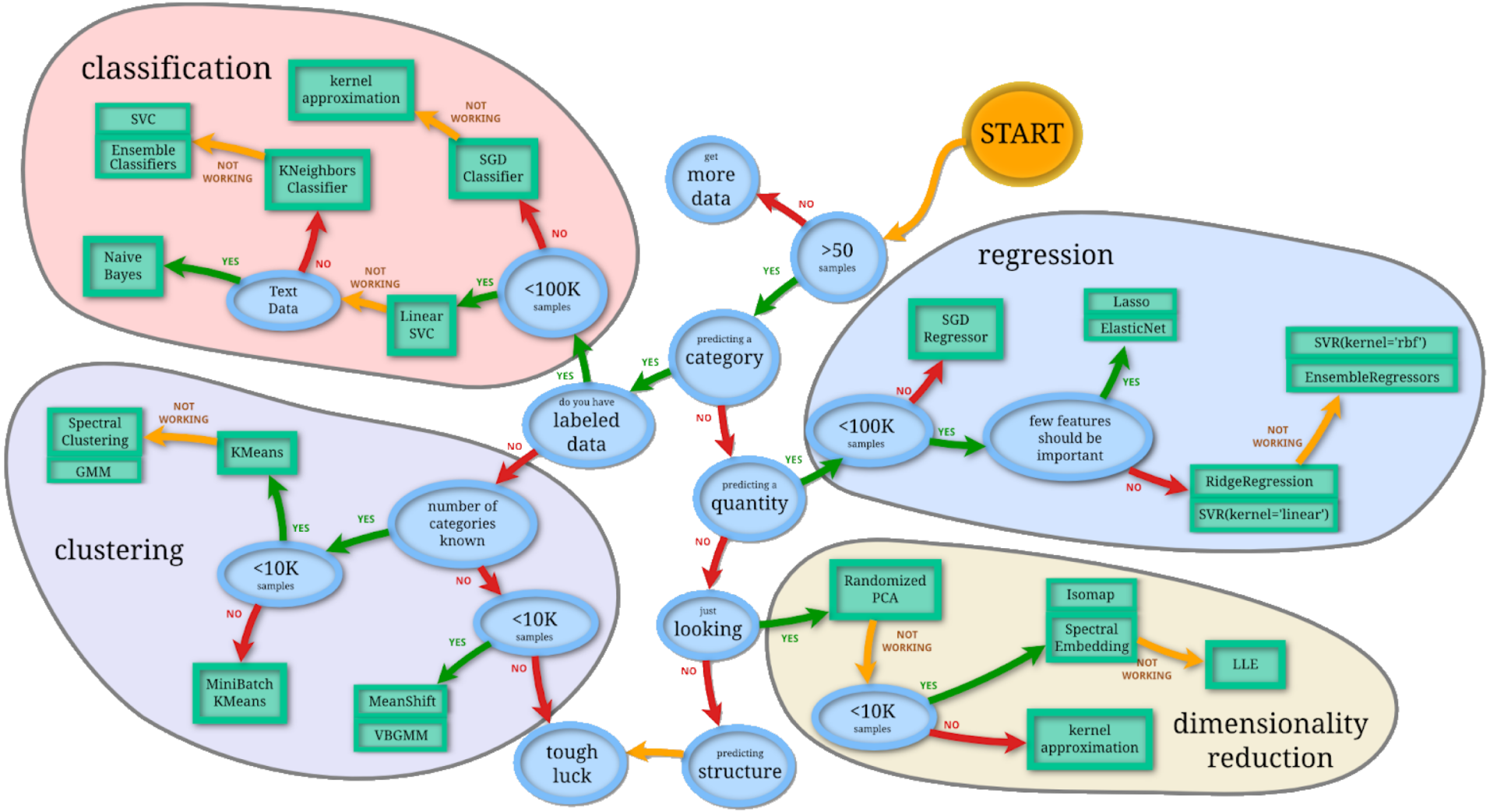
```
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.decomposition import PCA, KernelPCA
from sklearn.feature_selection import SelectKBest

union = FeatureUnion([("pca", PCA()),
                      ("kpca", KernelPCA(kernel="rbf"))])

Pipeline([("feat_union", union), ("feat_sel", SelectKBest(k=10)),
          ("log_reg", LogisticRegression(penalty="l2"))
         ]).fit(X_train, y_train).predict(X_test)
```

scikit-learn

Interface-Hierarchie



`scikit-learn` Algorithmen-Map von Scikit-Learn (Auswahl)



`tensorflow`

- Installation (bis Python Version 3.6)
  - Main: https://www.tensorflow.org/install/
  - Alternativ über wheel
    - Mac: pip install https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.12.0-py3-none-any.whl
    - Windows: https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.12.0-cp36-cp36m-win_amd64.whl

    (hier: v 1.12, ohne GPU support)

VORSICHT!

Stand 11.2019: Inkompatibilität von Tensorflow mit Python 3.7 (bedingt) und 3.8