Scikit-Learn Merkblatt-1/2

Scikit-Learn Beispiel

Mit Scikit-Learn können komplexe Aufgaben im Bereich maschinelles Lernen mit wenigen Zeilen Code ausgeführt werden.

Iris Datensatz clustern

In diesem Beispiel wird der in Scikit-Learn integrierte Iris-Datensatz mit dem **k-nearest-neighbour**-Verfahren in fünf Cluster unterteilt und anschliessend die Effizienz des Verfahrens bewertet.

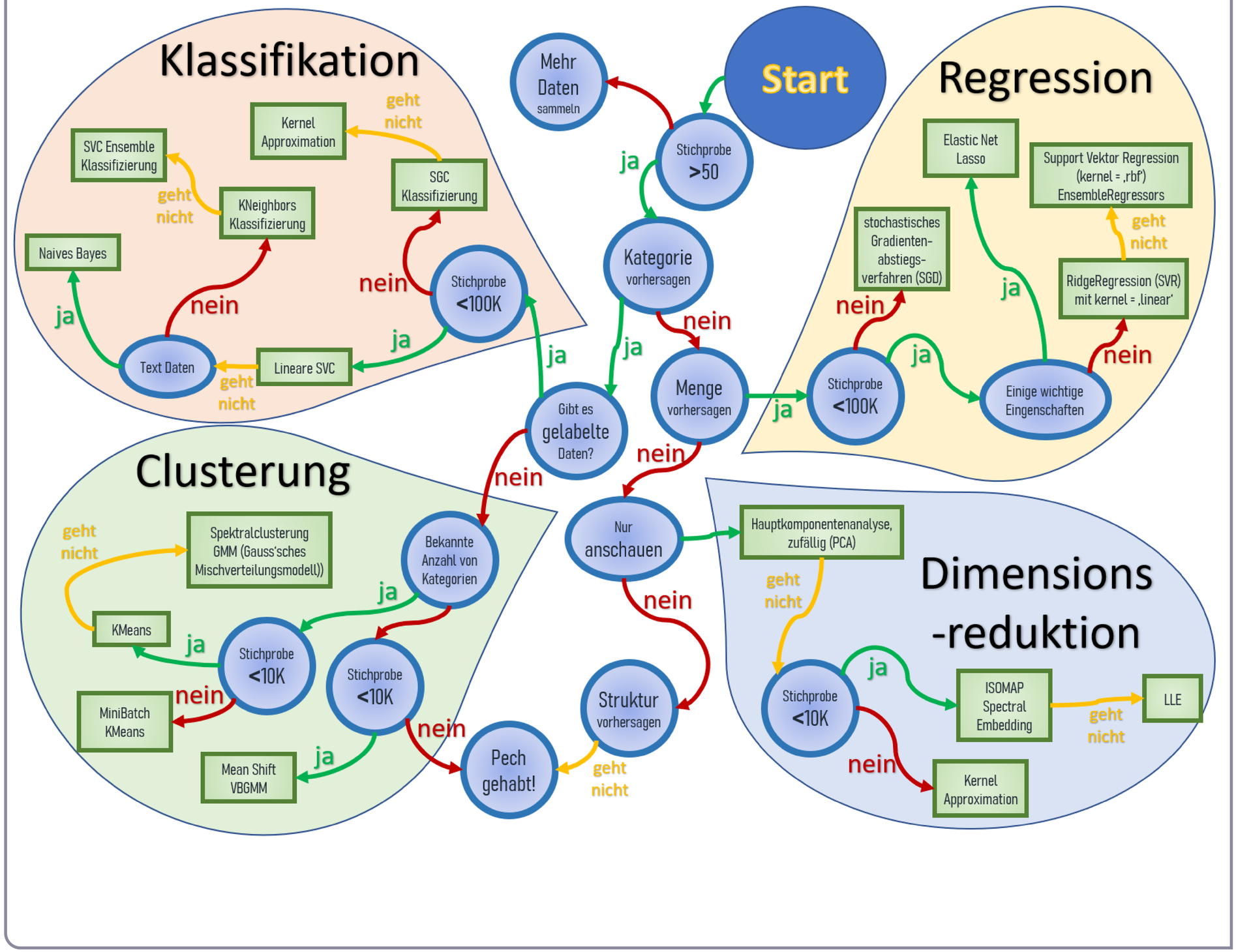
```
from sklearn import neighbors, datasets, \
    preprocessing
from sklearn.cross_validation import \
  train_test_split
from sklearn.metrics import \
    accuracy_score

iris = datasets.load_iris()
X, y = iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, random_state=33)

scaler = preprocessing.StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

knn = neighbors.KNeighborsClassifier(
  n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_score(y_test, y_pred)
```

Scikit-Learn Übersicht



Scikit-Learn Merkblatt-2/2

Scikit-Learn Grundlagen

Scikit-Learn ist ein Open-Source-Modul in Python, welches eine Reihe von Algorithmen der künstlichen Intelligenz sowie Methoden zur Datenvorbereitung, Validierung und Visualisierung zur Verfügung stellt.

Daten einlesen

Die Daten müssen in numerischer Form als NumPy-Arrays oder SciPy-Matrizen (Sparse Matrices) vorliegen. Pandas-Tabellen (DataFrames) können ebenfalls eingelesen werden.

```
import numpy as np
X = np.random.random((9,7))
y = np.array(['X','X','Y','X','Y','Y',
  'Y','X','Y','X','X'])
X[X < 0.5] = 0
```

Aufteilen in Training- und Testdaten

```
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,
  random_state=9)
```

Überwachte Lernverfahren

Lineare Regression:

```
from sklearn.linear_model import \
  LinearRegression
lr = LinearRegression(normalize=True)
```

Support Vector Machines:

```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
```

Naive Bayes:

```
from sklearn.naive_bayes import
GaussianNB
gnb = GaussianNB())
```

KNN:

```
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(
  n_neighbors=4)
```

Unüberwachte Lernverfahren

Hauptkomponentenanalyse:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.92)
```

K-Means:

```
from sklearn.cluster import
KMeans
k_means = KMeans(
  n_clusters=4, random_state=0)
```

Training und Evaluation

Modell trainieren

Überwachte Lernverfahren:

```
lr.fit(X, y)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
```

Unüberwachte Lernverfahren:

```
k_means.fit(X_train)
pca_model = pca.fit_transform(X_train)
```

Vorhersage

Überwachte Lernverfahren:

```
# svc und lr: Labels
y_pred = svc.predict(
  np.random.random((6,3)))
y_pred = lr.predict(X_test)

# knn: Wahrscheinlichkeit von Labels
y_pred = knn.predict_proba( X_test )
```

Unüberwachte Lernverfahren:

```
# Clustert nach Labels
y_pred = k_means.predict(X_test)
```

Klassifikationskennzahlen

Genauigkeit:

```
# Scoring des Lernverfahrens
knn.score(X_test, y_test)
# Unabhängige Scoringkennzahl
from sklearn.metrics import \
  accuracy_score
accuracy_score(y_test, y_pred)
```

Klassifikationsreport:

```
# Präzision, recall, f1-Score und Support
from sklearn.metrics import \
  classification_report
classification_report(y_test, y_pred)
```

Konfusionsmatrix:

```
from sklearn.metrics import \
  confusion_matrix
confusion_matrix(y_test, y_pred))
```

Kreuzvalidierung

```
from sklearn.cross_validation import \
  cross_val_score
print(cross_val_score(
  knn, X_train, y_train, cv=3))
print(cross_val_score(lr, X, y, cv=4))
```

Evaluation

Regressionskennzahlen

Absolute Abweichung vom Mittelwert:

```
from sklearn.metrics import \
  mean_absolute_error
y_true = [-1.1, 3, 2.2]
mean_absolute_error(y_true, y_pred)
```

Quadratische Abweichung vom Mittelwert:

```
from sklearn.metrics import \
  mean_squared_error
mean_squared_error(y_test, y_pred)
```

R^2-Score:

```
from sklearn.metrics import r2_score
r2_score(y_true, y_pred)
```

Clusterkennzahlen

Adjustierter Rand-Index:

```
from sklearn.metrics import \
  adjusted_rand_score
adjusted_rand_score(y_true, y_pred)
```

Homogenität:

```
from sklearn.metrics import \
  homogeneity_score
homogeneity_score(y_true, y_pred)
```

V-Measure:

```
from sklearn.metrics import \
  v_measure_score
metrics.v_measure_score(y_true, y_pred)
```

Modell verfeinern

Grid Search:

```
from sklearn.grid_search import \
  GridSearchCV
params = {"n_neighbors": np.arange(2,4),
  "metric": ["euclidean", "cityblock"]}
grid = GridSearchCV(estimator=knn,
  param_grid=params)
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_estimator_.n_neighbors)
```

Optimierung mit randomisierten Parametern:

```
from sklearn.grid_search import \
  RandomizedSearchCV
params = {"n_neighbors": range(1,3),
  "weights": ["uniform", "distance"]}
rsearch = RandomizedSearchCV(
  estimator=knn,
  param_distributions=params,
  cv=4, n_iter=7, random_state=4)
rsearch.fit(X_train, y_train)
print(rsearch.best_score_)
```

Preprocessing

Daten vorbereiten

Standardisierung:

```
from sklearn.preprocessing import \
  StandardScaler
scaler = StandardScaler().fit(X_train)
std_X = scaler.transform(X_train)
std_X_test = scaler.transform(X_test)
```

Normalisierung:

```
from sklearn.preprocessing import \
  Normalizer
scaler = Normalizer().fit(X_train)
nrm_X = scaler.transform(X_train)
nrm_X_test = scaler.transform(X_test)
```

Binärisierung:

```
from sklearn.preprocessing import \
  Binarizer
binarizer = Binarizer(threshold=0.0).fit(X)
bin_X = binarizer.transform(X)
```

Kategoriemerkmale encodieren:

```
from sklearn.preprocessing import \
  LabelEncoder
enc = LabelEncoder()
enc_y = enc.fit_transform(y)
```

Fehlende Werte einrechnen:

```
from sklearn.preprocessing import \
  Imputer
imp = Imputer(missing_values=0,
  strategy='mean', axis=0)
imp.fit_transform(X_train)
```

Polynomiale Features generieren:

```
from sklearn.preprocessing import \
  PolynomialFeatures
poly = PolynomialFeatures(4)
poly.fit_transform(X)
```