

```
import handout # Tool for generating report-type documents
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Load the dataset. We used the Car Evaluation dataset from <https://archive.ics.uci.edu/> To use the dataset for ML algorithms all variables are changed to numeric variables by label-encoder preprocessing.

```
features = ["Buying", "Maintenance", "Doors", "Persons", "Luggage", "Safety"]
target = 'Class'
df = pd.read_csv(
    'https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data',
    names=features + [target])
```

	Buying	Maintenance	Doors	Persons	Luggage	Safety	Class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Some setup code for the KNN classifier; data splitting and scaling:

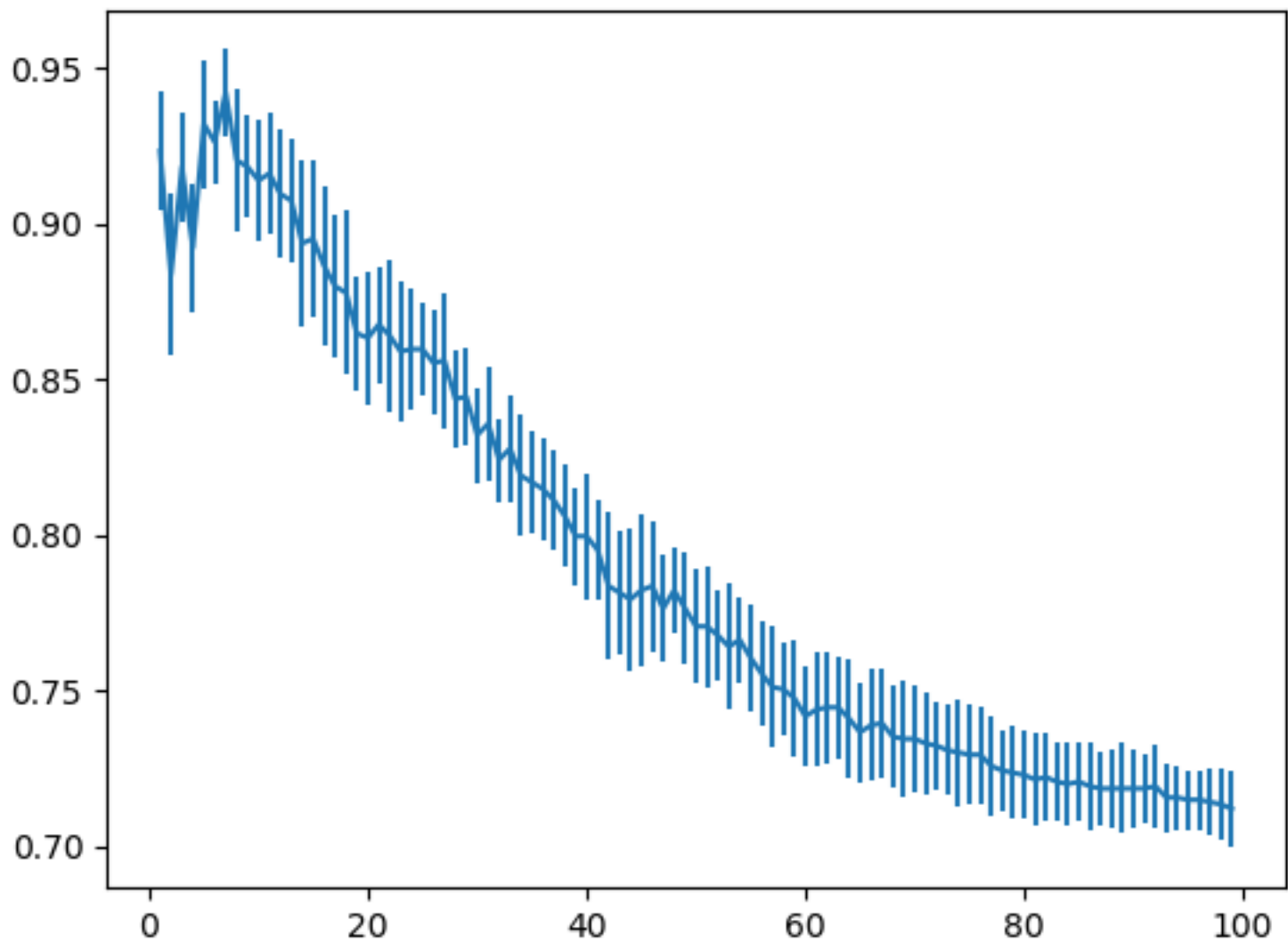
```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.2)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

data_scaled = pd.DataFrame(
    StandardScaler().fit_transform(df[features]),
    columns=features)
data_scaled[target] = df[target]
train_data_s, test_data_s = train_test_split(data_scaled,
    test_size=0.2)
```

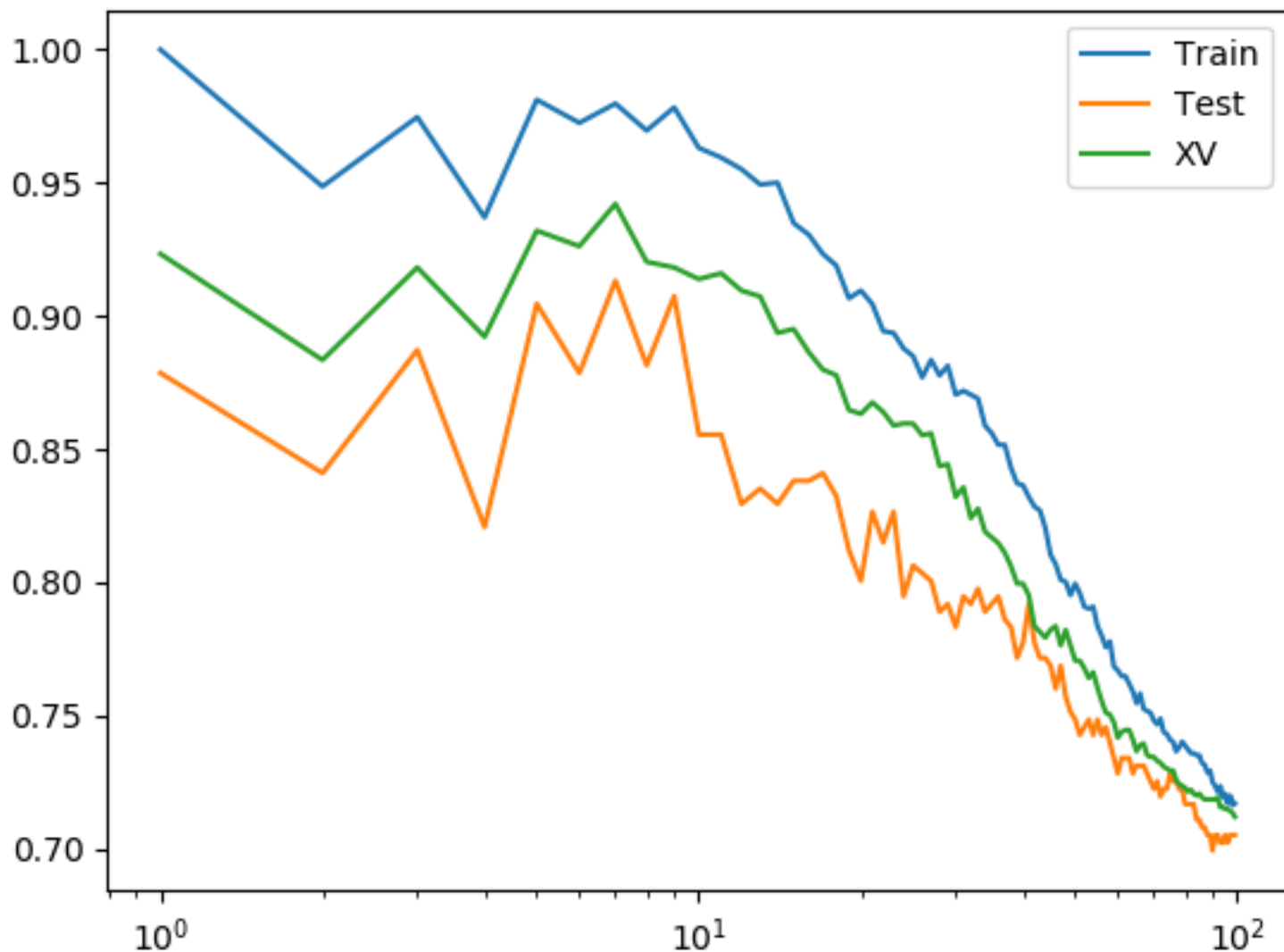
Using cross-validation to test which k produces the best results This code trains KNN models with different k using cross-validation. The point is to get an impression of how our choice of k will affect the expected accuracy on the test-set.

```
from sklearn.model_selection import cross_val_score
neighbor_ks = range(1, 100)
untrained_models = [KNeighborsClassifier(n_neighbors=k) for k in neighbor_ks]
k_fold_scores = [cross_val_score(estimator=m,
    X=train_data_s[features],
    y=train_data_s[target],
    cv=10) for m in untrained_models]
mean_xv_scores = [s.mean() for s in k_fold_scores]
plt.errorbar(neighbor_ks,
    mean_xv_scores,
    yerr=[s.std() for s in k_fold_scores])
```



Results of using different k's on the test-set This code compares KNN models trained with different k on the testset, plotted alongside results on the training data and the results from the cross-validation.

```
models = [KNeighborsClassifier(n_neighbors=k).fit(train_data_s[features],
train_data_s[target]) for k in neighbor_ks]
train_scores = [accuracy_score(train_data_s[target],
m.predict(train_data_s[features])) for m in models]
test_scores = [accuracy_score(test_data_s[target],
m.predict(test_data_s[features])) for m in models]
plt.semilogx(neighbor_ks, train_scores,
neighbor_ks, test_scores,
neighbor_ks, mean_xv_scores)
plt.legend(["Train", "Test", "XV"])
```



Conclusion We expect the model to be slightly more accurate than the cross-validation on the test-set because the cross-validation models are trained on a slightly smaller dataset. That is also what we observe in the above graph. For a deployment of the model to the "real-world", we'd assume the model to perform slightly worse (have a lower accuracy than shown in the graphs), because of possible sampling bias in the dataset we've used. Also, the dataset is a representation of only a (small) part of the 'real world', which means that the accuracy will always be lower. For the hyperparameter k , we can see that from $k > 10$ the accuracy of the model decreases, whereas a k around 6-8 seems to be the best option for this model.