The K-Nearest Neighbors Algorithm

February 4, 2026

Team 2

Computerpraktikum Teil 2

Inhaltsverzeichnis

Problemstellung

Problemstellung

- Aufgabe: Methode zur **binären Klassifikation** in Python
- Gegeben: Datenpuntke mit Labels

$$D = \{(y_i, x_i)\}_{i=1}^{n},\ y_i \in \{-1, +1\},\ x_i \in [-1, +1]^d$$

- Lerne einen Klassifikator

$$f_D : [-1, +1]^d \to \{-1, +1\}$$

- Ziel: **Minimierung der Fehlklassifikationsrate** auf unbekannten Testdaten D'
- Ansatz: k-**nächste-Nachbarn** mit Kreuzvalidierung zur Auswahl von k^* mit Ball-Tree

Aufbau von `classify.py`

Laden von Datensätzen: Alte Version

```
try:
    import csv
    data = []
    with open(args.datasetname, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            if not row:
                continue
            try:
                label = int(row[0])
                features = [float(v) for v in
                    row[1:]]
                data.append((label, features))
            except ValueError:
                print("Warning: skipping
                    malformed row", row)
    print(f"Dataset loaded successfully. ...
```

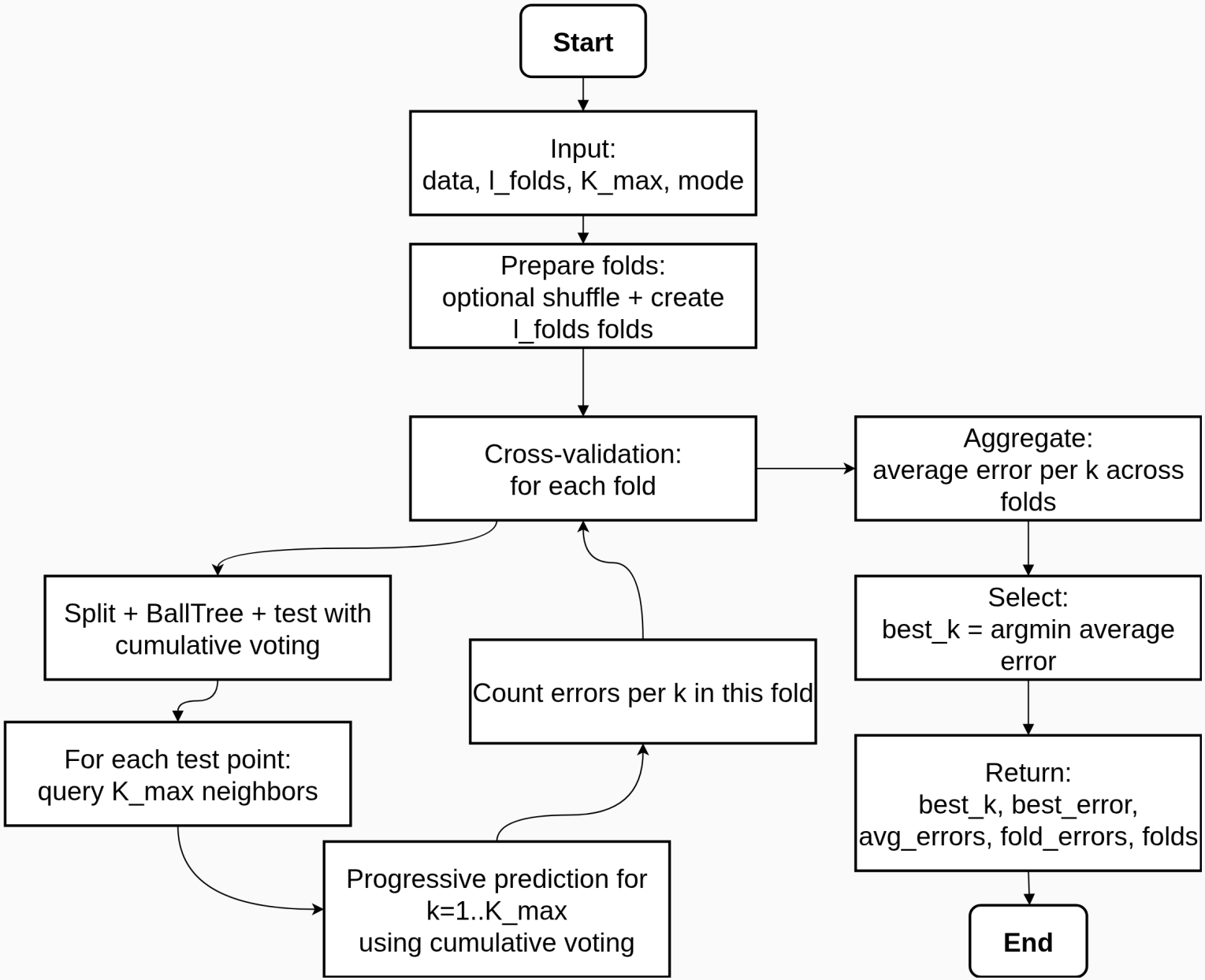
Laden von Datensätzen: Neue Version

```
def load_data(filename):
    data = []
    try:
        with open(filename, 'r') as f:
            for line in f:
                if not line.strip(): continue
                parts = line.split(',')
                data.append((float(parts[0]),
                    list(map(float, parts[1:]))))
    except Exception as e:
        print(f"Error while loading: {e}")
        sys.exit(1)
    return data
```

Erstellen vom Interface

```
usage: classify.py [-h] [-f l] [-k Kmax] [-d mode] [-n N] datasetname

KNN classification with l-fold cross validation.

The program determines the optimal k* ∈ {1, ..., kmax} using cross
 validation on the training data and then applies the resulting classifier
 f_D to the test data.

positional arguments:
  datasetname  Name of the dataset (without file extension).
               The following files are expected:
                 ../classification-data/<datasetname>.train.csv
                 ../classification-data/<datasetname>.test.csv

options:
  -h, --help   show this help message and exit
  -f l         Number of folds for cross validation (default: 5).
               The training data is split into l subsets D1, …, Dl.
  -k Kmax      Maximum value of k (default: 200).
               The set K = {1, 2, …, Kmax} is evaluated.
  -d mode      Mode for generating the folds (default: 0).
                 0: Random partitioning of the data
                 1: Deterministic partitioning as specified in the assignment:
                    D1 = (y1, x1), (yl+1, xl+1), (y2l+1, x2l+1), ...
                    D2 = (y2, x2), (yl+2, xl+2), (y2l+2, x2l+2), ...
                    ...
  -n N         Optional additional parameter.
               Uses only the first N training samples.
               This parameter does not affect the default behavior
               and is intended solely for testing or runtime experiments.
```

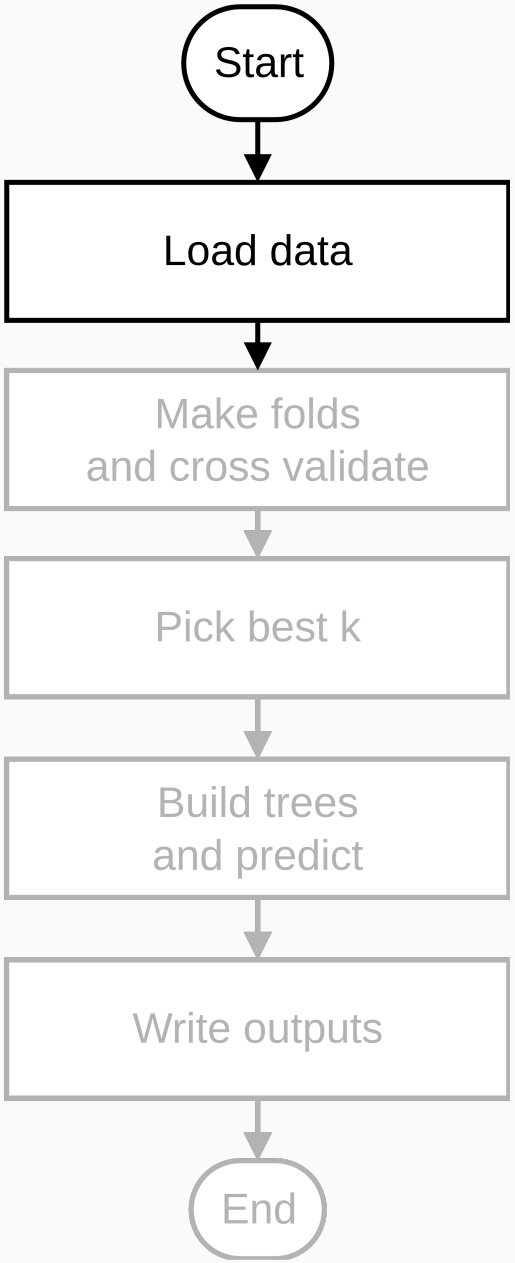
Die `run_cross_validation` Funktion



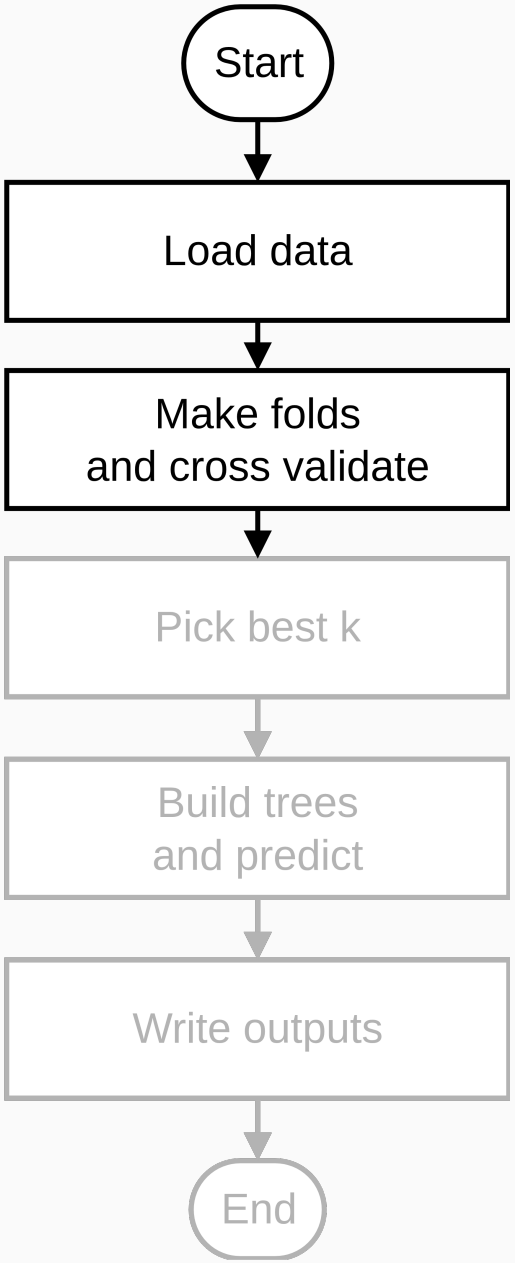
Die `run_cross_validation` Funktion

```
def run_cross_validation(data, l_folds, K_max, mode):
    n = len(data)
    if mode != 1: random.shuffle(data)
    folds = [data[i::l_folds] for i in range(l_folds)]
    fold_errors = {k: [] for k in range(1, K_max + 1)}
    for i in range(l_folds):
        test_set = folds[i]
        train_set = []
        for j in range(l_folds):
            if i != j: train_set.extend(folds[j])

        tree = BallTree(train_set)
        current_fold_counts = {k: 0 for k in range(1, K_max + 1)}

        for y_true, x_test in test_set:
            neighbors = tree.query(x_test, K_max)
            current_sum = 0
            for idx, label in enumerate(neighbors):
                k = idx + 1
                current_sum += label
                y_pred = 1.0 if current_sum >= 0 else -1.0
                if y_pred != y_true:
                    current_fold_counts[k] += 1

        for k in range(1, K_max + 1):
            fold_errors[k].append(current_fold_counts[k] / len(test_set))
    avg_errors = {k: sum(fold_errors[k]) / l_folds for k in range(1, K_max + 1)}
    best_k = min(avg_errors.items(), key=lambda x: x[1])[0]
```

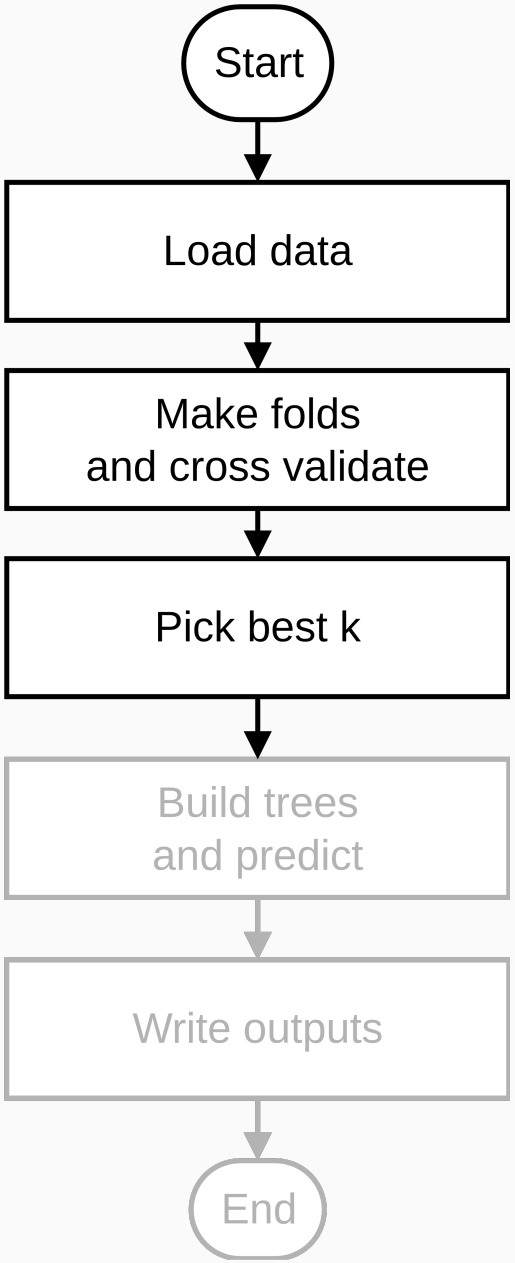
Die main Funktion



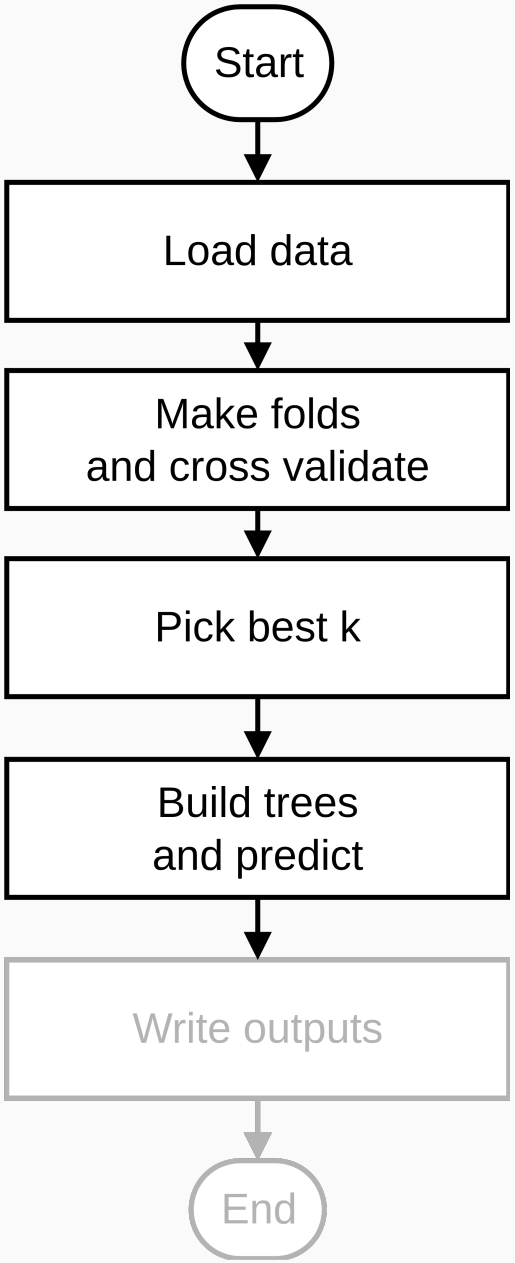
Die main Funktion



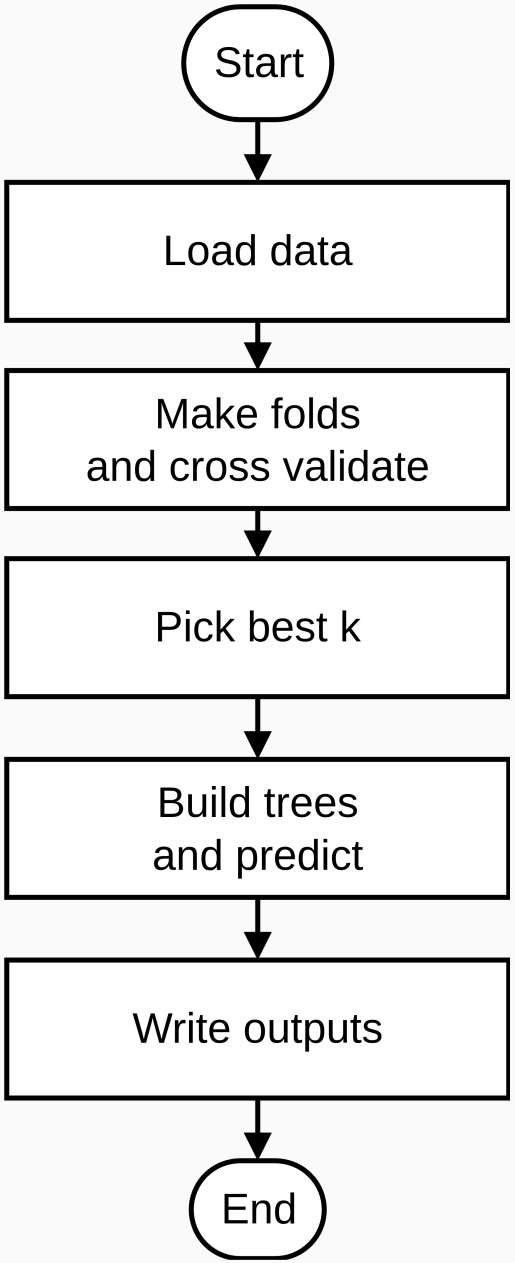
Die main Funktion



Die main Funktion



Die main Funktion



Aufbau von `ball_tree.py`

Fehlerreduktionsstrategien

Optimierung: Versuchte Methoden

- **stratified l-fold**
- additional distance metrics
- higher percision summation
- leaf size Variation

```
from collections import defaultdict
import random
def make_stratified_folds(data, l_folds, seed=42):
    rnd = random.Random(seed)
    buckets = defaultdict(list)
    for y, x in data:
        buckets[y].append((y, x))
    for y in buckets:
        rnd.shuffle(buckets[y])
    folds = [[] for _ in range(l_folds)]
    for y, items in buckets.items():
        for i, item in enumerate(items):
            folds[i % l_folds].append(item)
    return folds
```

Optimierung: Versuchte Methoden

- stratified l-fold
- **additional distance metrics**
- higher percision summation
- leaf size Variation

```
if self.metric == "l2":
  return sum((x - y) ** 2 for x, y in zip(a, b))
if self.metric == "l1":
  return sum(abs(x - y) for x, y in zip(a, b))
# linf
    return max(abs(x - y) for x, y in zip(a, b))
```

Optimierung: Versuchte Methoden

- stratified l-fold
- additional distance metrics
- **higher percision summation**
- **leaf size Variation**

`math.fsum(...)`

Lernerfolge

Lernerfolge

- GitHub Erfahrung
- Linux und Terminal Nutzung
- Fehlersucheingrenzung und Debugging
- Datenstrukturen analysieren
- Performance Optimierung