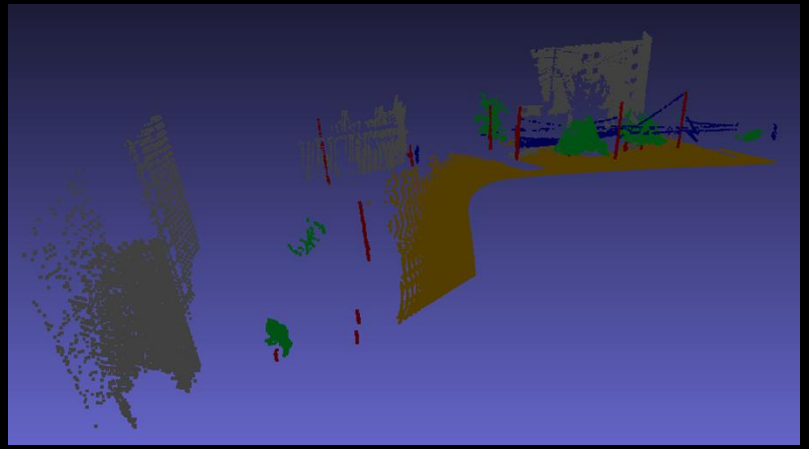
# Semantische Klassifikation von 3D-Punktwolken



k = 50

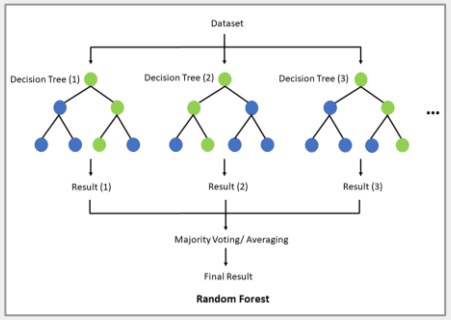
Ramon, Jan, Korvin

Übung 2 – Texturanalyse
Struktur- und Objektextraktion in 2D und 3D

# Gliederung

- Theorie Random Forest Classifier
- Programmcode
  - k-Nächste-Nachbarn
  - Covariance Features
  - Random Forest Klassifikator
- Ergebnisse
  - Qualitätsmetriken
  - Punktwolken

# Theorie




© Wikipedia

- Grundlage sind Entscheidungsbäume
- Bootstrap:
  - Nutzung von x% der Datenmenge, um den Entscheidungsbaum aufzubauen
  - Validierung mittels der ungenutzten 100-x% ⇒ Aussage durch out-of-bag-error (OOB)

- Viele Entscheidungsbäume bilden einen Random Forest (Ensemble Methode)
- Wie ist die Baumanzahl zu wählen?
  - Viele Features und wenige Bäume ⇒ nicht alle Features werden genutzt
  - Viele Daten und wenige Bäume ⇒ nicht alle Kombinationen abgedeckt
  - Viele Bäume kosten nur Rechenleistung, nicht Genauigkeit! (kein overfitting)

- Weitere mögl. Parameter des Random Forests:
  - max. Baumtiefe
  - max. Blattanzahl

# Code: k-Nächste-Nachbarn

- Verwende Funktion aus scikit-learn:

```
from sklearn.neighbors import NearestNeighbors
```

- Finde Indizes der `k = 50` nächsten Nachbarn

```
# Initialize the NearestNeighbors model
nbrs = NearestNeighbors(n_neighbors=k + 1, algorithm='auto').fit(points)

# Find the k+1 nearest neighbors (including the point itself)
distances, indices = nbrs.kneighbors(points)
```

- Gebe Indizes und Nachbarpunkte zurück

```
# Get the coordinates of the neighbors
points_neighbors = points[indices]

return indices, points_neighbors
```

# Code: Covariance Features

```python
# Compute the covariance features for each point
for i in range(n):
    # Get the neighbors coordinates (x,y,z) of the point i
    neighbors = points_neighbors[i]

    #! Compute the covariance matrix
    num_neighbors = neighbors.shape[0]           # number of neighbors
    sample_mean   = np.mean(neighbors, axis=0)   # mean of the neighbors
    cov_matrix    = 1/(num_neighbors - 1) * (neighbors - sample_mean).T @ (neighbors - sample_mean)
```

3D-Strukturtensor

```python
    #! Compute the eigenvalues
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    eigenvalues    = np.flip(eigenvalues)                    # order largest first
    # alternativ: eigenvalues[::-1, ::-1] # ist eig. genau was np.flip macht

    #! Compute the covariance features
    lbd1, lbd2, lbd3 = eigenvalues
```

Eigenwerte

# Code: Covariance Features

```
linearity     = (lbd1 - lbd2) / lbd1
planarity     = (lbd2 - lbd3) / lbd1
scattering    = lbd3 / lbd1
omnivariance  = np.cbrt(lbd1 * lbd2 * lbd3)
anisotropy    = (lbd1 - lbd3) / lbd1
eigenentropy  = - sum(lbd * np.log(lbd) for lbd in (lbd1, lbd2, lbd3))
sum_eigen     = sum(eigenvalues)
change_curv   = lbd3 / sum_eigen
```

Covariance-Features
→ aus Eigenwerten bestimmt

```
# Store the covariance features (at 0th to 7th column of cov_features)
cov_features[i,:8] = [linearity, planarity, scattering, omnivariance, anisotropy, eigenentropy, sum_eigen, change_curv]
```

# Code: Random Forest Klassifikator

- Verwende RandomForestClassifier aus scikit-learn:

```
from sklearn.ensemble import RandomForestClassifier
```

- Parameter beim Training:
  - n_estimators: Baumanzahl
  - bootstrap: Verwende % Punkte des Trainingsdatensatz oder alle Trainingsdaten
  - max_depth: Baumtiefe

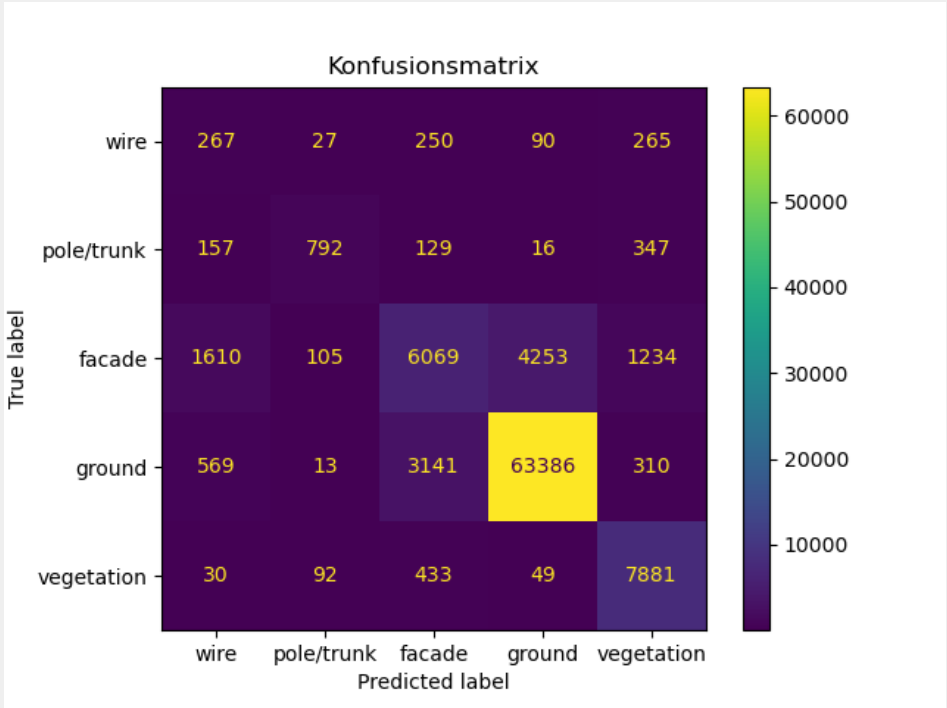
```
#  Initialize the Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=200,bootstrap=False,max_depth=None)
```

```
# Train the Random Forest Classifier
rfc = rfc.fit(X=cov_features_train,y=class_train)
```

- Anwendung des Klassifikators:

```
# Apply the Random Forest Classifier to the validation data
class_pred = rfc.predict(cov_features_valid)
```

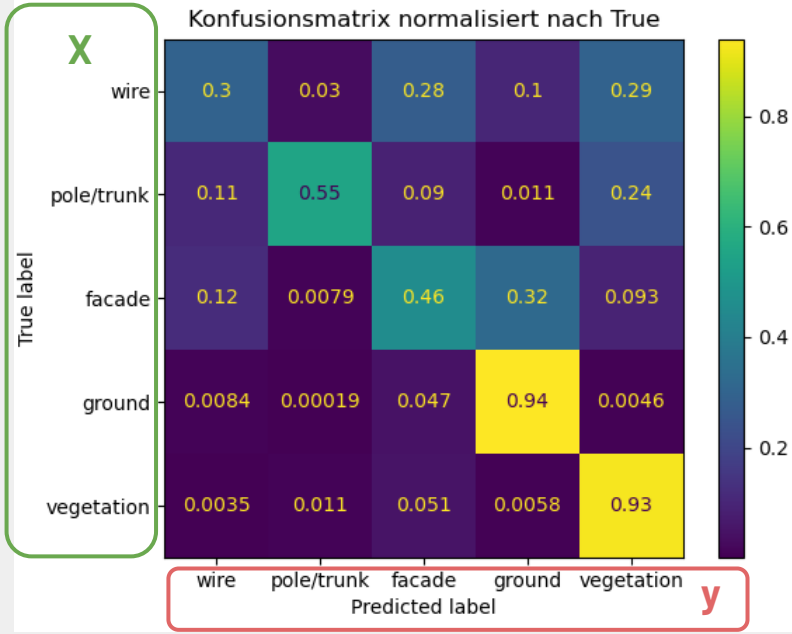
# Ergebnisse: Qualitätsmetriken

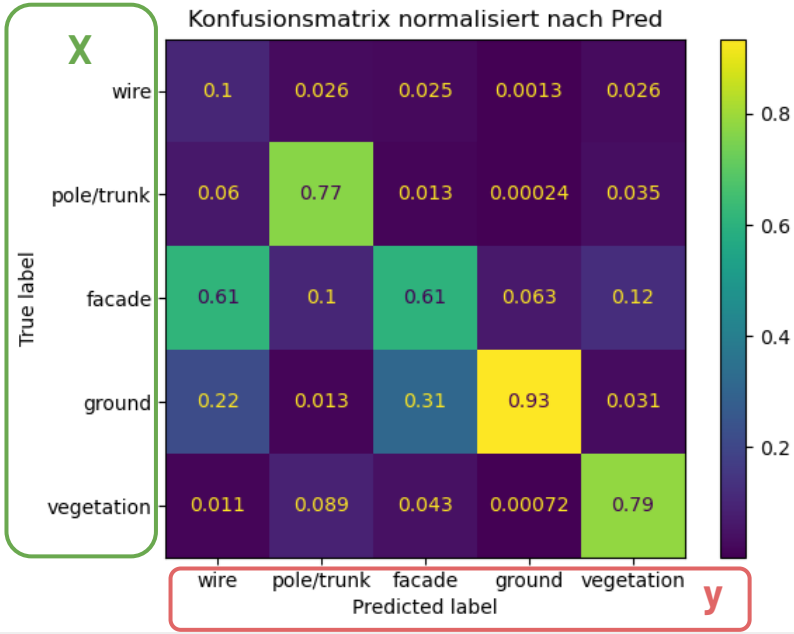



Classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| wire | 0.10 | 0.30 | 0.15 | 899 |
| pole/trunk | 0.77 | 0.55 | 0.64 | 1441 |
| facade | 0.61 | 0.46 | 0.52 | 13271 |
| ground | 0.93 | 0.94 | 0.94 | 67419 |
| vegetation | 0.79 | 0.93 | 0.85 | 8485 |
| accuracy | | | 0.86 | 91515 |
| macro avg | 0.64 | 0.63 | 0.62 | 91515 |
| weighted avg | 0.86 | 0.86 | 0.86 | 91515 |

# Ergebnisse: Qualitätsmetriken




Correctness:
"Klasse **X** wird zu p% als Klasse **y** prädiziert"

Completeness:
"Klasse **y** wurde zu p% aus Punkten der Klasse **X** prädiziert"

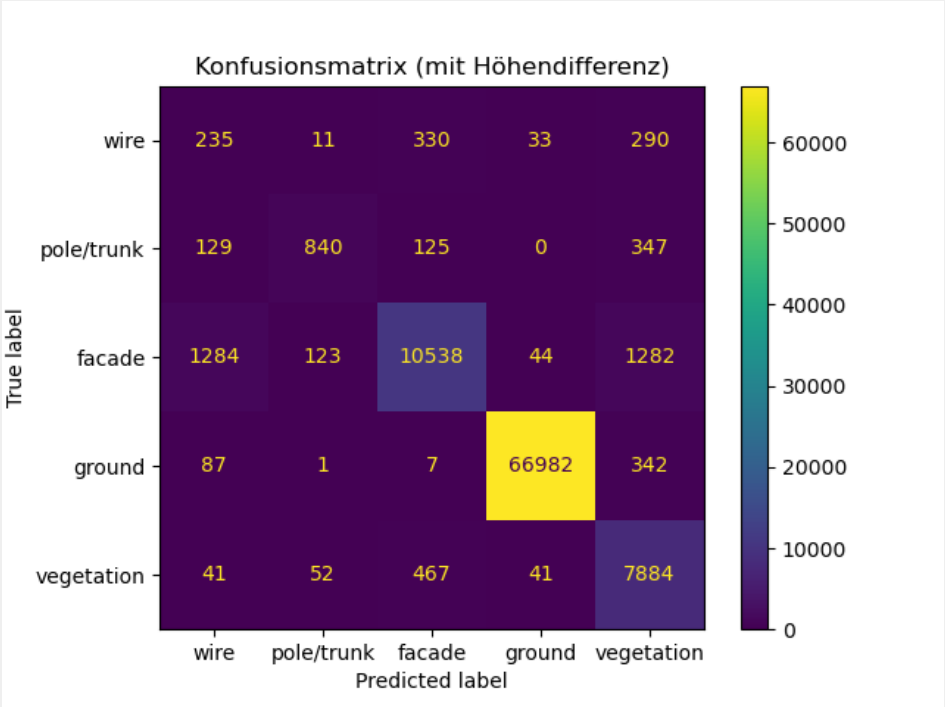
# Code: Erweiterung Feature-Vektor

Zusätzliches Feature: Höhendifferenz

```
##################################################################################################
if flag_dZ:
    '''
    Als zusätzliches Feature könnte man die Höhendifferenz der Nachbarn berechnen,
    um vor allem Fassade von Boden zu unterscheiden.
    '''
    #! Compute geometric features
    # height difference of the neighbors
    height_diff = np.max(neighbors[:,2]) - np.min(neighbors[:,2])

    # store the geometric features (at 9th column of cov_features)
    cov_features[i,8] = height_diff
##################################################################################################
```

# Ergebnisse: Qualitätsmetriken

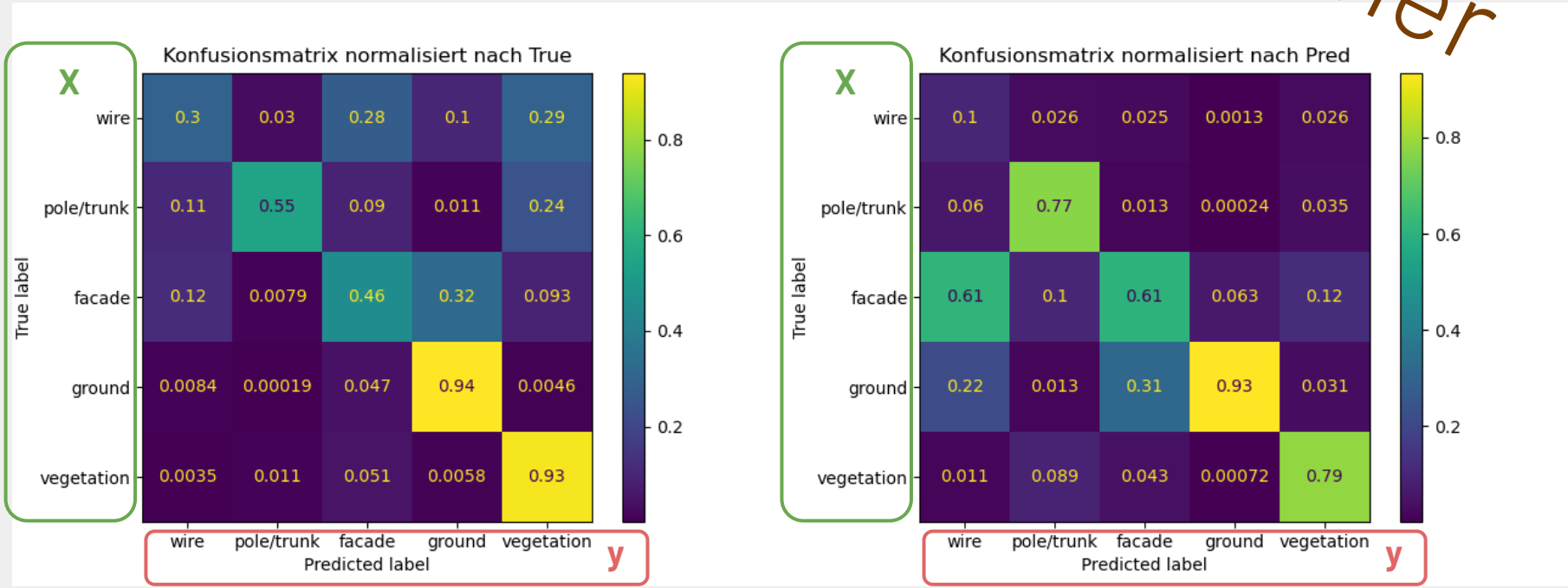



Classification report (no height difference):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| wire | 0.13 | 0.26 | 0.18 | 899 |
| pole/trunk | 0.82 | 0.58 | 0.68 | 1441 |
| facade | 0.92 | 0.79 (+0,33) | 0.85 | 13271 |
| ground | 1.00 | 0.99 | 1.00 | 67419 |
| vegetation | 0.78 | 0.93 | 0.85 | 8485 |
| accuracy | +0,09 | +0,08 | 0.94 (+0,08) | 91515 |
| macro avg | 0.73 | 0.71 | 0.71 | 91515 |
| weighted avg | 0.95 | 0.94 | 0.95 | 91515 |

# Ergebnisse: Qualitätsmetriken

Vorher

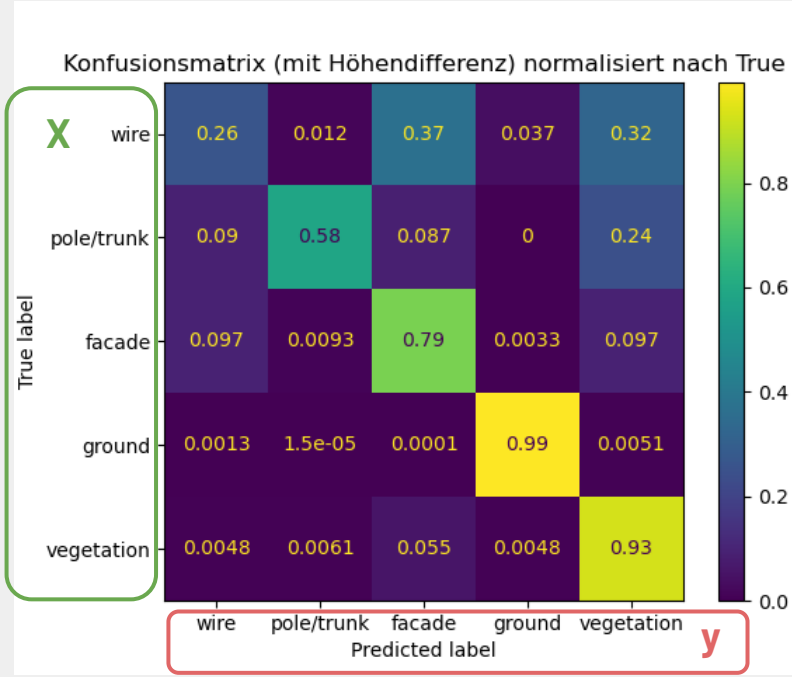


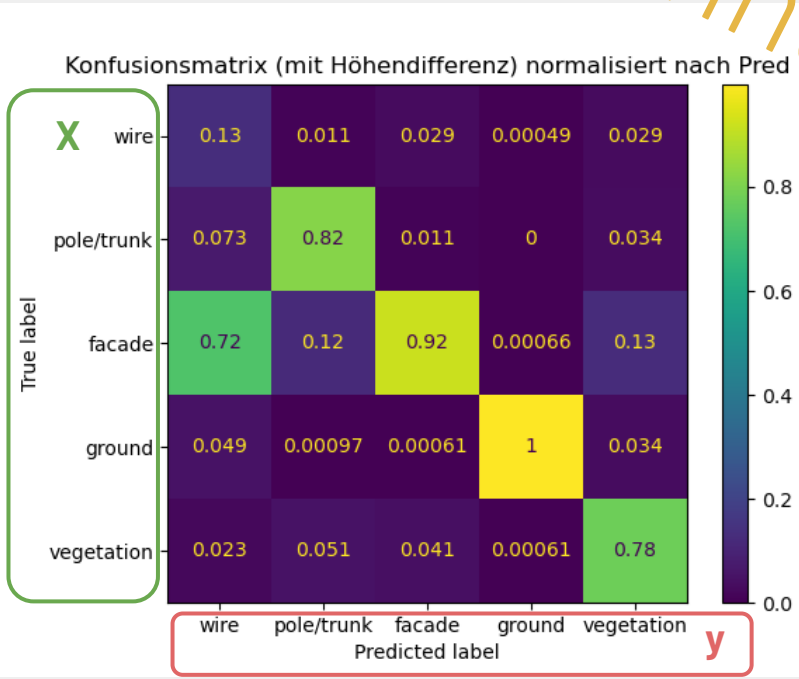

Correctness:
"Klasse **X** wird zu p% als Klasse **y** prädiziert"

Completeness:
"Klasse **y** wurde zu p% aus Punkten der Klasse **X** prädiziert"

# Ergebnisse: Qualitätsmetriken

Nachher

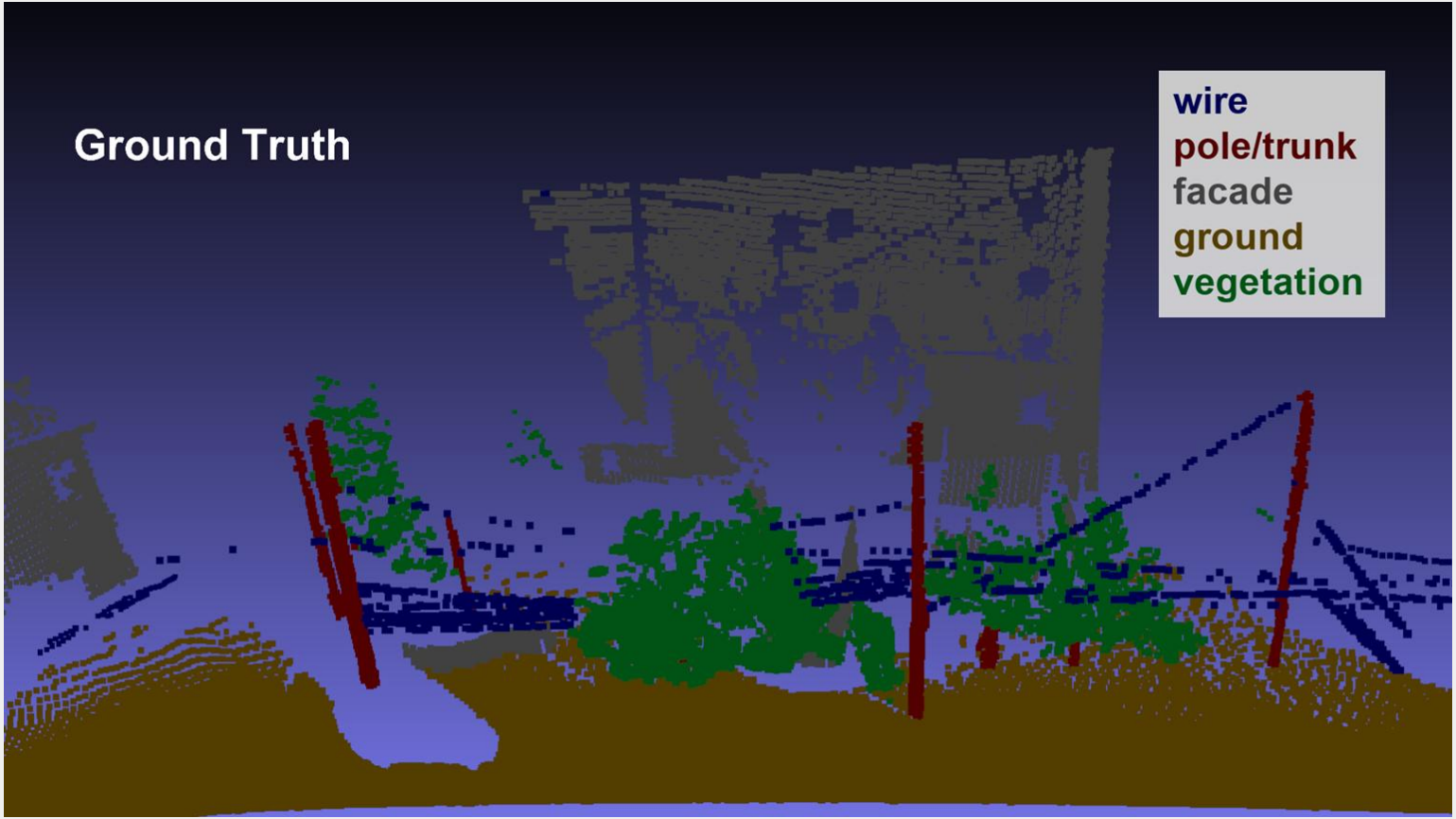


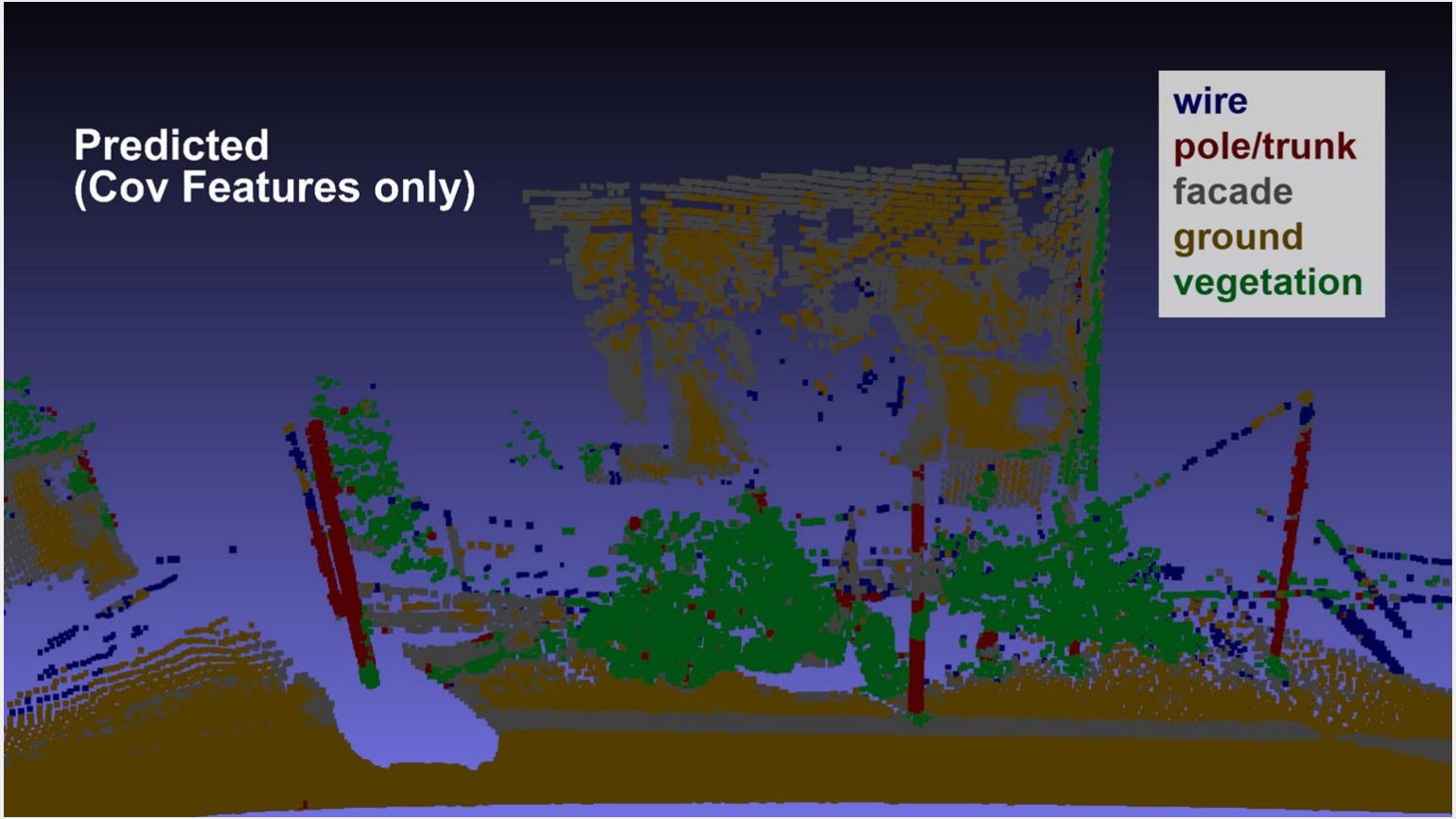

Correctness:
"Klasse **X** wird zu p% als Klasse **y** prädiziert"

Completeness:
"Klasse **y** wurde zu p% aus Punkten der Klasse **X** prädiziert"

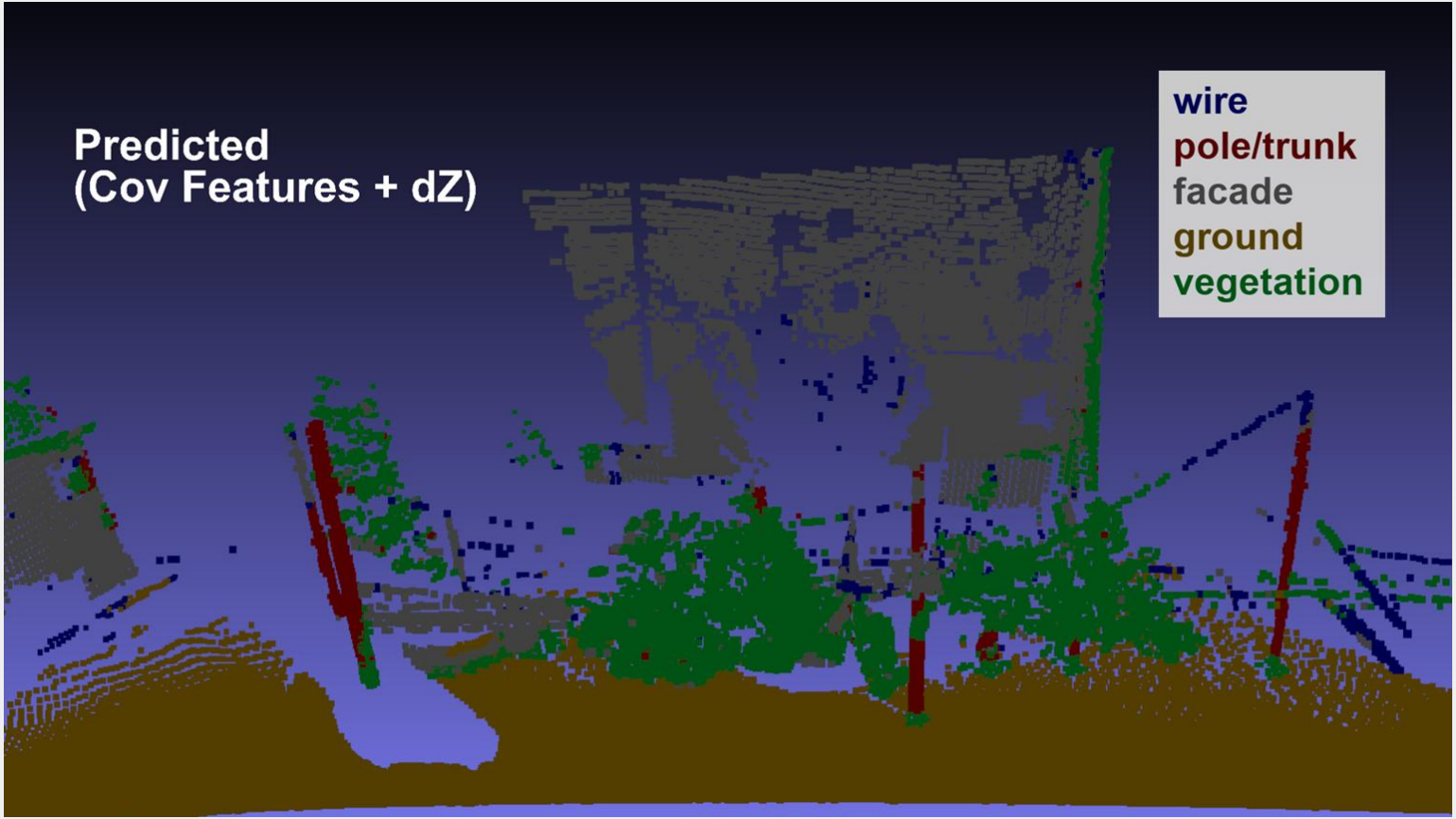
# Ergebnisse: Punktwolke



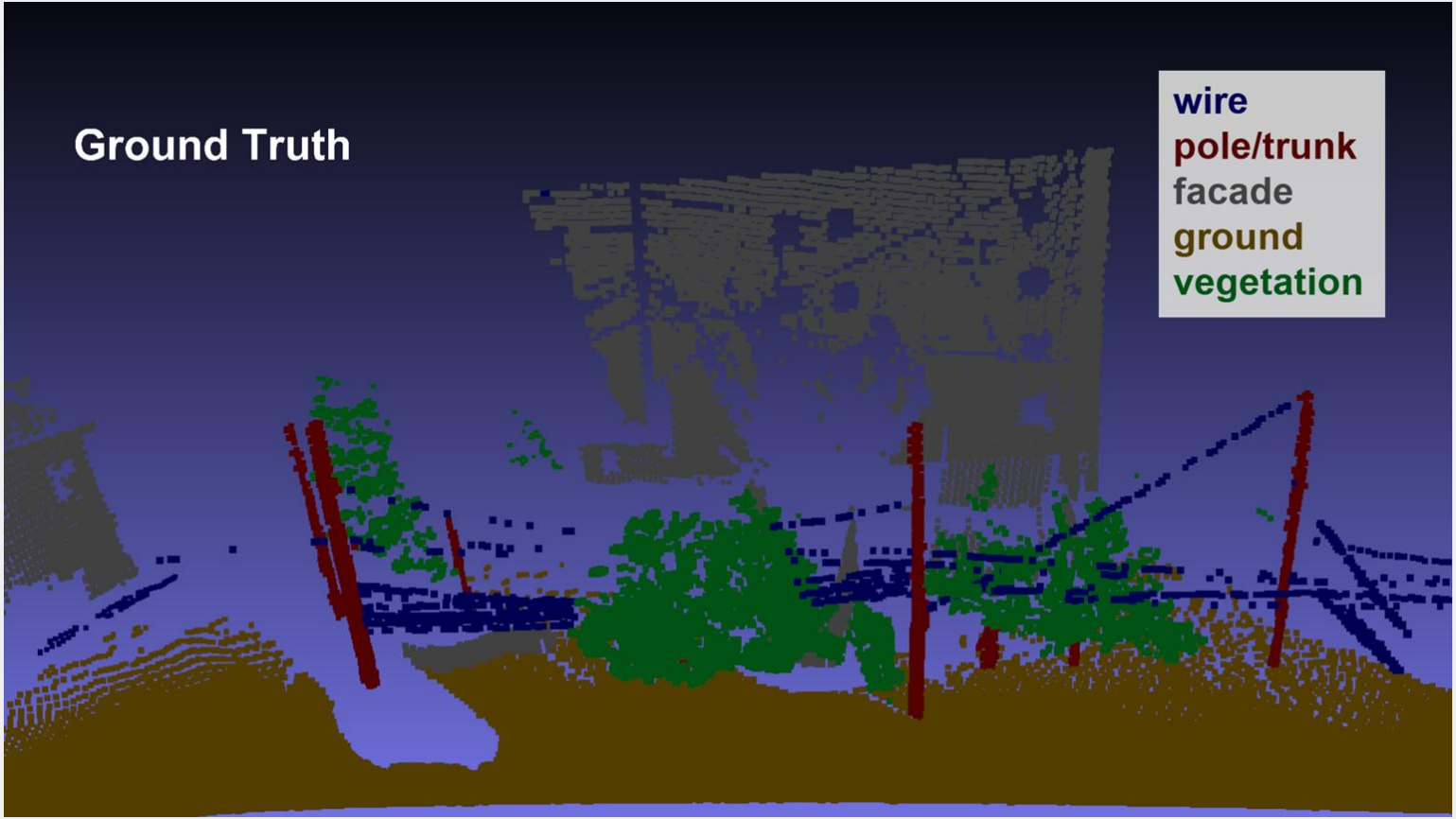
# Ergebnisse: Punktwolke



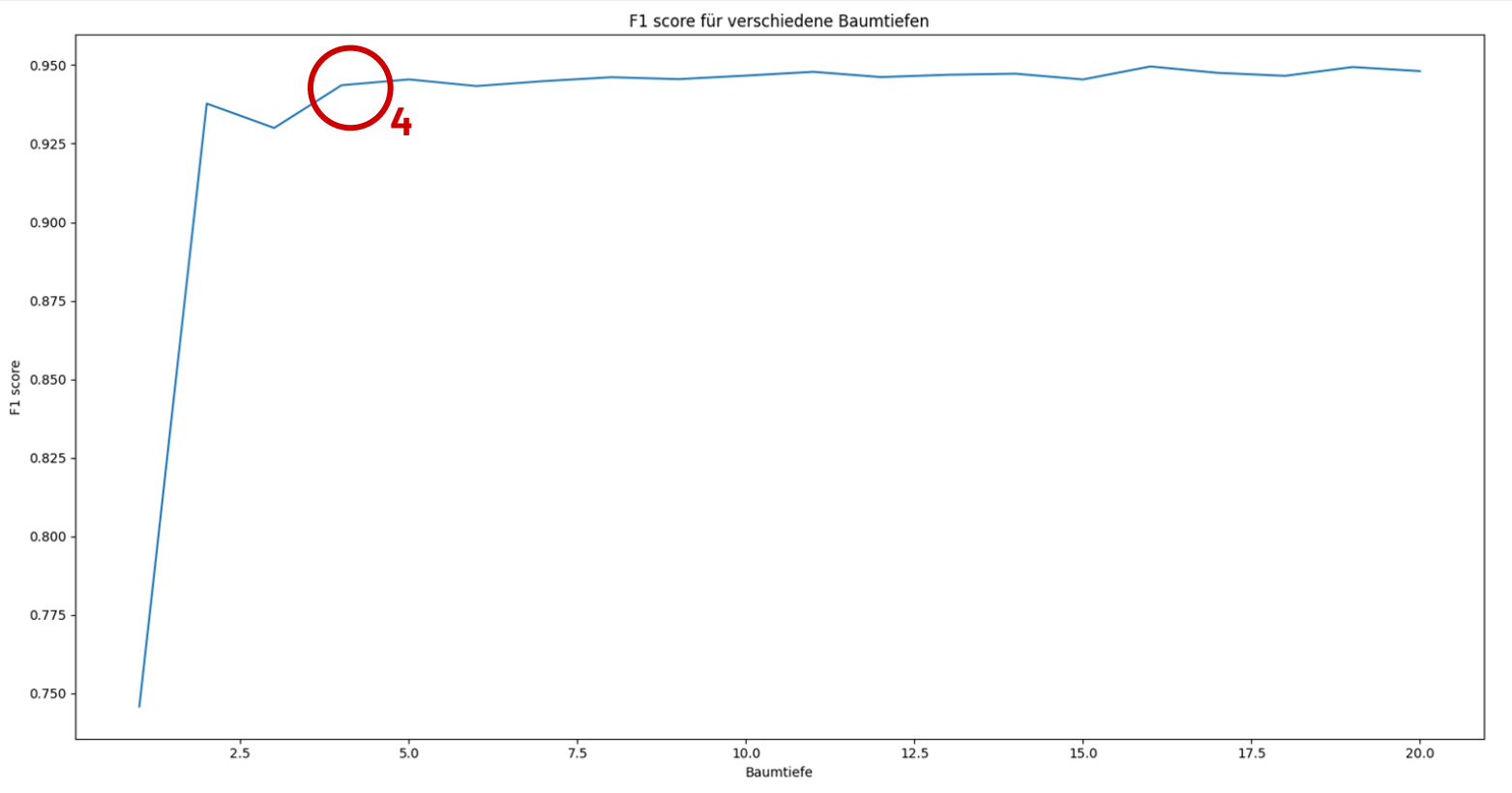
# Ergebnisse: Punktwolke



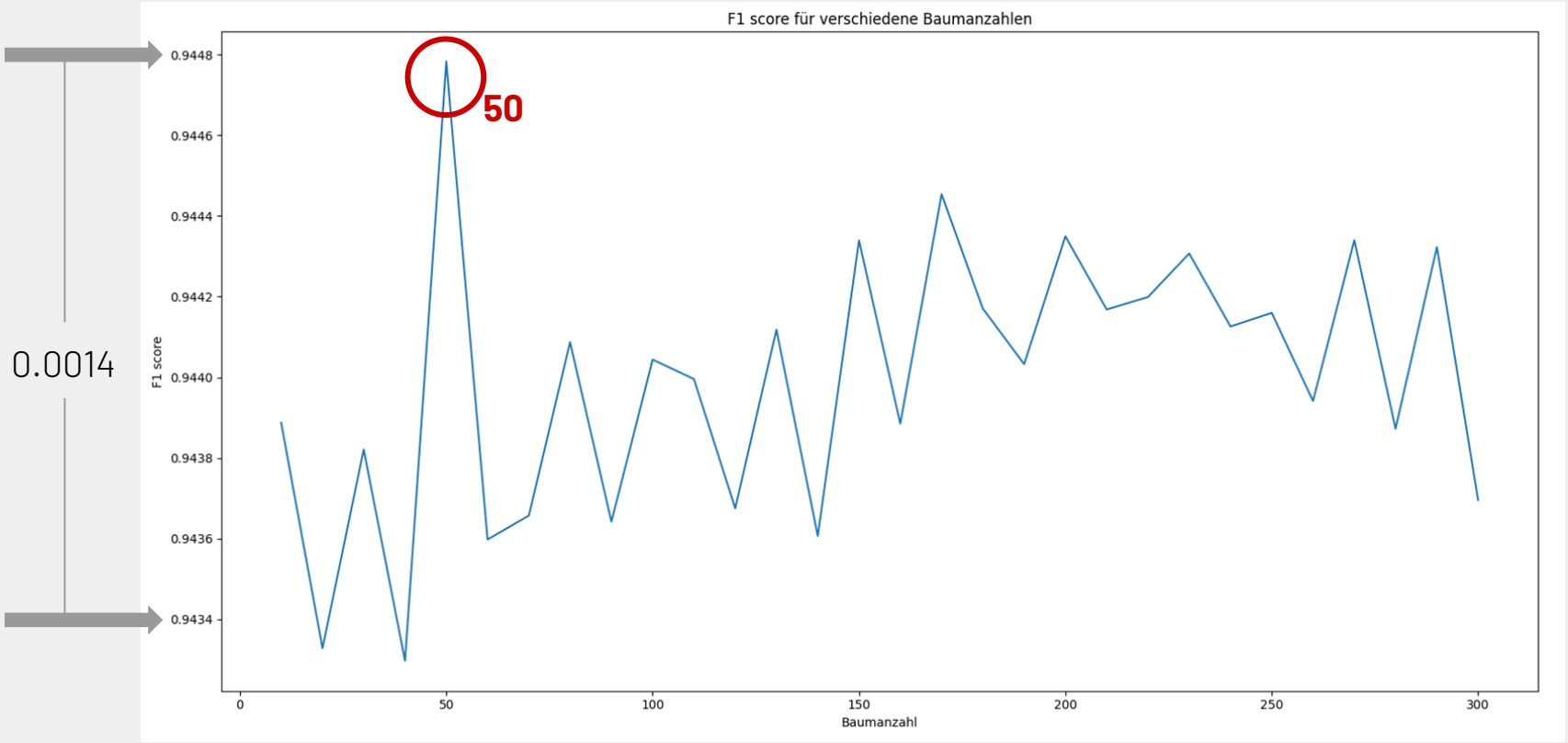
# Ergebnisse: Punktwolke



# Einfluss der Hyperparameter



# Einfluss der Hyperparameter



# Fazit

- Bei Verwendung von k = 50 gelingt die Klassifizierung recht gut (OA > 86%)
- Probleme treten insbesondere für die Klasse *wire*, *pole/trunk* und *façade* auf
  - *wire* wird oft als *facade* oder *vegetation* klassifiziert
  - *pole/trunk* wird oft als *vegetation* klassifiziert
  - *facade* wird oft als *ground* klassifiziert
- Verwendung eines geometrischen Features (z-Differenz in der Nachbarschaft) verbessert Klassifizierung (OA um 8% größer)
- Insbesondere wird *facade* wird nicht mehr als *ground* klassifiziert
  → Vorher zu 32% der Fall, nachher zu <1 % der Fall
- Hyperparameter haben mit zunehmender Magnitude geringen Einfluss

# Quellen und Links

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay

Link zum Programmcode: