3D geoinformation
Department of Urbanism
Faculty of Architecture and the Built Environment
Delft University of Technology

GEO5017
Machine Learning for the Built Environment

Lab Session
Random Forest in Scikit Learn, A2

Shenglan Du

RF in Scikit Learn

sklearn.ensemble.RandomForestClassifier¶

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

- Documentation:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- User guide:
  https://scikit-learn.org/stable/modules/ensemble.html#forest

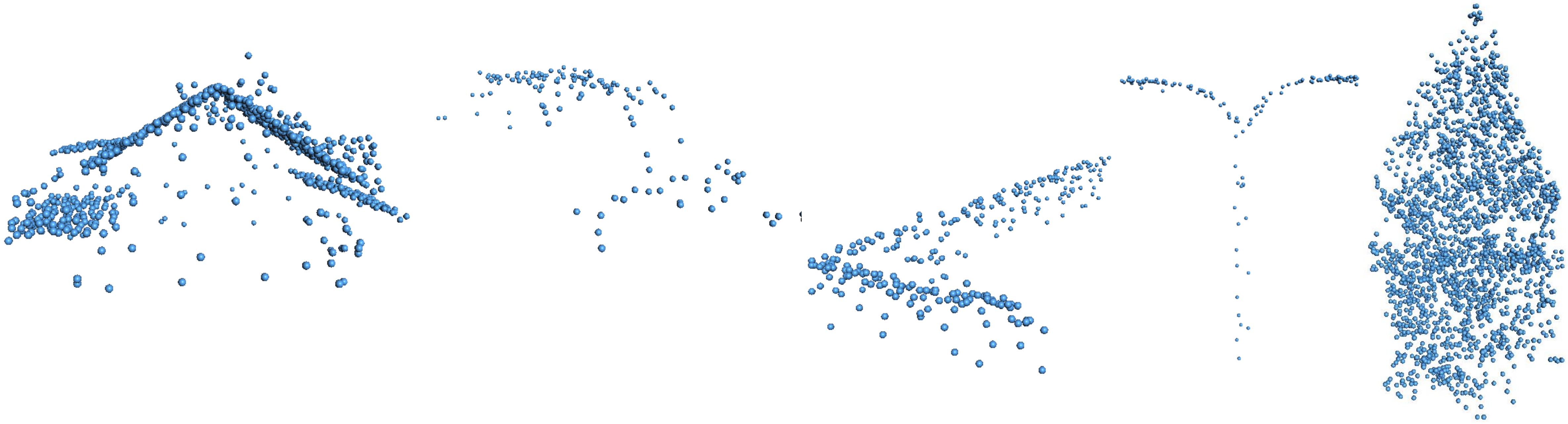
RF: Hyperparameters

- *Ensemble*: RF is a collection of individual tree classifiers
- *n_estimators*: number of trees in the forest
- *Criterion*: gini or entropy
- *max_features*: number of features to start splitting
- *Bootstrap*: whether bagging is used for building the trees
- *max_samples*: if bootstrap is true, then this is to determine how many max samples to draw from the original dataset (with replacement)

A2: Point Cloud Classification

A2: Point Cloud Classification

- 500 urban objects



A2: Point Cloud Classification

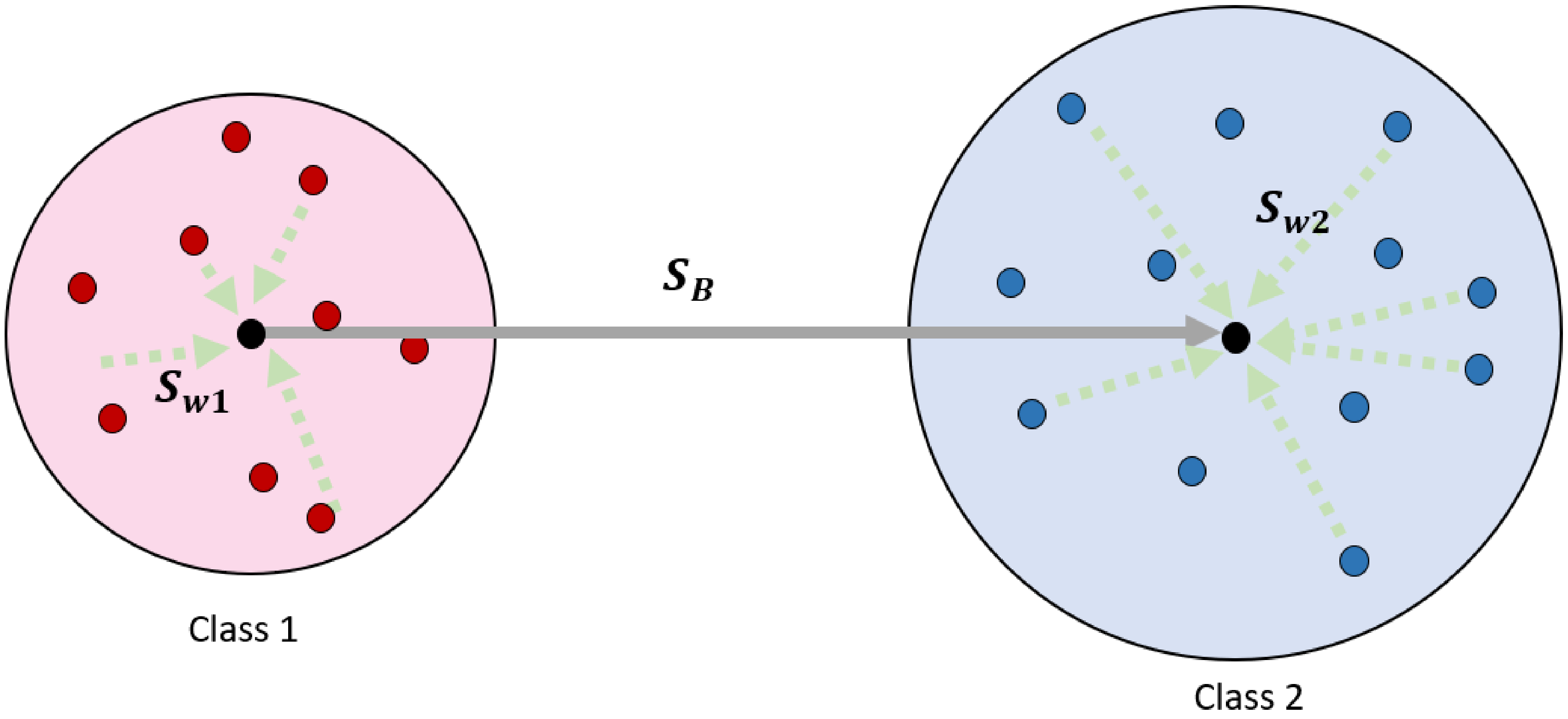
- You will use a classical ML model to perform point cloud classification (on object level)
- Focus on geometrical properties (color not available)
- Any useful property can be used, but need to make sense!
- What we evaluate: performance, analysis, visualization, reasoning……

A2: Point Cloud Classification

- Scikit learn is **Only** allowed to be used for data splitting, model training, model testing, and performance evaluation (e.g., accuracy, confusion matrix, errors)

- All other functions need to be implemented from scratch (only basic libraries are allowed such as numpy and scipy). This includes but is not restricted to:
  - Feature preprocessing
  - Hyperparameter tuning
  - Obtaining learning curves

A2: Good features

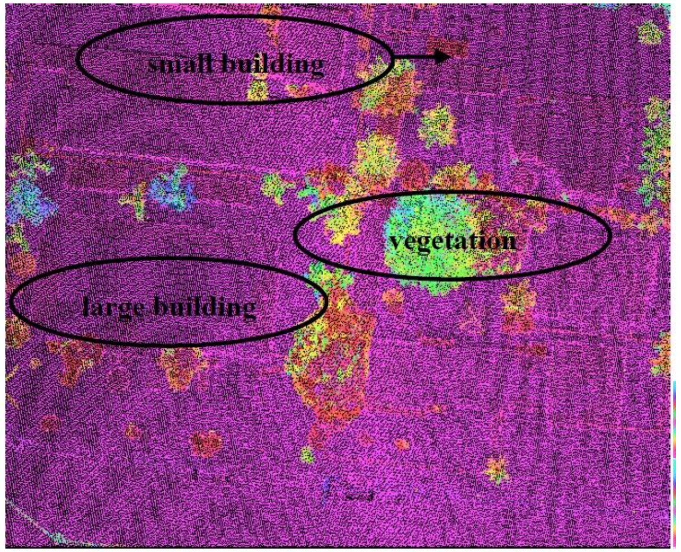


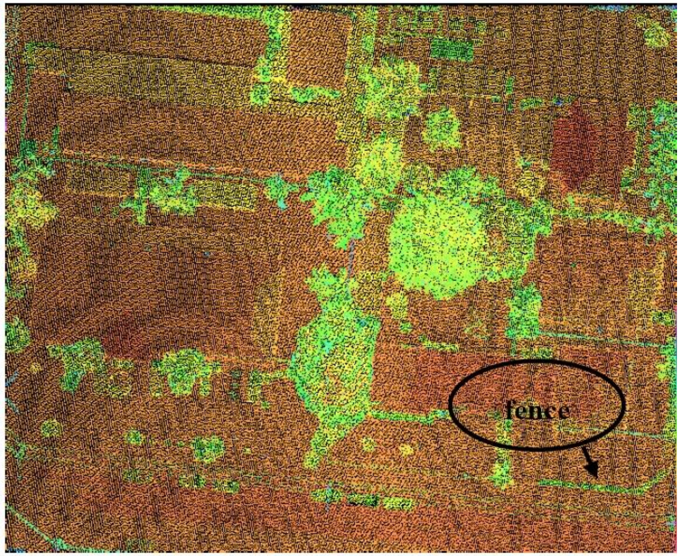
A2: Good features

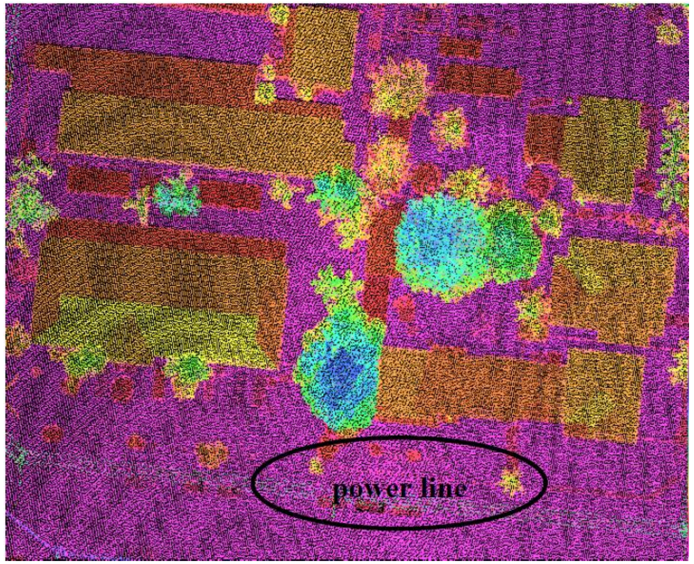
- Reasoning by other statistics
  - Histogram bins
  - Averaged feature values
  - ……

A2: Good features

- Reasoning by visualization







SVM-Based Classification of Segmented Airborne LiDAR Point Clouds in Urban Areas. Zhang et al., 2013

A2: a Demo

- Defining an urban object

```python
class urban_object:
    """
    Define an urban object
    """
    def __init__(self, filenm):
        """
        Initialize the object
        """
        # obtain the cloud name
        self.cloud_name = filenm.split('/\\')[-1][-7:-4]

        # obtain the cloud ID
        self.cloud_ID = int(self.cloud_name)

        # obtain the label
        self.label = math.floor(1.0*self.cloud_ID/100)

        # obtain the points
        self.points = read_xyz(filenm)

        # initialize the feature vector
        self.feature = []
```

```python
    def compute_features(self):
        """
        Compute the features, here we provide two example features. You're encouraged
        """
        # calculate the height
        height = np.amax(self.points[:, 2])
        self.feature.append(height)

        # get the root point and top point
        root = self.points[[np.argmin(self.points[:, 2])]]
        top = self.points[[np.argmax(self.points[:, 2])]]

        # construct the 2D and 3D kd tree
        kd_tree_2d = KDTree(self.points[:, :2], leaf_size=5)
        kd_tree_3d = KDTree(self.points, leaf_size=5)

        # compute the root point planar density
        radius_root = 0.2
        count = kd_tree_2d.query_radius(root[:, :2], r=radius_root, count_only=True)
        root_density = 1.0*count[0] / len(self.points)
        self.feature.append(root_density)

        # compute the 2D footprint and calculate its area
```

A2: a Demo

- Overall steps
  - Prepare features for each urban object, write each object ID with its features to a .txt
  - Load features from .txt
  - Visualize features
  - Classification

```
if __name__=='__main__':
    # specify the data folder
    """Here you need to specify your own path"""
    path = '../Data/pointclouds-500'

    # conduct feature preparation
    print('Start preparing features')
    feature_preparation(data_path=path)

    # load the data
    print('Start loading data from the local file')
    ID, X, y = data_loading()

    # visualize features
    print('Visualize the features')
    feature_visualization(X=X)

    # SVM classification
    print('Start SVM classification')
    SVM_classification(X, y)

    # RF classification
    print('Start RF classification')
    RF_classification(X, y)
```

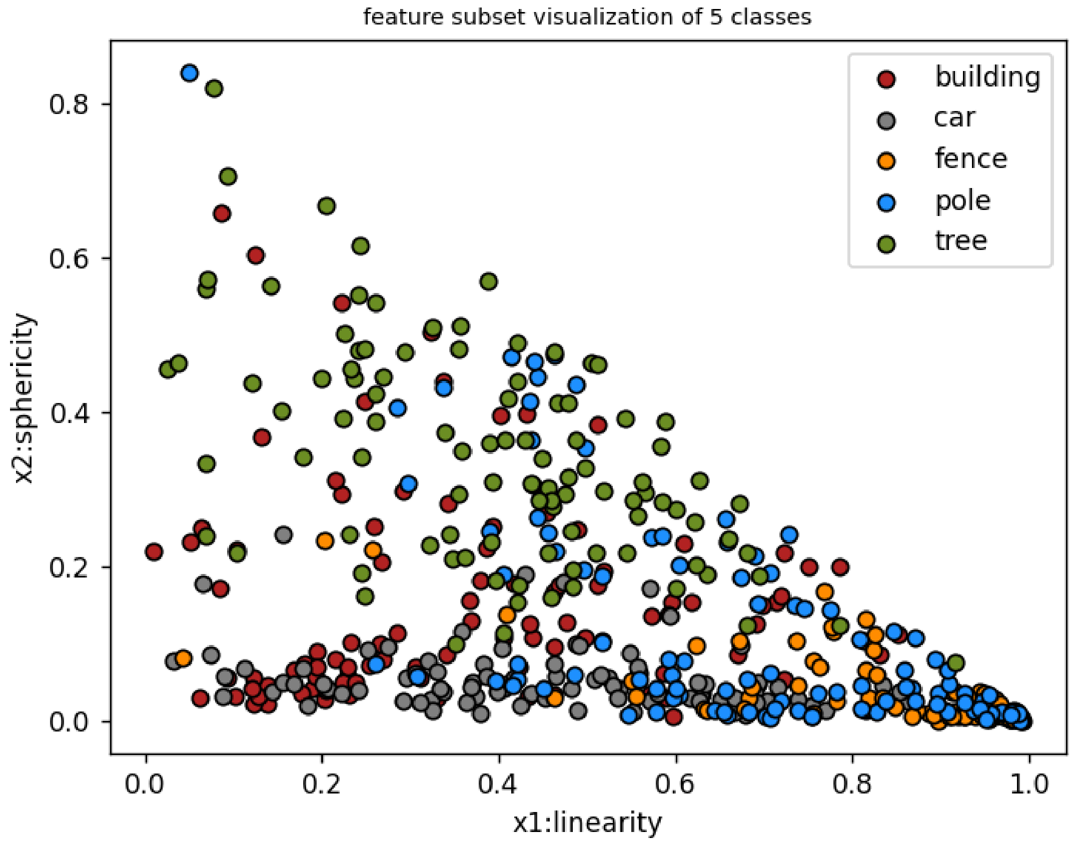
A2: a Demo

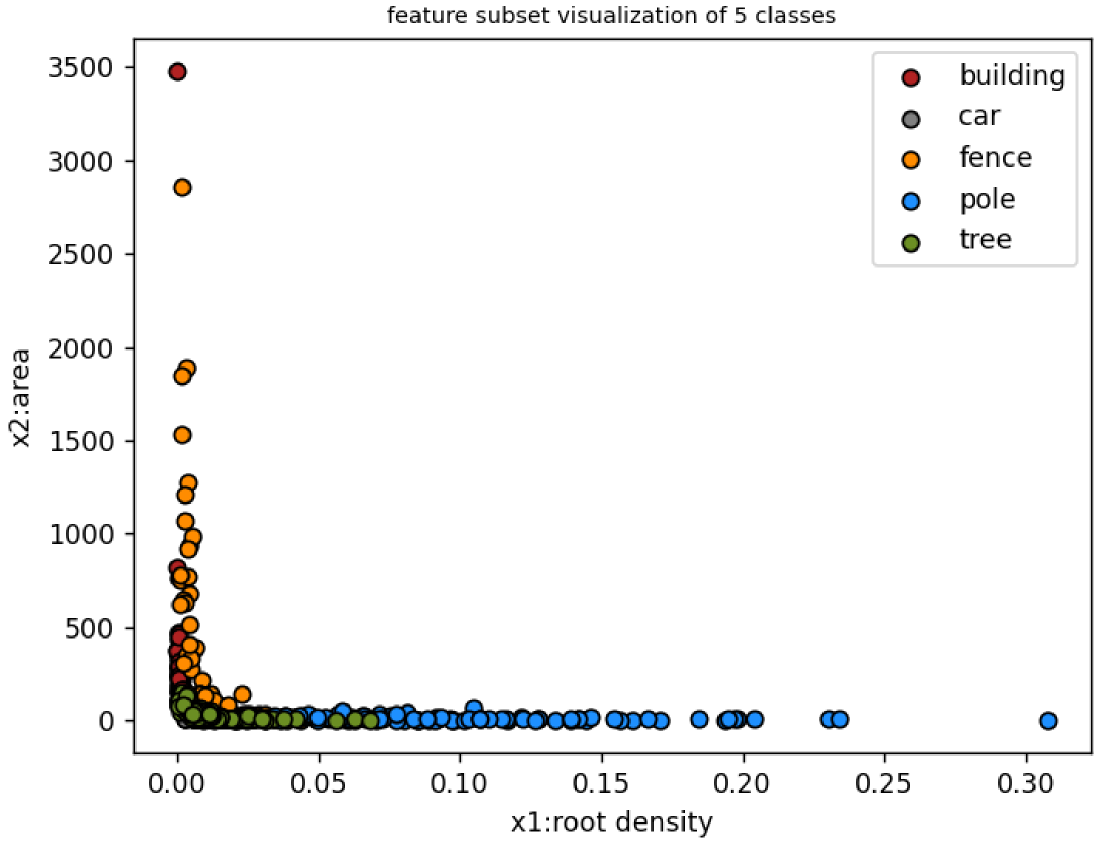
- Visualize 2 features to check if they are good

```python
def feature_visualization(X):
    """
    Visualize the features
        X: input features. This assumes classes are stored in a sequential manner
    """
    # initialize a plot
    fig = plt.figure()
    ax = fig.add_subplot()
    plt.title("feature subset visualization of 5 classes", fontsize="small")

    # define the labels and corresponding colors
    colors = ['firebrick', 'grey', 'darkorange', 'dodgerblue', 'olivedrab']
    labels = ['building', 'car', 'fence', 'pole', 'tree']

    # plot the data with first two features
    for i in range(5):
        ax.scatter(X[100*i:100*(i+1), 4], X[100*i:100*(i+1), 5], marker="o", c=colors[i], edgecolor="k", label=labels[i])

    # show the figure with labels
    """
    Replace the axis labels with your own feature names
    """
    ax.set_xlabel('x1:linearity')
    ax.set_ylabel('x2:sphericity')
    # ax.set_zlabel('x3:top sphericity')
    ax.legend()
    plt.show()
```

A2: a Demo

- Visualize 2 features to check if they are good





A2: a Demo

- SVM Classification

```python
def SVM_classification(X, y):
    """
    Conduct SVM classification
        X: features
        y: labels
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
    clf = svm.SVC()
    clf.fit(X_train, y_train)
    y_preds = clf.predict(X_test)
    acc = accuracy_score(y_test, y_preds)
    print("SVM accuracy: %5.2f" % acc)
    print("confusion matrix")
    conf = confusion_matrix(y_test, y_preds)
    print(conf)
```

```
Start SVM classification
SVM accuracy:  0.49
confusion matrix
[[29  0  2  0  8]
 [ 0 39  0  0  0]
 [ 4 19 12  0  8]
 [ 0 42  0  0  1]
 [ 1 16  0  0 19]]
```

A2: Hyperparameter Tuning

- Pseudo code of grid searching:

```
a = [a1, a2, a3, ……]
b = [b1, b2, b3, ……]
for ai in a:
    for bj in b:
            construct the model M(a, b)
            obtain and record M's performance
Return the best ai and bj
```

A2: Learning Curve Plotting

- Pseudo code:

check_interval = 0.1 (can also be smaller or larger)

for i in range(1/ check_interval -1):

    train test split ratio = (i+1)* check_interval

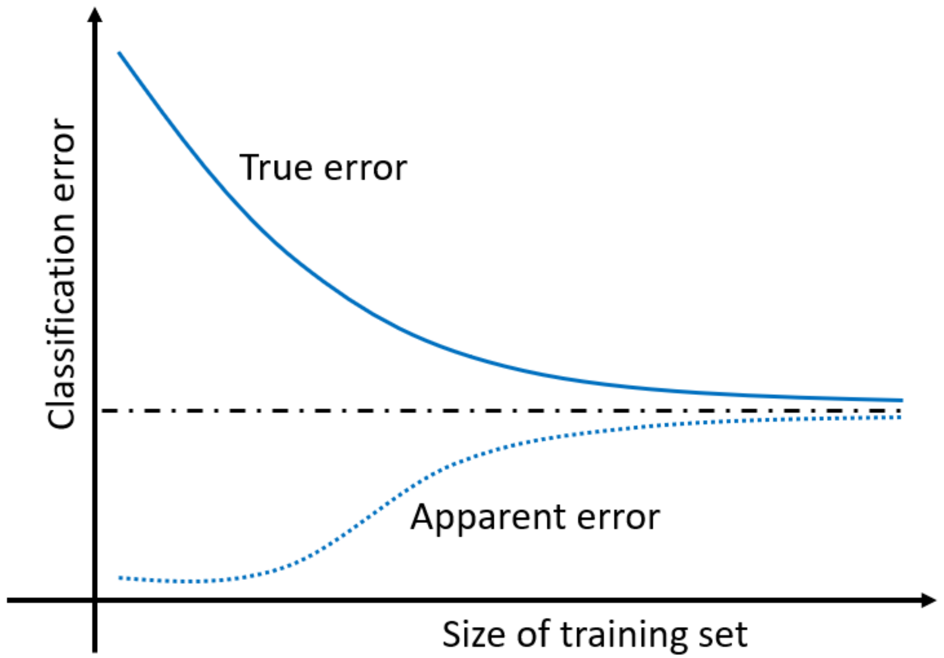
    split the data accordingly

    train and test model on the corresponding sets (multiple times) and record the (averaged) error rates

Plot the performances as curves

A2: Learning Curve Plotting

- Requirements:
  - X axis: training set size (0-500)
  - Y axis: classification error
  - Two curves need to be present:
    - Apparent error rate (on training set)
    - True error rate (approximated on testing set)
    - For each experiment, run multiple times so that the output curves are smooth



Learning Curve in Scikit-Learn

sklearn.model_selection.learning_curve¶

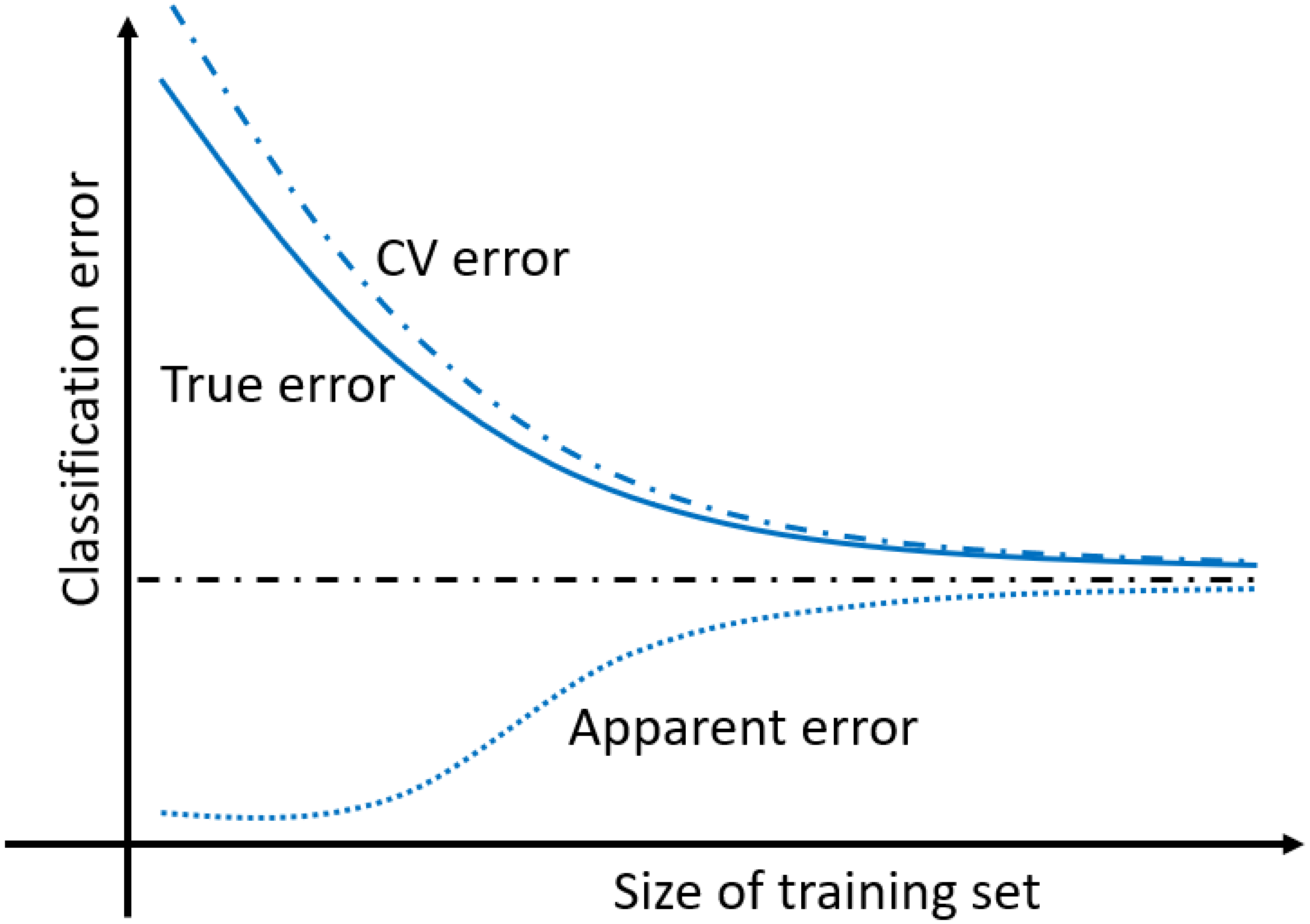
`sklearn.model_selection.learning_curve`(*estimator, X, y, *, groups=None, train_sizes=array([0.1, 0.33, 0.55, 0.78, 1.]), cv=None, scoring=None, exploit_incremental_learning=False, n_jobs=None, pre_dispatch='all', verbose=0, shuffle=False, random_state=None, error_score=nan, return_times=False, fit_params=None*) [source]

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html

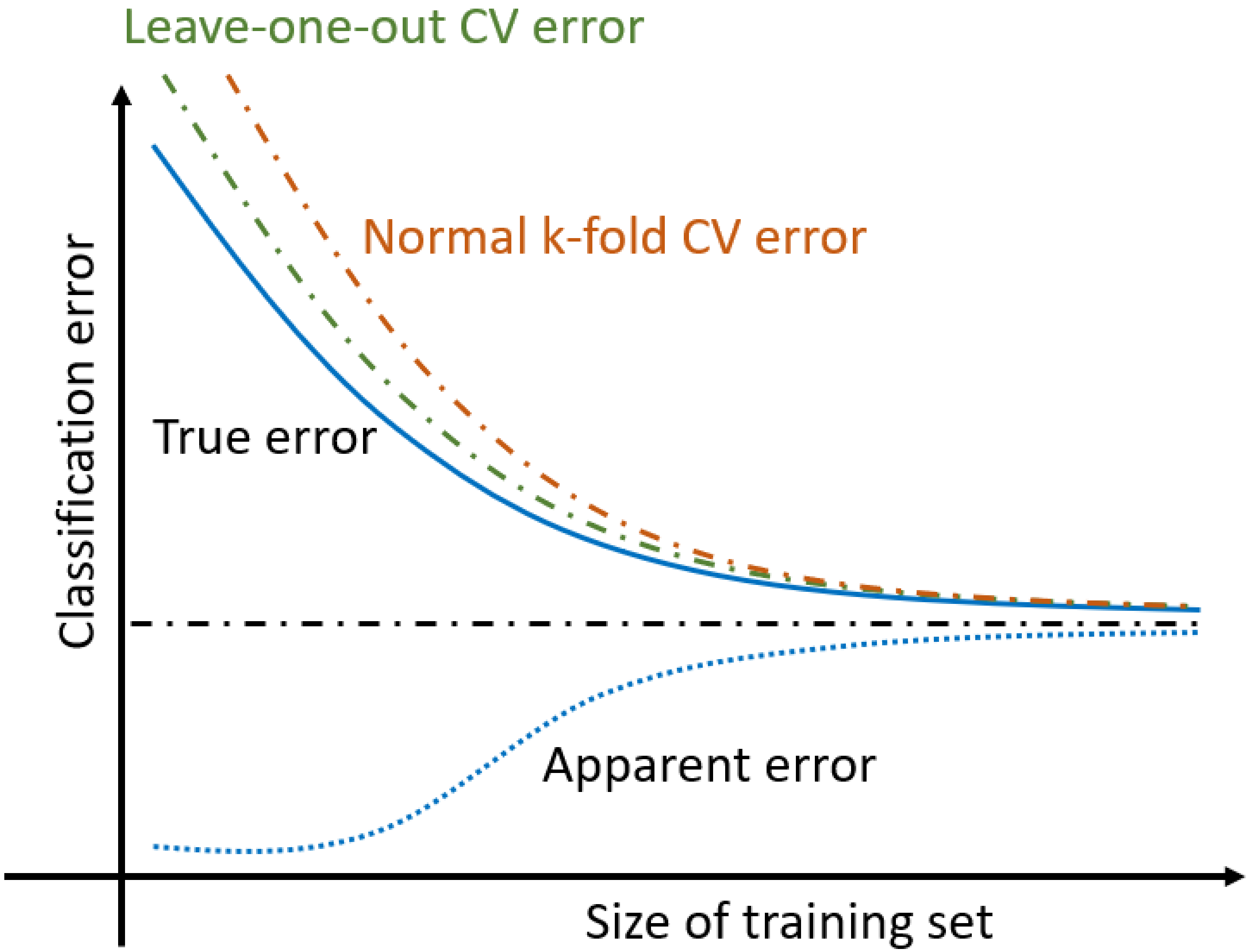
Learning Curve in Scikit-Learn

```
>>> from sklearn.datasets import make_classification
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.model_selection import learning_curve
>>> X, y = make_classification(n_samples=100, n_features=10, random_state=42)
>>> tree = DecisionTreeClassifier(max_depth=4, random_state=42)
>>> train_size_abs, train_scores, test_scores = learning_curve(
...     tree, X, y, train_sizes=[0.3, 0.6, 0.9]
... )
>>> for train_size, cv_train_scores, cv_test_scores in zip(
...     train_size_abs, train_scores, test_scores
... ):
...     print(f"{train_size} samples were used to train the model")
...     print(f"The average train accuracy is {cv_train_scores.mean():.2f}")
...     print(f"The average test accuracy is {cv_test_scores.mean():.2f}")
24 samples were used to train the model
The average train accuracy is 1.00
The average test accuracy is 0.85
48 samples were used to train the model
The average train accuracy is 1.00
The average test accuracy is 0.90
72 samples were used to train the model
The average train accuracy is 1.00
The average test accuracy is 0.93
```

Learning Curve in Scikit-Learn



Learning Curve in Scikit-Learn



A2 Overview

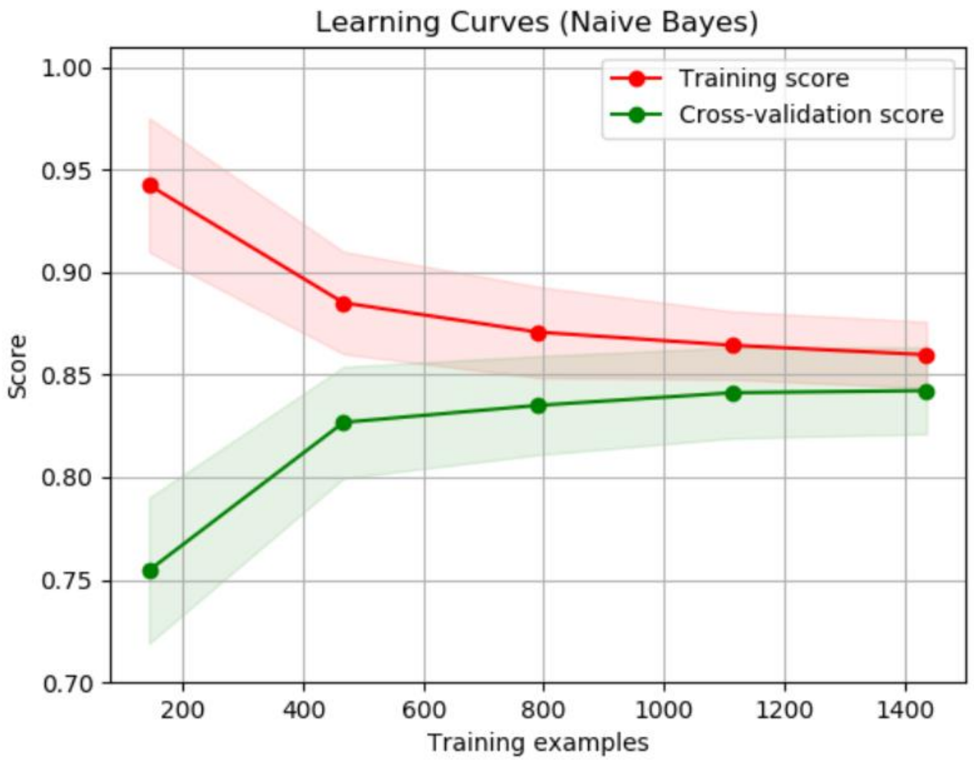
- You must implement your own functions for grid search and learning curve plotting.

- Scikit learn is **Not** allowed for hyperparameter tuning, and learning curve plotting.

- Visualization of learning curves can be done in Matplotlib or other plotting libraries.

A2 Visualization of Results

- Using any Google images for your submission is not allowed



https://en.wikipedia.org/wiki/Learning_curve_(machine_learning)

Questions?