Practical 4

```
import numpy as np
import cvxopt
from cvxopt import matrix as cvxopt_matrix
from matplotlib import pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import load_wine
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
```

Task 0: The Data

We will work with the data from practical 3. Load the data and split it into a training and test set. You can re-use the data splitting function from Practical 2.

```
# def split_data(X, y, frac=0.3, seed=None):

#     len_X = len(X)
#     indices = np.random.permutation(len_X)
#     indices_test, indices_train = indices[:int(len(indices)*frac)], indices[int(len(indice
s)*frac):]
#     y_train, y_test = y[indices_train], y[indices_test]
#     X_train, X_test = X[indices_train], X[indices_test]

#     return y_train, y_test, X_train, X_test

# from Practical 2
def split_data(X, y, frac=0.3, max_samples=None, seed=4711):

    if seed is not None:
        np.random.seed(seed)

    indices = np.random.permutation(len(X))
    indices = indices[:max_samples]
    indices_test, indices_train = indices[:int(len(indices)*frac)], indices[int(len(indices)*
frac):]

    assert (len(set(indices))) == len(indices)
    assert len(set(indices_test).intersection(indices_train)) == 0

    X_train, X_test = X[indices_train], X[indices_test]
    y_train, y_test = y[indices_train], y[indices_test]

    return X_train, X_test, y_train, y_test
# X_train, X_test, y_train, y_test = split_data(X_2d, t_2d)
```

```
# load data
X_2d, t_2d = np.load('data/nonlin_2d_data.npy')[:,:2], np.load('data/nonlin_2d_data.npy')[:,
2]
```

```
# split data
X_train, X_test, t_train, t_test = split_data(X_2d, t_2d, seed=1)
```

In SVMs, each data sample x_n has a corresponding lagrange multiplier α_n which indicates if x_n is a support vector. In the latter case $\alpha_n > 0$ holds. The goal of learning the SVM is to figure out which samples are support vectors by learning α. The dual SVM optimizes the following quadratic program.

$$\min \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) - \sum_{n=1}^{N} \alpha_n$$

subject to

$$0 \leq \alpha_n \leq C$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

The quadratic program solver expects the following form:

$$\min \frac{1}{2} \alpha^T P \alpha + \mathbf{q}^T \alpha$$

subject to

$$A\alpha = b$$

$$G\alpha \leq h$$

Here, A and G are matrices with one row per individual constraint. Similarly, b and h are vectors with one element per individual constraint.

Having trained the SVM, a prediction for an input $\mathbf{x}$ is made by:

$$y = sign([\sum_{n}^{N} \alpha_n t_n k(\mathbf{x}, \mathbf{x}_n)] + b)$$

Task 1.1

Use the code provided below as a basis to express the constrained optimization problem in terms of P, q, A, b, G and h and implement a function `fit_svm` which passes these variables to the provided QP solver. Fit a SVM on the training data and extract its parameters.

Hints: - The box constraint $0 \leq \alpha_n \leq C$ defines two constraints of the form $G\alpha_n \leq h$ for each α_n. - The inequality $x \geq 0$ is equivalent to $-x \leq 0$.

```
def rbf_kernel(x_n, x_m, gamma=1/5):
    tmp = np.square(x_n - x_m)
    tmp = -(1/gamma) * np.sum(tmp)
    return np.exp(tmp)
```

```
def accuracy(y_true, y_pred):
  n_c = np.sum((y_true == y_pred)) #number of correctly classified
  acc = n_c/len(y_true)
  return acc
```

```
def fit_svm(X, t, kernel, C=1.0):
    '''Fit SVM using data (X,t), specified kernel and parameter C.'''

    t = np.array([-1 if l == 0 else 1 for l in t])

    tmat = np.diag(t)
    Ptmp = np.array([kernel(xn,xm) for xn in X for xm in X]).reshape((len(X), len(X)))
    P = np.matmul(np.matmul(tmat, Ptmp), tmat)

    q = -1 * np.ones(len(X))
    A = np.array([float(k) for k in t]).reshape((1,len(X)))
    b = np.array([0.]).reshape((1,1))

    onev = np.array([-1, 1])
    G = np.kron(np.identity(len(X)), onev.reshape((2,1)))
    tmp = np.array([0,C])
    h = np.tile(tmp, len(X))

    assert P.shape == (len(X), len(X))
    assert len(q) == len(X)
    assert A.shape == (1, len(X)) and A.dtype == 'float'
    assert b.shape == (1, 1)
    assert len(G) == 2 * len(X)
    assert len(h) == 2 * len(X)

    return solve_quadratic_program(P, q, A, b, G, h)

def solve_quadratic_program(P, q, A, b, G, h):
    '''Uses cvxopt to solve the quadratic program.'''
    P, q, A, b, G, h = [cvxopt.matrix(var) for var in [P, q, A, b, G, h]]
    minimization = cvxopt.solvers.qp(P, q, G, h, A, b)
    lagr_mult = np.ravel(minimization['x'])
    return lagr_mult


def extract_parameters(X, t, kernel, lagr_mult, C, threshold=1e-5):
    '''Computes the intercept from the support vector constraints.

    Inputs
        X:         predictors
        t:         targets
        kernel:    a kernel to be used
        lagr_mult: the Lagrange multipliers obtained by solving the dual QP
        threshold: threshold for choosing support vectors

    Returns
        lagr_mult: lagrange multipliers for the support vectors
        svs:       set of support vectors
        sv_labels: targets t_n for the support vectors
        intercept: computed intercept
    '''
    t = np.array([-1 if l == 0 else 1 for l in t])

    # ---------------- INSERT CODE ----------------------

    sv_inds = (lagr_mult > threshold) & (lagr_mult < (C-threshold))
    svs, sv_labels, lagr_mult = X[sv_inds], t[sv_inds], lagr_mult[sv_inds]
```

```
    mlen = len(svs)

    intercept = 0
    for i in range(mlen):
      tmp = np.array([lagr_mult[m] * sv_labels[m] * kernel(svs[i], svs[m]) for m in range(mle
n)]).sum()
      intercept += (1/mlen) * (sv_labels[i] - tmp)

    # ---------------- END CODE ------------------------

    return lagr_mult, svs, sv_labels, intercept
```

```
# Fit SVM on training data
sols = fit_svm(X_train, t_train, rbf_kernel)

# Extract parameters
lagr, svs, sv_labels, intercept = extract_parameters(X_train, t_train, rbf_kernel, sols, 1.0)
```

```
     pcost       dcost       gap    pres   dres
 0: -1.2225e+01 -2.4493e+02  8e+02  1e+00  4e-16
 1: -9.2861e+00 -1.1119e+02  1e+02  1e-15  4e-16
 2: -1.4797e+01 -2.4685e+01  1e+01  3e-15  4e-16
 3: -1.5743e+01 -1.7893e+01  2e+00  1e-15  2e-16
 4: -1.5986e+01 -1.6566e+01  6e-01  1e-15  1e-16
 5: -1.6091e+01 -1.6235e+01  1e-01  2e-15  2e-16
 6: -1.6117e+01 -1.6144e+01  3e-02  2e-15  2e-16
 7: -1.6123e+01 -1.6125e+01  3e-03  3e-15  2e-16
 8: -1.6123e+01 -1.6123e+01  7e-05  1e-15  2e-16
 9: -1.6123e+01 -1.6123e+01  2e-06  4e-16  1e-16
Optimal solution found.
```

Task 1.2

Having learnt an SVM, we can use the calculated parameters to make predictions on novel samples. - Implement a function `svm_predict(X, kernel, lagr_mult, svs, sv_labels, intercept)` . - Use this function with your Gaussian RBF kernel (Practical 3) and compute the test accuracy on the 2d dataset.

```
def svm_predict(X, kernel, lagr_mult, svs, sv_labels, intercept):

    # ---------------- INSERT CODE ----------------------
    mlen = len(lagr_mult)

    prediction = []
    for x in X:
      tmp = np.array([lagr_mult[m] * sv_labels[m] * kernel(svs[m], x) for m in range(mlen)]).
sum()
      pred = int((intercept + tmp)>0)
      prediction.append(pred)


    # ---------------- END CODE ------------------------

    return np.array(prediction)
```

```
# Calculate test accuracy
y_train = svm_predict(X_train, rbf_kernel, lagr, svs, sv_labels, intercept)
y_test = svm_predict(X_test, rbf_kernel, lagr, svs, sv_labels, intercept)
print("The accuracy on the training set is", accuracy(t_train, y_train), "and on the test se
t", accuracy(t_test, y_test))
```

```
The accuracy on the training set is 0.9828571428571429 and on the test set 1.0
```

Task 1.3

- Instead of using the Gaussian RBF kernel, use the linear kernel (dot product) defined in Practical 3.
- Compare results on with both kernels with sklearn implementation (SVC)
- Visualize the predictions on the test set, the learned support vectors and the decision boundary for both kernels.

```
def linear_kernel(x_n, x_m):
    return np.matmul(x_n.T, x_m)
```

```
# Fit SVM with linear kernel and calculate the test accuracy
# Fit SVM on training data
sols_lk = fit_svm(X_train, t_train, linear_kernel)

# Extract parameters
lagr_lk, svs_lk, sv_labels_lk, intercept_lk = extract_parameters(X_train, t_train, linear_ker
nel, sols_lk, 1.0)
```

```
     pcost       dcost       gap    pres   dres
 0: -1.6799e+02 -3.7889e+02  9e+02  2e+00  2e-14
 1: -1.2879e+02 -2.5279e+02  1e+02  5e-15  1e-14
 2: -1.3986e+02 -1.4886e+02  9e+00  3e-15  9e-15
 3: -1.4000e+02 -1.4009e+02  9e-02  8e-16  8e-15
 4: -1.4000e+02 -1.4000e+02  9e-04  4e-15  9e-15
 5: -1.4000e+02 -1.4000e+02  9e-06  2e-15  1e-14
Optimal solution found.
```

```
y_train_lk = svm_predict(X_train, linear_kernel, lagr_lk, svs_lk, sv_labels_lk, intercept_lk)
y_test_lk = svm_predict(X_test, linear_kernel, lagr_lk, svs_lk, sv_labels_lk, intercept_lk)
print("The accuracy on the training set is", accuracy(t_train, y_train_lk), "and on the test
 set", accuracy(t_test, y_test_lk))
```

```
The accuracy on the training set is 0.52 and on the test set 0.41333333333333333
```

```
# Fit SVM using sklearn and calculate the test accuracy
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf.fit(X_train, t_train)
```

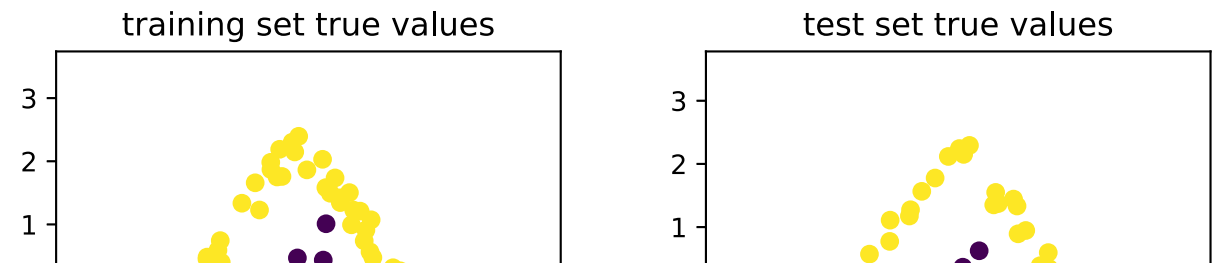
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

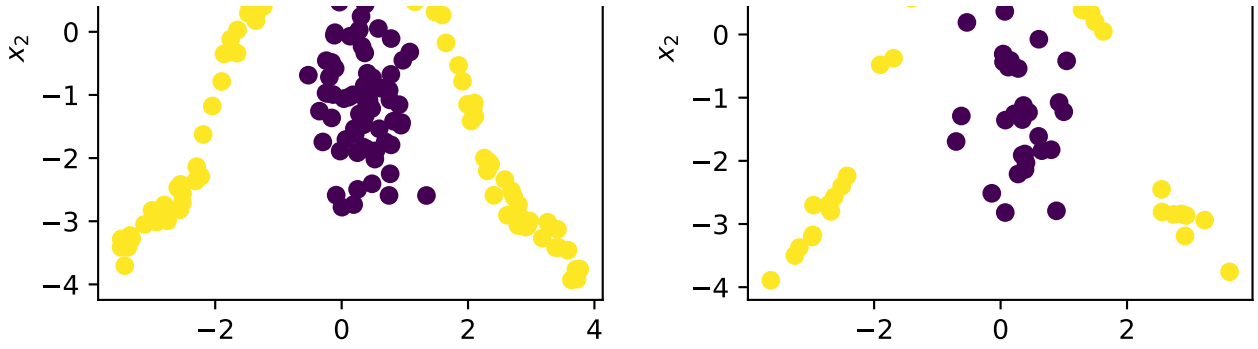
```
y_train_sk = clf.predict(X_train)
y_test_sk = clf.predict(X_test)
print("The accuracy on the training set is", accuracy(t_train, y_train_sk), "and on the test
 set", accuracy(t_test, y_test_sk))
```

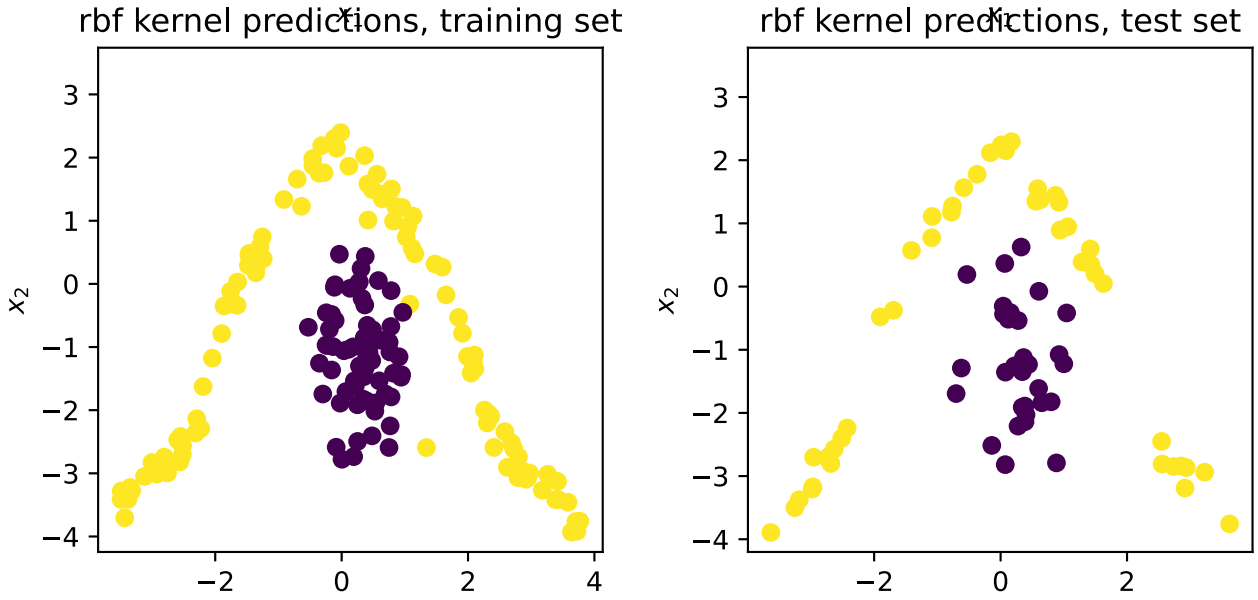
```
The accuracy on the training set is 0.9942857142857143 and on the test set 1.0
```

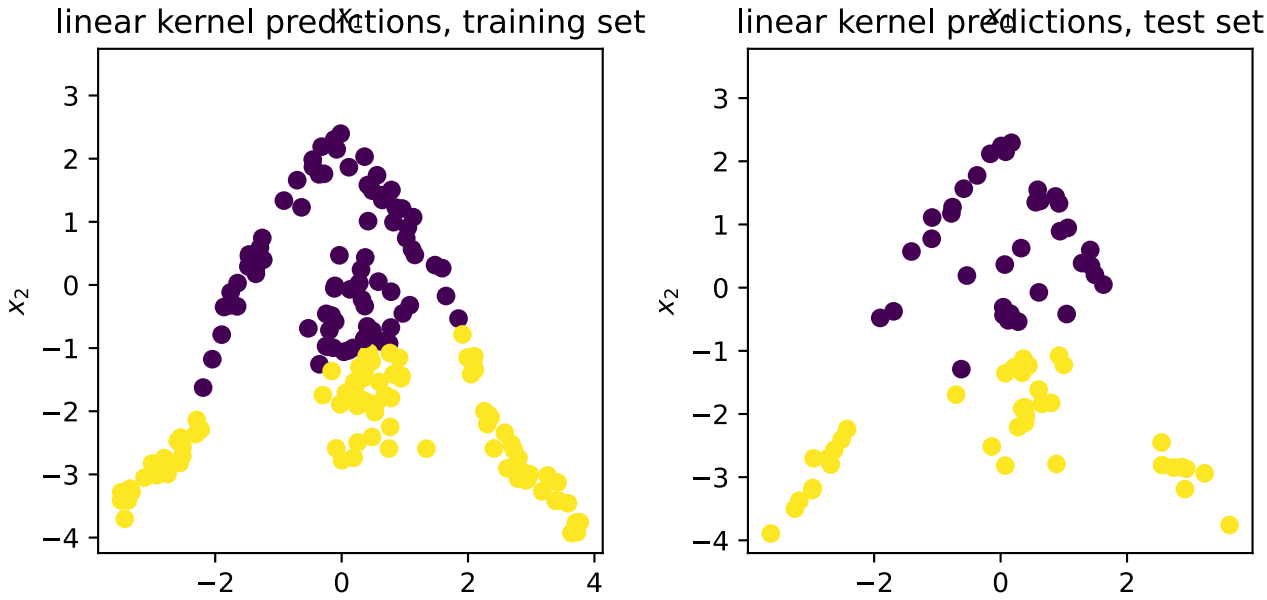
```
def plt2d(data, labels, title):
  plt.scatter(data[:,0], data[:,1], c = labels)
  plt.xlabel('$x_1$')
  plt.ylabel('$x_2$')
  plt.axis('square')
  plt.title(title)
```

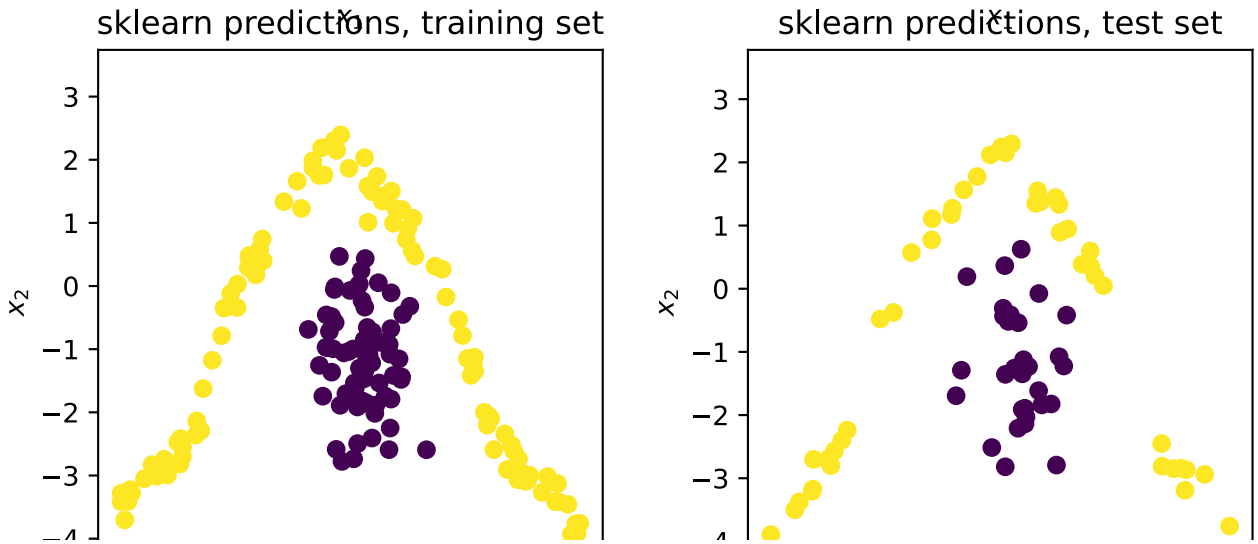
```
# Visualize
# Visualize results
plt.figure(figsize=(8,16))

plt.subplot(4,2,1)
plt2d(X_train, t_train, title = 'training set true values')

plt.subplot(4,2,2)
plt2d(X_test, t_test, title = 'test set true values')

plt.subplot(4,2,3)
plt2d(X_train, y_train, title = 'rbf kernel predictions, training set')

plt.subplot(4,2,4)
plt2d(X_test, y_test, title = 'rbf kernel predictions, test set')

plt.subplot(4,2,5)
plt2d(X_train, y_train_lk, title = 'linear kernel predictions, training set')

plt.subplot(4,2,6)
plt2d(X_test, y_test_lk, title = 'linear kernel predictions, test set')

plt.subplot(4,2,7)
plt2d(X_train, y_train_sk, title = 'sklearn predictions, training set')

plt.subplot(4,2,8)
plt2d(X_test, y_test_sk, title = 'sklearn predictions, test set')

plt.show()
```

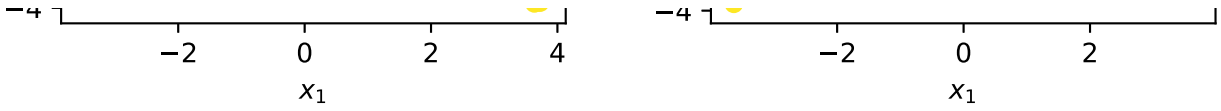



x_2
x_1


rbf kernel predictions, training set
rbf kernel predictions, test set
x_2
x_1

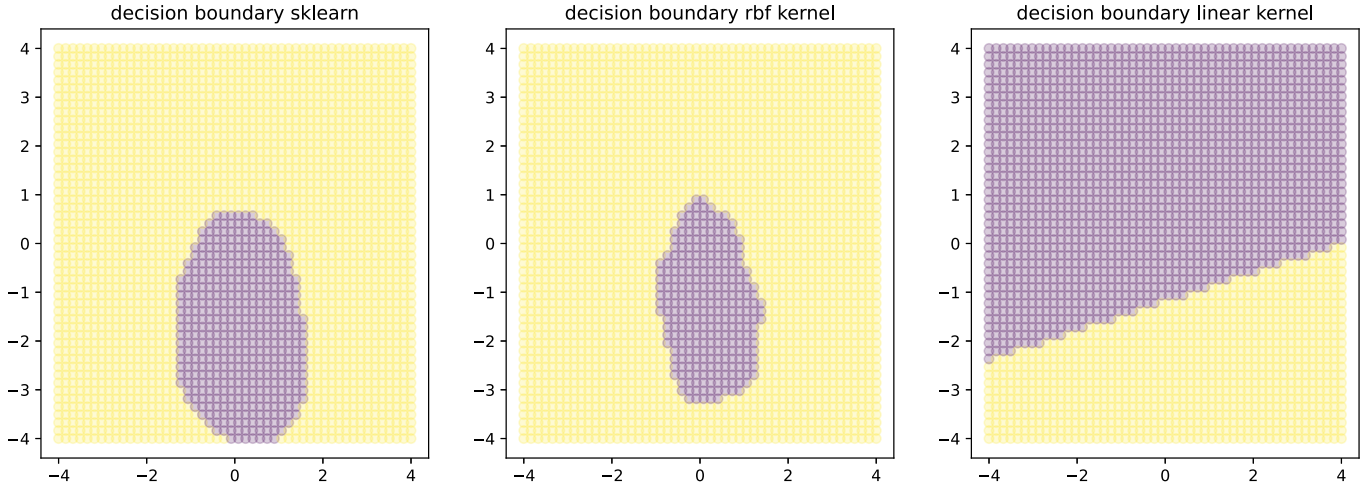

linear kernel predictions, training set
linear kernel predictions, test set
x_2
x_1


sklearn predictions, training set
sklearn predictions, test set
x_2
x_1



svg

```
def plot_decision_boundary(predict_fn):
    ''' Plot decision boundary.

    predict_fn: function handle to a predict function that takes
                as input a data point x and outputs the predicted
                label t
    '''
    grid = np.meshgrid(np.linspace(-4, 4, 50), np.linspace(-4, 4, 50))
    grid_x, grid_y = grid[0].flatten(), grid[1].flatten()

    grid_t = np.zeros(len(grid_x))
    for i, p in enumerate(zip(grid_x, grid_y)):
      tmp = np.array(p)
      tmp = np.reshape(tmp, (1, 2))
      grid_t[i] = predict_fn(tmp)

    plt.scatter(grid_x, grid_y, c=grid_t, alpha=0.2)


plt.figure(figsize = (15, 5))

plt.subplot(1, 3, 1)
predict_fn = lambda x: clf.predict(x)
plot_decision_boundary(predict_fn)
plt.title('decision boundary sklearn')

plt.subplot(1, 3, 2)
predict_fn = lambda x: svm_predict(x, rbf_kernel, lagr, svs, sv_labels, intercept)
plot_decision_boundary(predict_fn)
plt.title('decision boundary rbf kernel')

plt.subplot(1, 3, 3)
predict_fn = lambda x: svm_predict(x, linear_kernel, lagr_lk, svs_lk, sv_labels_lk, intercept
_lk)
plot_decision_boundary(predict_fn)
plt.title('decision boundary linear kernel')
plt.show()
```



svg

Task 2: Decision Trees

Next, we will implement a simple decision tree classifier using the Wine dataset, one of the standard sklearn datasets.

We will use the Gini impurity as a criterion for splitting. It is defined for a set of labels as

$$G = \sum_{i=0}^{C} p(i) * (1 - p(i))$$

Given labels l and split l_a and l_b, the weighted removed impurity can be computed by $G(l) - \frac{|l_a|}{|l|}G(l_a) - \frac{|l_b|}{|l|}G(l_b)$.

Here is a simple explanation of the Gini impurity that you may find useful: https://victorzhou.com/blog/gini-impurity/ (https://victorzhou.com/blog/gini-impurity/)

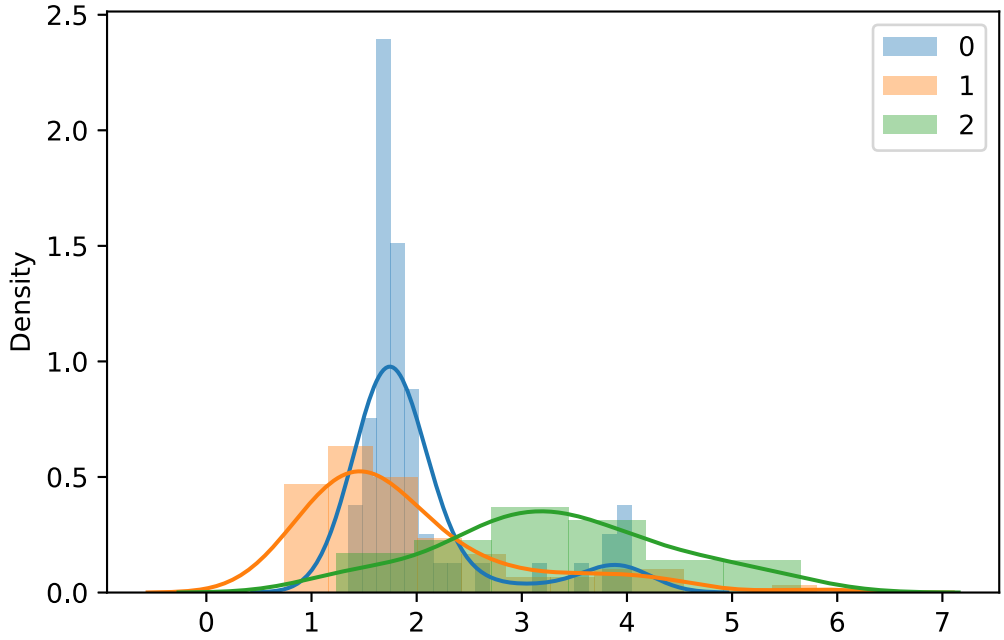
Task 2.1

1. Plot the distribution of the first feature of for each class of the wine dataset.
2. Implement a function `gini_impurity(t)` that computes the Gini impurity for an array of labels `t`.
3. Calculate the removed Gini impurity for a split after 50 samples, i.e. between `t[:50]` and `t[50:]`.

```
# Load Wine dataset and split into train+test set
from sklearn.datasets import load_wine
X, t = load_wine(return_X_y=True)
X_train, X_test, t_train, t_test = split_data(X = X, y = t)
```

```
# Plot distribution
import seaborn as sns
for i in np.unique(t):
    sns.distplot(X[(t==i)][:,1],kde=1,label='{}'.format(i))

plt.legend()
plt.show()
```

```
C:\Users\rene-\AppData\Local\Programs\Python\Python38\lib\site-packages\seaborn\distribution
s.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\rene-\AppData\Local\Programs\Python\Python38\lib\site-packages\seaborn\distribution
s.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\rene-\AppData\Local\Programs\Python\Python38\lib\site-packages\seaborn\distribution
s.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



svg

```
# Compute Gini impurity
def gini_impurity(data):
    temp = 0
    for i in np.unique(data):
        temp += np.sum(data==i)/len(data) * (1- np.sum(data==i)/len(data))
    return temp
```

```
print('Gini of full dataset: ', gini_impurity(t))
print('Gini of split dataset: ', gini_impurity(t[50:]))
print('Gini of split dataset: ', gini_impurity(t[:50]))
```

```
Gini of full dataset:  0.6583133442747128
Gini of split dataset:  0.5467529296875
Gini of split dataset:  0.0
```

```
gini_before = gini_impurity(t)
gini_after = len(t[:50])/len(t) * gini_impurity(t[:50]) + len(t[50:])/len(t) * gini_impurity
(t[50:])
reduced_gini = gini_before - gini_after
print("The removed gini based on a split after 50 samples is:",reduced_gini)
print("\n")
print("gini before:", gini_before)
print("gini after:", gini_after)
```

```
The removed gini based on a split after 50 samples is: 0.2651426982072971


gini before: 0.6583133442747128
gini after: 0.3931706460674157
```

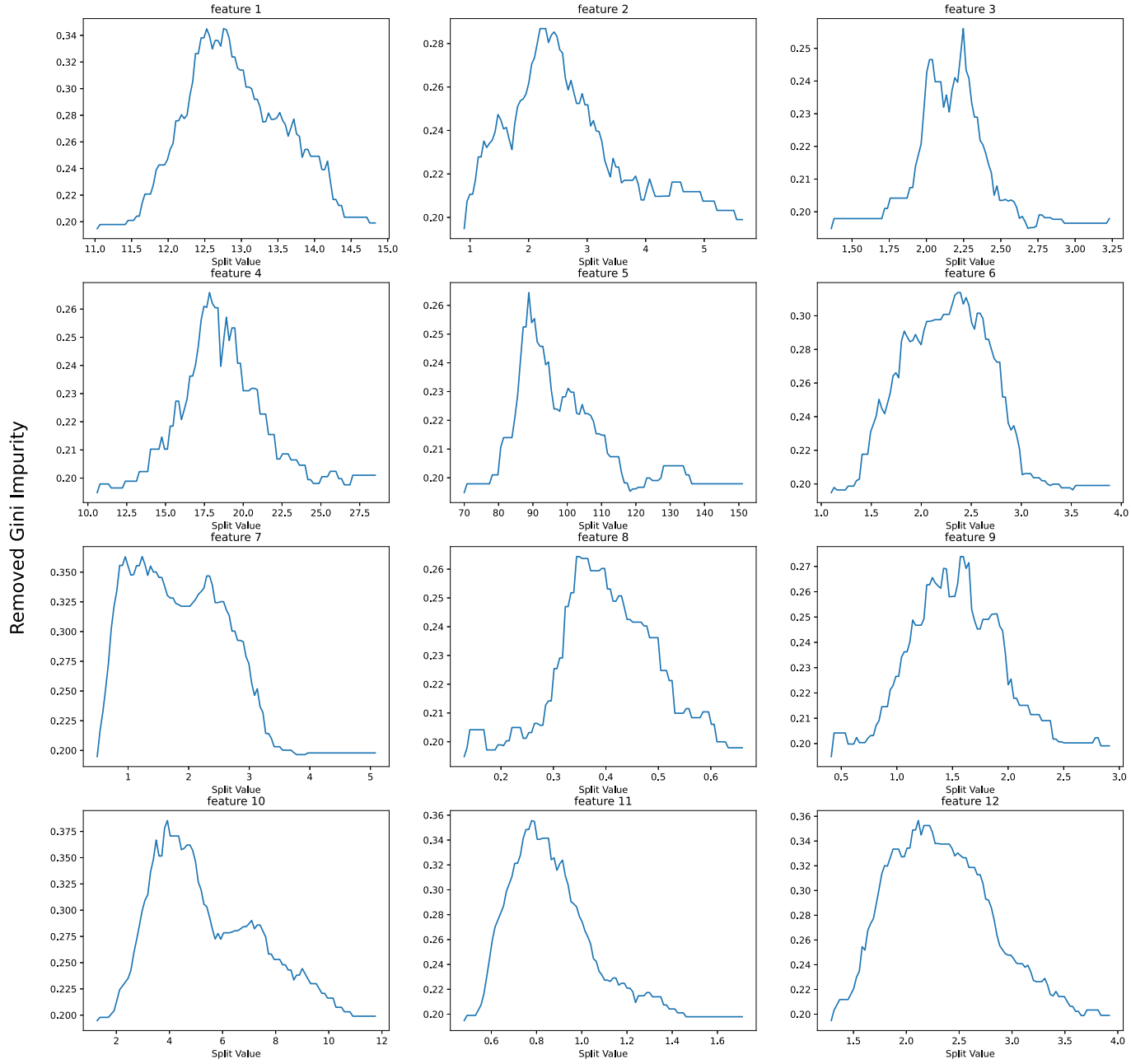
Task 2.2

Split the data along the first 12 features and plot the removed Gini impurity at different values of this feature. Which is the optimal split?

```
# Vereinfacher durch pandas:
import pandas as pd
X = pd.DataFrame(X)
t = pd.Series(t)
```

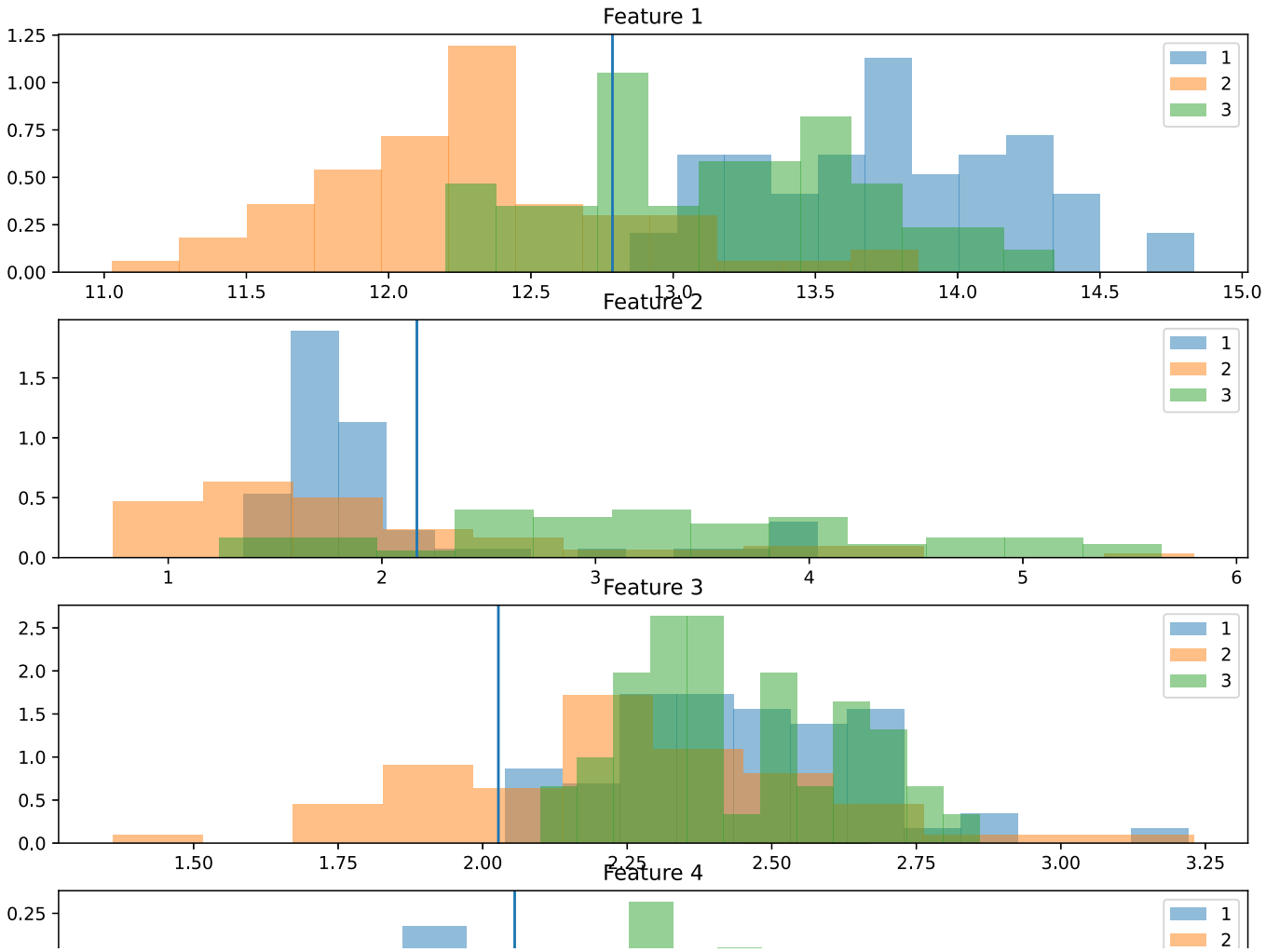
```
def splittest(wert, zeile, y):
    links, rechts = list(), list()
    n = len(zeile)
    for i in range(n):
        if zeile[i] < wert:
            links.append(int(y[i]))
        else:
            rechts.append(int(y[i]))
    # For format reasons:
    links, rechts = np.array(links), np.array(rechts)

    return links, rechts
```

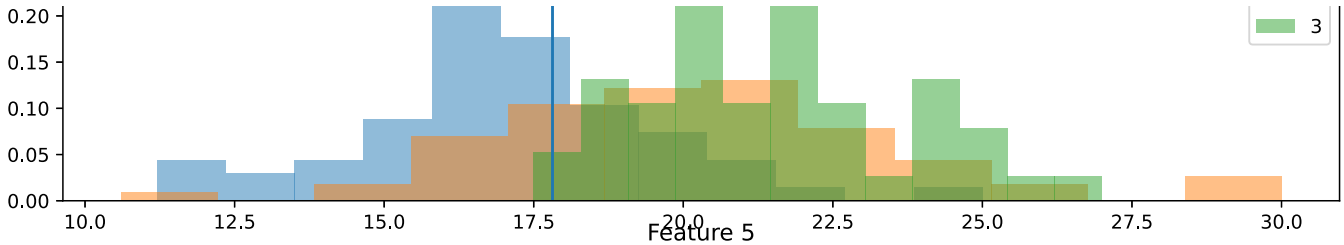
```
gini_reduc = np.ones((100,12))
thresholds = np.ones((100,12))
for jj in range(12):
    feat = X_train[:,jj]
    var_min = feat.min()
    var_max = feat.max()
    seq = np.linspace(var_min,var_max,100)
    thresholds[:,jj] = seq
    for ii in range(len(seq)):
        t_0 = t_train[feat < seq[ii]]
        t_1 = t_train[feat >= seq[ii]]
        gini_before = gini_impurity(t)
        gini_after = len(t_0)/len(t) * gini_impurity(t_0) + len(t_1)/len(t) * gini_impurity(t
_1)
        gini_reduc[ii,jj] = gini_before - gini_after

fig, ax = plt.subplots(4,3,figsize = (20,20))

fig.text(0.07, .5,"Removed Gini Impurity", fontsize = 20,va='center', rotation='vertical')
count = 0
for jj in range(4):
    for ii in range(3):
        ax[jj,ii].plot(thresholds[:,count],gini_reduc[:,count])
        ax[jj,ii].set_title("feature " + str(count+1))
        ax[jj,ii].set_xlabel("Split Value")
        count += 1
```


Removed Gini Impurity
feature 1
feature 2
feature 3
feature 4
feature 5
feature 6
feature 7
feature 8
feature 9
feature 10
feature 11
feature 12
Split Value

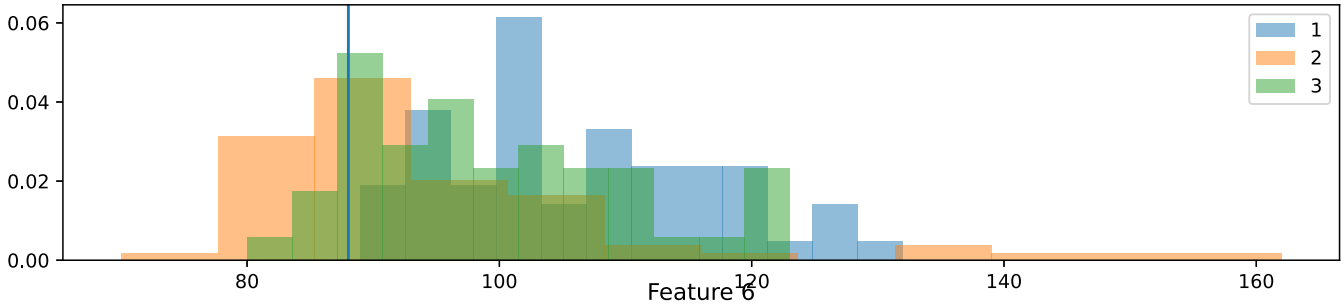
svg

```
fig, ax = plt.subplots(12, figsize = (12, 35))
for i in range(12):
    x_min, x_max = min(X.iloc[:, i]), max(X.iloc[:, i])
    x_range = np.linspace(start=x_min, stop=x_max, num=200)
    n, _ = X.shape
    bester_split, bester_gini = 0.0, 1.0
    for j in range(len(x_range)):
        links, rechts = splittest(wert = x_range[j], zeile=X.iloc[:, i], y=t) # Aufteilen
        links_gini, rechts_gini = gini_impurity(links), gini_impurity(rechts) #Gini für beide
Splits
        weighted_gini = (len(links)/n)*links_gini + (len(rechts)/n)*rechts_gini # Gewichteter
Gini
        if weighted_gini < bester_gini: # iteratives zuweisen
            bester_gini, bester_split = weighted_gini, x_range[j]
            bester_left, bester_right = links, rechts

    temp_ax = ax[i]
    feature1 = X.loc[t == 0, i]
    temp_ax.hist(feature1, alpha=0.5, bins=12, density=True, label = '1')
    feature2 = X.loc[t == 1, i]
    temp_ax.hist(feature2, alpha=0.5, bins=12, density=True, label = '2')
    feature3 = X.loc[t == 2, i]
    temp_ax.hist(feature3, alpha=0.5, bins=12, density=True, label = '3')
    temp_ax.axvline(x=bester_split)
    temp_ax.legend()
    name = 'Feature '
    name += str(i+1)
    temp_ax.set_title(name)
```

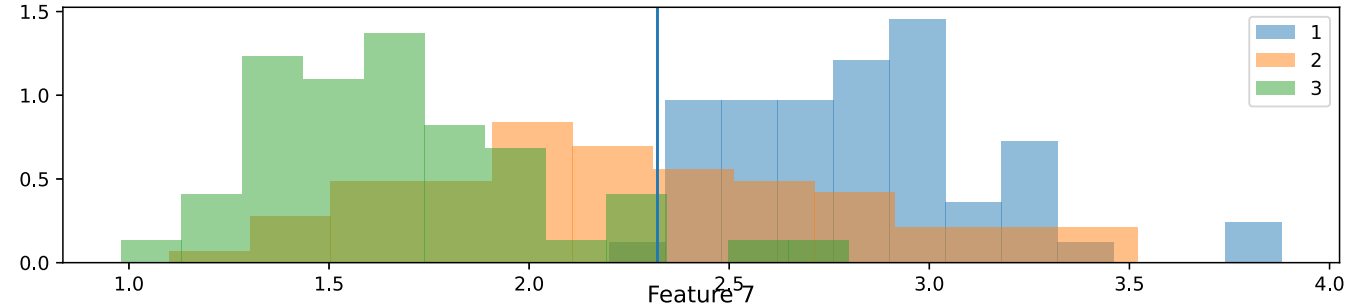




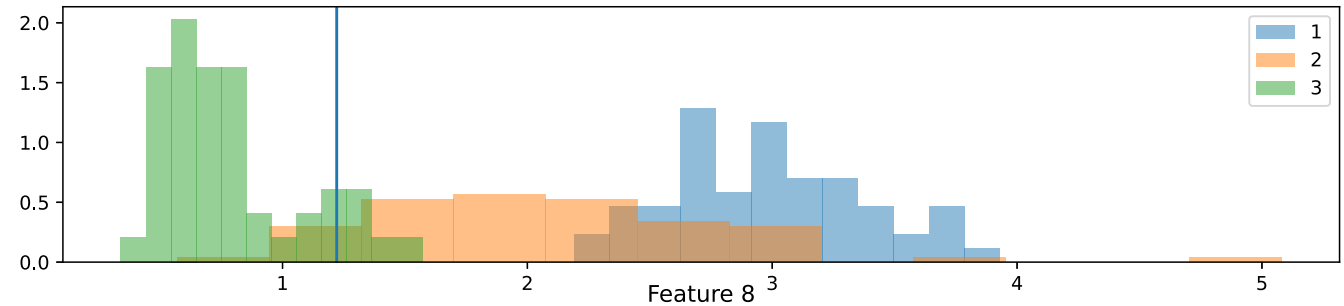
Feature 5



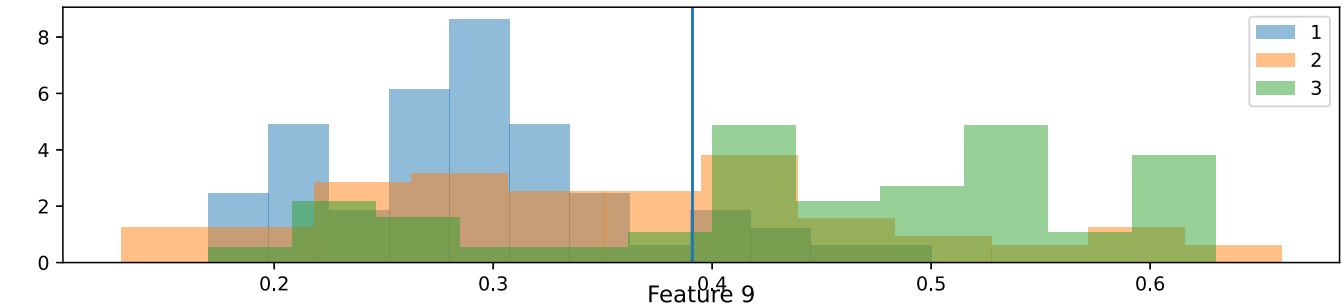
Feature 6



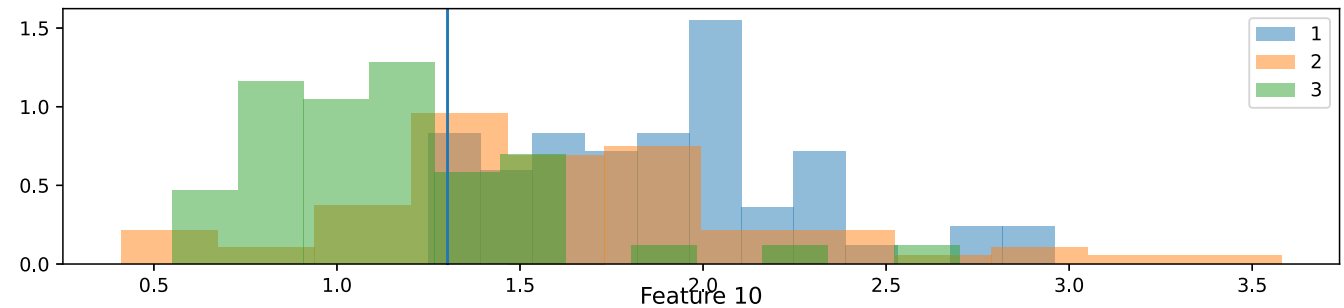
Feature 7



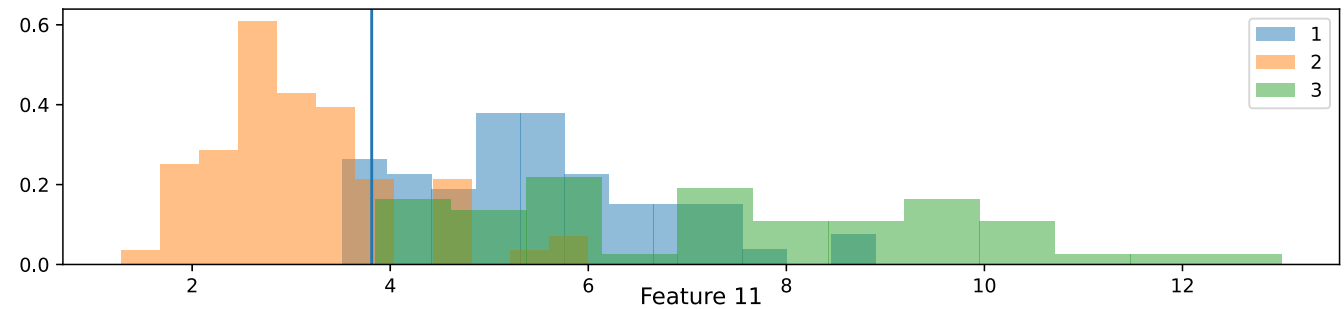
Feature 8



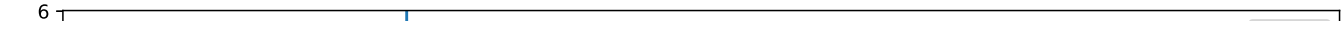
Feature 9

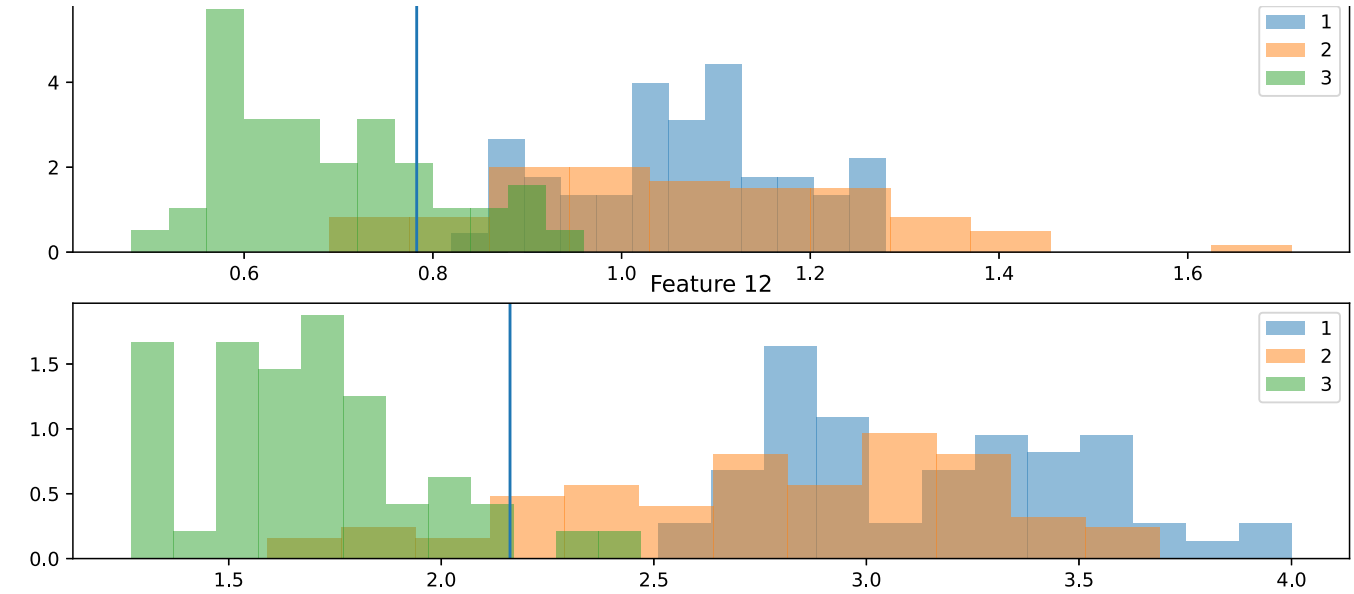


Feature 10



Feature 11





svg

Task 2.3

1. Implement a function `build_tree(X, t, depth)` which recursively builds a tree. Use the classes `Node` and `Leaf` as a data structure to build your tree.
2. Implement a function `predict_tree(tree, x)` which makes a prediction for sample `x`. Obtain scores for the `wine` dataset and compare to `sklearn.tree.DecisionTree`.
3. Switch back to the synthetic 2d dataset from the beginning (kernel methods). Compute scores and visualize the decisions in a 2d grid.

```
class Node:
    def __init__(self, links, rechts, n_feat, thresholdold):
        self.links = links
        self.rechts = rechts
        self.n_feat = n_feat
        self.thresholdold = thresholdold
    # Funktion um Format ausgeben zu können
    def desc(self):
      return(self.n_feat, self.thresholdold)
    # Folgt ein Split/Child Node?
    def followingSplits(self):
      return True
    # Prediction Function:
    def predict_tree(self, x):
      print(self.followingSplits())
      if(self.followingSplits()):
        feat_n, threshold = self.desc()
        feat_x = x[feat_n]
        if(feat_x < threshold):
          return self.links.predict_tree(x)
        else:
          return self.rechts.predict_tree(x)
      else:
        return self.desc()
```

```
class Leaf:
    def __init__(self, label):
        self.label = label
    # braucht man erst für die plots wirklich
    def desc(self):
      return(self.label)
    # Um zu stoppen
    def followingSplits(self):
      return False
```

```
def treeplot(node, x=[]):
    # cutoff points for plots
  if node.followingSplits():
    x.append(node.desc())
    if node.links is not None:
      treeplot(node.links,x)
    if node.rechts is not None:
      treeplot(node.rechts,x)
```

```
def build_tree(X, t, depth, max_depth=3, n_labels=2):
    # wegen: Cannot cast array data from dtype('float64') to dtype('int64') according to the
 rule 'safe'
    if len(X) <=0:
      return
    if len(X)<= 1:
      return Leaf(np.argmax(np.bincount(t)))
    if depth == max_depth:
      return Leaf(np.argmax(np.bincount(t)))

    #start werte
    current_gini, temp_feat, current_split = 1, None , 0.0
    for i in range(X.shape[1]):
      x_min, x_max = min(X.iloc[:, i]), max(X.iloc[:, i])
      x_range = np.linspace(start=x_min, stop=x_max, num=200)
      n = X.shape[0]

      for j in range(len(x_range)):
          links, rechts = splittest(wert = x_range[j], zeile=X.iloc[:, i], y=t) # split colum
n into links and rechts
          links_gini, rechts_gini = gini_impurity(links), gini_impurity(rechts)  # calculate
 gini coef for links and rechts
          weighted_gini = (len(links)/n)*links_gini + (len(rechts)/n)*rechts_gini # calculate
wighted gini
          if weighted_gini < current_gini:  # save best gini
              current_gini, current_split = weighted_gini, x_range[j]
              temp_links, temp_rechts, temp_feat = links, rechts, i

    links_X, rechts_X = [], []

    for k in range(X.shape[0]): # fröhliches iterieren
      if X.iloc[k,temp_feat] < current_split:
        links_X.append(X.iloc[k,:])
      else:
        rechts_X.append(X.iloc[k,:])

    node = Node(None,None,temp_feat,current_split)  # node class nutzen
    node.links = build_tree(pd.DataFrame(np.array(links_X)), temp_links, depth+1, max_depth,
 n_labels)
    node.rechts = build_tree(pd.DataFrame(np.array(rechts_X)), temp_rechts, depth+1, max_dept
h, n_labels)

    return node
```

```
def predict_tree(node, x):
  if(node.followingSplits()):
    feat_n, threshold = node.desc()
    feat_x = x[feat_n]
    if(feat_x < threshold):
      return predict_tree(node.links, x)
    else:
      return predict_tree(node.rechts, x)
  else:
      return node.desc()
```

```
def split_data(X, y, frac=0.3, max_samples=None, seed=4711):

    if seed is not None:
        np.random.seed(seed)

    indices = np.random.permutation(len(X))
    indices = indices[:max_samples]
    indices_test, indices_train = indices[:int(len(indices)*frac)], indices[int(len(indices)*
frac):]

    assert (len(set(indices))) == len(indices)
    assert len(set(indices_test).intersection(indices_train)) == 0

    X_train, X_test = X[indices_train], X[indices_test]
    y_train, y_test = y[indices_train], y[indices_test]

    return X_train, X_test, y_train, y_test
```

```
X, t = load_wine(return_X_y=True)
X_train, X_test, t_train, t_test = split_data(X = X, y = t)
```

```
tree = build_tree(pd.DataFrame(X_train),t_train, 0, max_depth=3, n_labels=3)
```

```
count = 0
for i in range(len(X_test)):
  if(t_test[i] == predict_tree(tree, X_test[i])):
    count += 1
```

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth=3)
clf = clf.fit(X_train, t_train)
predsk = clf.predict(X_test)
skcounter = 0
for i in range(len(predsk)):
  if(predsk[i] == t_test[i]):
    skcounter += 1
```

```
print("sklearn score:", skcounter/len(t_test))
print("Our score:", (count)/len(t_test))
```

```
sklearn score: 0.9245283018867925
Our score: 0.9245283018867925
```

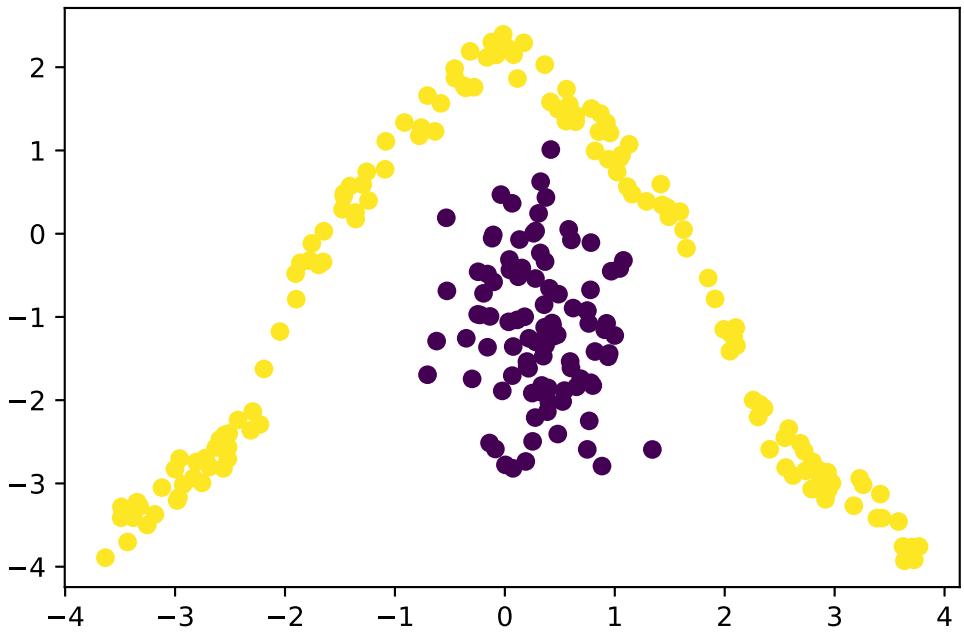
```
X_2d, t_2d = np.load('data/nonlin_2d_data.npy')[:,:2], np.load('data/nonlin_2d_data.npy')[:,
2]
X_train, X_test, y_train, y_test = split_data(X_2d, t_2d)
y_train = y_train.astype(int)
```

```
tree = build_tree(pd.DataFrame(X_train),y_train, 0, max_depth=3, n_labels=2)
```

```
correct = 0
for i in range(len(X_test)):
  if(y_test[i] == predict_tree(tree, X_test[i])):
    correct += 1
print(correct/len(y_test))
```

```
1.0
```

```
fig, ax = plt.subplots()
plt.scatter(X_2d[:,0], X_2d[:, 1], c=t_2d)
plt.show()
```

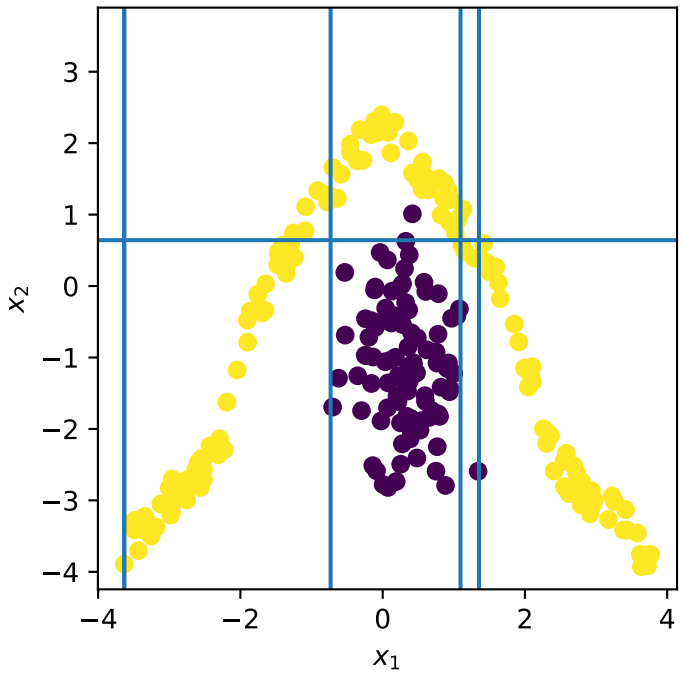


svg

```
cutoff = []
treeplot(tree, cutoff)
print(cutoff)
```

```
[(0, -0.7324396890211675), (0, -3.633659394608522), (0, -3.633659394608522), (0, 1.0942337613
46704), (1, 0.6419593869827454), (0, 1.35390440889765)]
```

```
fig, ax = plt.subplots()
plt.scatter(X_2d[:,0], X_2d[:, 1], c=t_2d)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.axis('square')
#ax.axhline(y=1)
for i in cutoff:
  if i[0]:
    ax.axhline(y=i[1])
  else:
    ax.axvline(x=i[1])
```



svg