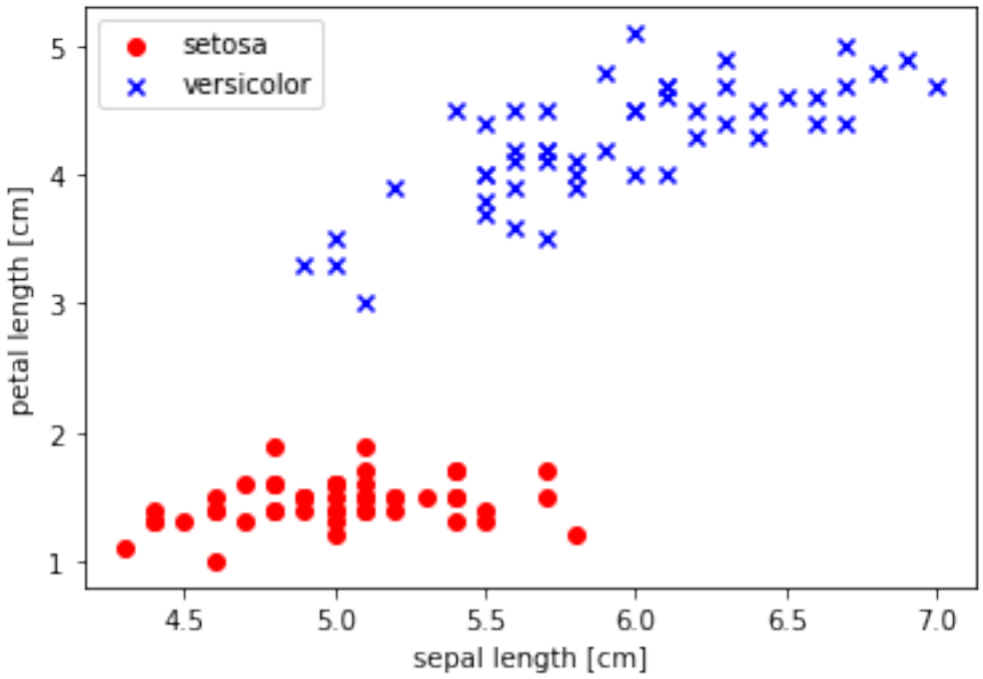
hw-assignment-2

February 15, 2023

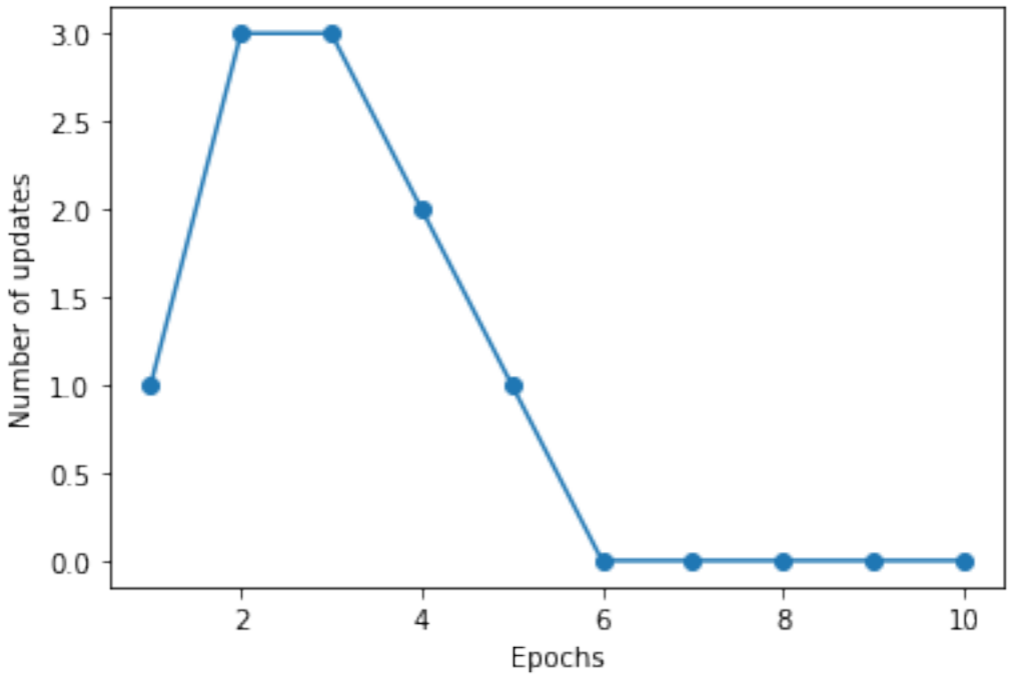
[4]:
```
  import os
import pandas as pd
s = os.path.join('https://archive.ics.uci.edu',
  'ml', 'machine-learning-databases',
  'iris','iris.data')
df = pd.read_csv(s,
  header=None,
  encoding='utf-8')
df.tail()

import matplotlib.pyplot as plt
import numpy as np
#     select setosa and
versicolor y = df.iloc[0:100,
4].values
y = np.where(y == 'Iris-setosa', -1, 1)
#     extract sepal length and petal
length X = df.iloc[0:100, [0,
2]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
  color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
  color='blue', marker='x', label='versicolor')

plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



[15]:
```
import numpy as np
class Perceptron(object):
  """Perceptron classifier.

  Parameters
  ------------
  eta : float
    Learning rate (between 0.0 and 1.0)
  n_iter : int
    Passes over the training dataset.
  random_state : int
    Random number generator seed for random
    weight initialization.

  Attributes
  -----------
  w_ : 1d-array
    Weights after fitting.
  errors_ : list
    Number of misclassifications (updates) in each epoch.

  """
  def __init__(self, eta=0.01, n_iter=50, random_state=1):
    self.eta = eta
    self.n_iter = n_iter
```

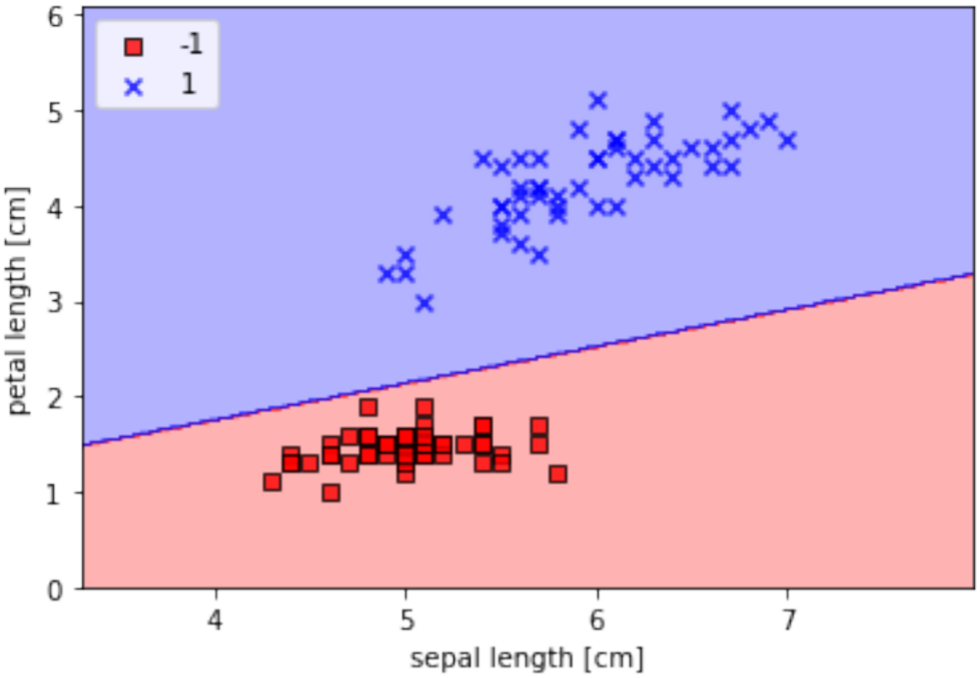
```
    self.random_state = random_state

  def fit(self, X, y):
    """Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_examples, n_features]
      Training vectors, where n_examples is the number of
      examples and n_features is the number of features.
    y : array-like, shape = [n_examples]
      Target values.

    Returns
    -------
    self : object

    """
    rgen = np.random.RandomState(self.random_state)
    self.w_ = rgen.normal(loc=0.0, scale=0.01,
      size=1 + X.shape[1])
    self.errors_ = []

    for _ in range(self.n_iter):
      errors = 0
      for xi, target in zip(X, y):
        update = self.eta * (target - self.predict(xi))
        self.w_[1:] += update * xi
        self.w_[0] += update
        errors += int(update != 0.0)
      self.errors_.append(errors)
    return self

  def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

  def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.net_input(X) >= 0.0, 1, -1)

ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1),
  ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
```

```
plt.show()
```



[19]:
```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
 #     setup marker generator and
 color map markers = ('s', 'x', 'o',
 '^', 'v')
 colors = ('red', 'blue', 'lightgreen', 'gray',
 'cyan') cmap =
 ListedColormap(colors[:len(np.unique(y))])

 # plot the decision surface
 x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
 x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
  resolution), np.arange(x2_min, x2_max,
  resolution))
 Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
 Z = Z.reshape(xx1.shape)
 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
 plt.xlim(xx1.min(), xx1.max())
 plt.ylim(xx2.min(), xx2.max())

 # plot class examples
 for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0],
                y=X[y == cl, 1],
```

```
                alpha=0.8,
                c=colors[idx],
                marker=markers[idx],
                label=cl,
                edgecolor='black')
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



[50]:
```
class AdalineGD(object):
    """ADAptive LInear NEuron classifier.

    Parameters
    ------------
    eta : float
      Learning rate (between 0.0 and 1.0)
    n_iter : int
      Passes over the training dataset.
    random_state : int
      Random number generator seed for random weight initialization.

    Attributes
```

```
    -----------
    w_ : 1d-array
      Weights after fitting.
    cost_ : list
      Sum-of-squares cost function value in each epoch.

    """
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
      self.eta = eta
      self.n_iter = n_iter
      self.random_state = random_state

    def fit(self, X, y):
      """ Fit training data.

      Parameters
      ----------
      X : {array-like}, shape = [n_examples, n_features]
        Training vectors, where n_examples
        is the number of examples and
        n_features is the number of features.
      y : array-like, shape = [n_examples]
        Target values.

      Returns
      -------
      self : object

      """
      rgen = np.random.RandomState(self.random_state)
      self.w_ = rgen.normal(loc=0.0, scale=0.01,
        size=1 + X.shape[1])
      self.cost_ = []


def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def activation(self, X):
    """Compute linear activation"""
    return X

def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.activation(self.net_input(X))
      >= 0.0, 1, -1)
```

```
 for i in range(self.n_iter):
     net_input = self.net_input(X)
     output = self.activation(net_input)
     errors = (y - output)
     self.w_[1:] += self.eta * X.T.dot(errors)
     self.w_[0] += self.eta * errors.sum()
     cost = (errors**2).sum() / 2.0
     self.cost_.append(cost)
 return self
```

```
---------------------------------------------------------------------------
NameError Traceback (most recent call last)
<ipython-input-50-bdf2665bafe9> in <module>
    58     return np.where(self.activation(self.net_input(X))
    59       >= 0.0, 1, -1)
---> 60 for i in range(self.n_iter):
    61     net_input = self.net_input(X)
    62     output = self.activation(net_input)

NameError: name 'self' is not defined
```

[51]:
```
class AdalineGD(object):
"""ADAptive LInear NEuron classifier.

Parameters
------------
eta : float
 Learning rate (between 0.0 and 1.0)
n_iter : int
 Passes over the training dataset.
random_state : int
 Random number generator seed for random weight initialization.

Attributes
-----------
w_ : 1d-array
 Weights after fitting.
cost_ : list
 Sum-of-squares cost function value in each epoch.

"""
def __init__(self, eta=0.01, n_iter=50, random_state=1):
 self.eta = eta
 self.n_iter = n_iter
 self.random_state = random_state
```

```
 def fit(self, X, y):
  """ Fit training data.

  Parameters
  ----------
  X : {array-like}, shape = [n_examples, n_features]
    Training vectors, where n_examples
    is the number of examples and
    n_features is the number of features.
  y : array-like, shape = [n_examples]
    Target values.

  Returns
  -------
  self : object

  """
  rgen = np.random.RandomState(self.random_state)
  self.w_ = rgen.normal(loc=0.0, scale=0.01,
    size=1 + X.shape[1])
  self.cost_ = []

  for i in range(self.n_iter):
      net_input = self.net_input(X)
      output = self.activation(net_input)
      errors = (y - output)
      self.w_[1:] += self.eta * X.T.dot(errors)
      self.w_[0] += self.eta * errors.sum()
      cost = (errors**2).sum() / 2.0
      self.cost_.append(cost)
  return self
def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def activation(self, X):
    """Compute linear activation"""
    return X

def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.activation(self.net_input(X))
      >= 0.0, 1, -1)
```

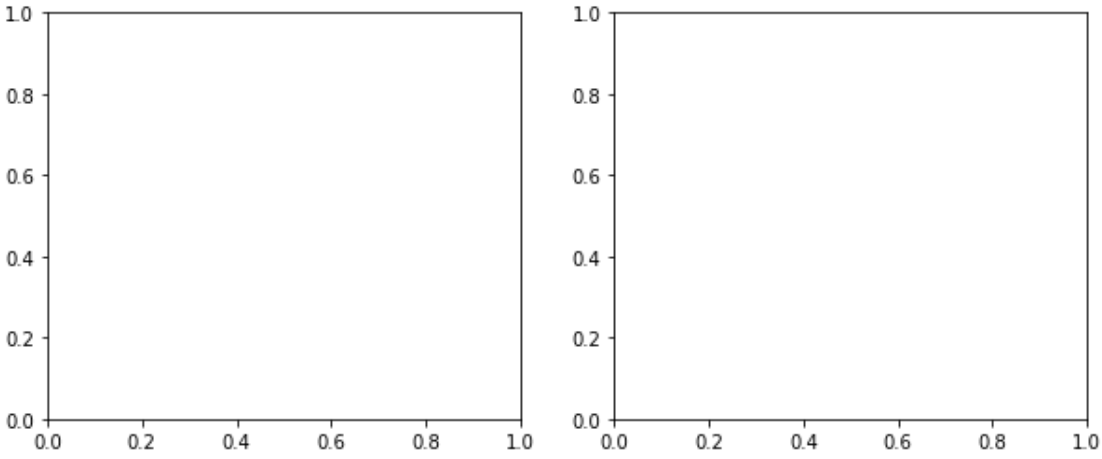
[52]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
ada1 = AdalineGD(n_iter=10, eta=0.01).fit(X, y)
ax[0].plot(range(1, len(ada1.cost_) + 1),
```

```
    np.log10(ada1.cost_), marker='o')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('log(Sum-squared-error)')
ax[0].set_title('Adaline - Learning rate 0.01')
ada2 = AdalineGD(n_iter=10, eta=0.0001).fit(X, y)
ax[1].plot(range(1, len(ada2.cost_) + 1),
    ada2.cost_, marker='o')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Sum-squared-error')
ax[1].set_title('Adaline - Learning rate 0.0001')
plt.show()
```

```
AttributeError
<ipython-input-52-02dcbdd79344> in <module>
            fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,
          1 4))
                                                                  Ada
                                                                  lin
                                                                  eGD
                                                                  (n_
                                                                  ite
                                                                  r=1
                                                                  0,
                                                                  eta
      ----> 2 ada1 =                                              =0.
```

```
01)
.fi
t(X
,
y)
```

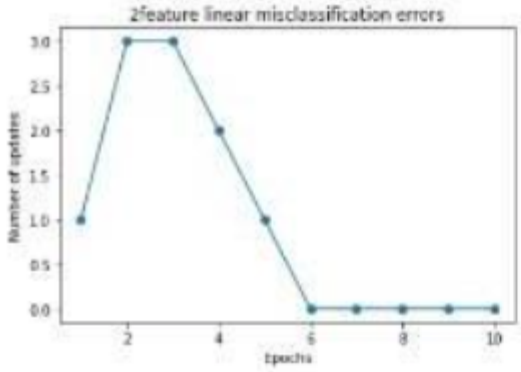
```
      3 ax[0].plot(range(1, len(ada1.cost_) + 1),
      4     np.log10(ada1.cost_),
        marker='o') 5
        ax[0].set_xlabel('Epochs')

<ipython-input-51-b72f8ec821a1> in fit(self, X, y)
     47
     48   for i in range(self.n_iter):
---> 49       net_input = self.net_input(X)
     50       output = self.activation(net_input)
     51       errors = (y - output)

AttributeError: 'AdalineGD' object has no attribute 'net_input'
```

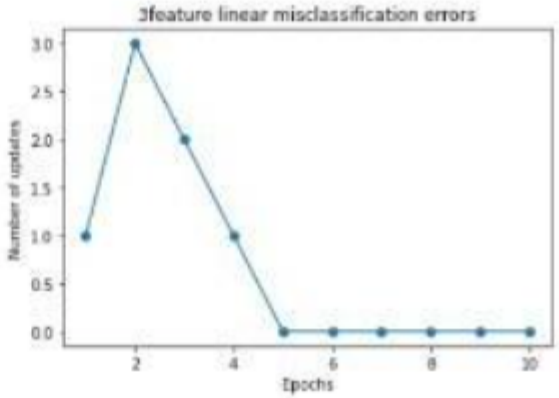


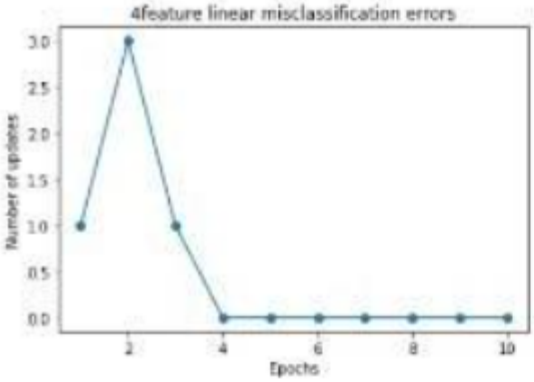
[53]:
```
  >>> X_std = np.copy(X)
>>> X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
>>> X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()
>>> ada_gd = AdalineGD(n_iter=15, eta=0.01)
>>> ada_gd.fit(X_std, y)
>>> plot_decision_regions(X_std, y, classifier=ada_gd)
>>> plt.title('Adaline - Gradient Descent')
>>> plt.xlabel('sepal length [standardized]')
>>> plt.ylabel('petal length [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.tight_layout()
>>> plt.show()
>>> plt.plot(range(1, len(ada_gd.cost_) + 1),
... ada_gd.cost_, marker='o')
>>> plt.xlabel('Epochs')
>>> plt.ylabel('Sum-squared-error')
>>> plt.tight_layout()
>>> plt.show()
```

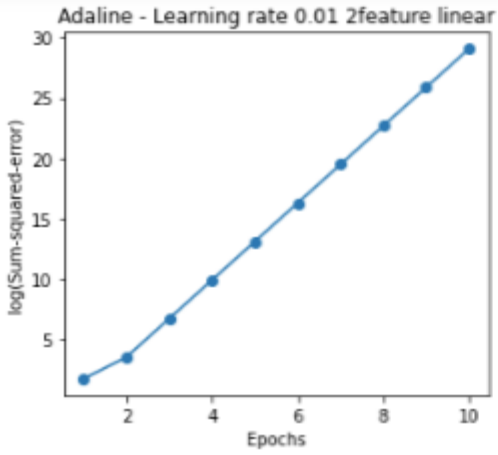
```
AttributeError
<ipython-input-53-2ba64ce02b6c> in <module>
      3 X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()
      4 ada_gd = AdalineGD(n_iter=15, eta=0.01)
```

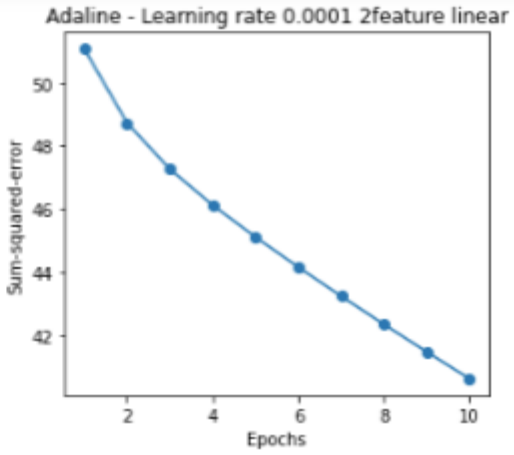
```
                    ----> 5 ada_gd.fit(X_std, y)
                            plot_decision_regions(X_std, y,
                          6 classifier=ada_gd)
                            plt.title('Adaline - Gradient
                          7 Descent')

<ipython-input-51-b72f8ec821a1> in fit(self, X, y)
     47
     48   for i in range(self.n_iter):
---> 49       net_input = self.net_input(X)
     50       output = self.activation(net_input)
     51       errors = (y - output)

AttributeError: 'AdalineGD' object has no attribute 'net_input'
```

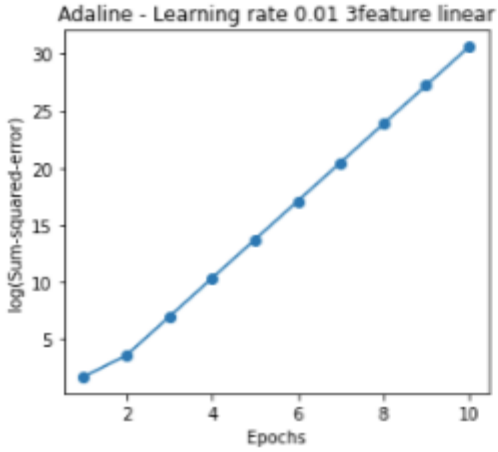

2feature linear misclassification errors
Number of updates
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2
4
6
8
10

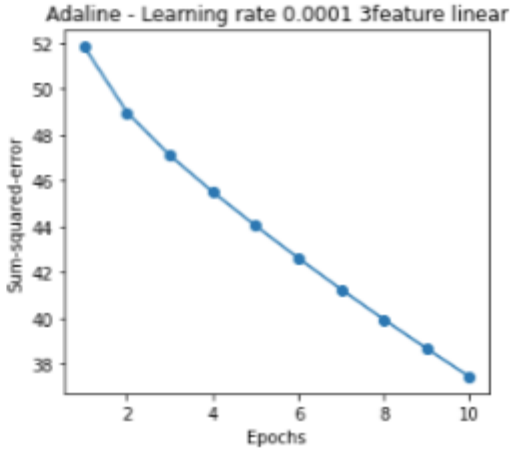

3feature linear misclassification errors
Number of updates
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2
4
6
8
10

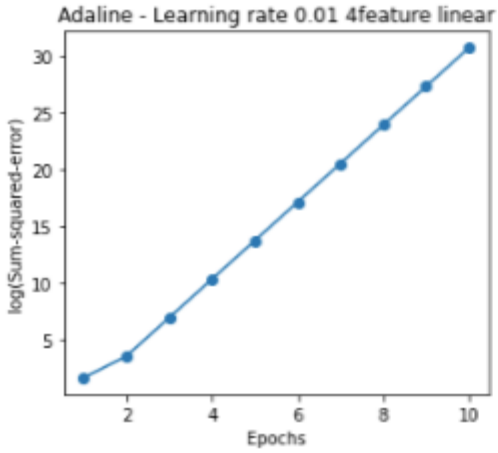

4feature linear misclassification errors
Number of updates
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2
4
6
8
10

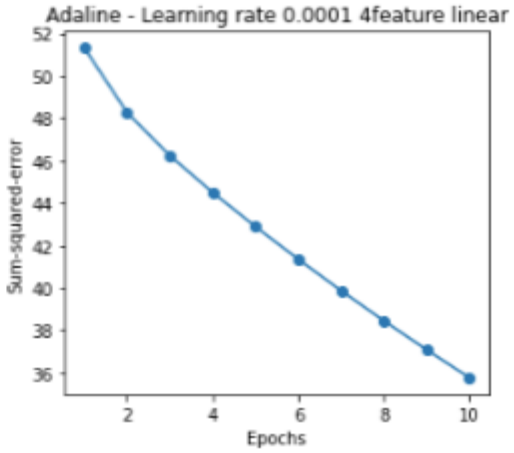

Adaline - Learning rate 0.01 2feature linear
log(Sum-squared-error)
Epochs

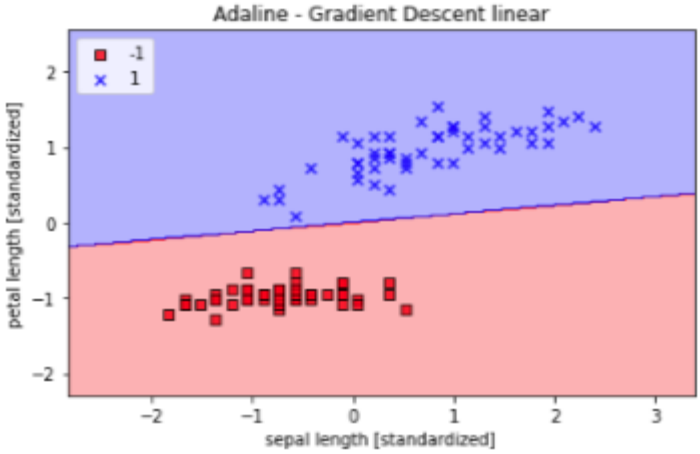

Adaline - Learning rate 0.0001 2feature linear
Sum-squared-error
Epochs

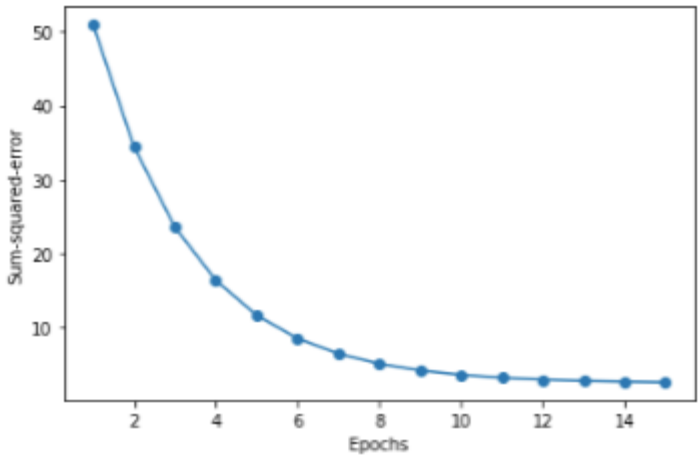

Adaline - Learning rate 0.01 3feature linear
log(Sum-squared-error)
Epochs

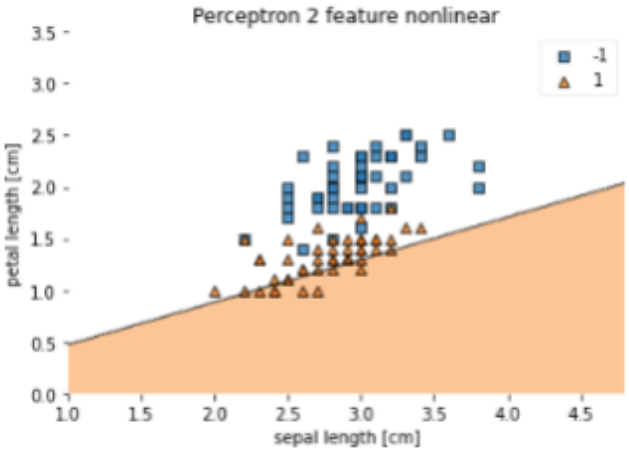

Adaline - Learning rate 0.0001 3feature linear
Sum-squared-error
Epochs

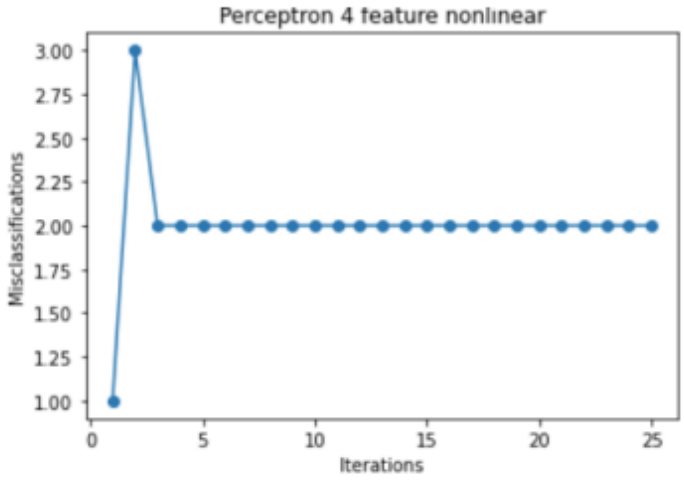

Adaline - Learning rate 0.01 4feature linear
log(Sum-squared-error)
Epochs

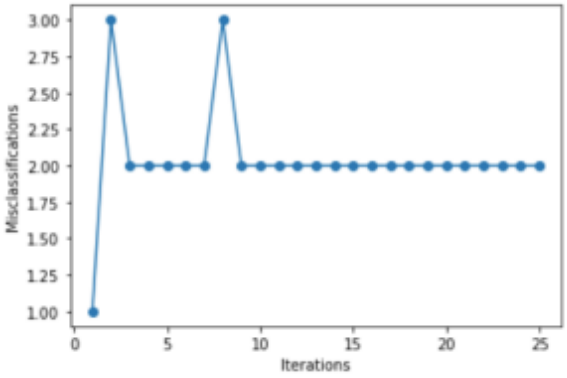

Adaline - Learning rate 0.0001 4feature linear
Sum-squared-error
Epochs

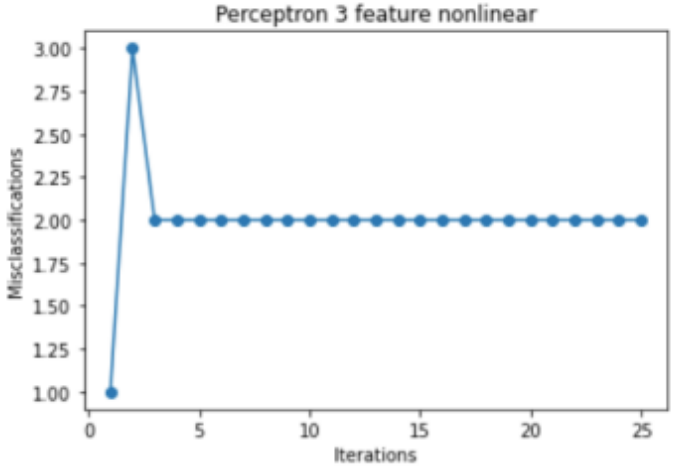

Adaline - Gradient Descent linear
-1
1
petal length [standardized]
sepal length [standardized]


Sum-squared-error
Epochs

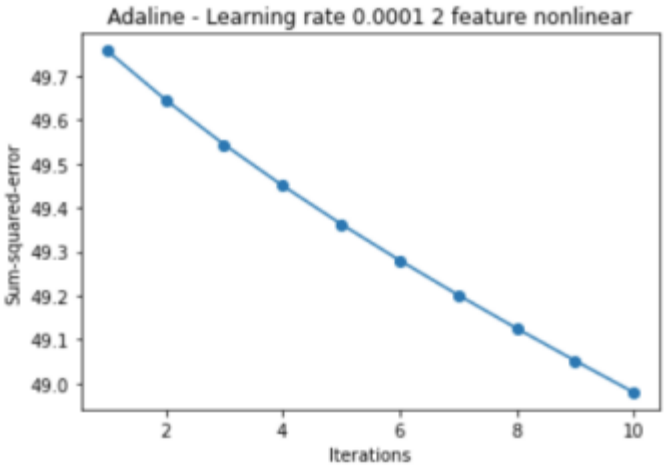


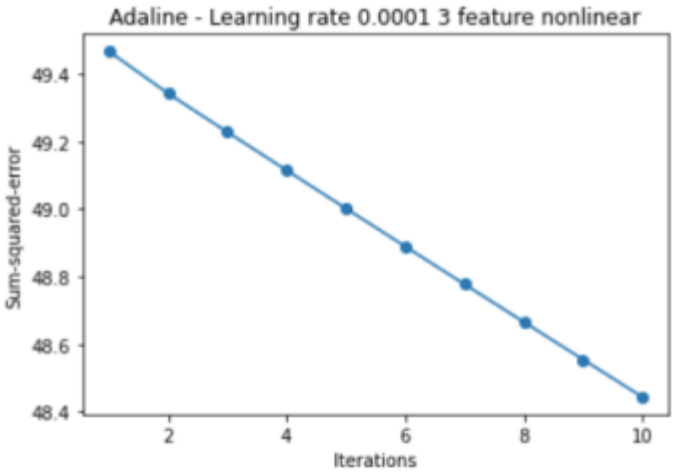


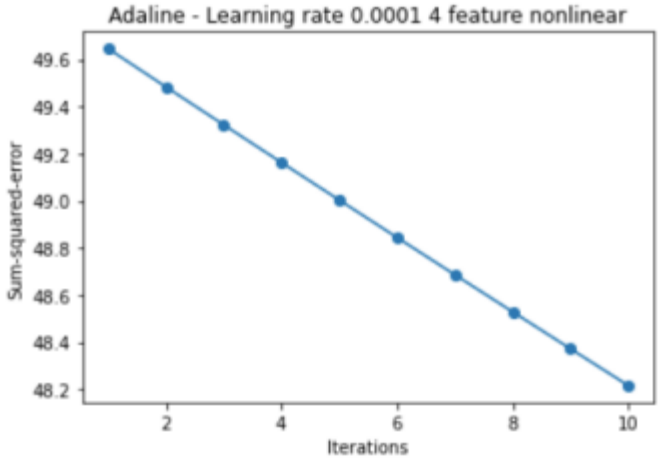


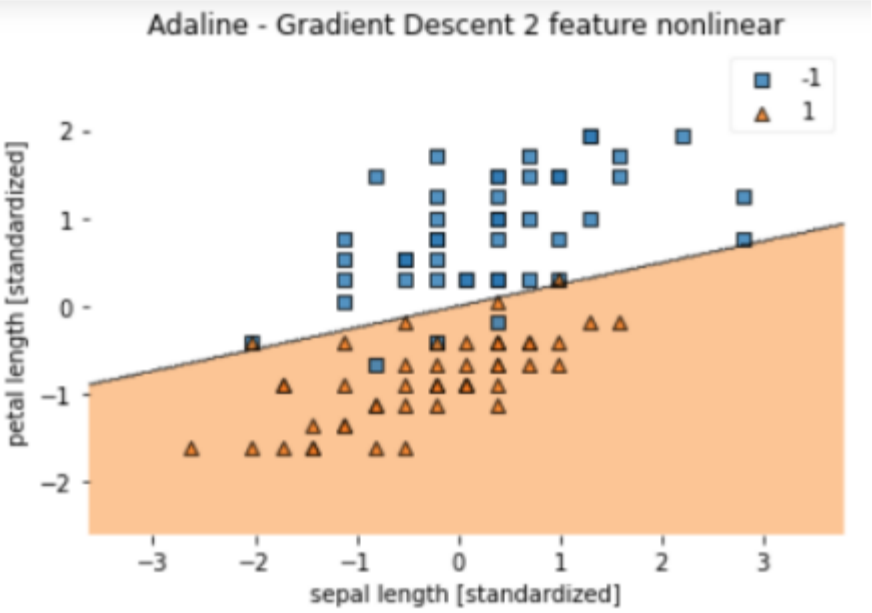


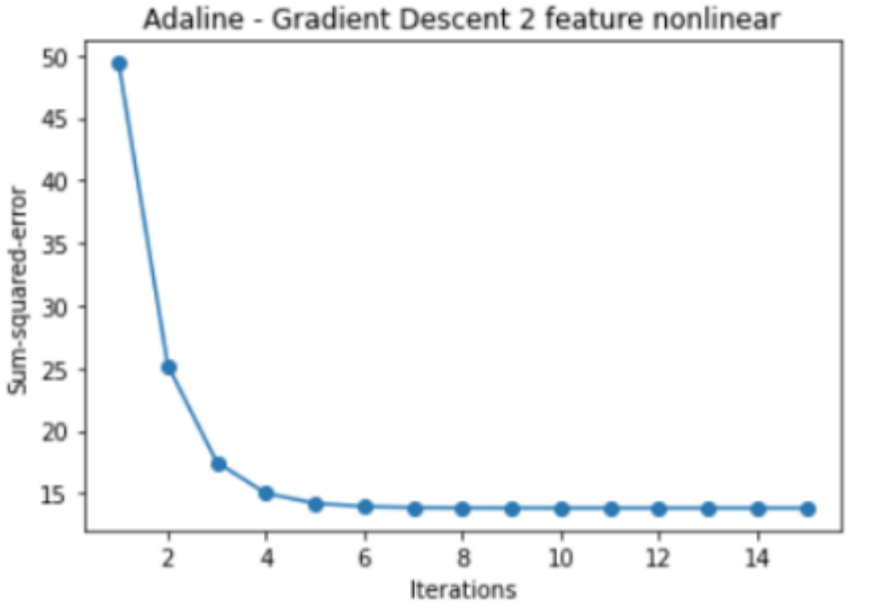

```
Weights: [ 0.     0.408  0.034 -0.528 -0.464]
```


Adaline - Learning rate 0.0001 2 feature nonlinear
Sum-squared-error
Iterations
49.7
49.6
49.5
49.4
49.3
49.2
49.1
49.0
2
4
6
8
10


Adaline - Learning rate 0.0001 3 feature nonlinear
Sum-squared-error
Iterations
49.4
49.2
49.0
48.8
48.6
48.4
2
4
6
8
10


Adaline - Learning rate 0.0001 4 feature nonlinear
Sum-squared-error
Iterations
49.6
49.4
49.2
49.0
48.8
48.6
48.4
48.2
2
4
6
8
10


Adaline - Gradient Descent 2 feature nonlinear
-1
1
petal length [standardized]
sepal length [standardized]


Adaline - Gradient Descent 2 feature nonlinear
Sum-squared-error
Iterations