Mustererkennung/Machine Learning - Assignment 8

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
%matplotlib inline
```

In [2]:

```
data = pd.read_csv("iris.data", header=None)
data.head(n=5)
```

Out[2]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Excercise 1. Perceptron

Implement the Perceptron algorithm using Python (incl. Numpy etc.) and use it on the Iris-Dataset. Train the algorithm to seperate Setosa from Versicolour and Virginica.

In [3]:

```
X_train, y_train = data[list(range(4))], data[4]

X_train_setosa = X_train[y_train=='Iris-setosa'].to_numpy()
X_train_versicolor = X_train[y_train=='Iris-versicolor'].to_numpy()
X_train_virginica = X_train[y_train=='Iris-virginica'].to_numpy()

X_s_vs_vv = np.vstack((X_train_setosa, X_train_versicolor, X_train_virginica))
y_s_vs_vv = np.hstack((np.ones((X_train_setosa.shape[0], )), np.zeros((X_train_vers

X_train_s_vs_vv, X_test_s_vs_vv, y_train_s_vs_vv, y_test_s_vs_vv = train_test_split
```

In [4]:

```
def accuracy(labels, predictions):
    return np.mean(labels == predictions)

def heavyside(x):
    return np.where(x > 0, np.ones_like(x), np.zeros_like(x))

def center_data(x):
    return x - np.mean(x, axis=0)

def vector_length(x):
    return np.linalg.norm(x)

class SingleLayerPerceptron():
    def fit(self, x, y, theta):
        x, y = x.copy(), y.copy()
        x = center_data(x)
        self.w = np.mean(x[y == 1], axis=0)
        while True:
            w_prime = self.w
            random_idx = np.random.randint(low=0, high=x.shape[0], size=(1, ))[0]
            v = x[random_idx]
            scalar = self.w.T @ v
            # v is from P
            if y[random_idx] == 1:
                if scalar > 0:
                    pass
                else:
                    w_prime = self.w + v
            # v is from N
            else:
                if scalar < 0:
                    pass
                else:
                    w_prime = self.w - v
            if vector_length(self.w - w_prime) < theta:
                self.w = w_prime
                break
            self.w = w_prime

    def predict(self, x):
        x = x.copy()
        x = center_data(x)
        return heavyside(x @ self.w)
```

In [5]:

```
clf = SingleLayerPerceptron()
clf.fit(X_train_s_vs_vv, y_train_s_vs_vv, theta=10**-6)
```

In [6]:

```
y_pred = clf.predict(X_test_s_vs_vv)
y_pred = y_pred.reshape((-1, ))
acc = accuracy(y_test_s_vs_vv, y_pred)
print(f"Accuracy is: {round(acc * 100, 4)}%")
```

```
Accuracy is: 93.3333%
```

(a) What happens if you use the algorithm to seperate Versicolour from Virginica? (Evaluate multiple runs)

In [7]:

```
X_v_vs_v = np.vstack((X_train_versicolor, X_train_virginica))
y_v_vs_v = np.hstack((np.ones((X_train_versicolor.shape[0], )), np.zeros((X_train_v
```

In [8]:

```
num_runs = 10
for i in range(1, num_runs + 1):
    X_train_v_vs_v, X_test_v_vs_v, y_train_v_vs_v, y_test_v_vs_v = train_test_split
    clf = SingleLayerPerceptron()
    clf.fit(X_train_s_vs_vv, y_train_s_vs_vv, theta=10**-6)
    y_pred = clf.predict(X_test_s_vs_vv)
    y_pred = y_pred.reshape((-1, ))
    acc = accuracy(y_test_s_vs_vv, y_pred)
    print(f"{i}. experiment: Accuracy of: {round(acc * 100, 4)}%")
```

```
1. experiment: Accuracy of: 93.3333%
2. experiment: Accuracy of: 93.3333%
3. experiment: Accuracy of: 93.3333%
4. experiment: Accuracy of: 93.3333%
5. experiment: Accuracy of: 100.0%
6. experiment: Accuracy of: 93.3333%
7. experiment: Accuracy of: 100.0%
8. experiment: Accuracy of: 93.3333%
9. experiment: Accuracy of: 93.3333%
10. experiment: Accuracy of: 93.3333%
```

(b) Find a way to solve the problem and obtain the accuracy.

We really don't know which problem should have occured. All runs have a very high and stable accuracy.

Excercise 2. Multilayer-Perceptron (MLP)

Implement a class that builds an MLP with both variable depth D (number of layers) and variable number of neurons ni for each layer i = 1, ..., D. Produce outputs on the ZIP-Dataset

In [9]:

```
class DenseLayer():
    def __init__(self, input_shape, output_shape, activation):
        """
        output_shape is equal to the number of neurons in the layer
        """
        self.w = np.random.uniform(low=-1, high=1, size=(input_shape, output_shape)
        self.b = np.ones((1, output_shape))
        self.activation = activation

    def forward(self, x):
        x_t = np.hstack((np.ones((x.shape[0], 1)), x))
        w_t = np.vstack((self.b, self.w))
        return self.activation(x_t @ w_t)
```

In [10]:

```
class NeuralNetwork():
    def __init__(self, layers):
        self.layers = layers

    def predict(self, x):
        x = x.copy()
        for layer in self.layers:
            x = layer.forward(x)
        return x

path_to_test = 'zip.test'
test_data = np.array(pd.read_csv(path_to_test, sep =' ',header=None))

X_test = test_data[:,1:-1]

layers = []
layers.append(DenseLayer(input_shape=X_test.shape[1], output_shape=64, activation=h
layers.append(DenseLayer(input_shape=64, output_shape=128, activation=heavyside))
layers.append(DenseLayer(input_shape=128, output_shape=64, activation=heavyside))
layers.append(DenseLayer(input_shape=64, output_shape=10, activation=heavyside))

clf = NeuralNetwork(layers)
```

In [11]:

```
y_pred = clf.predict(X_test)
print(y_pred)
```

```
[[0. 0. 1. ... 0. 1. 1.]
 [1. 0. 0. ... 1. 1. 0.]
 [1. 0. 1. ... 1. 1. 0.]
 ...
 [0. 0. 1. ... 1. 1. 1.]
 [1. 0. 0. ... 1. 1. 0.]
 [1. 0. 1. ... 1. 1. 0.]]
```