Freie Universität Berlin

**Prof. R. Rojas**

# Mustererkennung, WS17/18

## Übungsblatt 6

Boyan Hristov, Nedeltscho Petrov

26. November 2017

---

Link zum Git Repository: https://github.com/BoyanH/FU-MachineLearning-17-18/tree/master/Solutions/Homework6

## Perceptron Classifier

## Rückgabe des Programms und damit Score und beste Annäherung

(seed=8 in Initmethode)

```
Smallest error for Iris-virginica vs Iris-setosa: 0.0 mistaken probes
Score Iris-virginica vs Iris-setosa: 1.0
Score Iris-virginica vs Iris-setosa on train data: 1.0
Smallest error for Iris-setosa vs Iris-versicolor: 0.0 mistaken probes
Score Iris-setosa vs Iris-versicolor: 1.0
Smallest error for Iris-virginica vs Iris-versicolor: 2.0 mistaken probes
Score Iris-virginica vs Iris-versicolor: 1.0
Score Iris-virginica vs Iris-versicolor on train data: 0.9666666666666667
Score of LDA Iris-virginica vs Iris-setosa: 1.0
Score of LDA Iris-virginica vs Iris-setosa on train data: 0.9666666666666667
```

Und mit anderem zufällig ausgewälten initialen Gewichtsvektor. (seed=1 in Initmethode)

```
Smallest error for Iris-virginica vs Iris-setosa: 0.0 mistaken probes
Score Iris-virginica vs Iris-setosa: 1.0
Score Iris-virginica vs Iris-setosa on train data: 1.0
Smallest error for Iris-setosa vs Iris-versicolor: 0.0 mistaken probes
Score Iris-setosa vs Iris-versicolor: 1.0
Smallest error for Iris-virginica vs Iris-versicolor: 4.0 mistaken probes
Score Iris-virginica vs Iris-versicolor: 0.9
Score Iris-virginica vs Iris-versicolor on train data: 0.9555555555555556
Score of LDA Iris-virginica vs Iris-setosa: 1.0
Score of LDA Iris-virginica vs Iris-setosa on train data: 0.9666666666666667
```

## Analyse

Wie man von den Ergebnissen sieht, ist auf diesem Datensatz ein Perceptron Klassifikator fast so gut wie LDA. Man kann aber leicht ein "multi-layer network"von Perceptrons bauen mit denen man noch bessere Ergebnisse bekommen sollte. Weiter ist dieses Verfahren auch für önline learning"gut geigent, da man die Gewichte anhand von einzelnen Punkten ändert. D.h. man könnte mit der Zeit von weiteren Beispielen lernen und noch bessere Ergebnisse mit der Zeit bekommen.

Weiter sieht man, dass wir bessere Ergebnisse mit dem Testdatensatz bekommen haben, als mit dem Trainingdatensatz. Also wir lernen nicht mehr so viel äuswendig", das Algorithmus kann besser mit sehr üntypische"Beispiele umgehen.

## Implementierung

## Fit Methode

Wir haben das vorgeschlagene Entwurf aus der Vorlesung etwas geändert. Die Implementierung ist ähnlicher zu der im Buch Ëlements of Statistical Learning". Wir nehmen die Differenz aus berechnete und erwartete Klassifizierung und subtrahieren die von dem bisher berechnete Gewichtsvektor. Das erstetzt einfach die IF-Anweisungen, sonst gibt es keine Unterschied. Wir haben aber gelesen, dass es besser ist, ein Lernfaktor zu benutzen und die Differenz damit zu multiplizieren. So konvergiert man besser, da sonst ein Paar sehr üntypische"Punkte ganz schnell den Gewichtsvektor in falsche Richtung bewegen.

Weiter werden wir maximal 500 Iterationen ausführen, bei denen nicht bessere Ergebnisse rauskommen. So haben wir auch den Fall behandelt, in dem die Daten nicht linear separierbar sind.

Als "bessere"bzw ßchlechtereËrgebnisse haben wir am Anfang den Drehungsvinkel zwischen bisherigen und neuen Gewichtsvektor genommen. Wir haben aber gesehen, dass nur einige "nicht typische"Punkte aus dem Datensatz große Fehler erzeugen und man nicht bessere Ergebnisse bekommt, wenn man die korrigiert. Bei dem Datensatz war es deutlich besser, dafür zu sorgen, dass es wenigere falsche Klassifizierungen gibt, als dass der Gewichtsvektor möglichst nah ist an den falschen Punkten. Wir nehmen an das wird in den meisten Fällen so sein.

```
    def fit(self, X, y):
        t = 0
        least_error = None
        current_error = None
        best_w = self.w
        worse_iterations = 0

        while worse_iterations < 500 and (current_error is None or current_error > 0):  #
    < least_error:
            if least_error is not None and current_error > least_error:
                worse_iterations += 1

            current_error = 0

            for x, yi in zip(X, y):
                error = ((self.predict_single_normalized(x) - yi)/2)
                w_new = self.w - learning_rate*error*x
                # current_error += PerceptronClassifier.get_error(w_new, self.w)
                current_error += abs(error)
                self.w = w_new

            if current_error is not None and (least_error is None or current_error <
    least_error):
                least_error = current_error
                best_w = np.copy(self.w)
```

```
        self.w = best_w / np.linalg.norm(best_w)
        for x, yi in zip(X, y):
            error = (self.predict_single_normalized(x) - yi)
            assert(error == 0 or least_error > 0)
```

## Initialisierung

Wir haben initial als Gewichtsvektor ein beliebigen Vektor aus dem Datensatz genommen. Alternativ sollte man auch den Nullvektor nehmen können. Wir haben auch eine weitere Spalte in den Datensatz eingefügt mit Einsen. Das sollte im Prinzip bessere Ergebnisse liefern, wenn es z.B. deutlich wahrscheinlicher ist, dass ein Datenpunkt zu der einen Klasse gehört (wenn die eine Klasse häufiger vorkommt). Das sollte aber bei dem Datensatz keine große Rolle spielen.

```
    def __init__(self, X, y, class_a, class_b):
        ones = np.ones((len(X), 1), dtype=np.float64)
        X_normalized = np.append(ones, X, axis=1)
        y_normalized = np.vectorize(lambda x: -1 if x == class_a else 1)(y)
        np.random.seed(8)
        self.w = X_normalized[np.random.randint(0, X_normalized.shape[0], 1)][0]
        self.class_a = class_a
        self.class_b = class_b
        self.fit(X_normalized, y_normalized)
```

## Vollständiges Code

PerceptronClassifier.py

```
from Classifier import Classifier
import numpy as np
import math

classes_in_data_set = ['Iris-virginica', 'Iris-setosa', 'Iris-versicolor']
infinity = float('inf')
learning_rate = 0.0001

class PerceptronClassifier(Classifier):
    def __init__(self, X, y, class_a, class_b):
        ones = np.ones((len(X), 1), dtype=np.float64)
        X_normalized = np.append(ones, X, axis=1)
        y_normalized = np.vectorize(lambda x: -1 if x == class_a else 1)(y)
        np.random.seed(8)
        self.w = X_normalized[np.random.randint(0, X_normalized.shape[0], 1)][0]
        self.class_a = class_a
        self.class_b = class_b
        self.fit(X_normalized, y_normalized)

    def fit(self, X, y):
        t = 0
        least_error = None
        current_error = None
        best_w = self.w
        worse_iterations = 0

        while worse_iterations < 500 and (current_error is None or current_error > 0):  #
    < least_error:
            if least_error is not None and current_error > least_error:
                worse_iterations += 1

            current_error = 0
```

```
            # what happens here is really the same as in the lecture
            # just without if statements; if e.g x is positive and we predicted negative
            # predict single would be 0, y would be 1
            # => self.w + learning_rate*x

            # learning rate is something commonly used in this algorithm, in the lecture we learned
            # a simplified version where the learning rate is 1
            for x, yi in zip(X, y):
                error = ((self.predict_single_normalized(x) - yi)/2)
                w_new = self.w - learning_rate*error*x
                # current_error += PerceptronClassifier.get_error(w_new, self.w)
                current_error += abs(error)
                self.w = w_new

            if current_error is not None and (least_error is None or current_error < least_error):
                least_error = current_error
                best_w = np.copy(self.w)

        self.w = best_w / np.linalg.norm(best_w)
        for x, yi in zip(X, y):
            error = (self.predict_single_normalized(x) - yi)
            assert(error == 0 or least_error > 0)

        print('Smallest error for {} vs {}: {} mistaken probes'.format(self.class_a, self.class_b, least_error))

    @staticmethod
    def get_error(w_new, w):
        # err in degrees rotation
        return math.acos(np.clip((w_new / np.linalg.norm(w_new)).dot(w / np.linalg.norm(w)), -1.0, 1.0))

    def project_point(self, x):
        return x.dot(self.w / np.linalg.norm(self.w))

    def predict_single_normalized(self, x):
        return 1 if self.project_point(x) > 0 else -1

    def predict_single(self, x):
        x_normalized = np.append(np.array([1]), x)
        return self.class_a if self.predict_single_normalized(x_normalized) < 0 else self.class_b

    def predict(self, X):
        return list(map(lambda x: self.predict_single(x), X))
```

PerceptronClassifierDemo.py

```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from PerceptronClassifier import PerceptronClassifier, classes_in_data_set
from Parser import get_data_set, extract_classes_from_data_set

# Classes
# Iris-setosa, Iris-versicolour, Iris-virginica

X_train, X_test, y_train, y_test = get_data_set(1)

X_vi_se_train, y_vi_se_train = extract_classes_from_data_set(X_train, y_train, classes_in_data_set[:-1])
X_vi_se_test, y_vi_se_test = extract_classes_from_data_set(X_test, y_test, classes_in_data_set[:-1])
pc_vi_se = PerceptronClassifier(X_vi_se_train, y_vi_se_train, classes_in_data_set[0], classes_in_data_set[1])
score_vi_se = pc_vi_se.score(X_vi_se_test, y_vi_se_test)
```

```
score_vi_se_train = pc_vi_se.score(X_vi_se_train, y_vi_se_train)
print('Score {} vs {}: {}'.format(classes_in_data_set[0], classes_in_data_set[1],
    score_vi_se))
print('Score {} vs {} on train data: {}'.format(classes_in_data_set[0], classes_in_data_set
    [1], score_vi_se_train))


X_ve_se_train, y_ve_se_train = extract_classes_from_data_set(X_train, y_train,
    classes_in_data_set[1:])
X_ve_se_test, y_ve_se_test = extract_classes_from_data_set(X_test, y_test,
    classes_in_data_set[1:])
pc_ve_se = PerceptronClassifier(X_ve_se_train, y_ve_se_train, classes_in_data_set[1],
    classes_in_data_set[2])
score_ve_se = pc_ve_se.score(X_ve_se_test, y_ve_se_test)
print('Score {} vs {}: {}'.format(classes_in_data_set[1], classes_in_data_set[2],
    score_ve_se))

X_vi_ve_train, y_vi_ve_train = extract_classes_from_data_set(X_train, y_train, [
    classes_in_data_set[0],
                                                                classes_in_data_set[2]])
X_vi_ve_test, y_vi_ve_test = extract_classes_from_data_set(X_test, y_test, [
    classes_in_data_set[0],

    classes_in_data_set[2]])
pc_vi_ve = PerceptronClassifier(X_vi_ve_train, y_vi_ve_train, classes_in_data_set[0],
    classes_in_data_set[2])
score_vi_ve = pc_vi_ve.score(X_vi_ve_test, y_vi_ve_test)
score_vi_ve_train = pc_vi_ve.score(X_vi_ve_train, y_vi_ve_train)
print('Score {} vs {}: {}'.format(classes_in_data_set[0], classes_in_data_set[2],
    score_vi_ve))
print('Score {} vs {} on train data: {}'.format(classes_in_data_set[0], classes_in_data_set
    [2], score_vi_ve_train))

clf = LinearDiscriminantAnalysis()
clf.fit(X_vi_ve_train, y_vi_ve_train)
predictions = clf.predict(X_vi_ve_test)
predictions_train = clf.predict(X_vi_ve_train)
score_lda = np.mean(predictions == y_vi_ve_test)
score_lda_train = np.mean(predictions_train == y_vi_ve_train)
print('Score of LDA {} vs {}: {}'.format(classes_in_data_set[0], classes_in_data_set[1],
    score_lda))
print('Score of LDA {} vs {} on train data: {}'.format(classes_in_data_set[0],
    classes_in_data_set[1], score_lda_train))
```

Parser.py

```
import csv
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split


def parse_data():
    file_name = os.path.join(os.path.dirname(__file__), './Dataset/iris.data')
    return pd.read_csv(file_name, header=None).as_matrix()


def get_points_and_labels_from_data(data):
    points = np.array(data[:,:-1], dtype=np.float64)
    labels = data[:,-1]

    return points, labels


def extract_classes_from_data_set(X, y, classes):
    is_from_classes = np.vectorize(lambda y: y in classes)
    filter_arr = is_from_classes(y)
```

```
    return X[filter_arr], y[filter_arr]


def get_data_set(seed):
    data = parse_data()
    X, y = get_points_and_labels_from_data(data)
    # for determined results we use a seed for random_state, so that data is always split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9, test_size
    =0.1,
                                                        random_state=seed)

    return X_train, X_test, y_train, y_test
```

Classifier.py

```
import numpy as np

class Classifier:
    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
```