Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1 по курсу «Нейроинформатика»

Студент: В. В. Бирюков
Преподаватель: И. А. Рожлейс
Группа: М8О-407Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №1

Тема: Персептроны. Процедура обучения Розенблатта.

Цель работы: Исследование свойств персептрона Розенблатта и его применение для решения задачи распознавания образов.

Основные этапы работы:

1. Для первой обучающей выборки построить и обучить сеть, которая будет правильно относить точки к двум классам. Отобразить дискриминантную линию и проверить качество обучения.

2. Изменить обучающее множество так, чтобы классы стали линейно неразделимыми. Проверить возможности обучения по правилу Розенблатта.

3. Для второй обучающей выборки построить и обучить сеть, которая будет правильно относить точки к четырем классам. Отобразить дискриминантную линию и проверить качество обучения.

Вариант: 9

Data	Labels
$\begin{bmatrix} -1.1 & 1.8 & 4.8 & 1.2 & -1.2 & 2.5 \\ -4.3 & -1 & -1 & -3.5 & -3.4 & 3.7 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} 4.6 & -1 & -0.3 & -1.1 & 0.5 & 4.9 & 0.3 & -3.9 \\ 1.7 & 4.3 & -2.7 & 2 & 2.5 & 4.6 & 4.6 & -4.5 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

1 Исходный код

Лабораторная работа № 1

Вариант: 9

[1]:
```
import numpy as np
from tensorflow import keras
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import from_levels_and_colors, LinearSegmentedColormap
from matplotlib.markers import MarkerStyle
```

[2]:
```
%matplotlib inline
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg', 'pdf')
```

[3]:
```
COLORS = ['red', 'green', 'purple', 'yellow']

def plot_history(h, *metrics):
    for metric in metrics:
        print(f"{metric}: {h.history[metric][-1]:.4f}")
    figure = plt.figure(figsize=(5 * len(metrics), 3))
    for i, metric in enumerate(metrics, 1):
        ax = figure.add_subplot(1, len(metrics), i)
        ax.xaxis.get_major_locator().set_params(integer=True)
        plt.title(metric)
        plt.plot(h.history[metric], '-')
    plt.show()

def plot_line(a, b, c):
    xlim, ylim = plt.xlim(), plt.ylim()
    plt.axline((-c / a, 0), slope=-a/b)
    plt.xlim(xlim)
    plt.ylim(ylim)
```

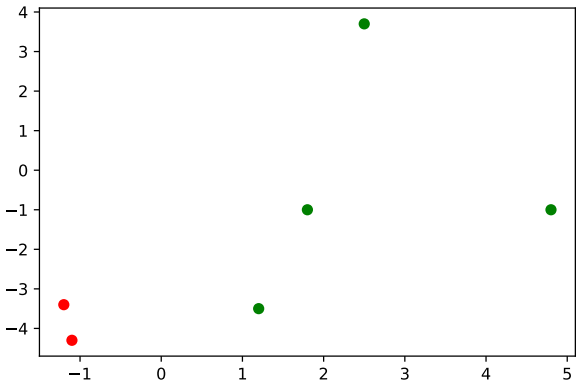
Классификация объектов двух классов

[4]:
```
data1 = np.array([(-1.1, -4.3), (1.8, -1), (4.8, -1), (1.2, -3.5), (-1.2, -3.4), (2.
↪5, 3.7)])
labels1 = np.array([0, 1, 1, 1, 0, 1])
```

[5]:
```
def plot_two_classes(data, labels, test=False):
    colors = [COLORS[i] for i in labels]
    plt.scatter(data[:, 0], data[:, 1], c=colors, marker=MarkerStyle('o', 'none' if␣
↪test else 'full'))
```

```
def plot_two_classes_decision_regions(model, n=100):
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()

    x = np.linspace(x_min, x_max, n)
    y = np.linspace(y_min, y_max, n)

    xv, yv = np.meshgrid(x, y)
    z = model.predict(np.c_[xv.ravel(), yv.ravel()]).reshape(n, n)

    cmap = LinearSegmentedColormap.from_list('cmap', COLORS[:2])

    plt.contourf(x, y, z, alpha=0.4, cmap=cmap, levels=10)
```

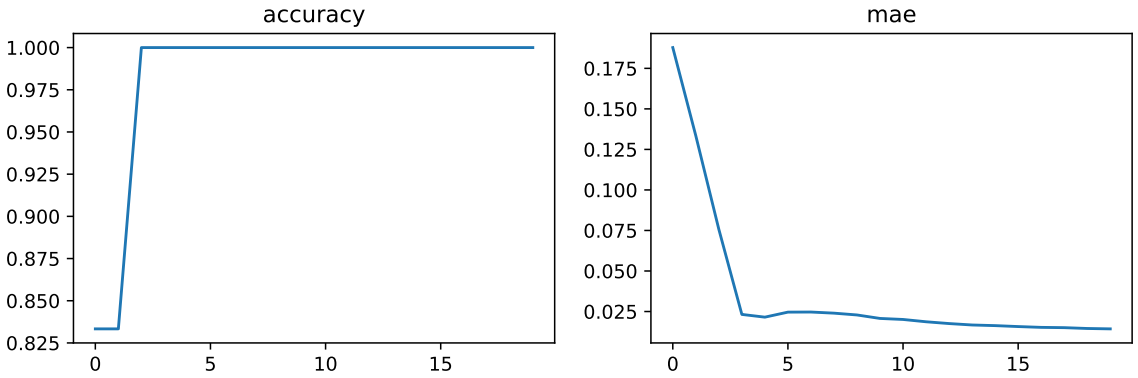
[6]:
```
plot_two_classes(data1, labels1)
plt.show()
```



[7]:
```
model1 = keras.models.Sequential([
    keras.layers.Dense(1, input_dim=2, activation='sigmoid')
])

model1.compile(keras.optimizers.Adam(0.1), 'mse', ['mae', 'accuracy'])

hist1 = model1.fit(data1, labels1, batch_size=1, epochs=20, verbose=0)
```

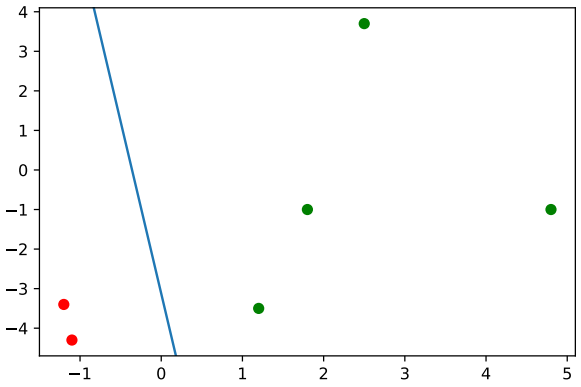
[8]:
```
plot_history(hist1, 'accuracy', 'mae')
```

```
accuracy: 1.0000
mae: 0.0143
```

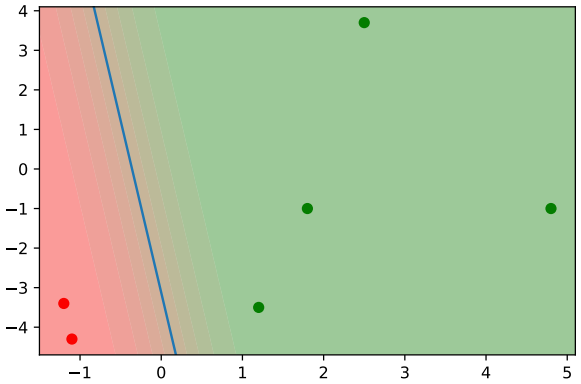


[9]:
```
weights1 = model1.get_weights()
```

[10]:
```
plot_two_classes(data1, labels1)
plot_line(weights1[0][0][0], weights1[0][1][0], weights1[1][0])
plt.show()
```



[11]:
```
plot_two_classes(data1, labels1)
plot_line(weights1[0][0][0], weights1[0][1][0], weights1[1][0])
plot_two_classes_decision_regions(model1)
plt.show()
```



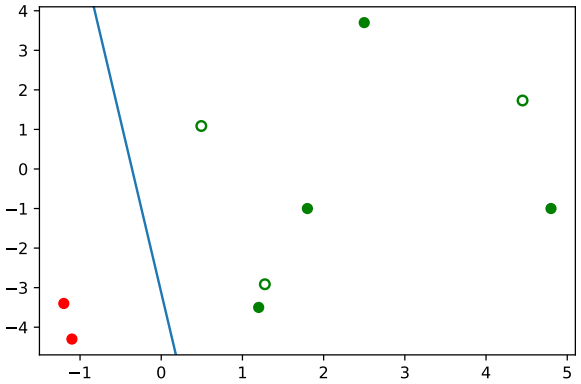
Тестирование

[12]:
```
test_size = 3

max_x, max_y = np.max(data1, axis=0)
min_x, min_y = np.min(data1, axis=0)

test_x = (max_x - min_x) * np.random.random(test_size) + min_x
test_y = (max_y - min_y) * np.random.random(test_size) + min_y

test1 = np.stack([test_x, test_y], axis=-1)
```

[13]:
```
test_labels1 = model1.predict(test1)
test_labels1
```

[13]:
```
array([[0.99999964],
       [0.9791605 ],
       [0.94757974]], dtype=float32)
```

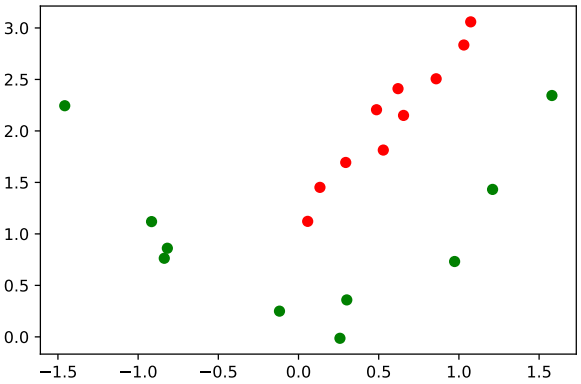
[14]:
```
plot_two_classes(data1, labels1)
plot_two_classes(test1, (test_labels1.flat >= 0.5).astype(int), test=True)
plot_line(weights1[0][0][0], weights1[0][1][0], weights1[1][0])
```



Линейно неразделимый случай

[15]:
```
x = np.linspace(-1.5, 1.3, 10) + (0.6 * np.random.random(10) - 0.3)
y = (x ** 2) + (0.6 * np.random.random(10) - 0.3)
data_nonlin = np.stack((x, y), axis=-1)
labels_nonlin = np.ones((10, ), dtype=int)

x = np.linspace(0, 1, 10) + (0.4 * np.random.random(10) - 0.2)
y = (2 * x + 1) + (0.6 * np.random.random(10) - 0.3)
data_nonlin = np.append(data_nonlin, np.stack((x, y), axis=-1), axis=0)
labels_nonlin = np.append(labels_nonlin, np.zeros((10, ), dtype=int))
```

[16]:
```
plot_two_classes(data_nonlin, labels_nonlin)
```

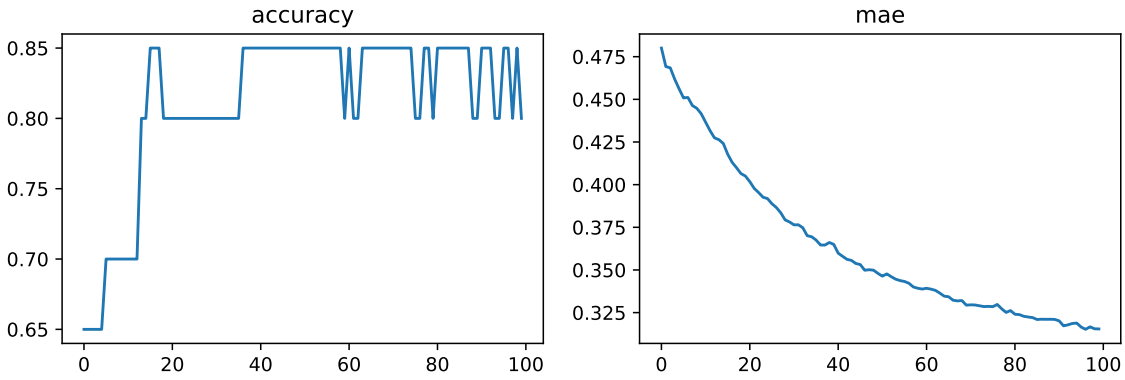


[17]:
```
model1_2 = keras.models.Sequential([
    keras.layers.Dense(1, input_dim=2, activation='sigmoid')
])
```

```
model1_2.compile(keras.optimizers.Adam(0.01), 'mse', ['mae', 'accuracy'])

hist1_2 = model1_2.fit(data_nonlin, labels_nonlin, batch_size=1, epochs=100,␣
↪verbose=0)
```

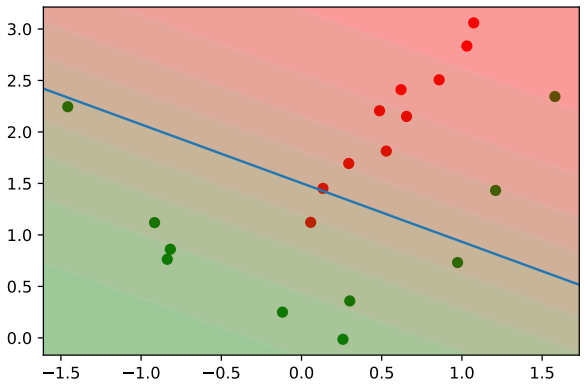
[18]:
```
plot_history(hist1_2, 'accuracy', 'mae')
```

```
accuracy: 0.8000
mae: 0.3154
```



[19]:
```
weights1_2 = model1_2.get_weights()
```

[20]:
```
plot_two_classes(data_nonlin, labels_nonlin)
plot_line(weights1_2[0][0][0], weights1_2[0][1][0], weights1_2[1][0])
plot_two_classes_decision_regions(model1_2)
plt.show()
```

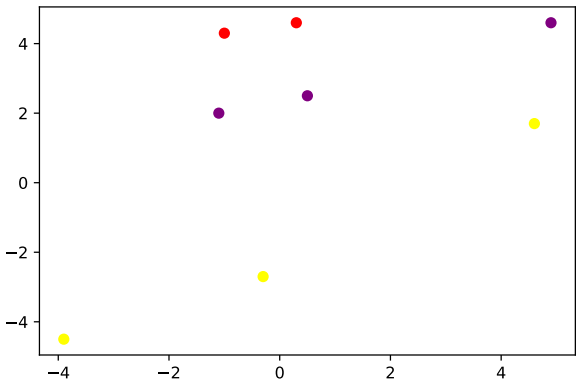


Классификация объектов четырех классов

[21]:
```
data2 = np.array([(4.6, 1.7), (-1, 4.3), (-0.3, -2.7), (-1.1, 2), (0.5, 2.5), (4.9,␣
↪4.6), (0.3, 4.6), (-3.9, -4.5)])
labels2 = np.array([(1, 1), (0, 0), (1, 1), (1, 0), (1, 0), (1, 0), (0, 0), (1, 1)])
```

[22]:
```
def plot_four_classes(data, labels, test=False):
    colors = [COLORS[i[0] * 2 + i[1]] for i in labels]
    plt.scatter(data[:, 0], data[:, 1], c=colors, marker=MarkerStyle('o', 'none' if␣
↪test else 'full'))

def accuracy_bin_encoded(labels, pred):
    """Calculates how often predictions match binary encoded labels"""
    correct = 0
    threshold = tf.constant([0.5])
    for i in range(len(labels)):
        if tf.experimental.numpy.all(tf.equal(tf.greater_equal(pred[i], threshold),␣
↪tf.cast(labels[i], tf.bool))):
            correct += 1
    return correct / len(labels)
```

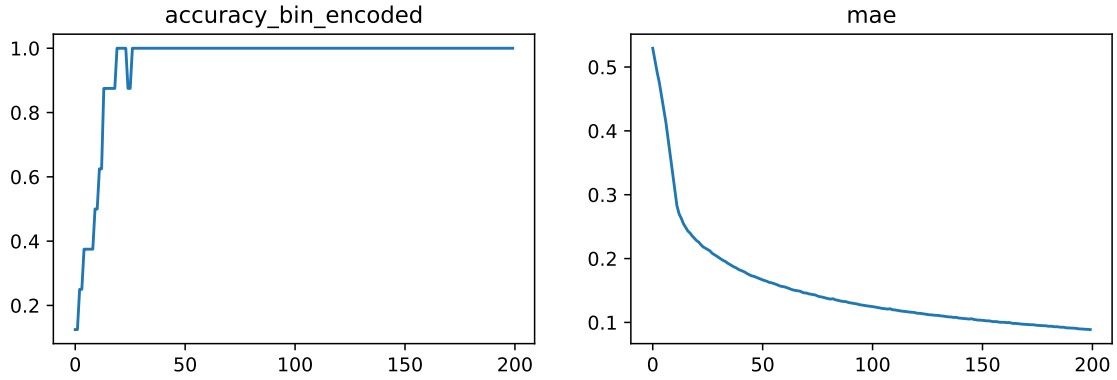
[23]:
```
plot_four_classes(data2, labels2)
plt.show()
```



[24]:
```
model2 = keras.models.Sequential([
    keras.layers.Dense(2, input_dim=2, activation='sigmoid')
])

model2.compile(keras.optimizers.Adam(0.01), 'mse', ['mae', accuracy_bin_encoded])

hist2 = model2.fit(data2, labels2, batch_size=1, epochs=200, verbose=0)
```

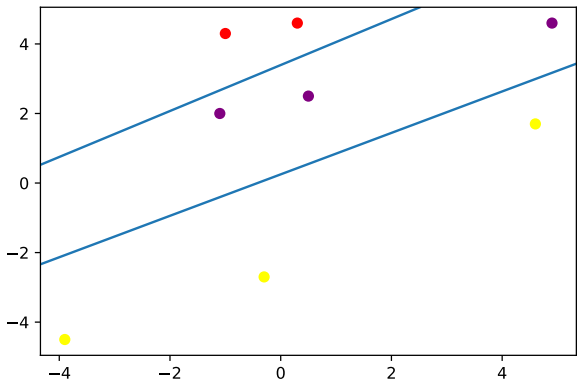
[25]:
```
plot_history(hist2, 'accuracy_bin_encoded', 'mae')
```

```
accuracy_bin_encoded: 1.0000
mae: 0.0888
```



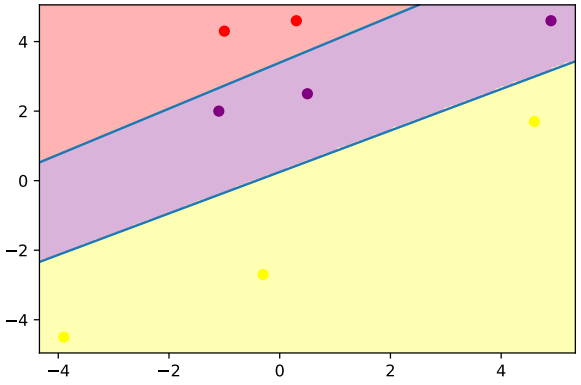
[26]:
```
weights2 = model2.get_weights()
```

[27]:
```
plot_four_classes(data2, labels2)
plot_line(weights2[0][0][0], weights2[0][1][0], weights2[1][0])
plot_line(weights2[0][0][1], weights2[0][1][1], weights2[1][1])
plt.show()
```



[28]:
```
plot_four_classes(data2, labels2)
plot_line(weights2[0][0][0], weights2[0][1][0], weights2[1][0])
plot_line(weights2[0][0][1], weights2[0][1][1], weights2[1][1])

x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()

n = 200
```

```
x = np.linspace(x_min, x_max, n)
y = np.linspace(y_min, y_max, n)

xv, yv = np.meshgrid(x, y)
z = (model2.predict(np.c_[xv.ravel(), yv.ravel()]) > 0.5).astype(int)
z = np.apply_along_axis(lambda x: 2 * x[0] + x[1], 1, z).reshape(n, n)

cmap, norm = from_levels_and_colors([-0.5, 0.5, 1.5, 2.5, 3.5], COLORS)

plt.imshow(z, alpha=0.3, extent=(x_min, x_max, y_min, y_max), aspect='auto',␣
 ↪origin='lower', cmap=cmap, norm=norm)
plt.show()
```



Тестирование

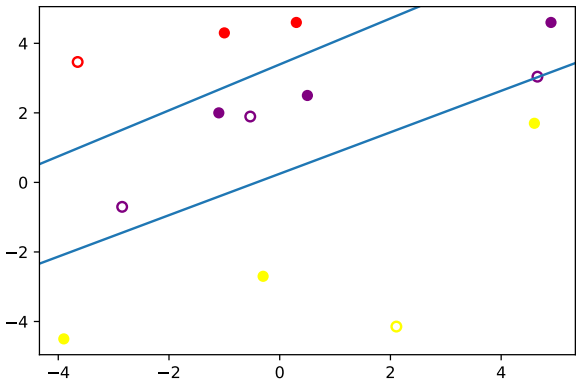
[41]:
```
test_size = 5

max_x, max_y = np.max(data2, axis=0)
min_x, min_y = np.min(data2, axis=0)

test_x = (max_x - min_x) * np.random.random(test_size) + min_x
test_y = (max_y - min_y) * np.random.random(test_size) + min_y

test2 = np.stack([test_x, test_y], axis=-1)
```

[42]:
```
test_labels2 = model2.predict(test2)
test_labels2
```

[42]:
```
array([[7.5612509e-01, 3.5026193e-02],
       [9.9985087e-01, 9.9992895e-01],
       [7.9546034e-02, 1.1077945e-04],
       [9.6717298e-01, 4.9302673e-01],
```

```
       [8.9885688e-01, 2.2142953e-01]], dtype=float32)
```

[43]:
```
plot_four_classes(data2, labels2)
plot_four_classes(test2, (test_labels2 >= 0.5).astype(int), test=True)
plot_line(weights2[0][0][0], weights2[0][1][0], weights2[1][0])
plot_line(weights2[0][0][1], weights2[0][1][1], weights2[1][1])
```



2 Выводы

В ходе выполнения первой лабораторной работы я освежил свои знания про перцептрон Розенблата. Применительно к задачам классификации единичный перцептрон показывает хорошие результаты только в линейно разделимом случае, что накладывает сильные ограничения на входные данные. Однако, он легко обобщается на случай многоклассовой классификации.