ps2-5-solve

May 14, 2021

1 (a)

1.1 i

for $\theta^{(i)}$:

$$\theta^{(i)} = \sum_{j=1}^{i} \alpha\big(y^{(j)} - h_{\theta^{(j-1)}}(\phi(x^{(j)}))\big)\phi(x^{(j)})$$

for $\theta^{(0)}$:

$$\theta^{(0)} = \vec{0}$$

1.2 ii

for $h_{\theta^{(i)}}(x^{(i+1)})$:

$$\begin{aligned}
h_{\theta^{(i)}}(x^{(i+1)}) &= g((\theta^{(i)})^T\phi(x^{(i+1)})) \\
&= sign\big(\sum_{j=1}^{i} \alpha\big(y^{(j)} - h_{\theta^{(j-1)}}(\phi(x^{(j)}))\big)\phi(x^{(j)})^T\phi(x^{(i+1)})\big) \\
&= sign\big(\sum_{j=1}^{i} \alpha\big(y^{(j)} - h_{\theta^{(j-1)}}(\phi(x^{(j)}))\big)\langle\phi(x^{(j)}), \phi(x^{(i+1)})\rangle\big) \\
&= sign\big(\sum_{j=1}^{i} \alpha\big(y^{(j)} - h_{\theta^{(j-1)}}(\phi(x^{(j)}))\big)K(x^{(j)}, x^{(i+1)})\big)
\end{aligned}$$

for $h_{\theta^{(0)}}(x^{(1)})$:

$$\begin{aligned}
h_{\theta^{(0)}}(x^{(1)}) &= g((\theta^{(0)})^T\phi(x^{(1)})) \\
&= g(0) \\
&= 1
\end{aligned}$$

1.3 iii

$$
\begin{aligned}
\theta^{(i+1)} &:= \theta^{(i)} + \alpha\big(y^{(i+1)} - h_{\theta^{(i)}}(\phi(x^{(i+1)}))\big)\phi(x^{(i+1)}) \\
&= \sum_{j=1}^{i+1} \alpha\big(y^{(j)} - h_{\theta^{(j-1)}}(\phi(x^{(j)}))\big)\phi(x^{(j)}) \\
&= \sum_{j=1}^{i+1} \alpha\Big(y^{(j)} - sign\big(\sum_{j=1}^{i} \alpha h_{\theta^{(j-1)}}(\phi(x^{(j)}))K(x^{(j)}, x^{(i+1)})\big)\Big)\phi(x^{(j)})
\end{aligned}
$$

let $\beta_i = \alpha\big(y^{(i)} - h_{\theta^{(i-1)}}(\phi(x^{(i)}))\big)$, then

$$
\begin{aligned}
\beta_{i+1} &:= \alpha\big(y^{(i+1)} - sign\big(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\big)\big) \\
\beta_1 &:= \alpha(y^{(1)} - 1) \\
h_{\theta^{(i)}}(x^{(i+1)}) &:= sign\big(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\big)
\end{aligned}
$$

2 (b)

code implements in `src/p05_percept.py`

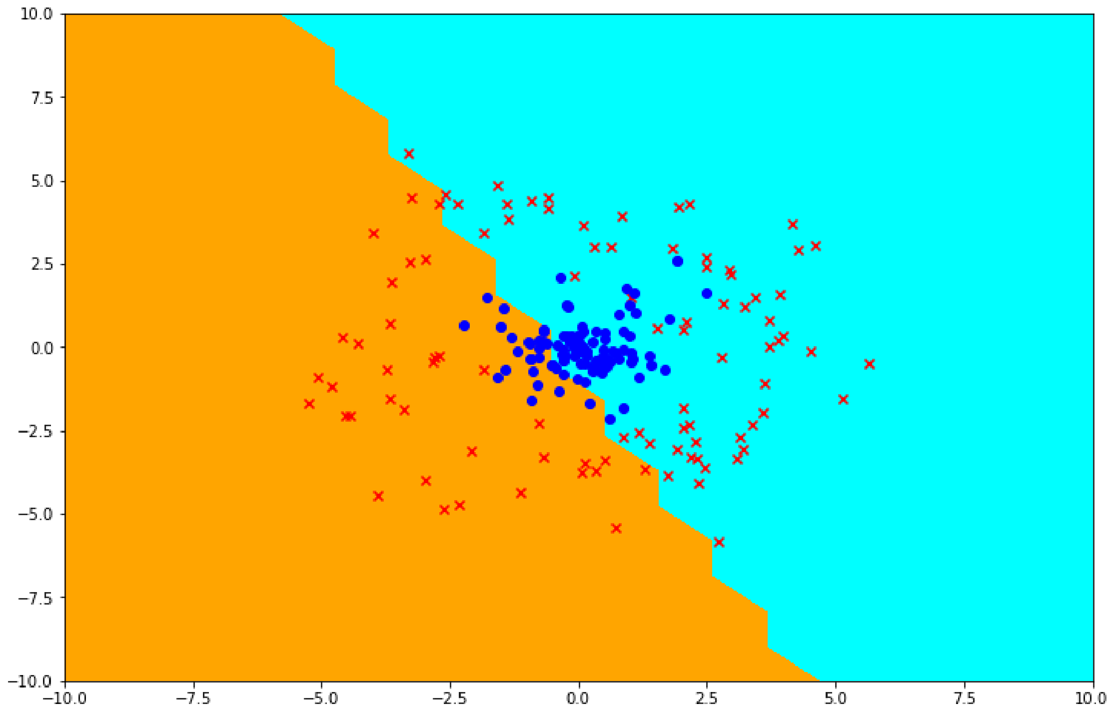
3 (c)

[1]:
```python
import math
import numpy as np
import matplotlib.pyplot as plt

from src import util
from src.p05_percept import initial_state, predict, \
                            update_state
```

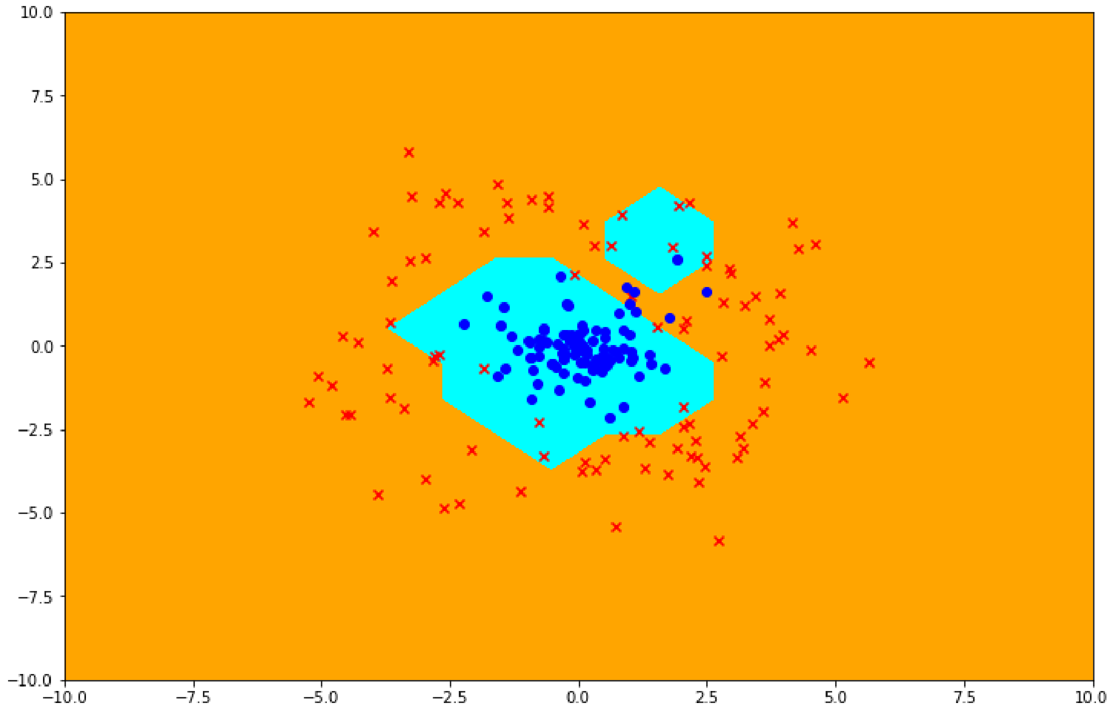
[2]:
```python
def dot_kernel(a, b):
    return np.dot(a, b)

def rbf_kernel(a, b, sigma=1):
    distance = (a - b).dot(a - b)
    scaled_distance = -distance / (2 * (sigma) ** 2)
    return math.exp(scaled_distance)

def train_perceptron(kernel_name, kernel, learning_rate):
    """
    same as train_perceptron in p05_percept.py
    """
```

```python
    train_x, train_y = util.load_csv('data/ds5_train.csv')

    state = initial_state()

    for x_i, y_i in zip(train_x, train_y):
        update_state(state, kernel, learning_rate, x_i, y_i)

    test_x, test_y = util.load_csv('data/ds5_train.csv')

    plt.figure(figsize=(12, 8))
    util.plot_contour(lambda a: predict(state, kernel, a))
    util.plot_points(test_x, test_y)
```

[3]:
```python
train_perceptron('dot', dot_kernel, 0.5)
```



[4]:
```python
train_perceptron('rbf', rbf_kernel, 0.5)
```



Dot kernel performs badly.

In fact, the dot kernel doesn't map x to a high dimension space.

$$K_{dot}(x,z) = x^T z$$
$$\phi(x)^T \phi(z) = x^T z$$
$$\phi(x) = x$$

So in this case, our algorithm with dot kernel is only separate 2-dimention space linearly.

```
[ ]:
```