Experiment 2 Linear Regression

Perceptron

- Given a set of n data points and their corresponding responses $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$
- we try to find a linear function $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{w^T x} + b$ to optimize
  - $\min\limits_{\{\boldsymbol{w},b\}} \max \frac{1}{2n} \sum\limits_{i=1}^{n} \left(\boldsymbol{w}^T \boldsymbol{x}_i + b - y_i\right)^2 + \alpha \left(\boldsymbol{w}^T \boldsymbol{w} - 1\right)$
  - Update the weights:
    - $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + (y_i - p_i)\, \boldsymbol{x}_i, b^{t+1} = b^t + (y_i - p_i)$
- Let $\boldsymbol{z} = \begin{pmatrix} \boldsymbol{W} \\ b \end{pmatrix}, \boldsymbol{y} = (y_1, \ldots, y_n)^T, \boldsymbol{A} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{1})^T$, where **1** is a vector with all ones. Hence
  - $\sum_{i=1}^n \left(\boldsymbol{w}^T \boldsymbol{x}_i + b - y_i\right)^2 = \|\boldsymbol{Az} - \boldsymbol{y}\|_2^2, \boldsymbol{w}^T \boldsymbol{w} - 1 = \boldsymbol{z}^T \boldsymbol{z} - b^2 - 1$

Experiment

- Download a regression dataset from UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets.php).
- 使用的数据集描述如下

> **Data Set Information:**
>
> These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
>
> I think that the initial data set had around 30 variables, but for some reason I only have the 13 dimensional version. I had a list of what the 30 or so variables were, but a.) I lost it, and b.), I would not know which 13 variables are included in the set.
>
> The attributes are (dontated by Riccardo Leardi, riclea **'@'** anchem.unige.it)
> 1) Alcohol
> 2) Malic acid
> 3) Ash
> 4) Alcalinity of ash
> 5) Magnesium
> 6) Total phenols
> 7) Flavanoids
> 8) Nonflavanoid phenols
> 9) Proanthocyanins
> 10)Color intensity
> 11)Hue
> 12)OD280/OD315 of diluted wines
> 13)Proline
>
> In a classification context, this is a well posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging.

- 对数据进行处理

```
import pandas as pd
import random
import numpy as np
import matplotlib.pyplot as plt

N = 11

# 读取数据
data_pd = pd.read_csv('winequality-red.csv', header=0, sep=';')
data = []
# 把输出分割出来
for _, row in data_pd.iterrows():
    data.append([[row['fixed acidity'], row['volatile acidity'], row['citric
acid'],
                 row['residual sugar'], row['chlorides'], row['free sulfur
dioxide'],
                 row['total sulfur dioxide'], row['density'], row['pH'],
                 row['sulphates'], row['alcohol']],row['quality']])


# 打乱顺序进行训练和测试
random.shuffle(data)

# 划分训练数据和测试数据
count = len(data)
ratio = 0.8
split_point = int(count * ratio)
tran_data = data[:split_point]
test_data = data[split_point:]
print('There {} datas in total, {} datas used for train, {} used for
test'.format(count, split_point, count - split_point))

# There 1599 datas in total, 1279 datas used for train, 320 used for test
```

- 样本数据的存储结构为 $data_x$: $[[x_1, x_2, \ldots], label]$
- 然后实现LinearRegression，将其封装成类
  - 首先对预测参数进行初始化，设置学习率
    - ```
      def __init__(self):
          super(LinearRegression, self).__init__()
          # 初始化参数
          self.dim = N
          self.w = [0 for i in range(self.dim)]
          self.b = 0
          self.learningRate = 0.00005
          self.alpha = 1
      ```
  - 然后实现正向传播和反向优化
    - Calculate the activation output:
      - $p_i = f\left(\boldsymbol{w}^t \cdot \boldsymbol{x}_i + b\right)$
    - Update the weights:
      - $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \left(y_i - p_i\right)\boldsymbol{x}_i, b^{t+1} = b^t + \left(y_i - p_i\right)$
      ```

- Compute error
  - $\min\limits_{\{\boldsymbol{w},b\}} \max \frac{1}{2n} \sum\limits_{i=1}^{n} \left(\boldsymbol{w}^T \boldsymbol{x}_i + b - y_i\right)^2 + \alpha \left(\boldsymbol{w}^T \boldsymbol{w} - 1\right)$
- 根据上述公式，进行代码实现
- ```
  def response(self, x):
      """计算预测结果：求和，激活"""
      y = sum([i * j for i, j in zip(self.w, x)]) + self.b
      if y >= 0:
          return 1
      else:
          return 0

  def updateWeights(self, x, iterError):
      """
          更新参数权重
          w(t+1) = w(t) + (yi - pi) * xi
          b(t+1) = b(t) + (yi - pi)
      """
      # self.w += self.learningRate * iterError * int(x)
      self.w = [i + self.learningRate * iterError * j for i, j in
  zip(self.w, x)]
      self.b += self.learningRate * iterError
  ```

- 在网络训练部分，对每一个训练样本，进行预测，如果预测错误，则进行权重更新。直到本次epoch，所有预测结果都准确
  - ```
    def response(self, x):
        """
        计算预测结果：求和，激活
        f(x;w,b)=w^T x+b
        """
        y = sum([i * j for i, j in zip(self.w, x)]) + self.b
        return y

    def updateWeights(self, x, iterError):
        """
        更新参数权重
        w(t+1) = w(t) + (yi - pi) * xi
        b(t+1) = b(t) + (yi - pi)
        """
        # self.w += self.learningRate * iterError * int(x)
        self.w = [ i + self.learningRate * iterError * j for i, j in
    zip(self.w, x)]
        self.b += self.learningRate * iterError
        self.alpha -= self.learningRate * iterError

    def computeError(self, x, y, r):
        """
        计算损失
        min_{w,b}   (max)_α  1/2n Σ_(i=1)^n ((w^T x_i+b−y_i))^2 +α(w^T
    w−1).
        Let z=((w b)), y=((y_1,…,y_n))^T, A=((x_1,…,x_n,1))^T,
        where 1 is a vector with all ones. Hence,
        Σ_(i=1)^n ((w^T x_i+b−y_i))^2 = ||Az−y||_2^2, w^T w−1=z^T
    z−b^2−1.
        """
    ```

```
    z = self.w + [self.b]
    A = x + [1]
    error1 = error2 = 0
    error1 = 1/10 * (sum([i * j for i, j in zip(A, z)]) - y) ** 2
    error2 = sum([i * j for i, j in zip(z, z)]) - self.b * self.b -
1
    return error1 + self.alpha * error2
```

- ```
  # 网络训练
  Net = LinearRegression()
  Net.train(tran_data)

  # Epoch 1980 finished, globalError is 72.82630575237867
  # Epoch 1981 finished, globalError is 71.77918532208149
  # Epoch 1982 finished, globalError is 70.54836659086041
  # Epoch 1983 finished, globalError is 68.97983245871387
  # end training
  ```

- ```
  # 可视化
  print('预设分割线：y =',w,'x +',b)
  print('预测分割线：', p.w[0],'* x + ', p.w[1],'* y =', - p.b)

  for data in data_x:
      if data[1] == 0:
          plt.plot(data[0][0],data[0][1],'g^')
      else:
          plt.plot(data[0][0],data[0][1],'ro')

  plt.plot([-7,7],[-7*w+b,7*w+b]) # 预设分割线
  plt.plot([-7,7],[-(-7.0*p.w[0]+p.b)/p.w[1],-
  (7.0*p.w[0]+p.b)/p.w[1]]) # 预测分割线
  plt.grid(True)
  ```

- 最后进行测试

  - ```
    # 测试部分
    total_error = 0
    for data in test_data:
        r = p.response(data[0])
        total_error += abs(data[1] - r)
    print(f'test data number: {len(test_data)} , total test loss is
    {total_error}')

    # test data number: 320 , total test loss is 155.0047567921725
    ```
    ```