Text Classification using Binary Perceptron Algorithm

201374125

Explain the Perceptron algorithm for the binary classification case, providing its pseudo code.

The **perceptron** algorithm is a supervised learning technique for binary classifiers. The algorithm maps a vector $\mathbf{x}$ to an output value $f(\mathbf{x})$ (a single value), under the conditions:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\mathbf{w}$ is a vector of weights, and b is the bias term.

For a binary classification case with a training vector $\mathbf{x}$ with two input vectors $\mathbf{x}_1, \mathbf{x}_2$ with labels $y_i \in \{-1, 1\}$, first the weight vector would be initialised with weights, one for each input, each taking a value of zero; $w_d(w_1 = w_2 = 0)$. Bias is also initialised as zero; $(b = 0)$.

The activation function may be computed which takes the dot product of the vectors $w = [w_0, w_1]$ and $x = [x_0, x_1]$, plus the bias;

$$a = \sum_{i=0}^{D} w_d x_d + b$$

If the output of $ya < 0$ then the function will update. As all weights were first computed as zero, the result of $w_1x_1 + w_2x_2 + b = 0$. This means that $a = 0$ and $ya = 0$ so the weight vector and bias must be updated. The weight vector is updated by adding the label y_i multiplied by the associated input vector x_d to the origin input vector value, represented; $w_d \leftarrow w_d + yx_d$, these updated training weights are then used to retrieve the next value of ya until $ya > 0$.

Algorithm 1 PERCEPTRONTRAIN(**D**, *MaxIter*)

```
1: $w_d \leftarrow o$, for all $d = 1 \ldots D$
2: $b \leftarrow o$
3: for $iter = 1 \ldots MaxIter$ do
4:     for all $(x,y) \in \mathbf{D}$ do
5:         $a \leftarrow \sum_{d=1}^{D} w_d x_d + b$
6:         if $ya \leq 0$ then
7:             $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$
8:             $b \leftarrow b + y$
   return $w_0, w_1, \ldots, w_D, b$
```

Algorithm 2 PerceptronTest($(w_0, w_1, \ldots, w_D, b, \hat{x})$)

```
1: a ← $\sum_{d=1}^{D} w_d \hat{x}_d + b$ return SIGN($a$)
```

Prove that for a linearly separable dataset the perceptron algorithm will converge.

For each data point x, for $\|\mathbf{x}\| < R$ where R is a constant number, $\gamma = (\theta^*)^T \mathbf{x_c}$, where $\mathbf{x}_c$ is a point closest to the linear separate hyperplane. Mathematically $\frac{\gamma}{\|\theta^*\|^2}$ is the distance d of the closest data point to the linear separate hyperplane, and the number of steps is bounded by $\frac{R^2}{d^2}$, i.e. a finite number of steps.

In written terms, each time the perceptron makes a mistake, the weight vector will move to decrease the squared distance from every vector in the generously feasible region. The squared distance decreases by at least the squared length of the input vector, and so after a number of mistakes, the weight vector will like in the feasible region, if it exists.

Implement a binary perceptron

Implemented in python code:

```python
import numpy as np
import random
import matplotlib.pyplot as plt
from typing import List

def get_data():
    train_data = np.genfromtxt('./data/train.data', delimiter=',', dtype='O')

    # class 1 vs class 2
    train_1 = train_data
    train_1[:, 4][train_1[:, 4] == b'class-1'] = -1
    train_1[:, 4][train_1[:, 4] == b'class-2'] = 1
    train_1 = train_1[train_1[:, 4] != b'class-3']
    train_1 = train_1.astype(float)

    train_2 = train_data
    train_2[:, 4][train_2[:, 4] == b'class-2'] = -1
    train_2[:, 4][train_2[:, 4] == b'class-3'] = 1
    train_2 = train_2[train_2[:, 4] != b'class-1']
    train_2 = train_2.astype(float)

    train_3 = train_data
    train_3[:, 4][train_3[:, 4] == b'class-1'] = -1
    train_3[:, 4][train_3[:, 4] == b'class-3'] = 1
    train_3 = train_3[train_3[:, 4] != b'class-2']
    train_3 = train_3.astype(float)
    return train_1, train_2, train_3
```

```
train_1, train_2, train_3 = get_data()



def perceptron(data, num_iter):
    features = data[:, :-1]
    labels = data[:, -1]

    # init weights as zero
    w = np.zeros(shape=(1, features.shape[1]+1))

    misclassified_ = []

    for _ in range(num_iter):
        misclassified = 0
        for x, label in zip(features, labels):
            x = np.insert(x, 0, 1)
            y = np.dot(w, x.transpose())
            target = 1.0 if (y > 0) else 0.0

            delta = (label - target)

            if(delta):
                misclassified += 1
                w += (delta*x)

        misclassified_.append(misclassified)
    return (w, misclassified_)


num_iter = 10
w, misclassified_ = perceptron(train_3, num_iter)
epochs = np.arange(1, num_iter+1)
plt.plot(epochs, misclassified_)
plt.xlabel('iterations')
plt.ylabel('misclassified')
plt.show()
```