ML hw2 Report B03202047 物理三 王昱翔

1 Logistic regression function

1.1 Preprocessing and Hyperparameters

- Feature Scaling

  I normalized the features, and the results were really so awful that I can know the test score would be very bad without submitting it to Kaggle. Therefore, I didn't do feature scaling on my features.

- Delete Features

# of deleted features	Public test score
16	0.91333
5	0.93000
1	0.93333
0	0.93333

Performance on different number of features

I have deleted the feature whose zeros are more than {80%,90%,95%} in datasets. The number of deleted features are {16,5,1}, however, none of them improve the public test score. So I also didn't delete any features on my datasets.

- Hyperparameters

Learning rate	1e-1
Epsilon*	1e-8
Beta1*	0.9
Beta2*	0.999
iteration	20000
W dim	(57, 1)

Hyper parameters

*: Hyperparameters in Adam optimizer

1.2 Training

- Objective function and the Loss function code

"

```
def sigmoid(self, z):
```

```
            return 1/(1+np.exp(-z/100))
    def cross_entropy(self):
        z = np.dot(self.x, self.w) + self.b
        loss = 0
        for i in range(self.y.shape[0]):
            if self.y[i][0] != self.sigmoid(z)[i][0]:
                loss += (-(self.y[i][0]*np.log(self.sigmoid(z)[i][0])
                + (1-self.y[i][0])*np.log(1-self.sigmoid(z)[i][0])))
        return loss'''
```

- Optimizer

  I use Adam as my optimizer, the following pseudo code is showed above

```
'''
loss     <-  dot(y - dot(w, x) + b)^2 + λw^2
grad_w <-  -2*dot(y – dot(w, x)) * (-x) + 2λw
grad_b <-   -2*dot(y – dot(w, x))
#Init:
 m_0 <- 0
 v_0 <- 0
 t    <- 0
#update
 t   <- t + 1
 lr_t <- learning_rate * sqrt (1 – beta2^t) / (1-beta1^t)
 m_t <- beta1 * m_{t-1} + (1 - beta1) * grad_w
  v_t <- beta2 * v_{t-1} + (1 - beta2) * grad_w * grad_w
 m_t <- beta1 * m_{t-1} + (1 - beta1) * grad_b
  v_t <- beta2 * v_{t-1} + (1 - beta2) * grad_b * grad_b
  w <- w - lr_t * m_t / (sqrt(v_t) + epsilon)'''
```

1.3 Public Test score and Discussion

- Different lambda values

λ value	Public Test Score
1	0.92000
1e-3	0.93000
1e-5	0.93333

0	0.93333

From the public test score, we can see if λ is bigger, the score is lower, which is out of my expectation. I think maybe the test datasets is similar to training set, so it didn't need regularization to avoid overfitting.

2 Support Vector Machine (linear SVM)

2.1 Preprocessing and Hyperparameters

Learning rate	1e-3
Epsilon, Beta1, Beta2, W dim	Same as Logistic
C ($=\frac{1}{\lambda}$)	1000

Hyperparameters of Linear SVM

Because the output of linear SVM is -1 or 1, therefore, I change all the 0 labels in datasetes to -1.

2.2 Training

- Objective function and the Loss function code

```
"def loss(self):
    z = np.dot(self.x, self.w) + self.b
    loss = 0
    for i in range(self.y.shape[0]):
        loss += self.C*max(0, 1-self.y[i]*z[i])
    loss += (1/2)*sum((self.w)**2)
    return loss
def svm_func(self):
    z = np.dot(self.x, self.w) + self.b
    return z
"
```

- Optimizer and output

  I use Adam as optimizer in Linear SVM. For the output, if z >= 0, then output 1, else output zero.

2.3 Public Test score and Discussion

Model	Public Score	Private Score
Logistic regression best	0.93677	0.92667
Linear SVM best	0.92677	0.92333

We can see that Logistic is a little bit better than linear SVN in this task. However, beacuse the dimension of features is small (less than 1000) and the size of training examples is intermediate (10-10000), Gaussian kernel SVM model will be a more appropriate model then logistic regression or linear SVM.