机器学习第三次作业

1、推导软-SVM 主问题的对偶问题

1. 构造拉格朗日函数得

$$L(w,b,\alpha,\xi,\beta)=\frac{1}{2}w^Tw+C\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\alpha_i\left(y_i(w^Tx_i+b)-1+\xi_i\right)-\sum_{i=1}^{n}\beta_i\xi_i$$

对 w, b 和 ξ_i 求偏导得

$$\frac{\partial L}{\partial w}=w-\sum_{i=1}^{n}\alpha_i y_i x_i=0$$

$$\frac{\partial L}{\partial b}=-\sum_{i=1}^{n}\alpha_i y_i=0 \longrightarrow \begin{cases} w=\sum_{i=1}^{n}\alpha_i y_i x_i \\ \sum_{i=1}^{n}\alpha_i y_i=0 \\ C=\alpha_i+\beta_i \end{cases}$$

$$\frac{\partial L}{\partial \xi_i}=C-\alpha_i-\beta_i=0$$

将其代入到 $L(w,b,\alpha,\xi,\beta)$ 中得

$$L=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i y_i x_i^T x_j \alpha_j y_j+\sum_{i=1}^{n}(\alpha_i+\beta_i)\xi_i-\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i y_i \alpha_j y_j x_j^T x_i$$

$$+\sum_{i=1}^{n}\alpha_i-\sum_{i=1}^{n}\alpha_i\xi_i-\sum_{i=1}^{n}\beta_i\xi_i$$

$$=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i y_i \alpha_j y_j x_i^T x_j$$

∴ 对偶问题为

$$\max_{\alpha}\ \sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$\text{s.t.}\ \sum_{i=1}^{n}\alpha_i y_i=0$$

$$0\le\alpha_i\le C \quad i=1,2,\cdots,n$$

2、上机实验题

（1）读取数据

```python
def load_data():
    '''
    读取数据集， X共有1900个特征，训练集有4000个数据，测试集有1000个数据
    :return: train_X, train_y, test_X, test_y
    '''
    train_dataFile = '垃圾邮件训练和测试数据/spamTrain.mat'
    test_dataFile = '垃圾邮件训练和测试数据/spamTest.mat'
    # 1899个特征值，4000个数据
    train_data = scio.loadmat(train_dataFile)
    train_X = train_data.get("X")
    train_y = train_data.get("y")
    # 1899个特征值，1000个数据
    test_data = scio.loadmat(test_dataFile)
    test_X = test_data.get("Xtest")
    test_y = test_data.get("ytest")
    return np.array(train_X), np.array(train_y), np.array(test_X), np.array(test_y)
```

（2）佩加索斯算法

```python
def pegasos(X, y, T=1000, lam=1):
    m, n = X.shape  # m表示X的样本个数， n表示每个样本的特征个数
    # w = np.random.randn(n, 1)
    # b = np.random.randn(1)
    w = np.zeros((n, 1))
    b = 0
    for t in range(1, T + 1):
        eta = 1.0 / (lam * t)
        i = np.random.randint(m)
        p = predict(w, X[i], b)
        if y[i] * p < 1:
            w = (1.0 - 1 / t) * w + eta * y[i] * (X[i].reshape(-1, 1))
            b += eta * y[i]
        else:
            w = (1.0 - 1 / t) * w
            b = b
    return w, b
```

（3）计算精度

```python
def accuracy(w, b, X, y):
    y_predict = np.array([predict(w, x, b) for x in X])
    y_predict = np.array([[1 if i > 0 else -1] for i in y_predict])
    num = 0
    for (i, j) in zip(y_predict, y):
        if i == j:
            num += 1
    acc = num / len(y) * 100
    return acc
```

（4）保存测试集的标签和预测结果的比较

```python
def save(y_test, y_predict):
    fn = "预测值与测试集标签比较.txt"
    y_predict = [[True if i > 0 else False] for i in y_predict]
    y_test = [[True if i > 0 else False] for i in y_test]
    num = 0
    with open(fn, "w") as file_obj:
        string = "(标签，预测值)        "
        file_obj.write(string + string + string + string + "\n")
        for i, j in zip(y_test, y_predict):
            num += 1
            if num % 4 == 0:
                string = str((i, j)) + '     '
                file_obj.write(string + "\n")
            else:
                string = str((i, j)) + '     '
                file_obj.write(string)
```

（5）试验结果

```
SVM
在测试集上精度为：97.2，时间为：2.127821922302246

Process finished with exit code 0
```

```
SVM
在测试集上精度为：97.3，时间为：1.9154663085937 5

Process finished with exit code 0
```

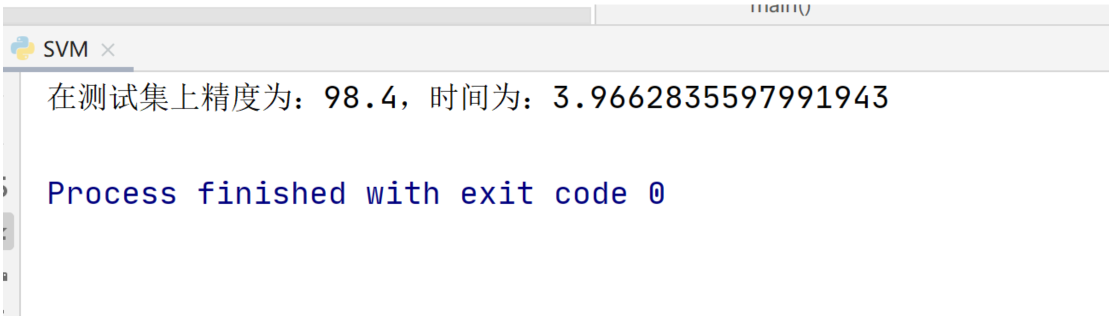
```
SVM
在测试集上精度为：97.5，时间为：2.176470994949341

Process finished with exit code 0
```

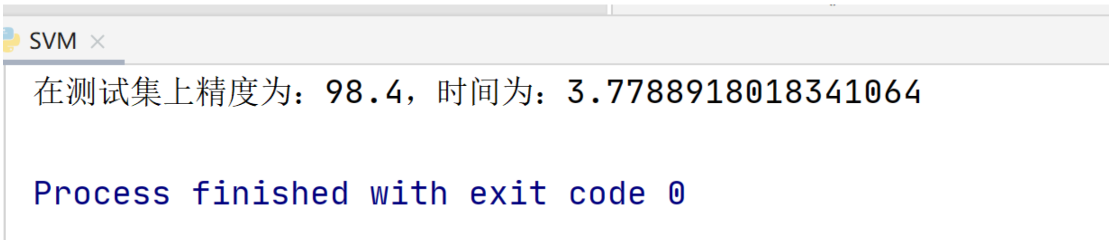
（标签，预测值）	（标签，预测值）	（标签，预测值）	（标签，预测值）
([True], [True])	([False], [False])	([False], [False])	([True], [True])
([True], [True])	([True], [True])	([True], [True])	([False], [False])
([True], [True])	([True], [False])	([False], [False])	([False], [False])
([False], [False])	([False], [False])	([True], [True])	([False], [False])
([False], [False])	([True], [True])	([True], [True])	([True], [True])
([False], [False])	([False], [True])	([False], [False])	([False], [False])
([True], [True])	([False], [False])	([False], [False])	([True], [True])
([False], [False])	([False], [False])	([False], [False])	([False], [False])
([False], [False])	([False], [False])	([True], [True])	([False], [False])
([True], [True])	([False], [False])	([True], [True])	([True], [True])
([True], [True])	([True], [True])	([True], [True])	([True], [True])
([True], [True])	([False], [False])	([False], [False])	([False], [False])
([True], [True])	([False], [True])	([False], [False])	([False], [False])
([True], [True])	([False], [False])	([False], [False])	([False], [False])
([False], [False])	([True], [True])	([False], [False])	([False], [False])
([False], [False])	([False], [False])	([False], [False])	([False], [False])
([True], [True])	([False], [False])	([False], [False])	([False], [False])
([True], [True])	([True], [True])	([False], [False])	([True], [True])
([True], [True])	([False], [True])	([False], [False])	([True], [True])

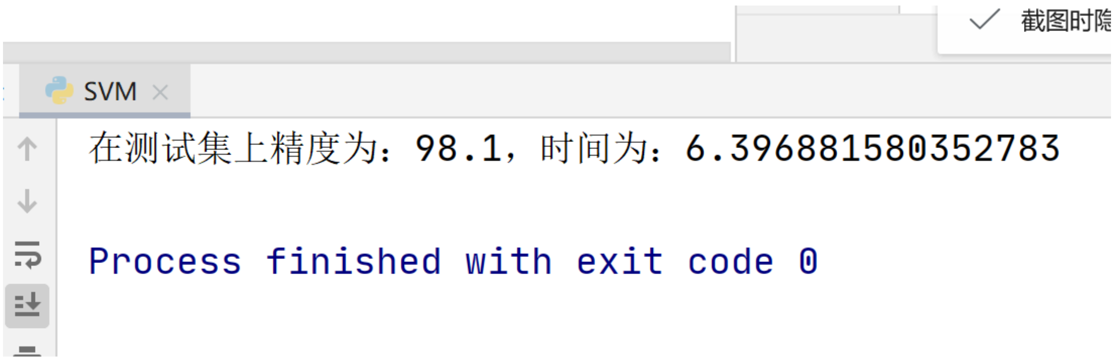
可以看出最后在测试集上的精度为百分之九十七点多。在前八十个结果中预测错了三个。

（6）采用佩加索斯算法的随机性太大，类比梯度下降，采用小批量佩加索斯算法

```
def batch_pegasos(X, y, T=1000, lam=0.1, size=10):
    m, n = X.shape   # m表示X的样本个数， n表示每个样本的特征个数
    w = np.zeros((n, 1))
    b = 0
    dataIndex = np.arange(m)
    for t in range(1, T + 1):
        eta = 1.0 / (lam * t)
        np.random.shuffle(dataIndex)
        for j in range(size):
            i = dataIndex[j]
            p = predict(w, X[i], b)
            if y[i] * p < 1:
                w = (1.0 - 1 / t) * w + eta * y[i] * (X[i].reshape(-1, 1))
                b += eta * y[i]
            else:
                w = (1.0 - 1 / t) * w
                b = b
    return w, b
```







可以看到在验证集上的精度再次提升到百分之九十八以上，运行时间会稍微长些，当时效果更好些。

注：具体代码和训练结果看附件