ML
LAB 1 report

王一鸣

PB21000024

目录

实验目的

实现逻辑斯蒂回归

实验环境

python3.11

windows11H23

实验步骤

- 切分数据
- 设计模型
- 训练
- 检验结果

切分数据

从维度为20，样本量10000中抽取数据，选取训练比率80，得到训练集和测试集

设计模型

梯度下降法

```
class LinearSVM:
    def __init__(self, learning_rate=0.0001, lambda_param=0.001, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None
        self.losses = []
        self.accuracies = []
    def fit(self, X, y , X_test, y_test):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)

        self.w = np.zeros(n_features)
        self.b = 0

        # Gradient Descent
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w -
(x_i*y_[idx]))
                    self.b -= self.lr * y_[idx]
            loss = self.compute_loss(X, y_)
            self.losses.append(loss)

            # 计算测试准确率
            acc = self.compute_accuracy(X_test, y_test)
            self.accuracies.append(acc)

    def predict(self, X):
        linear_output = np.dot(X, self.w) - self.b
        return np.sign(linear_output)
    def compute_loss(self, X, y):
        distances = 1 - y * (np.dot(X, self.w) - self.b)
        distances[distances < 0] = 0  # max(0, distance)
        hinge_loss = (np.sum(distances) / len(X))
        return hinge_loss

    def compute_accuracy(self, X, y):
        y_pred = self.predict(X)
        y_true = y
        #print(y_true.shape, y_pred.shape)
        total=0
        right=0
        for i in range(len(y_true)):
            total+=1
            if y_true[i][0]==y_pred[i]:
                right+=1
        return right/total
```

简单的梯度下降并加入了度量函数方便了解训练情况

smo

```
class SVM2:
    def __init__(self, X, y, C, tolerance, max_passes):
        self.X = X
        self.y = y
        self.C = C
        self.tol = tolerance
        self.max_passes = max_passes
        self.m, self.n = X.shape
        self.alphas = np.zeros(self.m, dtype=np.float32)
        self.b = 0
        self.w = np.zeros(self.n, dtype=np.float32)
    def f(self, x):
        return np.dot(x, self.w) + self.b
        return ((self.alphas * self.y)*np.dot(self.X, x.T)).sum() + self.b

    def choose_and_update(self, i):
        E_i = self.f(self.X[i]) - self.y[i]
        #print("i=",i,"E_i=",E_i)
        if ((self.y[i] * E_i < -self.tol and self.alphas[i] < self.C) or
            (self.y[i] * E_i > self.tol and self.alphas[i] > 0)):

            j = np.random.randint(0, self.m)
            while j == i:
                j = np.random.randint(0, self.m)

            E_j = self.f(self.X[j]) - self.y[j]
            alpha_i_old, alpha_j_old = self.alphas[i], self.alphas[j]

            if self.y[i] != self.y[j]:
                L = max(0, self.alphas[j] - self.alphas[i])
                H = min(self.C, self.C + self.alphas[j] - self.alphas[i])
            else:
                L = max(0, self.alphas[i] + self.alphas[j] - self.C)
                H = min(self.C, self.alphas[i] + self.alphas[j])

            if L == H:
                return

            eta = 2.0 * np.dot(self.X[i], self.X[j].T) - np.dot(self.X[i],
self.X[i].T) - np.dot(self.X[j], self.X[j].T)
            if eta >= 0:
                return

            self.alphas[j] -= self.y[j] * (E_i - E_j) / eta
            self.alphas[j] = min(H, self.alphas[j])
            self.alphas[j] = max(L, self.alphas[j])

            if abs(self.alphas[j] - alpha_j_old) < 1e-3:
                return

            self.alphas[i] += self.y[i] * self.y[j] * (alpha_j_old - 
self.alphas[j])

            b1 = self.b - E_i - self.y[i] * (self.alphas[i] - alpha_i_old) * 
np.dot(self.X[i], self.X[i].T) - self.y[j] * (self.alphas[j] - alpha_j_old) * 
np.dot(self.X[i], self.X[j].T)
            b2 = self.b - E_j - self.y[i] * (self.alphas[i] - alpha_i_old) * 
np.dot(self.X[i], self.X[j].T) - self.y[j] * (self.alphas[j] - alpha_j_old) * 
np.dot(self.X[j], self.X[j].T)

            if 0 < self.alphas[i] < self.C:
                self.b = b1
            elif 0 < self.alphas[j] < self.C:
                self.b = b2
            else:
                self.b = (b1 + b2) / 2.0
            self.w = 0
            for i in range(self.m):
                self.w += self.alphas[i] * self.y[i] * self.X[i]
            return
        return

    def fit(self):
        passes = 0
        for i in range(self.max_passes):
            self.choose_and_update(i%self.m)
            if passes%10==1:
                pass
                #print("passes=",passes)
            passes += 1



    def predict(self, X):
        y_pred = np.zeros((X.shape[0]))
        for i in range(X.shape[0]):
            if(i%10==1):
                print("i=",i)
            y_pred[i] = np.sign(self.f(X[i]))
        return y_pred
```

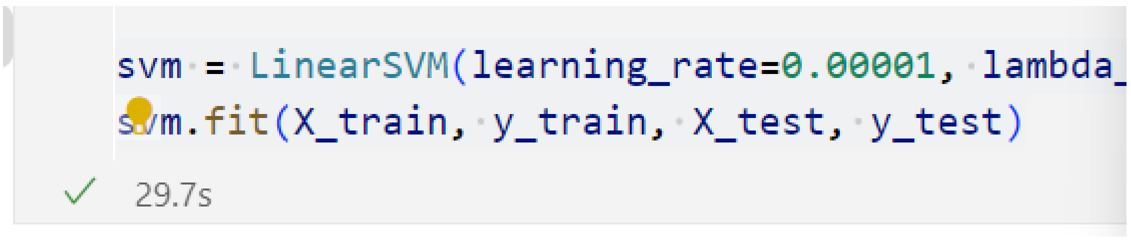
由于样本量比较大，要求参数全部满足ktt条件开销太大，故度量每次更新参数，取最大迭代次数为超参数进行调整

训练

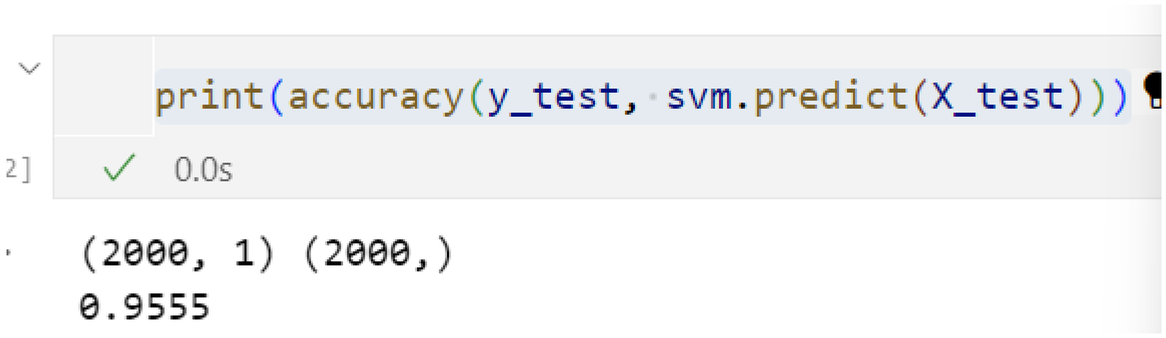
梯度下降

```
X_data, y_data, mislabel = generate_data(feature_num,data_size)
# split data
X_train, y_train = X_data[:split_id], y_data[:split_id]
X_test, y_test = X_data[split_id:], y_data[split_id:]
```

```python
import matplotlib.pyplot as plt
svm = LinearSVM(learning_rate=0.00001, lambda_param=0.00001, n_iters=30)
svm.fit(X_train, y_train, X_test, y_test)
print(accuracy(y_test, svm.predict(X_test)))
```

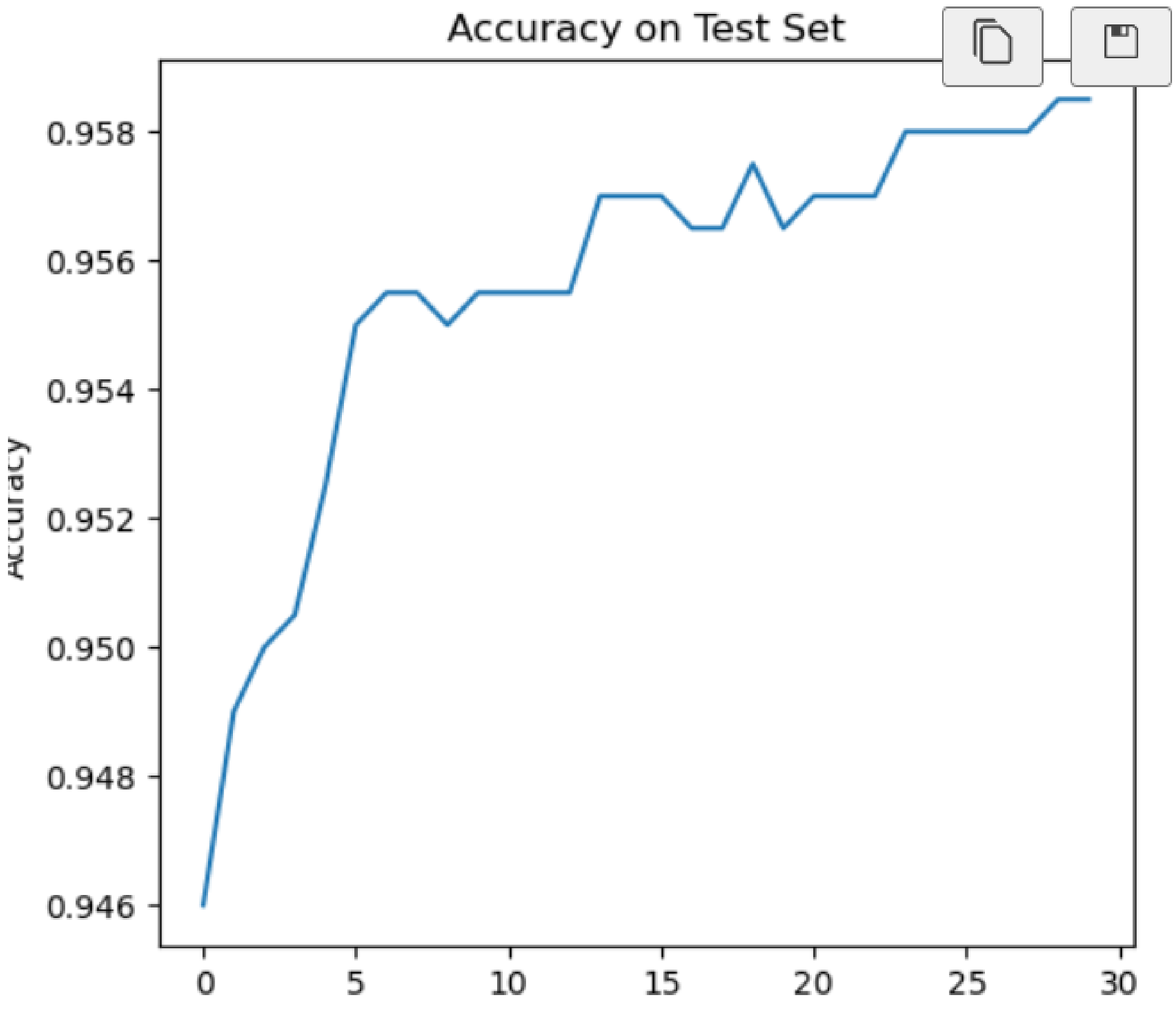


```
svm = LinearSVM(learning_rate=0.00001, lambda_
svm.fit(X_train, y_train, X_test, y_test)
```



```
print(accuracy(y_test, svm.predict(X_test)))
```

```
(2000, 1) (2000,)
0.9555
```

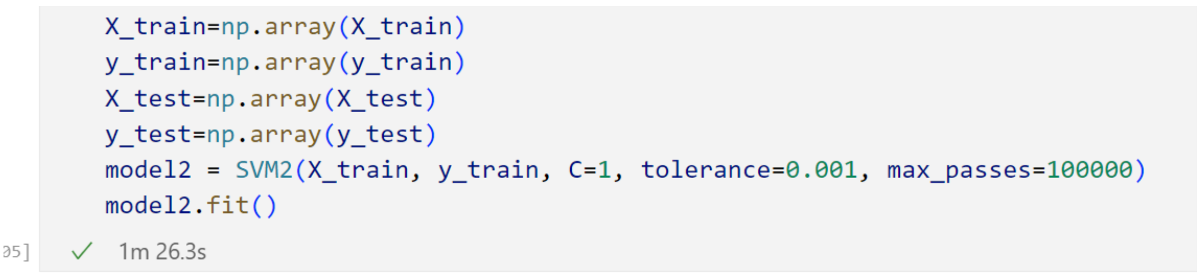


可以看到初始迭代的几次基本上已经取得了比较好的效果，由于smo算法时间开销比较大，所以没有用相同的方法来可视化。

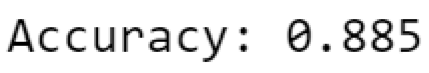
时间开销30s,准确率0.955

smo

```
feature_num=20
data_size=10000
train_ratio=0.8
split_id=int(data_size*train_ratio)
X_data, y_data, mislabel = generate_data(feature_num,data_size)
# split data
X_train, y_train = X_data[:split_id], y_data[:split_id]
X_test, y_test = X_data[split_id:], y_data[split_id:]
X_train=np.array(X_train)
y_train=np.array(y_train)
X_test=np.array(X_test)
y_test=np.array(y_test)
model2 = SVM2(X_train, y_train, C=1, tolerance=0.001, max_passes=100000)
model2.fit()
y_pred = model2.predict(X_test)
print(y_pred.reshape(-1,1))
print(y_test.reshape(-1,1))
print("Accuracy:", np.mean(y_pred.reshape(-1,1) == y_test.reshape(-1,1)))
```



```
    X_train=np.array(X_train)
    y_train=np.array(y_train)
    X_test=np.array(X_test)
    y_test=np.array(y_test)
    model2 = SVM2(X_train, y_train, C=1, tolerance=0.001, max_passes=100000)
    model2.fit()
```



```
Accuracy: 0.885
```

总结与反思

由于ddl突然提前，实验完成的比较仓促，很多细节来不及优化。主要问题是模型的稳定性没法保证，效果基本上是靠超参数试出来的。另一方面对于smo算法理解的不是很透彻，所以实现的时候颇费了一番功夫。