Introduction to Data Mining: 21191441
Zhejiang University — May 9, 2019
Professor Deng Cai — Homework 2

Homework 2

Collaborators:

Name: Jiang Shibiao
Student ID: 3170102587

Problem 2-1. A Walk Through Linear Models

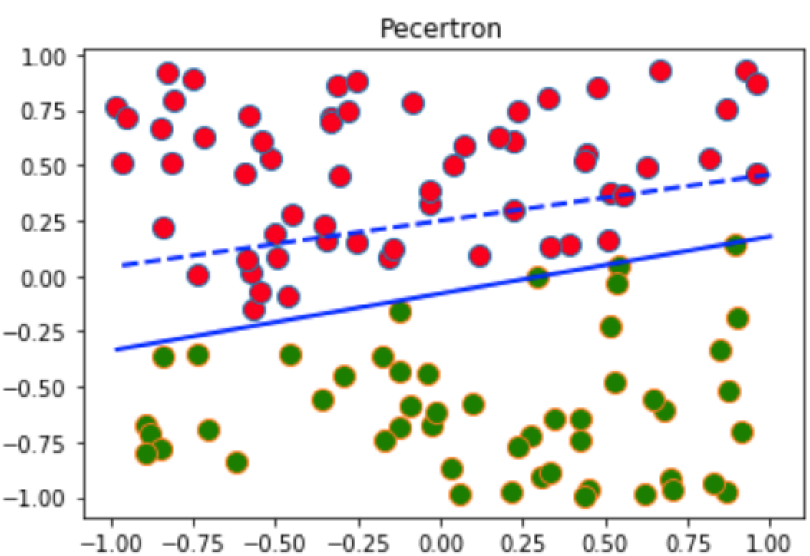
(a) Perceptron

Answer:

1. The learning rate $= 0.01$, and generate 100 more data for test.

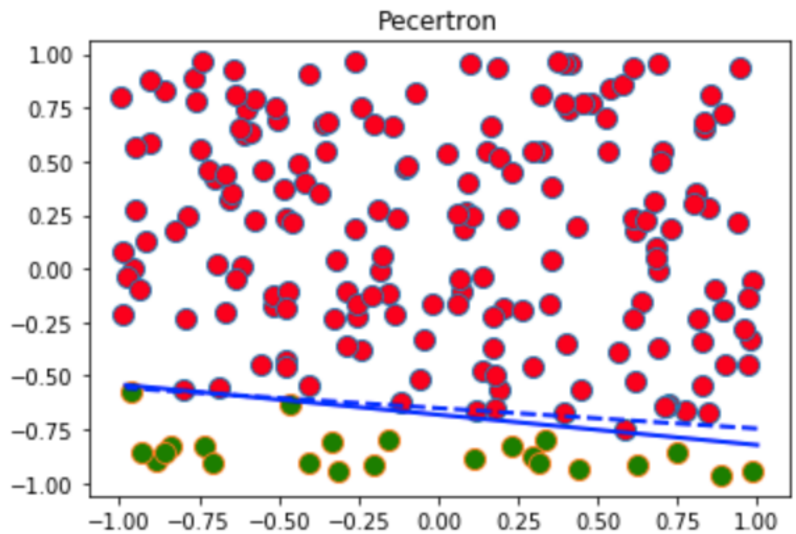
   When the size of training set is 10, training error is 0%, test error is $10.56(\pm 1)\%$.

```
E_train is 0.0, E_test is 0.10557000000000016
Average number of iterations is 5.637.
```



   When the size of training set is 100, training error is 0%, test error is $13.7(\pm 1)\%$.

```
E_train is 0.0, E_test is 0.013659999999999868
Average number of iterations is 58.358.
```



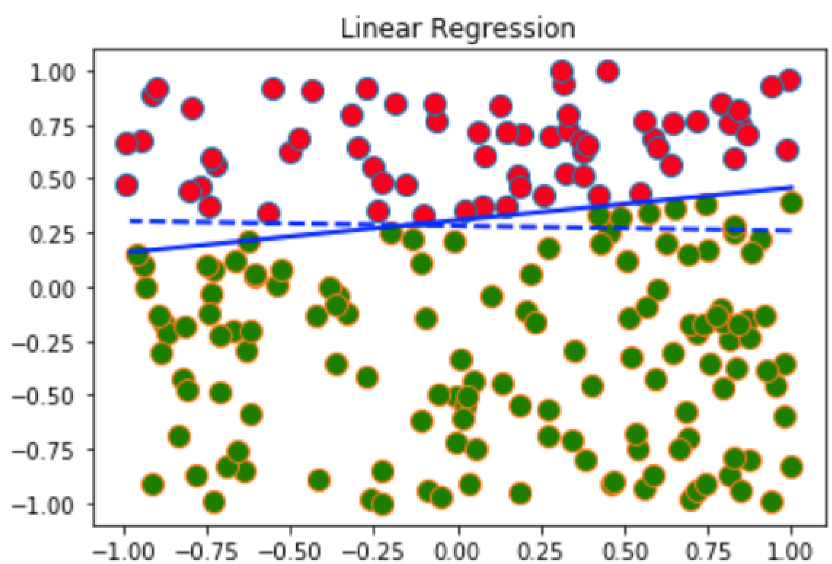
2. **When the size of training set is** 10, the average number of iterations is 5.6 ± 2.
   When the size of training set is 100, the average number of iterations is 58.4 ± 15.
3. It never converges.(i.e. the number of iterations $\rightarrow \infty$)

(b) Linear Regression

Answer:

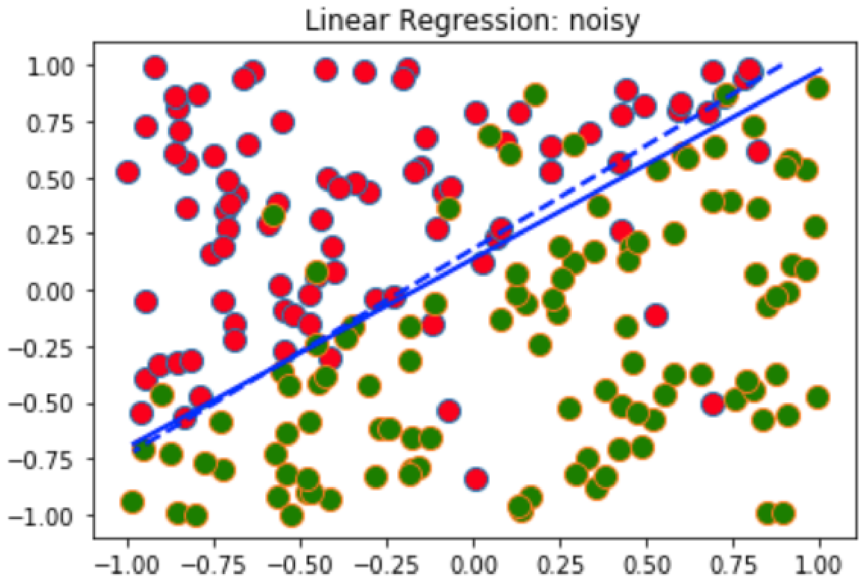
1. The training error is $4.1(\pm 0.2)\%$, the expected test error (number: 100) is $4.9(\pm 0.2)\%$.

```
E_train is 0.04068000000000084, E_test is 0.048570000000000065
```



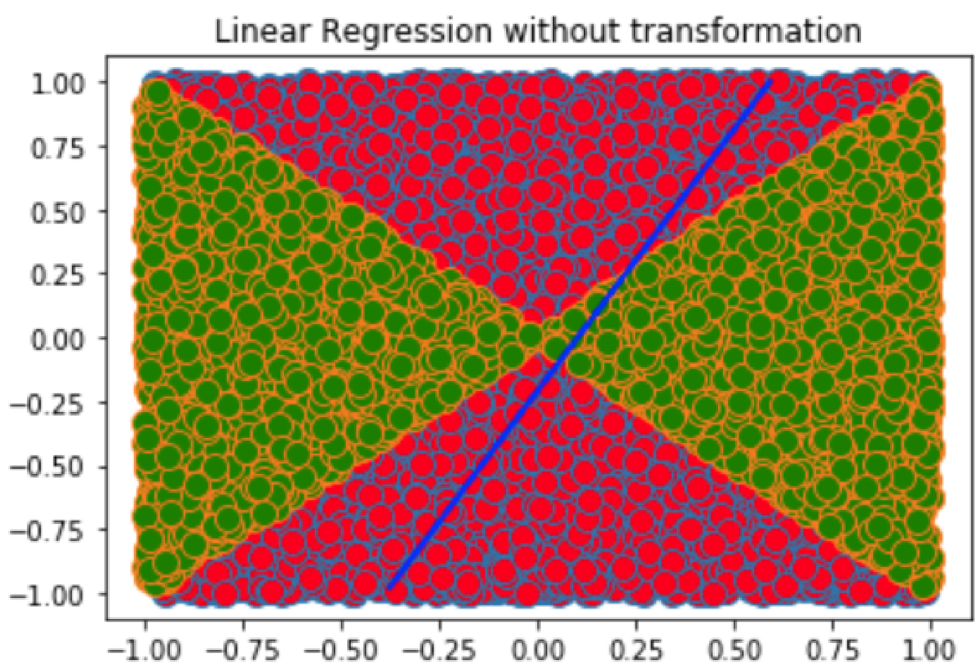
2. The training error is $13.3(\pm 0.5)\%$, the expected test error (number: 100) is $14.7(\pm 0.5)\%$.

   E_train is 0.1328700000000001, E_test is 0.1468000000000024



3. The training error is 40%, the testing error is 55.0%.
   I'm little surprised for this result, so I print the test results ($= 10000$ points). From the figure we can find that (pure) linear regression is not fit for non-linear cases.

   E_train is 0.49, E_test is 0.5496



4. The training error is 5.0%, the testing error is 6.6% (All reduce a lot).

```
# poly_fit with transform
X_train_t = np.array([X_train[0], X_train[1], X_train[0] * X_train[1], X_train[0] ** 2, X_train[1] ** 2])
X_test_t = np.array([X_test[0], X_test[1], X_test[0] * X_test[1], X_test[0] ** 2, X_test[1] ** 2])
w = linear_regression(X_train_t, y)

train_results = y_train * np.matmul(w.T, np.concatenate((np.ones((1, nTrain)), X_train_t), axis = 0))
E_train = np.sum(train_results <= 0) / nTrain
test_results = y_test * np.matmul(w.T, np.concatenate((np.ones((1, nTest)), X_test_t), axis = 0))
E_test = np.sum(test_results <= 0) / nTest

# Compute training, testing error
print('E_train is {}, E_test is {}'.format(E_train, E_test))
plotdata(X_test_t, y_test, w, w, 'Linear Regression with transformation');
```

```
E_train is 0.05, E_test is 0.066
Here we only support 2-d X data
```

(c) Logistic Regression

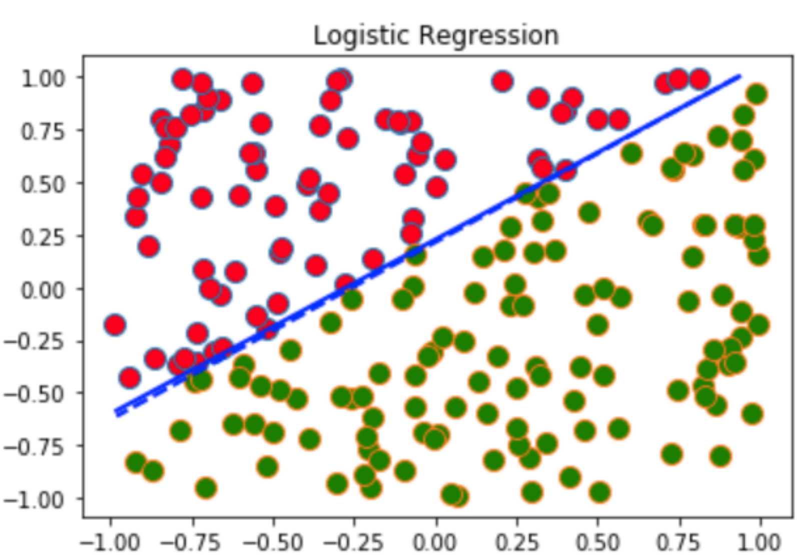
Answer:

1. I use dynamic learning rate which will be multiplied by a constant after each step.

```
step = 0
maxstep = 100
learning_rate = 1
smaller = 0.99
while step < maxstep:
    loss = - sum(np.log(h(w, X[:, y == 1]))) - sum(np.log(1 - h(w, X[:, y == 0])))
    grad = np.matmul(X, (h(w, X) - y).reshape((N, 1)))
    learning_rate *= smaller
    w = w - learning_rate * grad
    step += 1
```

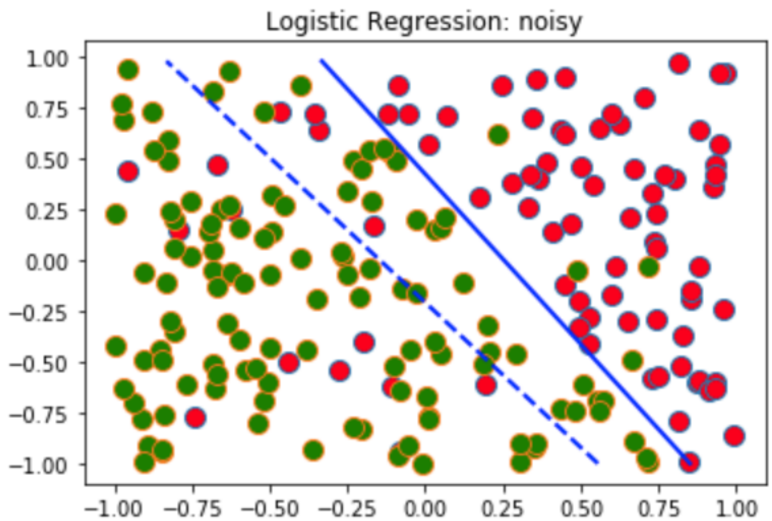
The training error is $0.23(\pm 0.15)\%$, the expected testing error is $1.21(\pm 0.2)\%$.

```
E_train is 0.0023000000000000001, E_test is 0.0121000000000000007
Average loss: 1.8541089114402332
```



2. The training error is $21.3(\pm 3)\%$, the expected testing error is $22.4(\pm 3)\%$.

```
E_train is 0.21349999999999997, E_test is 0.22380000000000005
Average loss: 110.51779073373709
```



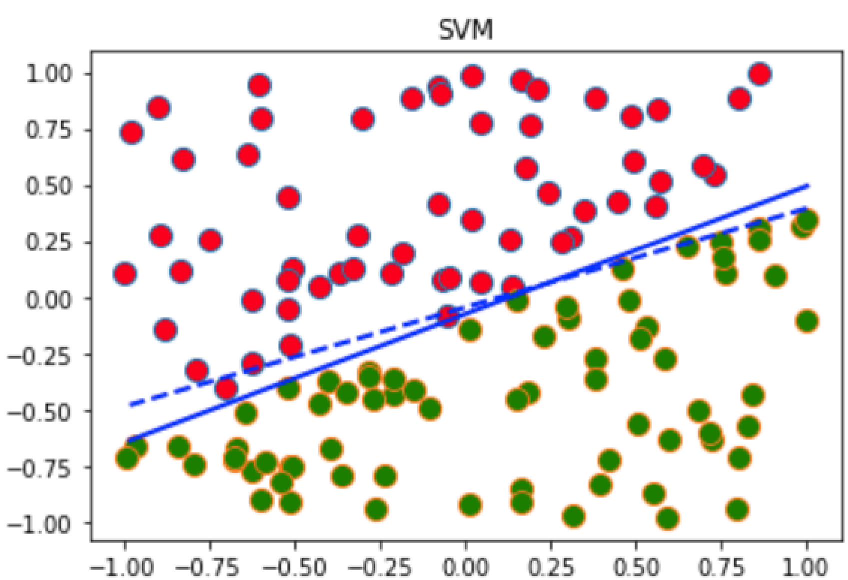
So we will find that it is **not robust** to the noisy.

(d) Support Vector Machine

Answer:

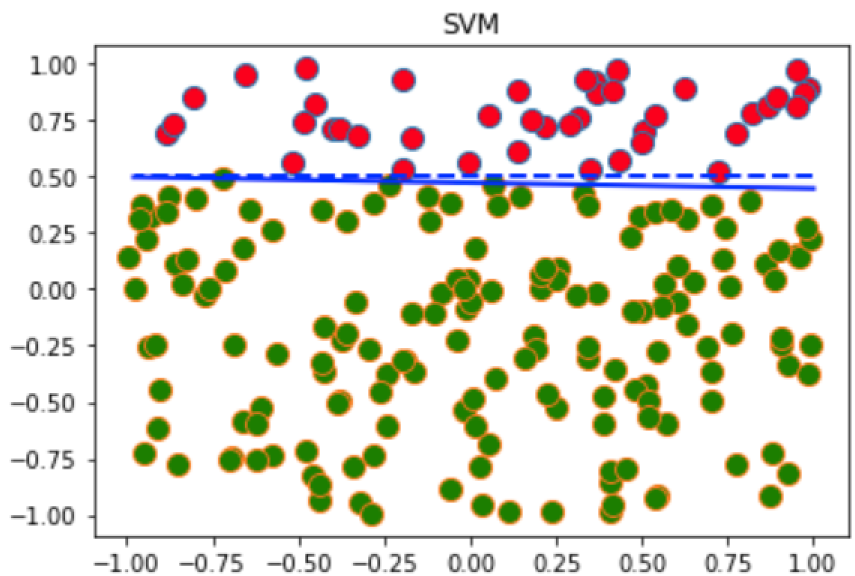
1. If the size of training set is 30, the training error rate is 0.0% while expected testing error rate is $4.1(\pm 1.5)\%$.

```
E_train is 0.0, E_test is 0.04139999999999965
Average number of support vectors is 3.02.
```



2. If the size of training set is 100, the training error rate is 0.0% while expected testing error rate is $0.86(\pm 0.5)\%$.

```
E_train is 0.0, E_test is 0.008600000000000003
Average number of support vectors is 2.92.
```



3. Average number of support vectors is $2.2(\pm 0.3)$.

Problem 2-2. Regularization and Cross-Validation

(a) Implement Ridge Regrssion, and use LOOCV to tune the regularization parameter λ.

Answer:

1. When doing *feature normalization*, I noticed that the variance of some feature **can become zero**. So we must do some adjustment.

```
def feature_normal(X):
    avg = np.average(X, axis = 1).reshape(X.shape[0], 1)
    std = np.std(X, axis = 1).reshape(X.shape[0], 1)
    std[std == 0] = 1.0
    return (X - avg) / std
```

When implementing *ridge.py*, I found that there is a terrible bug one may trap in: **The ω in the code is extended with b, but we should regularize only ω not b.** So we should add the following special judgement.

```
X = np.concatenate((np.ones((1, N)), X), axis = 0)
regular = np.identity(P + 1)
regular[0][0] = 0
w = np.matmul(np.matmul(scipy.linalg.pinv(np.matmul(X, X.T) + lmbda * regular), X), y.T)
```

In my first try, I use the average error rate (the number of error prediction divides the number of training data) to evaluate the validation. Although $\lambda = 1000$ seems the smallest, I think this feature **can not divide the λ clearly**.

```
0.001  Average validation error: 0.11
0.01   Average validation error: 0.11
0.1    Average validation error: 0.11
0.0    Average validation error: 0.345
1.0    Average validation error: 0.11
10.0   Average validation error: 0.06
100.0  Average validation error: 0.04
1000.0 Average validation error: 0.035
```

Then I use the average variance $\sum_i (y_i - \hat{w}_i X_i)^2$ to evaluate the validation. This way seems **more effective**. **I think choose $\lambda = 100$ or $\lambda = 1000$ are both OK**.

```
0.001  Average validation variance: [0.54211448]
0.01   Average validation variance: [0.54149283]
0.1    Average validation variance: [0.53543683]
0.0    Average validation variance: [38.70524012]
1.0    Average validation variance: [0.48737058]
10.0   Average validation variance: [0.33829825]
100.0  Average validation variance: [0.23433591]
1000.0 Average validation variance: [0.32183157]
```

2. With regularization $\lambda = 1000$, $\sum \omega_i^2 = 0.18$.
   With regularization $\lambda = 100$, $\sum \omega_i^2 = 0.36$.
   Without regularization($\lambda = 0$), $\sum \omega_i^2 = 1.01$.
3. With regularization $\lambda = 1000$, the training error is 1.0% and testing error is 5.5%.
   With regularization $\lambda = 100$, the training error is 0.0% and testing error is 6.1%.
   Without regularization, the training error is 0.0% and testing error is 12.3%.
   So regularization will effectively reduce the overfitting.

```
The square of w with lambda 1000: 0.18387598169379504
Training error with lambda  1000: 0.01
Testing error with lambda   1000: 0.055248618784530384
The square of w with lambda  100: 0.3644870309752901
Training error with lambda   100: 0.0
Testing error with lambda    100: 0.06127574083375188
The square of w without lambda:   1.0131787470699385
Training error without lambda:    0.0
Testing error without lambda:     0.12305374183827222
```

(b) Implement Logistic Regrssion, and use LOOCV to tune the regularization parameter.

Answer:

1. To make the training more steady, I change my logistic model from dynamic learning rate to the constant learning rate: 0.001.

   The training error are all 0.0%, so we can just compare the average validation variance among different λ. It's clearly that $\lambda = 100$ is the best.

```
0.001 Average validation error: 0.0
0.001 Average validation variance: [22.95213422]
0.01 Average validation error: 0.0
0.01 Average validation variance: [22.92972276]
0.1 Average validation error: 0.0
0.1 Average validation variance: [22.70751881]
0.0 Average validation error: 0.0
0.0 Average validation variance: [22.95462654]
1.0 Average validation error: 0.0
1.0 Average validation variance: [20.66472311]
10.0 Average validation error: 0.0
10.0 Average validation variance: [10.48175895]
100.0 Average validation error: 0.0
100.0 Average validation variance: [1.91789463]
1000.0 Average validation error: 0.0
1000.0 Average validation variance: [5.04255473]
```

2. With regularization $\lambda = 100$, the training error is 0.0% and testing error is 5.32%. Without regularization, the training error is 0.0% and testing error is 4.92%.
   The regularization seems not so suitable in this case. But we can still find that **the variance of solution still reduces a lot** with regularization.

```
The square of w with lambda  100: 0.6403386339032339
Training error with lambda   100: 0.0
Testing error with lambda    100: 0.053239578101456554
The square of w without lambda:   1.8812149422703894
Training error without lambda:    0.0
Testing error without lambda:     0.04922149673530889
```

Problem 2-3. Bias Variance Trade-off

Let's review the bias-variance decomposition first. Now please answer the following questions:

(a) True of False

Answer:

1. **False**. It may suffer overfitting instead of reducing a lot.
2. **False**. I believe that the best model will have steady performance.
3. **True**.
4. **False**. It will be more complex and may cause some problems.
5. **False**. If $\lambda \to \infty$, our target function will disppear. It's definitely not what we want.