# PM 2.5 Prediction by Linear Regression

Wei-Ning Chen, r05942078

ML Fall 2016

## A. **Linear Regression by Gradient Descent**

The goal of this project is to predict PM2.5 value by the previous 10 hours data. Since PM2.5 is integear-valued, regression is proper and promising for prediction.

### (1) Linear Regression Model

The empirical risk is defined as

$$f_{emp}(w) = \sum (w \cdot x_i^T - y_i)^2,$$

and our goal is to find an optimal $w^* = \arg\min f_{emp}(w)$.

To achieve the optimal $w^*$, we propose to use gradient descent as following:

### (2) Gradient Descent

The gradient descent algorithm update $w$ each steps by its current gradient values:

$$w_{n+1} \leftarrow w_n - \eta \cdot \nabla f_{emp}(w_n)$$

### (3) Adagrad

However, since the regular gradient descent faces to several shortcomings: the fixed learning rate make it easy to trap into local minimum; also it tends to oscillate when close to the optimal solution. Therefore, i propose to use adagrad to improve:

$$w_{n+1} \leftarrow w_n - \frac{\eta}{\| \nabla f_{emp}(w_n) \|_2} \nabla f_{emp}(w_n)$$

The detailed implementation can be found in appendix A ( with python).

## B. **Data Processing and Feature Extraction**

In this section, i briefly explain how i extract the feature from training data. First, we have 18 observed data per hour, and for each test data, we can access the previous 9 hours data. On the other hands, we have totally $20(days) \cdot 12(months) \cdot 24(hours) = 5760(hours)$ observed data. Therefore, i parse the observed data into $5751$ training data, each data consists of continuous 9 hours with the tenth hour's PM2.5 value as it's label, and hence reduce to a

162-dimesional (i.e. $18(\dim)\cdot 9(hours)$) linear regression problem.

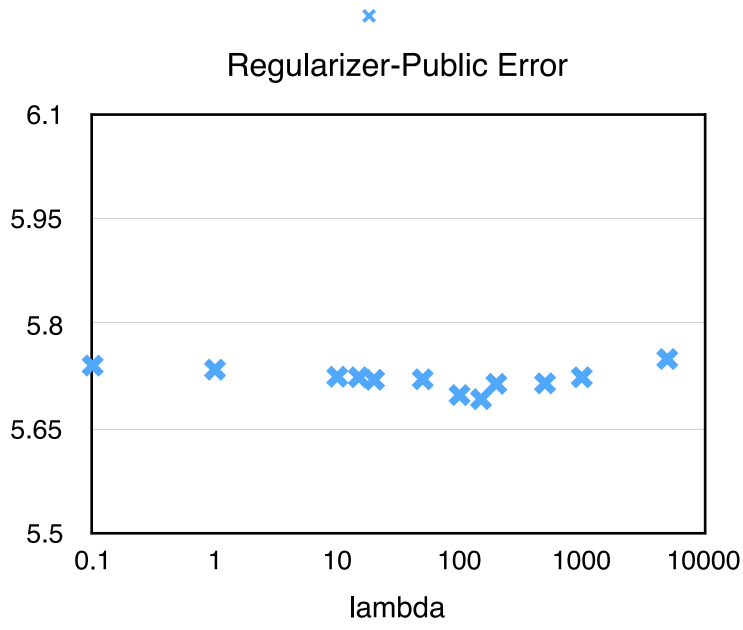
For the detailed implementation, please refer to appendix A.

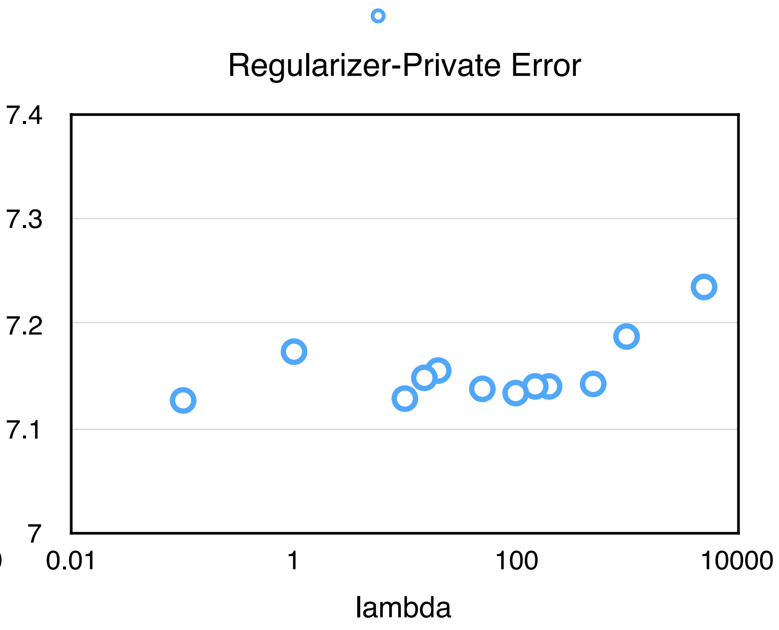
## C. Regularization

To avoid overfitting, we add an regularizer to original objective function. In particular, the regression problem can be rewritten as

$f_{emp}(w) = \sum (w\cdot x_i^T - y_i)^2 + \frac{1}{2}\lambda \| w \|^2$ , where $\lambda$ is regularization parameter.

To find an optimal $\lambda$ , we tried several different values and compared them as following:





From the figures, we can found that the error is minimized when $\lambda \approx 150$ .
The detailed result can be found in Appendix C.

## D. Learning Rate

The learning rate $\eta$ plays a crucial role in gradient descent. If $\eta$ too large, then the error to optimal solution will be considerable. On the other hand, if we choose smaller $\eta$ , then it takes a lot of iterations to converges.

Therefore, if we can adaptively rectify the learning rate, we can get a better result with less iterations.

Fortunately, we can apply adagrad to adjust the learning rate each steps, as described in section A.

## E. Discussion of Strategy in Kaggle

Finally, to get a better result, we can play another trick for this regression problem. High dimensional linear regression can be directly compute by some basic linear algebra fact( for more details please refer to appendix B). Therefore we can get the optimal result (without regularizer) by simply performing matrix multiplication.

The strategy is as following:

First we perform some matrix multiplication to get the optimal solution to the unregularized objective function

$$w_1 = \arg\min \sum (w \cdot x_i^T - y_i)^2$$

Then, we use this as the initial value of the gradient descent solver (adagrad) over the regularized objective function. We can expect this specific initial value will get a faster convergence.

The details can be found in appendix B.

## Appendix A

### (1) Adagrad Implementation (with python)

```
def AdaGrad(f, gf, n, trainSet, theta, T):
    gd_sq_sum = np.zeros(n, dtype=float)
    alpha = 1
    e = 1e-8
    for t in range(1,T):
        g = gf(trainSet, theta)
        gd_sq_sum += np.sum(g,g)
        for i in range(0, n):
            theta[i] -= alpha * g[i] / np.sqrt(gd_sq_sum + e)
        grad_norm = np.linalg.norm(gf(trainSet, theta))
        print "Itr = %d" % t
        print "f(theta) =", f(trainSet, theta)
        print "norm(grad) =", grad_norm
        if grad_norm < 1e-3:
            return theta
    return theta
```

### (2) Feature Extraction (with python and pandas imported)

```
def train_data_parser(filename):
  df = pd.read_csv(filename)
  rawData=[]
  trainSet = []
  tmp = []
  # reshape data into 5760(hours)x18(dimensions)
  for date in range(MONTH*DATE_PER_MONTH):
    for hour in range(HOUR):
      rawData.append(df.ix[date*DIM:date*DIM+DIM-1,str(hour)].values)
  # view each 9 hours as a feature vector
  for hour in range(MONTH*DATE_PER_MONTH*HOUR):
    rawData[hour] = [float_with_str(i) for i in rawData[hour]]

  for hour in range(MONTH*DATE_PER_MONTH*HOUR-9):
     for j in range(9):
       tmp.extend(rawData[hour+j])
     trainSet.append([np.array(tmp),rawData[hour+9][9]])
     tmp = []
  return trainSet
```

## Appendix B

### (1) Solution to Linear Regression

Suppose we want to find the optimal solution $f(x) = w^T \cdot x + b$ of linear regression with (vector valued) samples $x_1,...,x_m$ under $\ell_2$ loss function (i.e. square loss). Let

$X = \begin{pmatrix} {x_1}^T \\ \vdots \\ {x_m}^T \end{pmatrix}$ be a matrix, and $y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$ be the labels. Then the optimal solution is

$w^* = (X^T X)^{-1} X^T y$ .

## (2) Implementation of Predictor in Kaggle

```
#parse data
trainSet = train_data_parser("data/train.csv")
testSet = test_data_parser("data/test_X.csv")
X = []
y = []
for i in range(len(trainSet)):
  X.append(trainSet[i][0])
  y.append(trainSet[i][1])

# Direct calculate optimal w w/o regularizer
X = np.array(X)
y = np.array(y)
A = np.dot(X.T,y.T)
B = np.linalg.pinv(np.dot(X.T,X))
w_init = np.dot(A,B)
b = np.dot(w_init,np.sum(X,axis=0))-np.sum(y)
w_init = np.append(w_init,b)

#training models
model = AdaGrad(f_loss, grad_f, 163, trainSet, w_init, 50)

#get test labels
labels = [getTestLabel(testData, model) for testData in testSet]
ids = ['id_'+str(i) for i in range(len(labels))]
```

## Appendix C

Experiment: How Regularization Affects Regression Error

regularizer-error

| lambda | iterations | scores_pub | scores_priv |
|---|---|---|---|
| 5000 | 350000 | 5.74964 | 7.23533 |
| 1000 | 350000 | 5.72349 | 7.18795 |
| 500 | 250000 | 5.71475 | 7.14260 |
| 200 | 250000 | 5.71402 | 7.14026 |
| 150 | 250000 | 5.69210 | 7.14026 |
| 100 | 200000 | 5.69795 | 7.13384 |
| 50 | 200000 | 5.72058 | 7.13793 |
| 20 | 250000 | 5.71985 | 7.15542 |
| 15 | 200000 | 5.72349 | 7.14843 |
| 10 | 100000 | 5.72422 | 7.12858 |
| 1 | 100000 | 5.73440 | 7.17345 |
| 0.1 | 100000 | 5.74021 | 7.12683 |
| 0 | - | 6.07934 | 7.32063 |