COMP540 Statistical Machine Learning HW02

Yiqing Lu, Chuanwenjie Wei

February 6, 2017

1. Gradient and Hessian of $NLL(\theta)$ for logistic regression

1)

$$g(z) = \frac{1}{1+e^{-z}} \quad \Longrightarrow \quad \frac{\partial g(z)}{\partial z} = \frac{-1}{(1+e^{-z})^2} e^{-z}(-1) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$g(z)(1-g(z)) = \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$\Longrightarrow \frac{\partial g(z)}{\partial z} = g(z)(1-g(z))$$

2)

$$\frac{\partial}{\partial \theta} NLL(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \frac{1}{h_\theta(x^{(i)})} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)} + (1-y^{(i)}) \frac{-1}{1-h_\theta(x^{(i)})} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)}x^{(i)} - y^{(i)}h_\theta(x^{(i)})x^{(i)} - h_\theta(x^{(i)})x^{(i)} + y^{(i)}h_\theta(x^{(i)})x^{(i)}]$$

$$= \frac{1}{m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y^{(i)}]x^{(i)}$$

3)

Because

$$h_\theta(x^{(i)}) \in (0,1) \quad \Longrightarrow \quad (1-h_\theta(x^{(i)})) \in (0,1)$$

Let

$$S_1 = S^{0.5} = diag(\sqrt{h_\theta(x^{(1)})(1-h_\theta(x^{(1)}))},\ \dots\ ,\ \sqrt{h_\theta(x^{(1)})(1-h_\theta(x^{(1)}))})$$

$$\Longrightarrow S = S_1^T S_1 \qquad \Longrightarrow H = X^T S X = X^T S_1^T S_1 X = (S_1 X)^T (S_1 X)$$

Suppose

$$a = (a_1,\ a_2,\ \dots\ ,\ a_d)$$

$$\Longrightarrow a^T H a = \sum_{i=1}^{m}\sum_{j=1}^{d} (S_{1_{ii}} X_{ij} a_j)^2$$

and since X is full rank, so

$$a^T H a = \sum_{i=1}^{m}\sum_{j=1}^{d} (S_{1_{ii}} X_{ij} a_j)^2 = \sum_{i=1}^{m}\sum_{j=1}^{d} (S_{1_{ii}} a_j)^2 (X_{ij})^2 > 0$$

So H is positive definate.

2. Regularizing logistic regression

Since $\theta \sim MN\left(0,\ \alpha^2 I\right)$, **for regularized estimate** θ_{MAP}, **we have:**

$$\theta_{MAP} = argmax_\theta \left(2\pi\right)^{-\frac{d+1}{2}} |\alpha|^{-1} exp\left\{-\frac{1}{2\alpha^2}\theta^T\theta\right\} L\left(\theta\right)$$

$$= argmax_\theta exp\left\{-\frac{1}{2\alpha^2}\theta^T\theta\right\} L\left(\theta\right)$$

where $L\left(\theta\right)$ **denotes the likelihood of data**

We can rewrite above formula by taking log:

$$\theta_{MAP} = argmax_\theta LL\left(\theta\right) - \frac{1}{2\alpha^2}\theta^T\theta$$

$$= argmin_\theta NLL\left(\theta\right) + \frac{1}{2\alpha^2}||\theta||^2$$

And we know for θ_{MLE},

$$\theta_{MLE} \quad = \quad argmin NLL\left(\theta\right)$$

So

$$NLL\left(\theta_{MAP}\right) \quad \geq \quad NLL\left(\theta_{MLE}\right);\ NLL\left(\theta_{MAP}\right) + \frac{1}{2\alpha^2}||\theta_{MAP}||^2 \leq NLL\left(\theta_{MLE}\right) + \frac{1}{2\alpha^2}||\theta_{MLE}||^2$$

And thus

$$||\theta_{MAP}||^2 - ||\theta_{MLE}||^2 \quad \leq \quad 2\alpha^2\left(NLL\left(\theta_{MLE}\right) - NLL\left(\theta_{MAP}\right)\right) \leq 0$$

1 Implementing a k-nearest-neighbor classier

1.1 Distance matrix computation with two loops

```
for i in xrange(num_test):
  for j in xrange(num_train):
    #####################################################################
    # TODO:                                                             #
    # Compute the l2 distance between the ith test point and the jth    #
    # training point, and store the result in dists[i, j]. You should   #
    # not use a loop over dimension. 1 line of code expected            #
    #####################################################################
    dists[i,j]=np.sqrt(np.sum(np.square(X[i,:] - self.X_train[j,:])))

    pass
    #####################################################################
    #                       END OF YOUR CODE                            #
    #####################################################################
return dists
```

Fig.1 Distance matrix computation with two loops

Figure 1 shows the core code for computing distance matrix with two loops.

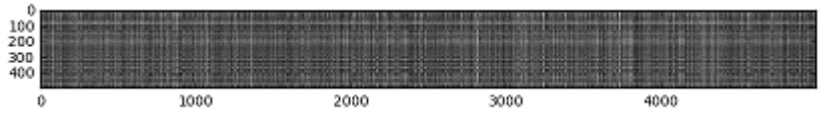

Fig.2 The distance matrix

Fig.2 shows the distance matrix. The x axis is the training data set, and the y axis is the test data set. The color in Fig.1 shows the distance between the i^{th} training data and the j^{th} test data. Since black indicates low distances while white indicates high distances, the distinctly bright rows indicate that the j^{th} test data is far away from all the training data(suppose axis_y=j). The distinctly bright columns indicate that the i^{th} training data is far away from all the teat data(suppose axis_x=i).

1.2 Compute majority label

When k=1, we get 137 / 500 correct => accuracy: 0.274000.

When k=5, we get 139 / 500 correct => accuracy: 0.278000.

We can see that for k=5, the performance is slightly better than for k=1.

1.3 Distance matrix computation with one loop

The code is as following:

```
for i in xrange(num_test):
  #########################################################################
  # TODO:                                                                 #
  # Compute the l2 distance between the ith test point and all training   #
  # points, and store the result in dists[i, :].                          #
  #########################################################################
  dists[i,:]= np.sqrt(np.sum(np.square(X[i,:]-self.X_train),axis=1))

  pass
  #########################################################################
  #                       END OF YOUR CODE                                #
  #########################################################################
return dists
```

Fig.3 Distance matrix computation with one loop

1.4 Distance matrix computation with no loops

We could devide $(x-y)^2$ to x^2, y^2, xy. The code is as following:

```
#########################################################################
sum_test = np.sum(X ** 2, axis=1)
sum_train = np.sum(self.X_train ** 2, axis=1)
s = sum_test.reshape((num_test, 1)) + sum_train - 2 * X.dot(self.X_train.T)
dists = np.sqrt(s)

pass
#########################################################################
#                       END OF YOUR CODE                                #
#########################################################################
return dists
```

Fig.4 Distance matrix computation with no loop

The running time of each circumstance is as following:

1) Two loop version took 35.989207 seconds.

2) One loop version took 42.476688 seconds.

3) No loop version took 0.297603 seconds.

The No loop version is the fastest, the one loop version is the slowest. The speed of the no loop version is about 100 times faster than the two loop version.

1.5 Choosing k by cross validation

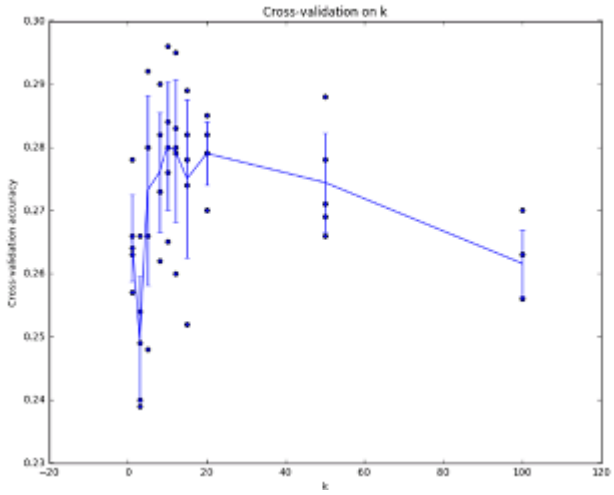

Fig.5 Choosing k by crossvalidation on the CIFAR-10 dataset

From Fig.5, wo can see the relationship between cross-validation accuracy and k. When k=10, the cross-validation accuracy is the highest.

When k=5, we get 141 / 500 correct => accuracy: 0.282000.

2 Implementing logistic regression

2.1 Logistic regression

2.1.1 Implementing logistic regression : the sigmoid function

The code is as following:

```
def sigmoid (z):

sig = np.zeros(z.shape)

sig=1/(1+np.e**(-z))

return sig
```

2.1.2 Cost function and gradient of logistic regression

The code of the cost function in logistic regression is

```
for i in range(len(y)):

J=J+1.0/len(y)*(-y[i]*math.log(utils.sigmoid(np.dot(X[i,:],theta)))-(1-
y[i])*math.log(1-utils.sigmoid(np.dot(X[i,:],theta))))
```

The code of the gradient of the cost is

```
y_pred=utils.bin_features(np.dot(X,self.theta))
```

Loss on all-zeros theta vector (should be around 0.693) = 0.69314718056.

Gradient of loss wrt all-zeros theta vector (should be around [-0.1, -12.01, -11.26]) =

[-0.1 -12.00921659 -11.26284221].

Theta found by fmin_bfgs: [-25.16056945 0.20622963 0.20146073].

Final loss = 0.203497702351.

2.1.3 Prediction using a logistic regression model

Let XXX=np.array([1,45,85])

Then calculate the admission probability pred_prob=1.0/(1+np.e**(-np.dot(XXX,theta_opt)))

For a student with 45 on exam 1 and 85 on exam 2, the probability of admission = 0.776246678481.

The prediction is y_pred=utils.bin_features(np.dot(X,self.theta))

And then calculate the accuracy by comparing predy and y. Accuracy on the training set = 0.89.

2.1.4 Visualizing the decision boundary

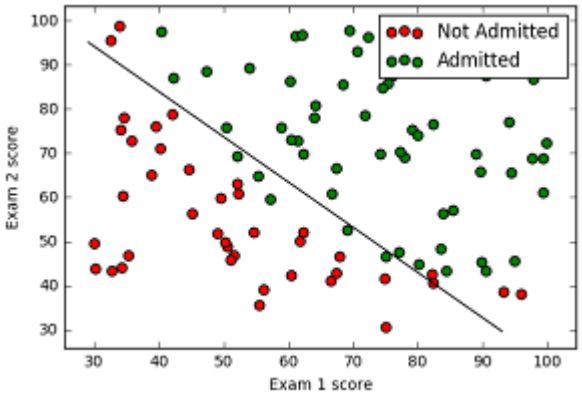

Fig.6 The decision boundary

Theta found by sklearn: [[-25.15293066 0.20616459 0.20140349]].

2.2 Regularized logistic regression

2.2.1 Cost function and gradient for regularized logistic regression

The code of the cost function in logistic regression is

```
J=-np.dot(np.log(utils.sigmoid(np.dot(X,theta))),y)/float(len(y))-
  np.dot((np.log(1-utils.sigmoid(np.dot(X,theta)))),(1-y))/float(len(y))
  +reg*np.sum(theta**2)/2/len(y)
```

The code of the gradient of the cost is

```
        grad=np.dot((utils.sigmoid(np.dot(X,theta))-y), X)/len(y)+reg/
len(y)*np.vstack([[0],theta[1:len(theta)].reshape(len(theta)-1,1)]).reshape(1,len(the
ta))[0]
```

And we can get the results as follows:

```
Theta found by fmin_bfgs:  [ 1.22936289  0.63244485  1.19453878 -1.974858   -0.91081349 -1.31288739
  0.1278769  -0.37280756 -0.38208417 -0.1659471  -1.45886406 -0.09425439
 -0.59816669 -0.28669511 -1.18162912 -0.2465451  -0.22356158 -0.06635879
 -0.26965296 -0.30821223 -0.50672174 -1.06313076 -0.0044257  -0.2889401
 -0.00588416 -0.31972433 -0.15633712 -0.98598224]
Final loss =  0.46378580834
```

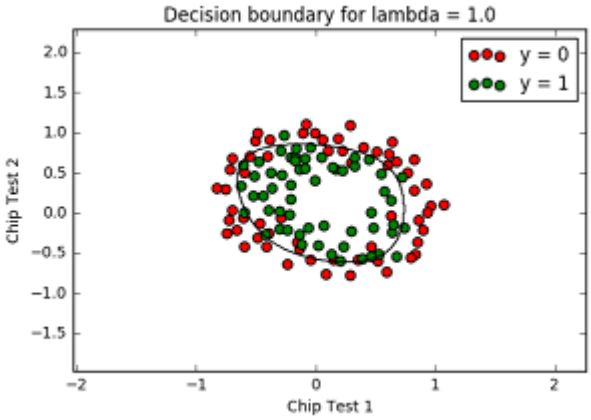

Fig.7 Training data with decision boundary for lambda = 1

The decision boundary for $\lambda = 1$ is shown in Fig.7.

2.2.2 Prediction using the model

The prediction is

```
y_pred=utils.bin_features(np.dot(X,self.theta))
```

And then calculate the accuracy by comparing predy and y. Accuracy on the training set = 0.830508474576.

2.2.3 Varying λ

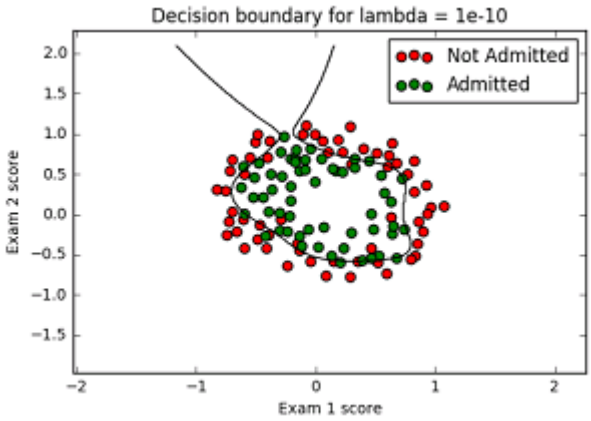

Fig.8 Training data with decision boundary for lambda = 1e-10

When $\lambda = 1e-10$, the decision boundary is shown in Fig.8.

Loss with sklearn theta in this circunstance is = 0.288688297553.

The Loss for $\lambda = 1e-10$ is smaller than the loss for $\lambda = 1$. But from Fig.5 we can see that the shape of the boundary is complicated and thus overfitting the data.

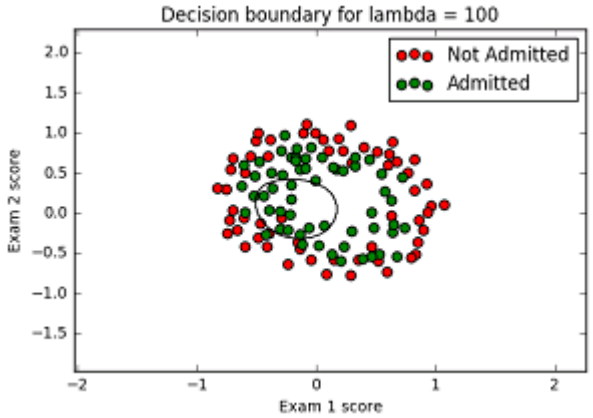

Fig.9 Training data with decision boundary for lambda = 100

When $\lambda = 100$, the decision boundary is shown in Fig.9.

Loss with sklearn theta in this circunstance is = 0.68061702032 .

The Loss for $\lambda = 100$ is bigger than the loss for $\lambda = 1$. From Fig.6 we can see that it isn't a good fit since the decision boundary couldn't seperate $\lambda = 0$ and $\lambda = 0$ so well, and thus under-fitting the data.

2.2.4 Exploring L1 and L2 penalized logistic regression

1) When $\lambda = 1e-10$, for L1 regularization,

[13.19262074 17.05679171 17.98667207 -130.06023689 -68.54331573 -57.11046281 -124.57574116 -181.31793228 -131.22278537 -66.67498367 443.8285247 434.67970794 591.80142823 275.27483618 109.13299628 213.69270243 430.17711434 588.53283175 526.38591995 281.95283652 98.95953208 -503.56853285 -753.16005167 -1206.75032914 -1062.6707846 -1030.08136553 -464.81150706 -125.05997931].

Loss with sklearn theta in this circumstance is = 0.241582677247.

For L2 regularization,

Theta found by sklearn with L1 reg:

[37.88232479 55.16468351 97.05718661 -366.11742011 -175.48255227
-192.34465906 -363.39467927 -833.87822194 -711.84451441 -505.74201103
1172.75644476 1267.91351361 1889.7642785 905.01882567 508.54517527
569.41179555 1615.8775152 2530.14066931 2887.85006006 1759.89516688
775.49260334 -1248.14525317 -2241.31588546 -4106.14478791 -4249.30884635
-4185.63356572 -2032.24073123 -741.21398609]

Loss with sklearn theta in this circumstance is = 0.219293553778 .

2) When $\lambda = 1$, for L1 regularization,

Theta found by sklearn with L1 reg:

[1.87011455 0.68677055 1.28054137 -4.86297952 -1.62201229 -2.34506375
0. 0. 0. 0. 0. 0. 0. 0. -2.36481728 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.]

Loss with sklearn theta in this circumstance is = 0.438133986889.

For L2 regularization,

[[1.1421418 0.60133123 1.16720816 -1.87175318 -0.91573602 -1.26954353
0.12666993 -0.36872263 -0.34519427 -0.1737859 -1.42387402 -0.04858047
-0.60640851 -0.26932551 -1.16317987 -0.24308611 -0.20707255 -0.0432088
-0.28027502 -0.28694562 -0.46911911 -1.03616566 0.02922704 -0.29262593
0.01734536 -0.32896299 -0.13795013 -0.93199552]]

Loss with sklearn theta in this circumstance is = 0.468432631539.

3) When $\lambda = 100$, for L1 regularization,

**Theta found by sklearn with L1 reg: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.].**

Loss with sklearn theta in this circumstance is = 0.69314718056.

For L2 regularization,

[0.00468643 -0.01726866 0.00641955 -0.05402668 -0.01327561 -0.03727149
-0.01821201 -0.0076104 -0.00885312 -0.02224573 -0.0428837 -0.00238587
-0.01393198 -0.00354832 -0.04072376 -0.0207858 -0.00467204 -0.0035498
-0.00624896 -0.00500397 -0.03153157 -0.03381516 -0.0010832 -0.00694193
-0.00039451 -0.00788596 -0.00157686 -0.04058855]

Loss with sklearn theta in this circumstance is = 0.680617011756.

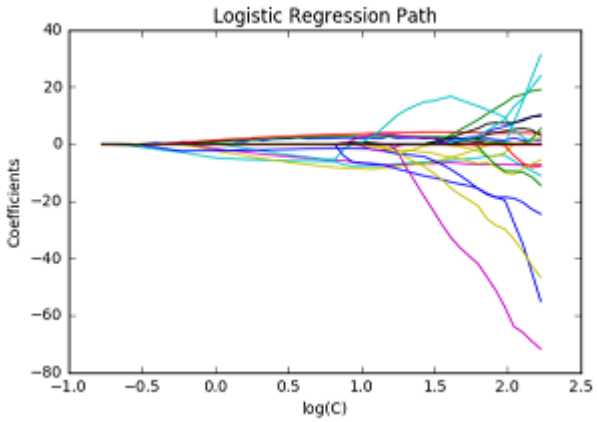

Fig.10 The logistic regression path

From the above data, we can see that for L1 regularization, the larger the λ is, the less the number of non-zero coefficients is. For L2 regularization, the larger the λ is, the closer the elements of the coefficient are to 0.

The loss for L1 regularization and L2 regularization are close to each other of the same λ . And for both L1 regularization and L2 regularization, the bigger the λ is, the bigger the loss is.

2.3 Logistic regression for spam classication

2.3.1 Feature transformation

1) Standardize features:

```
def std_features(X):
mu = np.mean(X,axis=0)
sigma = np.std(X,axis=0)
X_norm = (X - mu) / sigma
return X_norm, mu, sigma
```

2) Log transform features:

```
def log_features(X):
logf = np.zeros(X.shape)
logf=np.log(1+X)
return logf
```

3) Binarize features:

```
def bin_features(X):
tX = np.zeros(X.shape)
tX=np.where(X > 0, 1, 0)
return tX
```

2.3.2 Fitting regularized logistic regression models (L2 and L1)

1) For "std" with L2 regression:

best_lambda = 3.1

Coefficients = [-1.78181781] [[-0.01866499 -0.22087451 0.13045695 0.54959949 0.25976796 0.18567584 0.91334912 0.31058932 0.14206366 0.06141135 -0.05916022 -0.15322367 -0.05299372 0.02529411 0.2368854 0.77416088 0.47734971 0.0765331 0.26541873 0.22333128 0.26194678 0.41573354 0.7566006 0.2525873 -1.95449231 -0.6198795 -2.15472417 -0.10367899 -0.74932045 -0.15329143 -0.31104183 -0.20677323 -0.42104413 -0.49274202 -0.3765311 0.34288776 0.01709408 -0.14617388 -0.38989726 -0.10566088 -0.72265062 -1.03667715 -0.33733305 -0.76159976 -0.82397828 -1.24560991 -0.13763984 -0.72018041 -0.33921641 -0.15808517 -0.11526349 0.22628354 1.53555041 0.55853798 -0.15239951 0.92103955 0.38161314]]

Accuracy on set aside test set for std = 0.921875

2) For "logt" with L2 regression:

best_lambda = 1.1

Coefficients = [-4.50575491] [[-0.40935479 -0.27795478 -0.02701633 0.56259276 1.17127547 0.86303868 2.62418436 1.3857304 0.23160282 0.33391684 -0.26077718 -0.47746371 -0.56861136 0.35378058 0.55753631 1.50334495 1.27835828 0.14237831 0.36168092 0.57748645 0.5355757 0.34585533 1.87816184 1.46761657 -2.85217887 -0.74493414 -3.83688801 -0.12708452 -1.01843191 -0.21415669 -0.61832201 -0.2650045 -0.96253695 -0.34180064 -0.87412 1.07740416 -0.92149783 -0.28414598 -1.01642284 -0.60014002 -1.03727518 -2.04235221 -1.12927993 -1.57870834 -1.00931631 -2.53103313 -0.51300913 -1.57944258 -1.3999715 -0.56234564 -0.54017017 2.02515268 3.5030211 0.60454134 0.65627859 0.20338546 0.39217144]]

Accuracy on set aside test set for logt = 0.942708333333

3) For "bin" with L2 regression:

best_lambda = 1.1

Coefficients = [-1.83742964] [[-1.91463199e-01 -1.66872958e-01 -3.93802023e-01 2.39462779e-01 9.83292893e-01 1.75311414e-01 2.12183419e+00

7.92547596e-01 1.94566579e-01 3.34388296e-01 -2.90824615e-01 -4.20297341e-01 -9.06380381e-01 2.56299856e-01 5.15189474e-01 1.47014136e+00 8.76696476e-01 -8.32760956e-02 2.41264180e-01 5.01801273e-01 7.37046896e-01 1.15518007e+00 9.11195183e-01 1.36902984e+00 -2.35248856e+00 -4.17190306e-01 -3.79772643e+00 6.88337611e-01 -6.07237597e-01 -1.61622832e-01 -9.24671804e-01 -6.04558748e-01 -6.91161481e-01 -3.85638230e-02 -6.71440136e-01 3.52732370e-01 -1.05408408e+00 5.28551480e-01 -7.65306731e-01 -2.46067578e-01 -1.27643951e+00 -1.90613122e+00 -7.90184279e-01 -1.57619158e+00 -7.64312034e-01 -2.22366816e+00 -8.34144234e-02 -1.39371572e+00 -3.06993897e-01 2.00231957e-01 -1.70968577e-01 1.20762876e+00 1.45771409e+00 3.79908694e-02 5.31813494e-04 5.31813494e-04 5.31813494e-04]]

Accuracy on set aside test set for bin = 0.927734375

4) For "std" with L1 regression:

best_lambda = 3.1

Coefficients = [-2.11722355] [[-0.01612798 -0.17694905 0.12523865 0.32334392 0.25359601 0.1888663 0.91025235 0.29160398 0.15143888 0.05178758 -0.04233988 -0.14672014 -0.01983542 0.0106062 0.16824292 0.77678448 0.47435973 0.06229814 0.26045369 0.22713572 0.24820687 0.35352076 0.72794987 0.22512442 -2.57305367 -0.37349253 -4.84643842 -0.02844991 -0.5086759 0. 0. 0. -0.34824494 0. -0.09527807 0.29293866 0. -0.12662302 -0.34794345 -0.06091905 -0.39509061 -0.95242066 -0.23080425 -0.68053142 -0.7874126 -1.32803236 -0.10534206 -0.62272747 -0.28159212 -0.133717 -0.06007871 0.21366222 1.66866098 0.39241402 0. 0.74766639 0.3427959]]

Accuracy on set aside test set for std = 0.923828125

5) For "logt" with L1 regression:

best_lambda = 1.1

Coefficients = [-4.59100701] [[-4.31079221e-01 -1.30757253e-01 0.00000000e+00 2.68229808e-01 1.21200161e+00 7.91872887e-01 2.97219544e+00 1.42216833e+00 5.41110345e-02 3.38654254e-01 -8.03808680e-02 -4.81620345e-01 -4.78050029e-01 2.18226749e-01 0.00000000e+00 1.50277801e+00 1.43641850e+00 0.00000000e+00 3.72956564e-01 3.22285678e-01 4.95131654e-01 4.14471605e-01 1.86544675e+00 1.35050720e+00 -3.58757501e+00 -2.83769045e-01 -9.19261028e+00 0.00000000e+00 -8.07813255e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00 -8.48628918e-01 0.00000000e+00 -5.48707580e-01 1.09989557e+00 -7.49276158e-01 0.00000000e+00 -9.97195630e-01 0.00000000e+00 -7.55592950e-01 -2.72158032e+00 -1.31698140e+00 -1.99155671e+00 -1.23554613e+00 -3.07986922e+00 0.00000000e+00 -2.58453611e+00 -1.52522565e+00 -4.72818663e-01

0.00000000e+00 2.03642138e+00 5.69428860e+00 5.22949514e-03 6.98895539e-01 1.66895040e-01 4.26880709e-01]]

Accuracy on set aside test set for logt = 0.9453125

6) For "logt" with L1 regression:

best_lambda = 1.1

Coefficients = [-0.17033236] [[-1.36805151e-01 -1.10151227e-01 -3.58690700e-01 0.00000000e+00 1.00534621e+00 1.64663580e-01 2.24867636e+00 8.04848724e-01 1.70283910e-01 3.52358569e-01 -2.73628524e-01 -4.28605439e-01 -9.06322268e-01 1.84727318e-01 3.65273399e-01 1.48189828e+00 9.79519530e-01 -5.85048948e-02 1.91059140e-01 4.85241103e-01 7.13983839e-01 1.18670896e+00 8.63116673e-01 1.45287702e+00 -2.70674269e+00 -1.81011306e-01 -5.05237089e+00 7.01590134e-01 -3.67740966e-01 0.00000000e+00 -8.75548373e-01 0.00000000e+00 -6.45082455e-01 0.00000000e+00 -5.65047951e-01 2.73290690e-01 -1.03010240e+00 2.04256762e-01 -7.55376773e-01 -1.89174389e-03 -1.53644138e+00 -2.17691311e+00 -7.35298139e-01 -1.71793777e+00 -7.81602839e-01 -2.43496579e+00 0.00000000e+00 -1.62179358e+00 -3.00689612e-01 1.70317381e-01 -2.81970791e-02 1.20493261e+00 1.54112359e+00 0.00000000e+00 -6.66903117e-01 -1.93755987e-01 -7.65652386e-01]]

Accuracy on set aside test set for bin = 0.921875

So, from the data, we recommend for "logt" with L1 regression because it has the maximum accuracy.