Implement LARS for Linear and Lasso Regression with Application in Auto-MPG Data Set*

1st Yihang Li
Mathematics
University of Central Florida
Orlando, America
lyh970828@Knights.ucf.edu

I. Introduction

A. Literature Review

The Lasso proposed by Tibshirani(1996) [1], deals with multicollinearity and variable selection for OLS. It is widely used in statistical applications and is especially effective in the settings of low to moderate multicollinearity where the solution is believed to be sparse [2]. Though there is no closed form solution to compute the Lasso estimates, it can be generated by a minor modification of the LARS algorithm.

Least Angle Regression(LARS), proposed by (Efron et al. (2004)) [3], can be viewed as a form of stagewise variable selection algorithm with a little modification, it efficiently computes the entire sequence of Lasso solutions by exploiting the geometry of the Lasso Problem [2].

B. Our Goal

In this project, our goal is to write our own code to implement LARS algorithm to fit linear regression(its original form) and Lasso(its modification). And applying them in the Auto-MPG Data Set [4].

More specifically, for fitting the linear regression using LARS, we need to produce the plot of correlation and coefficient trajectories for all steps, for fitting the Lasso using LARS, we need to produce the coefficient trajectories plot(solution path), and choose the optimal value tuning parameter using proper method.

II. Implement LARS Algorithm

A. Symbolic Representation

Following Efron et al. (2004) [3], the symbolic representation can be seen in Appendix A.

B. Algorithm Summary

The LARS algorithm can be summarized as follows:

Algorithm 1: Least Angle Regression

1. Standardize X, y to the mean 0 and l_2 unit length. Start with $\hat{\mu}^0 = 0, \hat{\beta}^0 = 0$, and initialize $\mathbb{A}$ as empty set, $\mathbb{A}^c$ with indices range from 0 to $p-1$.
2. For $i = 1, ..., p$:

(a) Let $j = argmax_j|\hat{c}^{i-1}|$. Where $\hat{c}^{i-1} = X^T(y - \hat{\mu}^{i-1})$, then add j into $\mathbb{A}$, and remove it from $\mathbb{A}^c$.

(b) **[Find Direction]** : Let $s^{i-1} = sign(\hat{c}_{\mathbb{A}}^{i-1})$, $X_{\mathbb{A}}^{i-1} = X[:, \mathbb{A}] \times s^{i-1}$, $g_{\mathbb{A}}^{i-1} = (X_{\mathbb{A}}^{i-1})^T X_{\mathbb{A}}^{i-1}$, $A_{\mathbb{A}}^{i-1} = [I_{\mathbb{A}}^T (g_{\mathbb{A}}^{i-1})^{-1} I_{\mathbb{A}}]^{-1/2}$. Then the direction, equiangular vector, is $u_{\mathbb{A}}^{i-1} = X_{\mathbb{A}}^{i-1}\omega_{\mathbb{A}}^{i-1}$, where $\omega_{\mathbb{A}}^{i-1} = A_{\mathbb{A}}^{i-1}(g_{\mathbb{A}}^{i-1})^{-1} I_{\mathbb{A}}$.

(c) **[Find Stepsize]** : Define inner product $a^{i-1} = X^T u_{\mathbb{A}}^{i-1}$, then the stepsize is $\hat{\gamma}^{i-1} = min^+_{k \in \mathbb{A}^c}\{\frac{\hat{c}_j^{i-1} - \hat{c}_k^{i-1}}{A_{\mathbb{A}}^{i-1} - a^{i-1}}, \frac{\hat{c}_j^{i-1} + \hat{c}_k^{i-1}}{A_{\mathbb{A}}^{i-1} + a^{i-1}}\}$.

(d) **[Update the estimates]** : $\hat{\mu}^i = \hat{\mu}^{i-1} + \hat{\gamma}^{i-1} u_{\mathbb{A}}^{i-1}$, also, we have $\hat{\beta}^i = \hat{\beta}^{i-1} + \hat{\gamma}^{i-1} s^{i-1} \omega_{\mathbb{A}}^{i-1}$.

3. Return $\hat{\beta}^p$.

C. Remark

For every step, LARS algorithm find the predictor most correlated with the current residual. In geometric view, it is equivalent to find the predictor that has least angle with the current residual. As for obtaining the direction and stepsize, this can be seen in Efron et al. (2004) [3].

III. Implement Lasso Modification

Suppose we have just completed a LARS step, then the Lasso modification can be summarized as follows:

Algorithm 2: Lasso Modification

- In the 2.(c) **[Find Stepsize]** : Let $\hat{d}^{i-1}$ be the p-vector equaling $s^{i-1}\omega_{\mathbb{A}}^{i-1}$ for $\mathbb{A}$ and 0 for $\mathbb{A}^c$. Let $z_k^{i-1} = -\frac{\hat{\beta}_k^i}{\hat{d}_k^{i-1}}$ for $k \in \mathbb{A}$. And $\hat{z}^{i-1} = min_{z_k^{i-1}>0}\{z_k^{i-1}\}$.
- If $\hat{z}^{i-1} < \hat{\gamma}^{i-1}$, set $\hat{\gamma}^{i-1} = \hat{z}^{i-1}$, remove k from $\mathbb{A}$ and add it into $\mathbb{A}^c$.
- Then back into this iteration's 2(b) **[Find Direction]** , get the $u_{\mathbb{A}}^{i-1}$
- Continue the 2(d) and the next iteration.

A. Remark

In my perspective, as the LARS is similar to stagewise regression or forward regression, the Lasso based on modifying LARS is extremely similar to stepwise regression because it has one more step of removing index. In this algorithm, there seems no parameter to tune, however, we can still try to control the iteration.

IV. Data Preparation

A. Data Preview

By using pandas.read csv, we successfully load the data as autompg. Fig 1 is the first five rows of Auto-MPG Data Set. The first column is our response y and the next 6 columns are our predictors X.

B. Data Clean

Then we thy to check the data type, only to find the abnormal of the type of horsepower, which should have be int64 or float64 like the other predictors, but get the object type. What reason makes it different? In order to explore this problem, we convert the horsepower column into numeric by using pd.to numeric(), and set its parameter errors as 'coerce', this will return a NAN for any data that can not be convert. Finally, we find there are 6 invalid values, and back to the original data set located by their index, only to find they are '?'. Since the missing value's totall number are just 6, so, one solution is to delete the rows that include them. Checking the type again, now the type of horsepower is float64.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

Fig. 1. Header of Auto-MPG Data Set

V. LARS Application

Let's use our self-coding LARS to fit on the data. Table I is the correlation among our predictors for every iteration. Fig 2 is its graphs. Fig 3 is our coefficients' trajectory.

TABLE I
Correlation by iteration

iter time	cyl	disp	hp	kg	acc	yr
0	-0.778	-0.805	-0.778	-0.832	0.423	0.581
1	-0.451	-0.465	-0.463	-0.468	0.271	0.468
2	-0.254	-0.259	-0.261	-0.261	0.160	0.261
3	-0.109	-0.107	-0.109	-0.109	0.073	0.109
4	-0.024	-0.020	-0.024	-0.024	0.024	0.024
5	-0.003	0.003	-0.003	-0.003	-0.003	-0.003

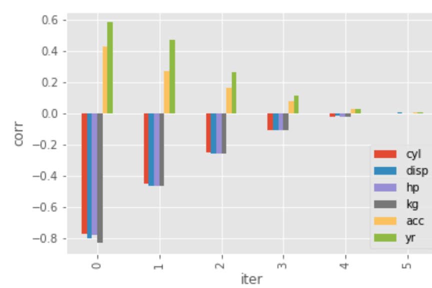

Fig. 2. Correlation among predictors for every iteration

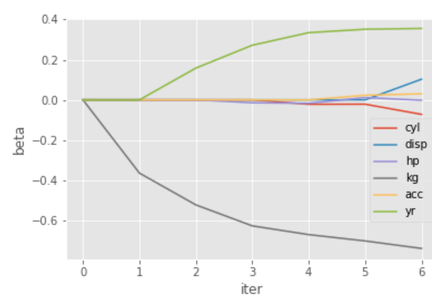

Fig. 3. Trajectory of coefficients for LARS

VI. Lasso Application

After tuning the iteration times, Fig 4 is the trajectory of our coefficients.

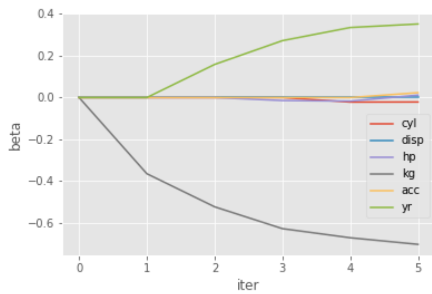

Fig. 4. Trajectory of coefficients for Lasso

VII. Compared with LassoLars in sklearn

We import linear_model from sklearn. Let reg = linear_model.LassoLarsCV() and set cv=10, n_jobs=3, max_iter = 200, normalize=True.

By 10-fold Cross-Validation, we get the best alpha is 0.000029 given the smallest mse, 0.000556. Then we let reg = linear_model.LassoLars(), set alpha = 0.000029. And reg.fit(X,y) to fit on our Auto-MPG Data Set. Fig 5 is our coefficients' trajectory.

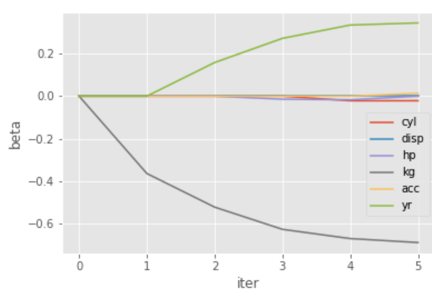

Fig. 5. Trajectory of coefficients for Lasso by sklearn

VIII. Conclusion

After this project, I stand at a new point to view the Lasso Regression. Also, I deeply get familiar with the LARS algorithm and exercised my coding ability by using python to implement the algorithm in the python class and inheritance. Finally, I used what I code by exploring on the Auto-MPG Data Set.

References

[1] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[2] Y. Samizo, "Secure statistical analysis on vertically distributed databases," 2016.

[3] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[4] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

Appendix A
Symbolic Representation

- X : $n \times p$ matrix of all predictors.
- y : response variable with length n.
- $\hat{\beta}$: current solution with length p.
- $\hat{\mu} = X\hat{\beta}$: the current estimate.
- $\hat{c} = c(\hat{\mu}) = X^T(y - \hat{\mu})$: the length p vector of current correlations
- $\mathbb{A} = argmax_j\{j : |\hat{c_j}|\}$: Active set. (Also denote $\mathbb{A}^c$ as Inactive set)
- $X_{\mathbb{A}}$: $n \times |\mathbb{A}|$ matrix of $|\mathbb{A}|$ predictors, where $|\mathbb{A}|$ denote the number of elements in $\mathbb{A}$
- $s_{=}sign$

Appendix B
All My Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
class LeastAngleRegression:

   def __init__(self):

      self.beta = None
      self.beta0 = None
      self.est = None
      self.Active_set = None
      self.steps = None
      self.corr = None


   @staticmethod
   def Standardize(x, y):
      """
      x and y both have mean 0 and unit
          length, use L_2 norm to
          normarlize it.
      """
      x = np.asanyarray(x, dtype =
          np.float) #asanyarray: The
          input will be returned uncopied
      #iff it's a compatible ndarray or
          subclass like matrix
          (copy=False, subok=True).
      y = np.asanyarray(y, dtype =
          np.float)
      #center x, y
      x_mean = np.mean(x, axis = 0)
      x -= x_mean
      y_mean = np.mean(y, axis = 0)
      y -= y_mean
      #l2-normarlize x and y!
      x_norm = np.sum(x**2, axis = 0)**0.5
      x /= x_norm
      y_norm = np.sum(y**2, axis = 0)**0.5
      y /= y_norm
      return x, y

   def fit(self, x, y, max_iter = None):
      """
      Fit LARS
      """
      n_samples, n_features = x.shape
      x, y = self.Standardize(x, y)
          #Standardize our data
      Gram = np.dot(x.T, x)# Precompute
          x^T*x


      if max_iter is not None:
         if max_iter <= 0:
            raise ValueError("max_iter
                must be > 0")

      if (max_iter == None) or (max_iter
          > n_features):
         max_iter = n_features

      #We begin at mu_0 = 0 and build mu
          by steps.
      mu = np.zeros(n_samples)

      #Our two index sets
      Active_set = []
      Inactive_set =
          list(range(n_features))

      #At first, all betas are zero, so
          are the ests
      beta = np.zeros(n_features, dtype =
          np.float)
      est = np.zeros((max_iter + 1,
          n_features), dtype=np.float)

      #initialize our corr path
      corr = list()

      for i in range(max_iter):

         # the vector of current
             correlations
         c = np.dot(x.T, (y - mu))

         # corr path
         corr.append(c)

         # the Active_set is the set of
             indices corresponding to
             covariates
         #with the greatest absolute
             current correlations
         ct = c.copy()
         ct[Active_set] = 0 # avoid
```

```
        reselection
    ct_abs = np.abs(ct)
    j = np.argmax(ct_abs)
    C = ct_abs[j]
    Active_set.append(j)
    Inactive_set.remove(j)

    # let s_j = sign(c^hat_j)
    s = np.sign(c[Active_set])

    # each predictor in the
        Active_set face the
        direction of their
        correlation with the current
        residual
    # and collecting them in a
        matrix x_A
    x_A = x[:, Active_set] * s

    # compute gram matrix and its
        inverse, also the A_A, which
        is a scalar that satisfies
        the equal angle equation
    g_A = np.dot(x_A.T, x_A)
    g_A_inverse = np.linalg.inv(g_A)
    A_A =
        (np.sum(g_A_inverse))**(-0.5)

    # compute the equiangular vector
        (#direction)
    w_A = np.sum(A_A * g_A_inverse,
        axis=1)
    u_A = np.dot(x_A, w_A)

    # define inner product vector a
        between each predictor and
        u_A.
    a = np.dot(x.T, u_A)

    # gamma_hat (#stepsize)
    g1 = (C - c[Inactive_set])/(A_A
        - a[Inactive_set])
    g2 = (C + c[Inactive_set])/(A_A
        + a[Inactive_set])
    #Join a sequence of arrays along
        an existing axis.
    g = np.concatenate((g1, g2))
    # the positive components
    g = g[g > 0.0]
    if g.shape[0] == 0:
       #the discussion after
           (2.22),gamma_hat is not
           defined since Active_set
           contain all covariates
       gammahat = C / A_A
    else:
       #by definition of (2.13)
       gammahat = np.min(g)

    # the new estimate of
        beta^hat_j, for j belongs
        Active_set
    beta[Active_set] =
        beta[Active_set] + gammahat
        * w_A*s

    est[i+1, Active_set] =
        beta[Active_set]

    # augment our current estimate
        mu, where gammahat is
        stepsize, and u_A is
        direction
    mu = mu + (gammahat * u_A)

 self.Active_set, self.est,
    self.steps, self.corr =
    np.asarray(Active_set), est,
    max_iter, corr

#      self.est /= np.sum(x**2, axis =
    0)**0.5
 self.beta = np.copy(self.est[-1])
 self.beta0 = np.mean(y, axis = 0) -
    np.dot(np.mean(x, axis = 0),
    self.beta)

def prediction(self, x):

   tarr = np.asarray(x, dtype=np.float)

   if tarr.ndim > 2 or tarr.ndim < 1:
      raise ValueError("x must be an
          1d or a 2d array_like
          object")

   try:
      pre = np.dot(tarr, self.beta) +
          self.beta0
   except ValueError:
      raise ValueError("x, beta: shape
          mismatch")

   return pre

def get_est(self):

   return self.est

def get_Active(self):

   return self.Active_set

def get_beta(self):

   return self.beta

def get_beta0(self):

   return self.beta0
```

```
    def get_steps(self):

        return self.steps

    def plot_corr(self, columns):
        """
        Plot the all step's correlation

        :Parameters:
          columns : the columns of X
        """
        if self.corr != None:
            corr_pd =
                pd.DataFrame(self.corr,
                columns = columns)
            try:
                corr_pd.plot.bar()
                plt.xlabel('iter')
                plt.ylabel('corr')
            except Exception as e:
                print(e)

    def plot_trajectory(self, columns):
        """
        Plot the trajectory

        :Parameters:
         columns : the columns of X

        """
        try:
            for i in self.get_est().T:
                plt.plot(i)
            plt.legend(columns, loc='center
                left', bbox_to_anchor=(0.8,
                0.37))
            plt.xlabel('iter')
            plt.ylabel('beta')
        except Exception as e:
            print(e)
```

```
#Load our Data
autompg =
    pd.read_csv('DataSet/auto-mpg.csv')
#To see the fist five rows
autompg.head()
#split x and y
y = autompg['mpg']
x = autompg.iloc[:, 1:7]
#check if there is something wrong
x.dtypes
#If 'coerce', then invalid parsing will
    be set as NaN
x.horsepower =
    pd.to_numeric(x.horsepower,
    errors='coerce')
print(x[pd.isnull(x).any(axis=1)])
x.drop(x.index[[32, 126, 330, 336, 354,
    374]], inplace = True)
y.drop(y.index[[32, 126, 330, 336, 354,
    374]], inplace = True)
x.dtypes
columns = x.columns
columns
x = np.asarray(x, dtype = float)
y = np.asarray(y, dtype = float)
#Let's try to use our self-coding LARS to
    fit on the data
lar1 = LeastAngleRegression()
lar1.fit(x, y)
lar1.corr
columns = ['cyl', 'disp', 'hp', 'kg',
    'acc', 'yr']
corr_pd = pd.DataFrame(lar1.corr, columns
    = columns)
lar1.plot_corr(columns)
plt.savefig('corr_lar.png')
lar1.plot_trajectory(columns)
plt.savefig('traj_lars.png')
```

```
class Lasso_by_Lars(LeastAngleRegression):

    def __init__(self):
        super(LeastAngleRegression,
            self).__init__()
        self.d = None
        self.trajectory = None


    def fit(self, x, y, max_iter = None):
        """
        Overriding the fit method
        Fitting Lasso by using lars
        """
        n_samples, n_features = x.shape
        x, y = self.Standardize(x, y)
            #Standardize our data
        Gram = np.dot(x.T, x)# Precompute
            x^T*x


        if max_iter is not None:
            if max_iter <= 0:
                raise ValueError("max_iter
                    must be > 0")

        if (max_iter == None) or (max_iter
            > n_features):
            max_iter = n_features

        #We begin at mu_0 = 0 and build mu
            by steps.
        mu = np.zeros(n_samples)

        #Our two index sets
```

```
Active_set = []
Inactive_set =
    list(range(n_features))

#At first, all betas are zero, so
    are the ests
beta = np.zeros(n_features, dtype =
    np.float)
est = np.zeros((max_iter + 1,
    n_features), dtype=np.float)

#initialize our corr path
corr = list()

#initialize direction d
d = list()


for i in range(max_iter):

   # the vector of current
       correlations
   c = np.dot(x.T, (y - mu))

   # corr path
   corr.append(c)

   # the Active_set is the set of
       indices corresponding to
       covariates
   #with the greatest absolute
       current correlations
   ct = c.copy()
   ct[Active_set] = 0 # avoid
       reselection
   ct_abs = np.abs(ct)
   j = np.argmax(ct_abs)
   C = ct_abs[j]
   Active_set.append(j)
   Inactive_set.remove(j)

   # let s_j = sign(c^hat_j)
   s = np.sign(c[Active_set])

   # each predictor in the
       Active_set face the
       direction of their
       correlation with the current
       residual
   # and collecting them in a
       matrix x_A                    #
   x_A = x[:, Active_set] * s

   # compute gram matrix and its
       inverse, also the A_A, which
       is a scalar that satisfies
       the equal angle equation
   g_A = np.dot(x_A.T, x_A)
   g_A_inverse = np.linalg.inv(g_A)
   A_A =
       (np.sum(g_A_inverse))**(-0.5)

   # compute the equiangular vector
       (#direction)

   w_A = A_A * np.sum(g_A_inverse,
       axis=1)
   u_A = np.dot(x_A, w_A)
       #equiangular vector

   # define inner product vector a
       between each predictor and
       u_A.
   a = np.dot(x.T, u_A)

   # gamma_hat (#stepsize)
   g1 = (C - c[Inactive_set])/(A_A
       - a[Inactive_set])
   g2 = (C + c[Inactive_set])/(A_A
       + a[Inactive_set])
   g = np.concatenate((g1, g2))
       #Join a sequence of arrays
       along an existing axis.
   g = g[g > 0.0] # the positive
       components
   if g.shape[0] == 0:
      gammahat = C / A_A #the
          discussion after
          (2.22),gamma_hat is not
          defined since Active_set
          contain all covariates
   else:
      gammahat = np.min(g)#by
          definition of (2.13)

   # the new estimate of
       beta^hat_j, for j belongs
       Active_set
   beta_copy = beta.copy()
   beta[Active_set] =
       beta[Active_set] + gammahat
       * w_A*s

   # define d_hat to be the
       m-vector equaling s_j *
       w_A_j for j belongs
       Active_set and 0 elsewhere
   d_hat = np.zeros(n_features,
       dtype = np.float)
       #initialize d_hat
     print(len(s))
   d_hat[Active_set] = s*w_A
   d.append(d_hat)

   # for gamma that make beta
       change sign(where beta ==
       0), rename as z
   z = -beta[Active_set]/
   d_hat[Active_set]
   z_p = z[z > 0]
```

```
        if z_p.shape[0] == 0:
            z_tuta = np.inf
        else:
            z_tuta = np.min(z_p)

        Remove = False
        #Lasso modification
        if z_tuta < gammahat:
            Remove = True
            # some betas have changed sign
            index = list(z).index(z_tuta)
            #stop the ongoing LARS step
                at:
            gammahat = z_tuta
            #remove j_tuta from the
                calculation of the next
                equiangular direction
            Active_set.remove(index)
            Inactive_set.append(index)
            ### Since Active Set changed,
                these things should be
                revised!
            s = np.sign(c[Active_set])

            # each predictor in the
                Active_set face the
                direction of their
                correlation with the
                current residual
            # and collecting them in a
                matrix x_A
            x_A = x[:, Active_set] * s

            # compute gram matrix and its
                inverse, also the A_A,
                which is a scalar that
                satisfies the equal angle
                equation
            g_A = np.dot(x_A.T, x_A)
            g_A_inverse =
                np.linalg.inv(g_A)
            A_A =
                (np.sum(g_A_inverse))**(-0.5)

            # compute the equiangular
                vector (#direction)

            w_A = A_A *
                np.sum(g_A_inverse,
                axis=1)
            u_A = np.dot(x_A, w_A)
            #update mu and beta
            mu = mu + gammahat*u_A
            beta[Active_set] =
                beta_copy[Active_set] +
                gammahat*w_A*s

        if Remove == False:
            # augment our current
                estimate mu, where
                gammahat is stepsize, and
                u_A is direction
            mu = mu + (gammahat * u_A)

        est[i+1, Active_set] =
            beta[Active_set]



    self.Active_set, self.est,
        self.steps, self.corr, self.d =
        np.asarray(Active_set), est,
        max_iter, corr, d

#       self.est /= np.sum(x**2, axis =
    0)**0.5
    self.beta = np.copy(self.est[-1])
    self.beta0 = np.mean(y, axis = 0) -
        np.dot(np.mean(x, axis = 0),
        self.beta)
lasso = Lasso_by_Lars()
lasso.fit(x, y)
try:
    for i in lasso.get_est().T:
        plt.plot(i[:-1])
    plt.legend(columns, loc='center left',
        bbox_to_anchor=(0.8, 0.37))
    plt.xlabel('iter')
    plt.ylabel('beta')
except Exception as e:
    print(e)
plt.savefig('lasso.png')
```

```
from sklearn import linear_model
reg = linear_model.LassoLarsCV(
    cv=10, n_jobs=3, max_iter = 200,
        normalize=True)
reg.fit(x, y)
index = np.where(reg.cv_alphas_ ==
    reg.alpha_)
_mse_v = np.mean(reg.cv_mse_path_[index,
    :])
print("mse value: %f" % _mse_v)

print("best alpha: %f" % reg.alpha_)
best_alpha = reg.alpha_
reg = linear_model.LassoLars(
alpha=best_alpha)
reg.fit(x, y)
plt.plot(reg.coef_path_.T)
plt.legend(columns, loc='center left',
    bbox_to_anchor=(0.8, 0.37))
plt.xlabel('iter')
plt.ylabel('beta')
plt.savefig('sklearn.png')
```