Regression Models with **Regularization**

Cesar Acosta Ph.D.

OVERVIEW

- Norm of a vector
- Linear Regression Loss function
- Ridge and LASSO regression
- Example: Regression with regularization
- Logistic Regression Loss function
- Example: Logistic regression with regularization

INTRODUCTION

Regression models with regularization

- Ridge regression (L2 regularization)
- LASSO regression (L1 regularization)
- Elastic net regression

Ridge and LASSO Regression Models

Cesar Acosta Ph.D.

LOSS FUNCTION = COST FUNCTION

Linear Regression loss function (sum of squared errors)

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Find $b_0, \ldots, b_p$
that minimize SSE

LOSS FUNCTIONS

Linear Regression loss function

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Find $b_0, \ldots, b_p$
that minimize SSE

Ridge Regression loss function

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{i=1}^{p} b_i^2$$

penalty

This term prevents
large $b_1, \ldots, b_p$

RIDGE REGRESSION

Ridge Regression loss function

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{i=1}^{p} b_i^2$$

- α is the *regularization* parameter
- If α = 0 (no regularization)

LASSO REGRESSION

LASSO Regression model

loss function

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{i=1}^{p} |b_i|$$

- This loss function prevents large regression coefficients
- If α is large, some regression coefficients are equal to zero resulting in a model with less predictors

RIDGE AND LASSO LOSS FUNCTIONS

.

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\|b\|_2^2$$

$$\underset{b_0,\ldots,b_p}{\text{Min}} \quad SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\|b\|_1$$

Example
Baseball Players

Shrinkage Methods

- The Hitters.csv file includes data about baseball players, such as their salary and 19 player's performance measures
- To predict the player's salary we will fit regression models with regularization
- We start by removing all rows with missing values in column Salary

Shrinkage Methods

- Fit 100 ridge regression models with $10^{-2} < \alpha < 10^{10}$
- Show how the coefficients $b_1, b_2 \ldots, b_{19}$ shrink when α increases
- Find the best value for α using
  - holdout cross validation
  - 5-fold cross validation
- Use the best α value to fit a ridge regression model
- Compute the test MSE
- **Repeat with LASSO regression**

Lasso Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# for holdout cv
from sklearn.model_selection import train_test_split

# for K-fold cv
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from sklearn.linear_model import Ridge, Lasso
from sklearn.linear_model import LinearRegression
```

EXAMPLE – BASEBALL PLAYERS

```
df = pd.read_csv('Hitters.csv')
df.shape
```

```
(322, 20)
```

first 16 columns

```
df[:5]
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts
0	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805

EXAMPLE – BASEBALL PLAYERS

```
df = pd.read_csv('Hitters.csv')
df.shape
```

```
(322, 20)
```

last 15 columns

```
df.iloc[:5,-15:]
```

Y

	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
0	14	1	293	66	1	30	29	14	A	E	446	33	20	NaN	A
1	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0	N
2	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0	A
3	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0	N
4	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	N

```
# drop NAs
```

```
d0 = df.dropna()
d0.shape
```

```
(263, 20)
```

EXAMPLE – BASEBALL PLAYERS

```
df = pd.read_csv('Hitters.csv')
df.shape
```

```
(322, 20)
```

```
df.iloc[:5,-15:]
```

	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
0	14	1	293	66	1	30	29	14	A	E	446	33	20	NaN	A
1	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0	N
2	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0	A
3	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0	N
4	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	N

```
# drop NAs
```

```
d0 = df.dropna()
d0.shape
```

```
(263, 20)
```

EXAMPLE – BASEBALL PLAYERS

```
y = d0.Salary
x0 = d0.drop(['Salary'],axis=1)
```

```
x0.dtypes
```

AtBat	int64
Hits	int64
HmRun	int64
Runs	int64
RBI	int64
Walks	int64
Years	int64
CAtBat	int64
CHits	int64
CHmRun	int64
CRuns	int64
CRBI	int64
CWalks	int64
League	object
Division	object
PutOuts	int64
Assists	int64
Errors	int64
NewLeague	object

EXAMPLE – BASEBALL PLAYERS

```
y = d0.Salary
x0 = d0.drop(['Salary'],axis=1)
```

```
# substitute categorical cols with dummy vars

x = pd.get_dummies(x0,
                   columns = ['League','Division','NewLeague'],
                   drop_first=True)
```

Assists	Errors	League_N	Division_W	NewLeague_N
43	10	1	1	1
82	14	0	1	0
11	3	1	0	1
40	4	1	0	1

```
X = x.astype('float64')
```

```
x.dtypes
```

```
AtBat          int64
Hits           int64
HmRun          int64
Runs           int64
RBI            int64
Walks          int64
Years          int64
CAtBat         int64
CHits          int64
CHmRun         int64
CRuns          int64
CRBI           int64
CWalks         int64
PutOuts        int64
Assists        int64
Errors         int64
League_N       uint8
Division_W     uint8
NewLeague_N    uint8
```

EXAMPLE – BASEBALL PLAYERS

Lasso regression

Lasso Regression

```
df = pd.read_csv('Hitters.csv')
d0 = df.dropna()
```

```
y = d0.Salary
x0 = d0.drop(['Salary'],axis=1)
```

```
x = pd.get_dummies(x0,
                   columns = ['League','Division','NewLeague'],
                   drop_first=True)
```

```
# Create a 1D array of 100 alpha values
# ranging from very small to very large
```

```
alphas = 10**np.linspace(10,-2,100)
```

Lasso Regression

```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
model = Lasso(max_iter = 10000)
```

```
coefs = []
for a in alphas:
    model.set_params(alpha = a)
    model.fit(X_scaled, y)
    coefs.append(model.coef_)
```

Lasso Regression

```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
model = Lasso(max_iter = 10000)
```

```
coefs = []
for a in alphas:
    model.set_params(alpha = a)
    model.fit(X_scaled, y)
    coefs.append(model.coef_)
```

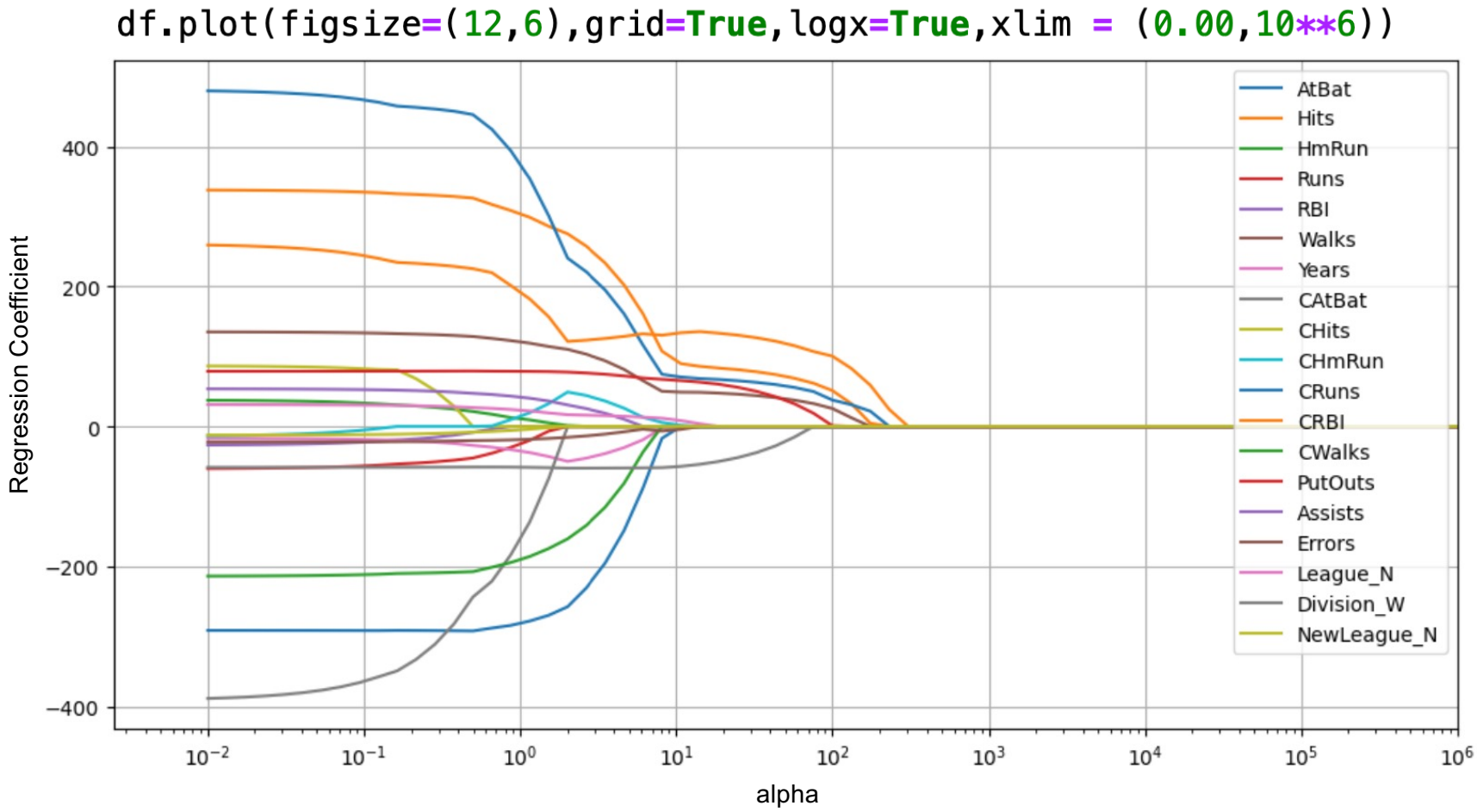
```
# Store regression coeffs in a DataFrame
df = pd.DataFrame(coefs)
df.columns = x.columns
df.index = alphas
df.index.name = 'alpha'
```

alpha	AtBat	Hits	HmRun	Runs
1.000000e+10	0.000000	0.000000	0.000000	0.000000
7.564633e+09	0.000000	0.000000	0.000000	0.000000
5.722368e+09	0.000000	0.000000	0.000000	0.000000
4.328761e+09	0.000000	0.000000	0.000000	0.000000
3.274549e+09	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...
3.053856e-02	-291.162954	336.804902	36.485262	-59.279202
2.310130e-02	-291.145771	337.053896	36.818406	-59.593809
1.747528e-02	-291.133099	337.242730	37.070513	-59.832017
1.321941e-02	-291.123510	337.385571	37.261221	-60.012210
1.000000e-02	-291.116409	337.493850	37.405530	-60.148623

100 rows × 19 columns

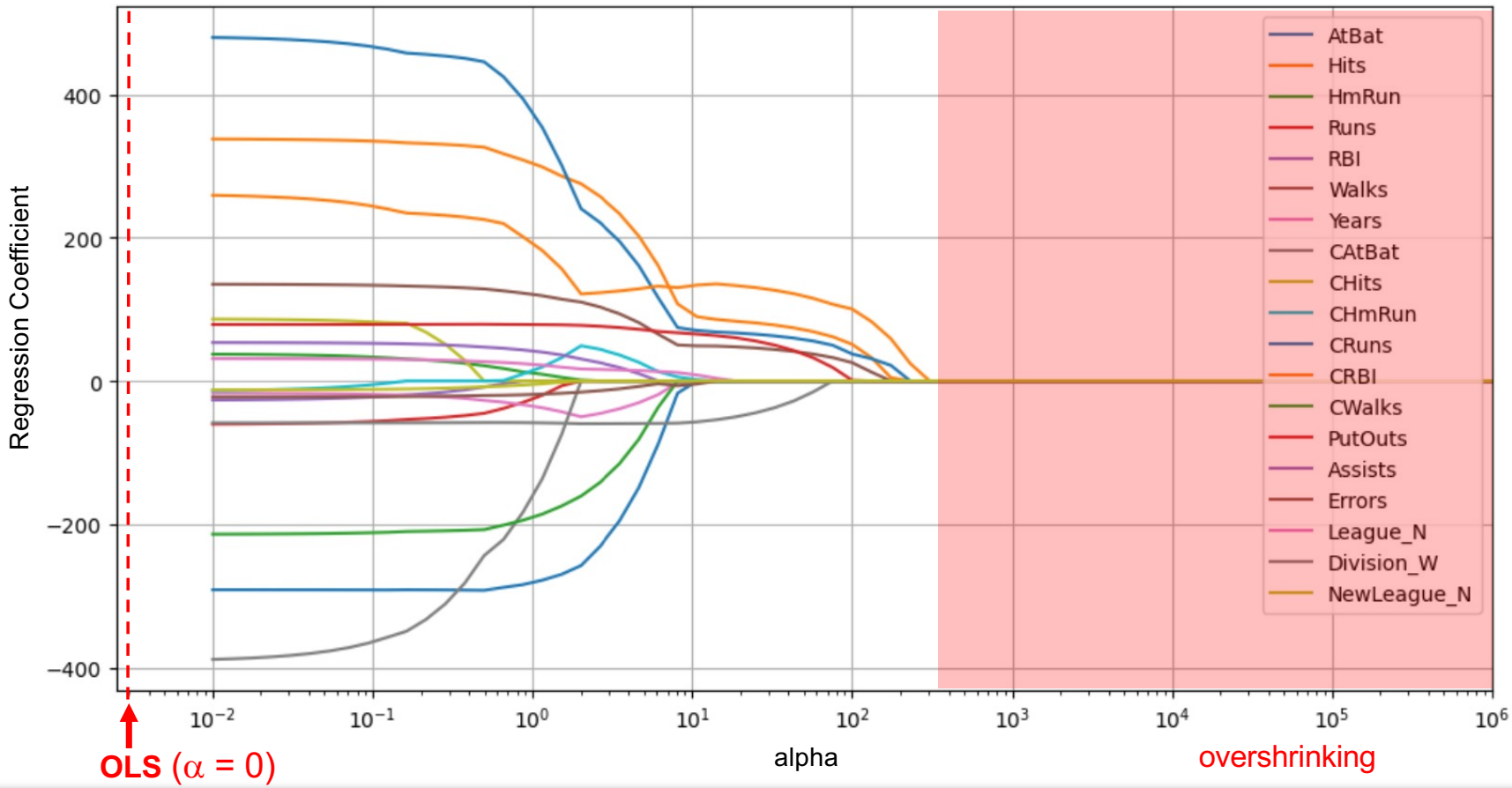
Lasso Regression Coefficients

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.00,10**6))
```



Lasso Regression Coefficients

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.00,10**6))
```



Lasso regression
-Holdout Cross Validation-

Holdout cross validation with alpha known

```
X_train,X_test,y_train,y_test = train_test_split(X,y,
                                                 test_size=0.5,
                                                 random_state=1)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

fit model with alpha = 4

```
lasso2 = Lasso(alpha=4)
lasso2.fit(X_train_scaled, y_train)
pred2 = lasso2.predict(X_test_scaled)
mspe = mean_squared_error(y_test, pred2)
mspe
```

```
105286.61132387113
```

fit model with huge alpha = 10^9

```
lasso3 = Lasso(alpha=10**9)
lasso3.fit(X_train_scaled, y_train)
pred3 = lasso3.predict(X_test_scaled)
mspe = mean_squared_error(y_test, pred3)
mspe
```

```
172862.23592080915
```

Holdout cross validation with alpha known

fit model with alpha = 4

```
lasso2 = Lasso(alpha=4)
lasso2.fit(X_train_scaled, y_train)
pred2 = lasso2.predict(X_test_scaled)
mspe = mean_squared_error(y_test, pred2)
mspe
```

```
105286.61132387113
```

```
# cannot use Lasso(alpha=0)
# so use LinearRegression()
```

```
ols_model = LinearRegression()
ols_model.fit(X_train_scaled, y_train)
pred = ols_model.predict(X_test_scaled)
ols_mse = mean_squared_error(y_test, pred)
ols_mse
```

```
116690.46856661158
```

fit model with huge alpha = 10^9

```
lasso3 = Lasso(alpha=10**9)
lasso3.fit(X_train_scaled, y_train)
pred3 = lasso3.predict(X_test_scaled)
mspe = mean_squared_error(y_test, pred3)
mspe
```

```
172862.23592080915
```

Holdout Cross Validation searching for alpha

```
X_nontest,X_test,y_nontest,y_test = train_test_split(X,y,
                                                     test_size=0.40,
                                                     random_state=1)

X_train,X_validation,y_train,y_validation = train_test_split(X_nontest,y_nontest,
                                                     test_size=0.5,
                                                     random_state=1)
```

Lasso Regression – Holdout Cross Validation searching for alpha

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_validation_scaled = scaler.transform(X_validation)
```

```
lasso_model = Lasso(max_iter = 10000)
```

```
validation_mspes = []

for a in alphas:
    lasso_model.set_params(alpha = a)
    lasso_model.fit(X_train_scaled, y_train)
    yhat = lasso_model.predict(X_validation_scaled)
    mspe = mean_squared_error(y_validation,yhat)
    validation_mspes.append(mspe)
```

```
df = pd.DataFrame(validation_mspes,
                  columns = ['Valid_MSE'])
df.index = alphas
df.index.name = 'alpha'
df
```

alpha	Valid_MSE	
1.000000e+10	161825.800308	
7.564633e+09	161825.800308	
5.722368e+09	161825.800308	
4.328761e+09	161825.800308	
3.274549e+09	161825.800308	
.		
.		
.		
0.000000	116690.468566	OLS

Lasso Regression – Holdout Cross Validation searching for alpha

```
df = pd.DataFrame(validation_mspes,
                  columns = ['Valid_MSE'])
df.index = alphas
df.index.name = 'alpha'
```

alpha	Valid_MSE
1.000000e+10	161825.800308
7.564633e+09	161825.800308
5.722368e+09	161825.800308
4.328761e+09	161825.800308
3.274549e+09	161825.800308
...	...
3.053856e-02	112731.183827
2.310130e-02	112783.012264
1.747528e-02	112822.687475
1.321941e-02	112852.966674
1.000000e-02	112876.025516

```
min1 = df.Valid_MSE.min()
min1
```

```
77542.23372579267
```

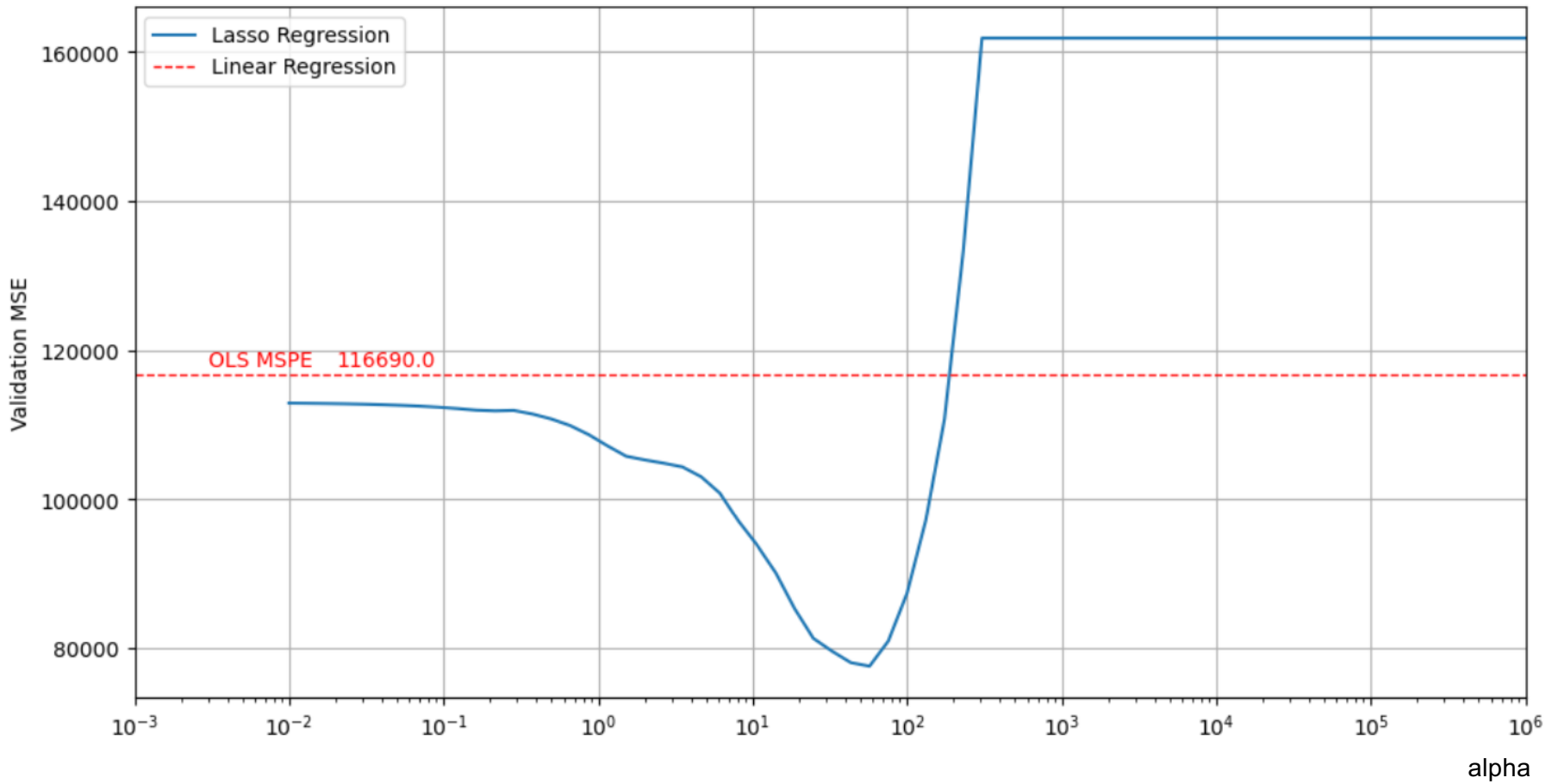
```
df[df.Valid_MSE == min1]
```

alpha	Valid_MSE
57.223677	77542.233726

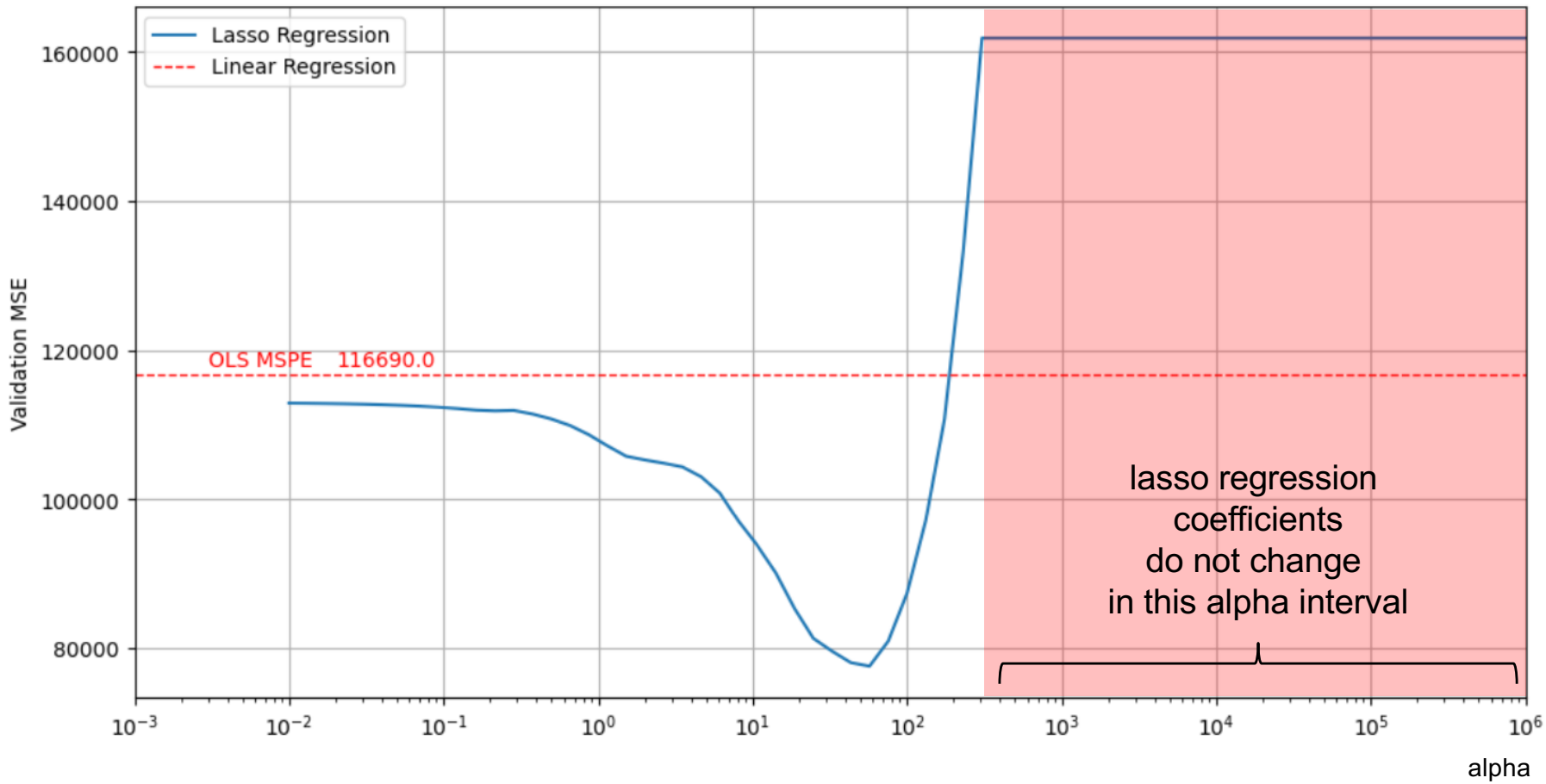
```
# best alpha giving min Validation MSE
best_alpha = df.Valid_MSE.idxmin()
best_alpha
```

```
57.2236765935022
```

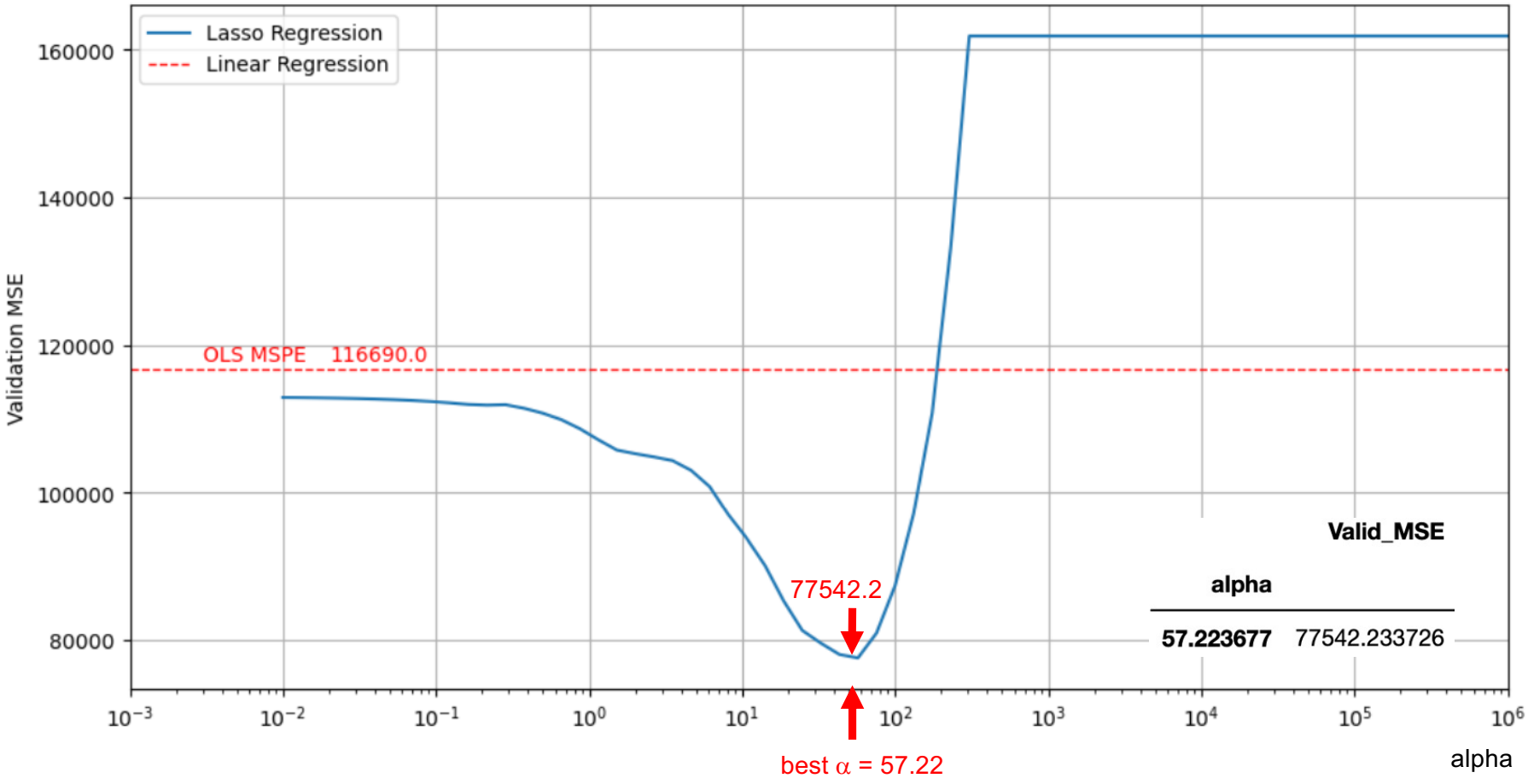
Lasso Regression – Plot for Validation MSE



Lasso Regression – Plot for Validation MSE



Lasso Regression – Plot for Validation MSE



Lasso Regression – Holdout Cross Validation searching for alpha

```
min1 = df.Valid_MSE.min()
min1
```

```
77542.23372579267
```

← Validation MSE

```
df[df.Valid_MSE == min1]
```

	Valid_MSE
alpha	
57.223677	77542.233726

```
# best alpha giving min Validation MSE
best_alpha = df.Valid_MSE.idxmin()
best_alpha
```

```
57.2236765935022
```

```
# test MSE with best alpha
```

```
scaler = StandardScaler()
scaler.fit(X_nontest)
X_nontest_scaled = scaler.transform(X_nontest)
X_test_scaled = scaler.transform(X_test)
```

```
lasso2 = Lasso(alpha = best_alpha)
lasso2.fit(X_nontest_scaled, y_nontest)
pred2 = lasso2.predict(X_test_scaled)
mspe = mean_squared_error(y_test, pred2)
mspe
```

```
109556.92691129888
```

← Test MSE

116690.0

← OLS Test MSE

Lasso regression
- 5-fold Cross Validation -

Lasso Regression 5-fold cross validation to find best alpha

```
X_train,X_test,\
y_train,y_test = train_test_split(X,y,test_size=0.5,
                                  random_state=1)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
scaler = StandardScaler()
model = Lasso(max_iter = 10000)
pipe1 = Pipeline([('scaler', scaler),('lasso', model)])
```

```
param_grid = {'lasso__alpha': alphas}
```

```
lassocv = GridSearchCV(pipe1,param_grid,cv = 5,
                       scoring = 'neg_mean_squared_error')
lassocv.fit(X_train, y_train);
```

Lasso Regression 5-fold cross validation to find best alpha

```
X_train,X_test,\
y_train,y_test = train_test_split(X,y,test_size=0.5,
                                  random_state=1)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
scaler = StandardScaler()
model = Lasso(max_iter = 10000)
pipe1 = Pipeline([('scaler', scaler),('lasso', model)])
```

```
param_grid = {'lasso__alpha': alphas}
```

```
lassocv = GridSearchCV(pipe1,param_grid,cv = 5,
                       scoring = 'neg_mean_squared_error')
lassocv.fit(X_train, y_train);
```

```
# best alpha (minimizing validation MSE)
```

```
lassocv.best_params_
```

```
{'lasso__alpha': 32.745491628777316}
```

```
alpha1 = lassocv.best_params_['lasso__alpha']
alpha1
```

```
32.745491628777316
```

```
# test MSE with best alpha
```

```
# fit lasso model with best alpha
lasso1 = Lasso(alpha = alpha1)
lasso1.fit(X_train_scaled,y_train);
```

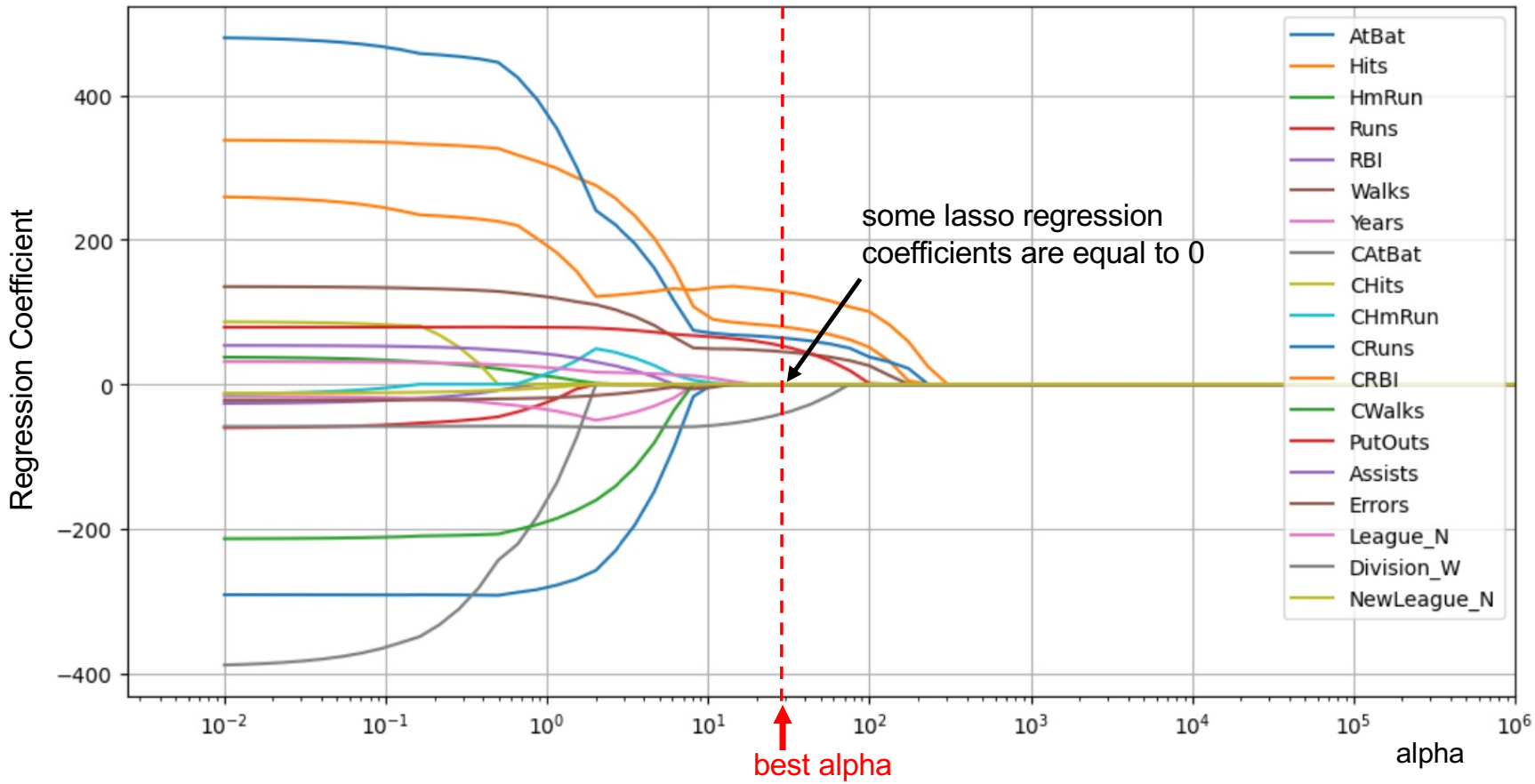
```
yhat = lasso1.predict(X_test_scaled)
best_mspe = mean_squared_error(y_test, yhat)
best_mspe
```

```
105405.44883970029
```

Lasso regression
- feature selection -

Lasso Regression

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.00,10**6))
```



Lasso Regression Coefficients using best alpha

```
df4 = pd.DataFrame(lasso1.coef_,index=X.columns,
                   columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	47.461295
HmRun	0.000000
Runs	0.000000
RBI	1.279976
Walks	64.820579
Years	0.000000
CAtBat	0.000000
CHits	0.000000
CHmRun	16.730628
CRuns	0.000000
CRBI	179.522896
CWalks	0.000000
PutOuts	107.689142
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-39.374659
NewLeague_N	0.000000

12 regression coeffs equal to zero

features selected by LASSO

```
df4[df4.lasso_coeff != 0]
```

	lasso_coeff
Hits	47.461295
RBI	1.279976
Walks	64.820579
CHmRun	16.730628
CRBI	179.522896
PutOuts	107.689142
Division_W	-39.374659

Comparison of Lasso and Ridge Regression coefficients

```
df4 = pd.DataFrame(lasso1.coef_,index=X.columns,
                   columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	47.461295
HmRun	0.000000
Runs	0.000000
RBI	1.279976
Walks	64.820579
Years	0.000000
CAtBat	0.000000
CHits	0.000000

	lasso_coeff
CHmRun	16.730628
CRuns	0.000000
CRBI	179.522896
CWalks	0.000000
PutOuts	107.689142
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-39.374659
NewLeague_N	0.000000

12 regression coeffs equal to zero

	ridge_coeff
AtBat	3.911359
Hits	36.096360
HmRun	1.736680
Runs	19.611659
RBI	32.219132
Walks	43.972410
Years	8.496447
CAtBat	17.992839
CHits	32.545056

	ridge_coeff
CHmRun	41.512925
CRuns	33.271936
CRBI	41.571334
CWalks	25.535833
PutOuts	75.761366
Assists	-2.475953
Errors	-0.792667
League_N	8.214298
Division_W	-41.608368
NewLeague_N	5.211955

No regression coeffs are equal to zero

PREDICTION

predict salary of first player in test set

```
X_test[:1]
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks
126	282.0	78.0	13.0	37.0	51.0	29.0	5.0	1649.0	453.0	73.0	211.0	280.0	138.0

```
newval = X_test_scaled[:1]
newval
```

```
array([[-0.75403349, -0.59086622,  0.20177987, -0.63314908, -0.00973381,
        -0.45405721, -0.50193827, -0.42197414, -0.40314783,  0.01162816,
        -0.43551916, -0.17330772, -0.4452582 ,  1.27997267, -0.33878194,
        -0.44171974, -1.03892496,  0.99239533, -1.00766295]])
```

```
lasso1.predict(newval)
```

Predicted Salary →

```
array([543.06680376])
```

```
y_test[:1]
```

Actual Salary →

```
126    500.0
Name: Salary, dtype: float64
```

Shrinkage Methods

Logistic regression with Regularization

PROBABILITY FUNCTIONS

Let $Y_1, \ldots, Y_n$ be independent Bernoulli random variable with pdfs

$$\begin{aligned} P[Y_1 = y_1] &= \pi_1^{y_1} (1 - \pi_1)^{1-y_1} \\ P[Y_2 = y_2] &= \pi_2^{y_2} (1 - \pi_2)^{1-y_2} \\ &\vdots \qquad\qquad \vdots \\ P[Y_n = y_n] &= \pi_n^{y_n} (1 - \pi_n)^{1-y_n} \end{aligned}$$

Then the joint probability function is

$$P[Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n] = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

LOSS FUNCTION

Joint probability function

$$P[Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n] = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

likelihood function

$$L(\pi_1, \pi_2, ..., \pi_n) = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

log-likelihood function

$$\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [y_i \log \pi_i + (1 - y_i) \log (1 - \pi_i)]$$

Cross-entropy loss function

$$-\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [-y_i \log \pi_i - (1 - y_i) \log (1 - \pi_i)]$$

LOSS FUNCTION

Joint probability function

$$P[Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n] = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

likelihood function

$$L(\pi_1, \pi_2, ..., \pi_n) = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

log-likelihood function

$$\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [y_i \log \pi_i + (1 - y_i) \log (1 - \pi_i)]$$

Cross-entropy loss function

$$-\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [-y_i \log \pi_i - (1 - y_i) \log (1 - \pi_i)]$$

← loss function for logistic regression to minimize

CROSS ENTROPY LOSS FUNCTIONS

Cross-entropy loss function

$$-\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [-y_i \log \pi_i - (1 - y_i) \log (1 - \pi_i)]$$

Cross-entropy loss function with regularization

$$-\log L(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n} [-y_i \log \pi_i - (1 - y_i) \log (1 - \pi_i)] + \alpha \sum_{i=p}^{n} b_i^2$$

penalty term

LOGISTIC REGRESSION WITH REGULARIZATION - sklearn

sklearn includes regularization by means of argument C = $1/\alpha$

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)
```

is the same as

```
model = LogisticRegression(penalty="l2", C=1)
model.fit(X, y)
```

default

LOGISTIC REGRESSION WITH REGULARIZATION - sklearn

sklearn includes regularization by means of argument C = $1/\alpha$

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)
```

To do logistic regression with No Regularization, use C = 1e20 (which is $\alpha \cong 0$)

Logistic regression

Example
Cancer data

Cancer Data - EXAMPLE

- The Cancer data from *sklearn* contains data from 569 patients.
- Build a Logistic Regression model to predict whether the tumor of a patient is malignant.
- The data has 30 lab measurements associated with breast cancer tumors
- These measurements should help predict if the patient has cancer
- Use Cross validation to find the test accuracy rate

Cancer Data

30 measurements

Y	<--------	----------	----------	-------	average	values	----------	----------	----------	---------->	<--------	--------	----------	--------	worst	values	----------	----------	----------
out	radius	texture	perimeter	area	smoothness	compactness	concavity	concave p	symmetry	fractal_dir	radius	texture	perimeter	area	smoothness	compactness	concavity	concave p	symmetry
M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	25.38	17.33	184.6	2019	0.1622	0.6656	0.7119	0.2654	0.4601
M	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	24.99	23.41	158.8	1956	0.1238	0.1866	0.2416	0.186	0.275
M	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	23.57	25.53	152.5	1709	0.1444	0.4245	0.4504	0.243	0.3613
M	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	14.91	26.5	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638
M	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	22.54	16.67	152.2	1575	0.1374	0.205	0.4	0.1625	0.2364
M	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	15.47	23.75	103.4	741.6	0.1791	0.5249	0.5355	0.1741	0.3985
M	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	22.88	27.66	153.2	1606	0.1442	0.2576	0.3784	0.1932	0.3063
M	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	17.06	28.14	110.6	897	0.1654	0.3682	0.2678	0.1556	0.3196
M	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	15.49	30.73	106.2	739.3	0.1703	0.5401	0.539	0.206	0.4378
M	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243	15.09	40.68	97.65	711.4	0.1853	1.058	1.105	0.221	0.4366
M	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	19.19	33.88	123.8	1150	0.1181	0.1551	0.1459	0.09975	0.2948
M	15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082	20.42	27.28	136.5	1299	0.1396	0.5609	0.3965	0.181	0.3792
M	19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078	20.96	29.94	151.7	1332	0.1037	0.3903	0.3639	0.1767	0.3176
M	15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338	16.84	27.66	112	876.5	0.1131	0.1924	0.2322	0.1119	0.2809
M	13.73	22.61	93.6	578.3	0.1131	0.2293	0.2128	0.08025	0.2069	0.07682	15.03	32.01	108.8	697.7	0.1651	0.7725	0.6943	0.2208	0.3596
M	14.54	27.54	96.73	658.8	0.1139	0.1595	0.1639	0.07364	0.2303	0.07077	17.46	37.13	124.1	943.2	0.1678	0.6577	0.7026	0.1712	0.4218
M	14.68	20.13	94.74	684.5	0.09867	0.072	0.07395	0.05259	0.1586	0.05922	19.07	30.88	123.4	1138	0.1464	0.1871	0.2914	0.1609	0.3029
M	16.13	20.68	108.1	798.8	0.117	0.2022	0.1722	0.1028	0.2164	0.07356	20.96	31.48	136.8	1315	0.1789	0.4233	0.4784	0.2073	0.3706
M	19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395	27.32	30.88	186.8	2398	0.1512	0.315	0.5372	0.2388	0.2768
B	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766	15.11	19.26	99.7	711.2	0.144	0.1773	0.239	0.1288	0.2977
B	13.08	15.71	85.63	520	0.1075	0.127	0.04568	0.0311	0.1967	0.06811	14.5	20.49	96.09	630.5	0.1312	0.2776	0.189	0.07283	0.3184
B	9.504	12.44	60.34	273.9	0.1024	0.06492	0.02956	0.02076	0.1815	0.06905	10.23	15.66	65.13	314.9	0.1324	0.1148	0.08867	0.06227	0.245
M	15.34	14.26	102.5	704.4	0.1073	0.2135	0.2077	0.09756	0.2521	0.07032	18.07	19.08	125.1	980.9	0.139	0.5954	0.6305	0.2393	0.4667
M	21.16	23.04	137.2	1404	0.09428	0.1022	0.1097	0.08632	0.1769	0.05278	29.17	35.59	188	2615	0.1401	0.26	0.3155	0.2009	0.2822

Logistic Regression - EXAMPLE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

```
cancer = load_breast_cancer()
cancer.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names',
```

X **y**

Response is in target and target_names

Predictors are in data and feature_names

```
y = cancer.target
X = cancer.data
```

Logistic Regression – SCALE predictors

```
X_train,X_test,y_train,y_test = train_test_split(X,y,
                                                 stratify=y,
                                                 random_state=66)
scaler = MinMaxScaler()

# Find min/max of each feature in Train set

scaler.fit(X_train);

# Now transform data into (0,1)
# substracting the train set Min,
# dividing by the train set range

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression – Holdout Cross Validation

.

not scaling the data

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000)
model.fit(X_train,y_train)
yhat = model.predict(X_test)
model.score(X_test,y_test)
```

```
0.9440559440559441
```

scaling the data

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000)
model.fit(X_train_scaled,y_train)
yhat = model.predict(X_test_scaled)
model.score(X_test_scaled,y_test)
```

```
0.972027972027972
```

Logistic Regression – Holdout Cross Validation

.

not scaling the data

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000)
model.fit(X_train,y_train)
yhat = model.predict(X_test)
model.score(X_test,y_test)
```

```
0.9440559440559441
```

C = 1 is default value

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000,C=1)
model.fit(X_train_scaled,y_train)
yhat = model.predict(X_test_scaled)
model.score(X_test_scaled,y_test)
```

```
0.972027972027972
```

Logistic Regression – Holdout Cross Validation

.

not scaling the data

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000)
model.fit(X_train,y_train)
yhat = model.predict(X_test)
model.score(X_test,y_test)
```

```
0.9440559440559441
```

C = 1e20 means $\alpha = 0$ (no regularization)

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000,C=1e20)
model.fit(X_train_scaled,y_train)
yhat = model.predict(X_test_scaled)
model.score(X_test_scaled,y_test)
```

```
0.9440559440559441
```

Holdout Cross Validation – Search best C value

```
model = LogisticRegression(solver = 'lbfgs',max_iter=1000)
```

```
Cvalues = np.arange(0.01,1.50,0.001)
arates = []
```

```
for i in Cvalues:
    model.set_params(C = i)
    model.fit(X_train_scaled,y_train)
    arate = model.score(X_test_scaled,y_test)
    arates.append(arate)
```

```
arates[:5]
```

```
[0.7972027972027972,
 0.7972027972027972,
 0.8041958041958042,
 0.8041958041958042,
 0.8181818181818182]
```

Holdout Cross Validation – Search best C value

```
df2 = pd.DataFrame(arates,columns = ['Test accuracy'])
df2.index = Cvalues
df2.index.name = 'C values'
df2[:5]
```

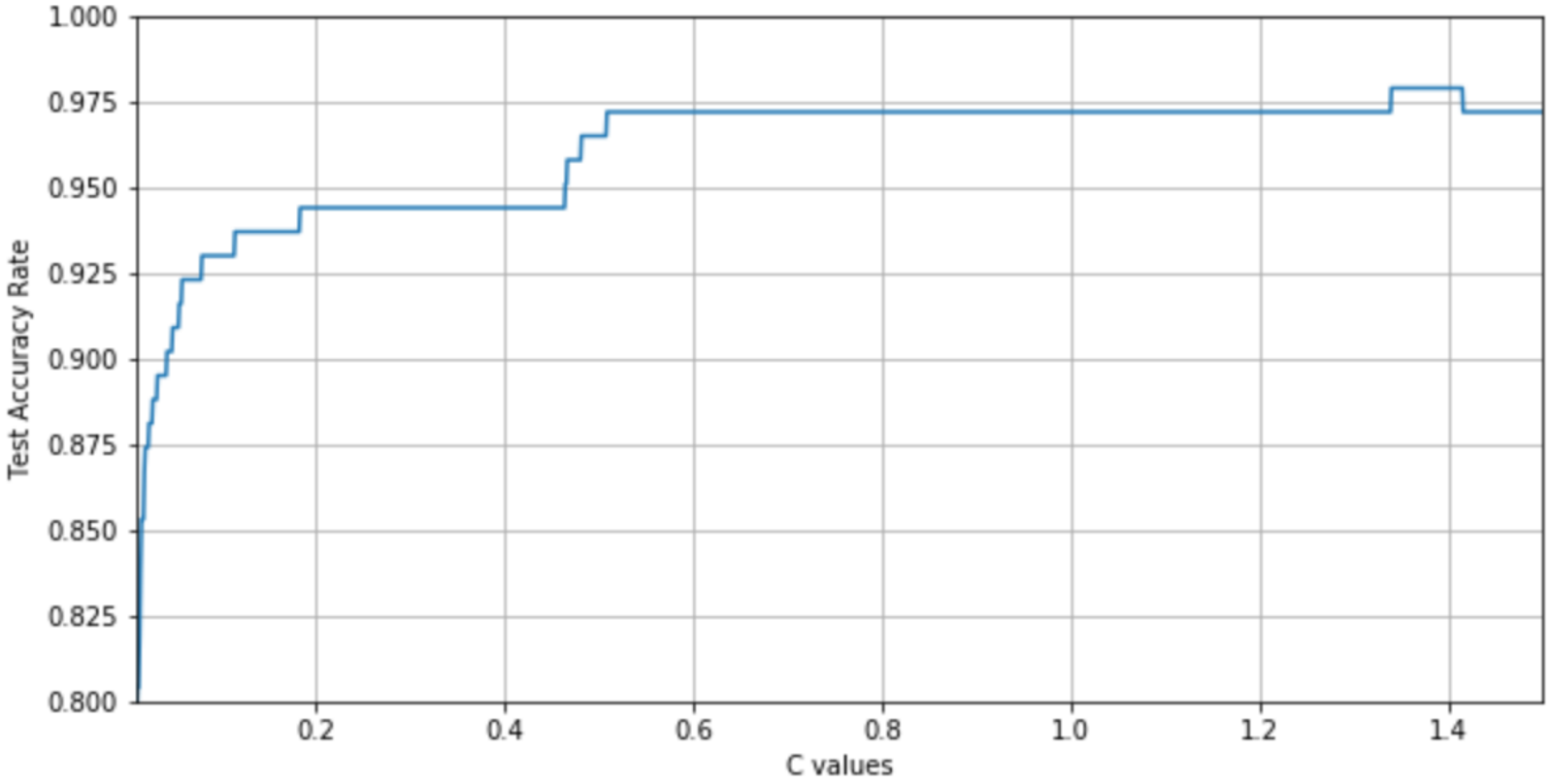
```
arates[:5]
```

```
[0.7972027972027972,
 0.7972027972027972,
 0.8041958041958042,
 0.8041958041958042,
 0.8181818181818182]
```

C values	Test accuracy
0.010	0.797203
0.011	0.797203
0.012	0.804196
0.013	0.804196
0.014	0.818182

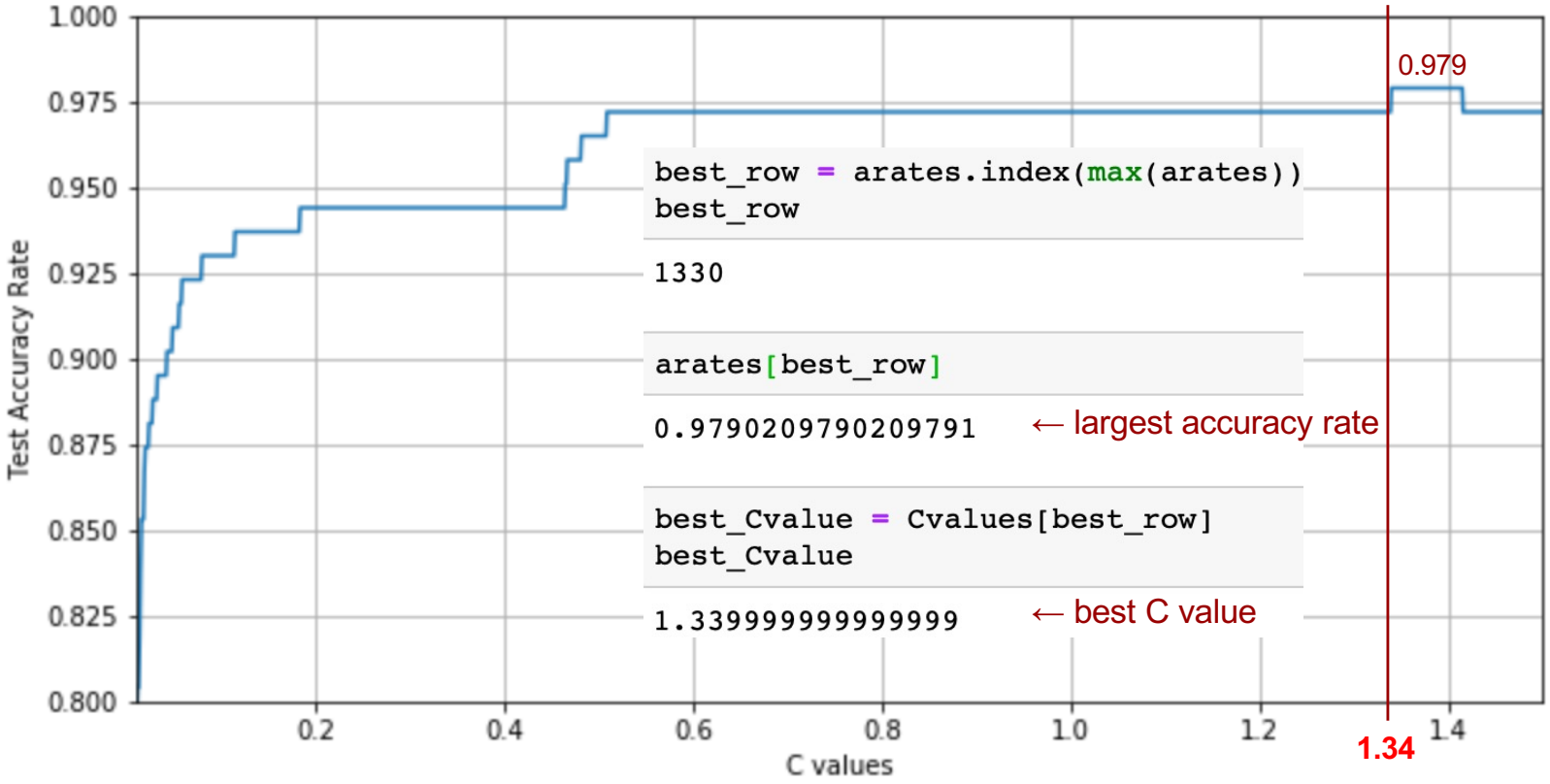
Logistic Regression – Holdout Cross Validation – Search best C value

```
df2.plot(figsize = (10,5),grid=True,legend = False,ylim = (0.80,1.0))
plt.ylabel('Test Accuracy Rate');
```



Logistic Regression – Holdout Cross Validation – Search best C value

```
df2.plot(figsize = (10,5),grid=True,legend = False,ylim = (0.80,1.0))
plt.ylabel('Test Accuracy Rate');
```




```
best_row = arates.index(max(arates))
best_row
```

```
1330
```

```
arates[best_row]
```

```
0.9790209790209791
```

← largest accuracy rate

```
best_Cvalue = Cvalues[best_row]
best_Cvalue
```

```
1.339999999999999
```

← best C value