Regression Models with **Regularization**

When getting more data isn't possible, the next best solution is to modulate the quantity of information that your model is allowed to store, or to add constraints on the smoothness of the model curve. If a network can only afford to memorize a small number of patterns, or very regular patterns, the optimization process will force it to focus on the most prominent patterns, which have a better chance of generalizing well. The process of fighting overfitting this way is called *regularization* We'll review regularization techniques in depth in section 5.4.4.

OVERVIEW

- Norm of a vector
- Linear Regression Loss function
- Encoding methods
- Ridge and LASSO regression
- Example –sklearn
- Logistic Regression Loss function
- Example -sklearn

5.4.4 page 148

Regression models with regularization

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

Shrinkage Methods

Norm of a Vector

Shrinkage Methods

Norm of a vector → a measure of the length of a vector

Vector $\underline{b}' = [\, b_1, \ldots, b_m \,]$

ℓ_2 norm $\qquad \|b\|_2 = \sqrt{b_1^2 + \cdots + b_m^2}$

ℓ_1 norm $\qquad \|b\|_1 = |b_1| + \cdots + |b_m|$

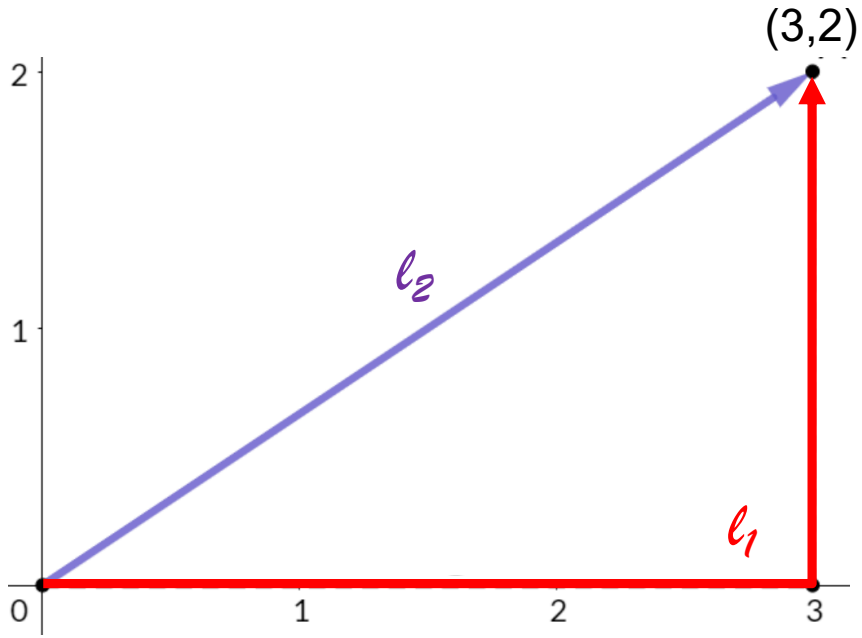
Shrinkage Methods

Norm of a vector a measure of the length of a vector

Vector $\underline{b}' = [3,2]$

ℓ_2 norm $\quad \|b\|_2 = \sqrt{3^2 + 2^2}$

ℓ_1 norm $\quad \|b\|_1 = |3| + |2|$



Ridge and LASSO Regression Models

LOSS 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 shrinkage) we get linear regression loss function

RIDGE REGRESSION

Ridge Regression model

loss function

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

with solution

$$\underline{b} = [X'X + \alpha\, I\,]^{-1} X'Y$$

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 yielding a regression model with less predictors

RIDGE AND LASSO LOSS FUNCTIONS

.

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

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

RIDGE AND LASSO LOSS FUNCTIONS

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

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

- the value of tuning parameter α is chosen by cross validation
- *LogisticRegression( )* uses $C = 1/\alpha$ as tuning parameter

Encoding Methods

Categorical Predictors – LABEL ENCODING

Consider the following dataset

X_1	X_2	Y
S	-0.10	19.19
S	2.53	22.74
S	4.86	23.91
M	0.26	7.07
M	2.55	7.93
M	4.87	8.93
L	0.08	20.63
L	2.62	23.46
L	5.09	25.75

$n = 9$
$p = 2$

```
data0 = pd.read_csv('small.csv')
data0
```

	X1	X2	Y
0	S	-0.10	19.19
1	S	2.53	22.74
2	S	4.86	23.91
3	M	0.26	7.07
4	M	2.55	7.93
5	M	4.87	8.93
6	L	0.08	20.63
7	L	2.62	23.46
8	L	5.09	25.75

Categorical Predictors – LABEL ENCODING

.

X_1	X_2	Y
S	-0.10	19.19
S	2.53	22.74
S	4.86	23.91
M	0.26	7.07
M	2.55	7.93
M	4.87	8.93
L	0.08	20.63
L	2.62	23.46
L	5.09	25.75

label encoding

X_1	X_2	Y
0	-0.10	19.19
0	2.53	22.74
0	4.86	23.91
1	0.26	7.07
1	2.55	7.93
1	4.87	8.93
2	0.08	20.63
2	2.62	23.46
2	5.09	25.75

Categorical Predictors – ONE-HOT ENCODING

Replace X_1 with binary columns

X_1	X_2	Y
S	-0.10	19.19
S	2.53	22.74
S	4.86	23.91
M	0.26	7.07
M	2.55	7.93
M	4.87	8.93
L	0.08	20.63
L	2.62	23.46
L	5.09	25.75

ONE-HOT ENCODING

X_{10}	X_{11}	X_{12}	X_2	Y
1	0	0	-0.10	19.19
1	0	0	2.53	22.74
1	0	0	4.86	23.91
0	1	0	0.26	7.07
0	1	0	2.55	7.93
0	1	0	4.87	8.93
0	0	1	0.08	20.63
0	0	1	2.62	23.46
0	0	1	5.09	25.75

Categorical Predictors – ONE-HOT ENCODING

Replace X_1 with binary columns

X_1	X_2	Y
S	-0.10	19.19
S	2.53	22.74
S	4.86	23.91
M	0.26	7.07
M	2.55	7.93
M	4.87	8.93
L	0.08	20.63
L	2.62	23.46
L	5.09	25.75

n = 9
p = 2

ONE-HOT ENCODING

X_{11}	X_{12}	X_2	Y
0	0	-0.10	19.19
0	0	2.53	22.74
0	0	4.86	23.91
1	0	0.26	7.07
1	0	2.55	7.93
1	0	4.87	8.93
0	1	0.08	20.63
0	1	2.62	23.46
0	1	5.09	25.75

n = 9
p = 3

Shrinkage Methods

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 decreases
- Split the data set into a train and a test set (50%).
- Find the best value for α using
  - holdout cross validation
  - 10-fold cross validation
- Use the best α value to refit a ridge regression model with the full dataset and make a prediction
- Repeat with LASSO regression

Ridge Regression

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

```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, Lasso
```

```
# for holdout cv
from sklearn.model_selection import train_test_split
```

```
# for K-fold cv
from sklearn.linear_model import RidgeCV, LassoCV
```

Ridge Regression

```
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

Ridge Regression

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

```
(322, 20)
```

last 15 columns

```
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

Ridge Regression

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

```
(322, 20)
```

```
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)
```

Ridge Regression

```
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)
```

Ridge Regression

```
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

Ridge Regression – One-hot Encoding with pd.get_dummies( )

```
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)
```

```
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
```

Ridge Regression – Find 100 values in the interval $0.01 < \alpha < 10^{10}$

split interval (10,-2) in 100 subintervals

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

```
alphas.min()
```

```
0.01
```

```
alphas.max()
```

```
10000000000.0
```

```
# fit 100 Ridge regression models,
# one for each alpha (normalizing all cols)
```

```
model = Ridge(normalize = True)
```

```
coefs = []
```

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

coefs is a list of 1D arrays (vectors)

The arrays have the regression coefficients

Ridge Regression

split interval (10,-2) in 100 subintervals

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

```
alphas.min()
```

```
0.01
```

```
alphas.max()
```

```
10000000000.0
```

```
# fit 100 Ridge regression models,
# one for each alpha (normalizing all cols)
```

```
model = Ridge(normalize = True)
```

```
coefs = []
```

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

store coefs in dataframe df

```
df = pd.DataFrame(coefs)
df.columns = x.columns
df.index = alphas
df.index.name = 'alpha'
df
```

← column names
← row names

Each row is a Ridge regression model with 19 beta coefficients

- DataFrame with ridge regression coefficients

	predictors					
	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
alpha						
1.000000e+10	0.000	0.000	-0.000	-0.000	-0.000	-0.000
7.564633e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
5.722368e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
4.328761e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
3.274549e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
...	...	...	...	...	...	...
3.053856e-02	0.272	0.215	-3.842	57.629	-124.203	-23.693
2.310130e-02	0.274	0.235	-3.855	58.938	-124.215	-25.357
1.747528e-02	0.276	0.253	-3.842	59.905	-123.955	-26.519
1.321941e-02	0.278	0.270	-3.810	60.611	-123.493	-27.252
1.000000e-02	0.279	0.286	-3.767	61.128	-122.894	-27.641

100 rows x 19 columns

Each row is a Ridge regression model with 19 beta coefficients

- DataFrame with ridge regression coefficients

model 1

model 100

	predictors					
	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
alpha						
1.000000e+10	0.000	0.000	-0.000	-0.000	-0.000	-0.000
7.564633e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
5.722368e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
4.328761e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
3.274549e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
...	...	...	...	...	...	...
3.053856e-02	0.272	0.215	-3.842	57.629	-124.203	-23.693
2.310130e-02	0.274	0.235	-3.855	58.938	-124.215	-25.357
1.747528e-02	0.276	0.253	-3.842	59.905	-123.955	-26.519
1.321941e-02	0.278	0.270	-3.810	60.611	-123.493	-27.252
1.000000e-02	0.279	0.286	-3.767	61.128	-122.894	-27.641

100 rows x 19 columns

Each row is a Ridge regression model with 19 beta coefficients

- DataFrame with ridge regression coefficients

	predictors					
	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
alpha						
1.000000e+10	0.000	0.000	-0.000	-0.000	-0.000	-0.000
7.564633e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
5.722368e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
4.328761e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
3.274549e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
...	...	...	...	...	...	...
3.053856e-02	0.272	0.215	-3.842	57.629	-124.203	-23.693
2.310130e-02	0.274	0.235	-3.855	58.938	-124.215	-25.357
1.747528e-02	0.276	0.253	-3.842	59.905	-123.955	-26.519
1.321941e-02	0.278	0.270	-3.810	60.611	-123.493	-27.252
1.000000e-02	0.279	0.286	-3.767	61.128	-122.894	-27.641

α very large

α very small

100 rows x 19 columns

Each row is a Ridge regression model with 19 beta coefficients

- DataFrame with ridge regression coefficients

α very large

	predictors					
	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
alpha						
1.000000e+10	0.000	0.000	-0.000	-0.000	-0.000	-0.000
7.564633e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
5.722368e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
4.328761e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
3.274549e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
...	...	...	...	...	...	...
3.053856e-02	0.272	0.215	-3.842	57.629	-124.203	-23.693
2.310130e-02	0.274	0.235	-3.855	58.938	-124.215	-25.357
1.747528e-02	0.276	0.253	-3.842	59.905	-123.955	-26.519
1.321941e-02	0.278	0.270	-3.810	60.611	-123.493	-27.252
1.000000e-02	0.279	0.286	-3.767	61.128	-122.894	-27.641

100 rows x 19 columns

Each row is a Ridge regression model with 19 beta coefficients

- DataFrame with ridge regression coefficients
- How does each ridge regression coefficient changes with alpha?

19 predictors

alpha	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
1.000000e+10	0.000	0.000	-0.000	-0.000	-0.000	-0.000
7.564633e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
5.722368e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
4.328761e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
3.274549e+09	0.000	0.000	-0.000	-0.000	-0.000	-0.000
...	...	...	...	...	...	...
3.053856e-02	0.272	0.215	-3.842	57.629	-124.203	-23.693
2.310130e-02	0.274	0.235	-3.855	58.938	-124.215	-25.357
1.747528e-02	0.276	0.253	-3.842	59.905	-123.955	-26.519
1.321941e-02	0.278	0.270	-3.810	60.611	-123.493	-27.252
1.000000e-02	0.279	0.286	-3.767	61.128	-122.894	-27.641

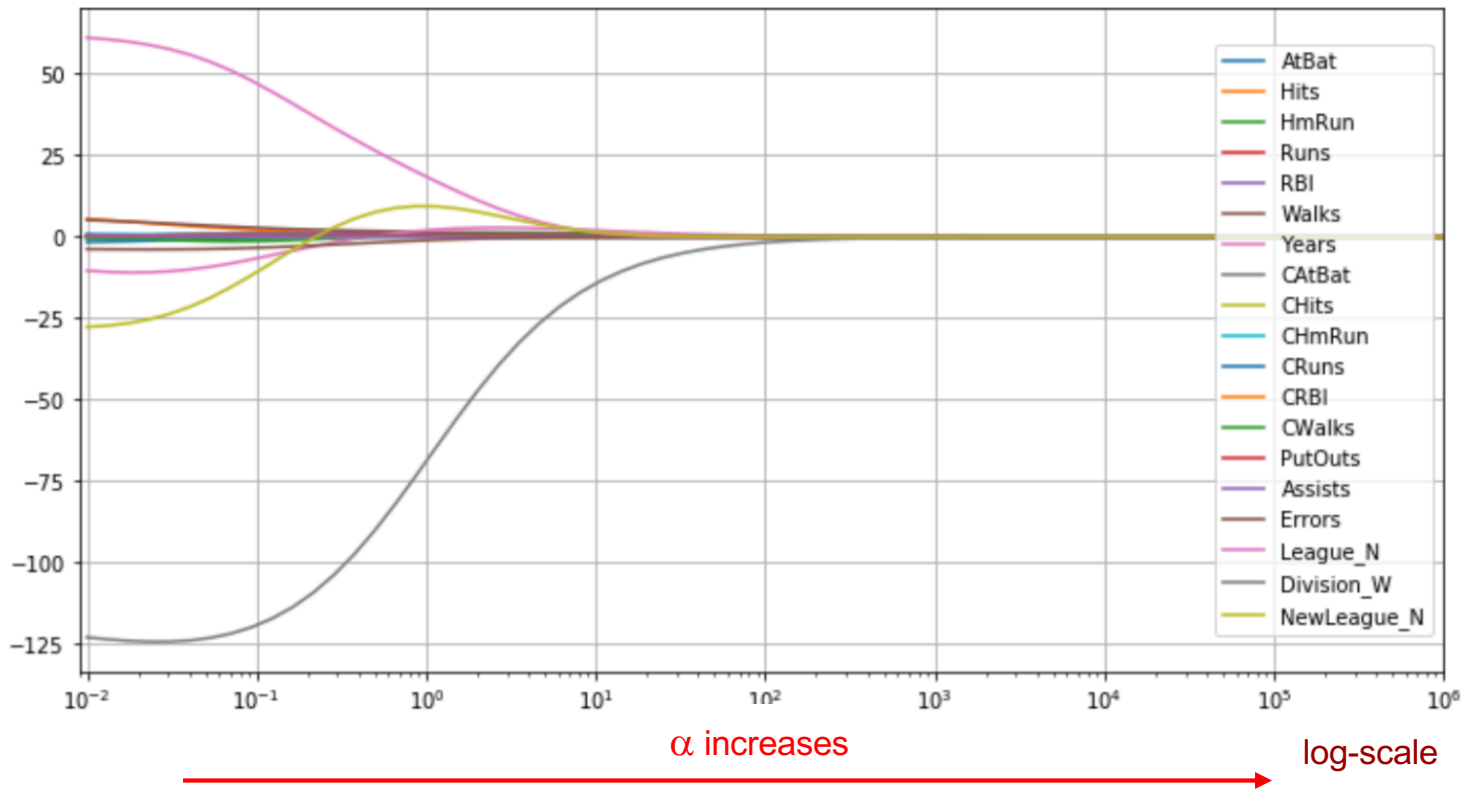
100 rows x 19 columns

Ridge Regression

There are 19 curves one for each predictor

Each curve shows how the value of a ridge regression coefficient changes when α increases

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.009,10**6))
plt.legend(loc='right');
```



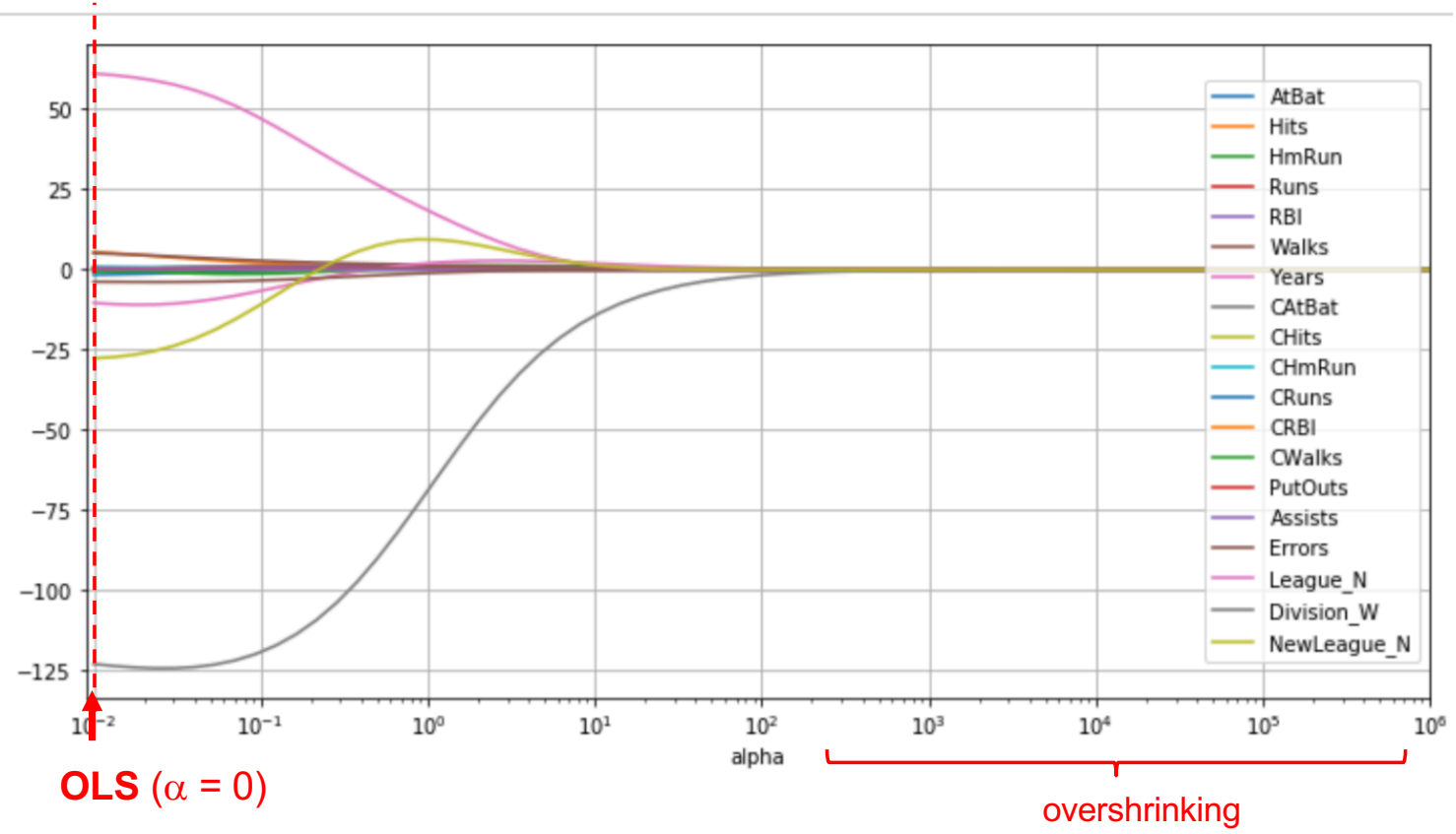
Ridge Regression

There are 19 curves one for each predictor

Each curve shows how the value of a ridge regression coefficient changes when α increases

All coefficients *shrink* to zero as alpha increases

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.009,10**6))
plt.legend(loc='right');
```

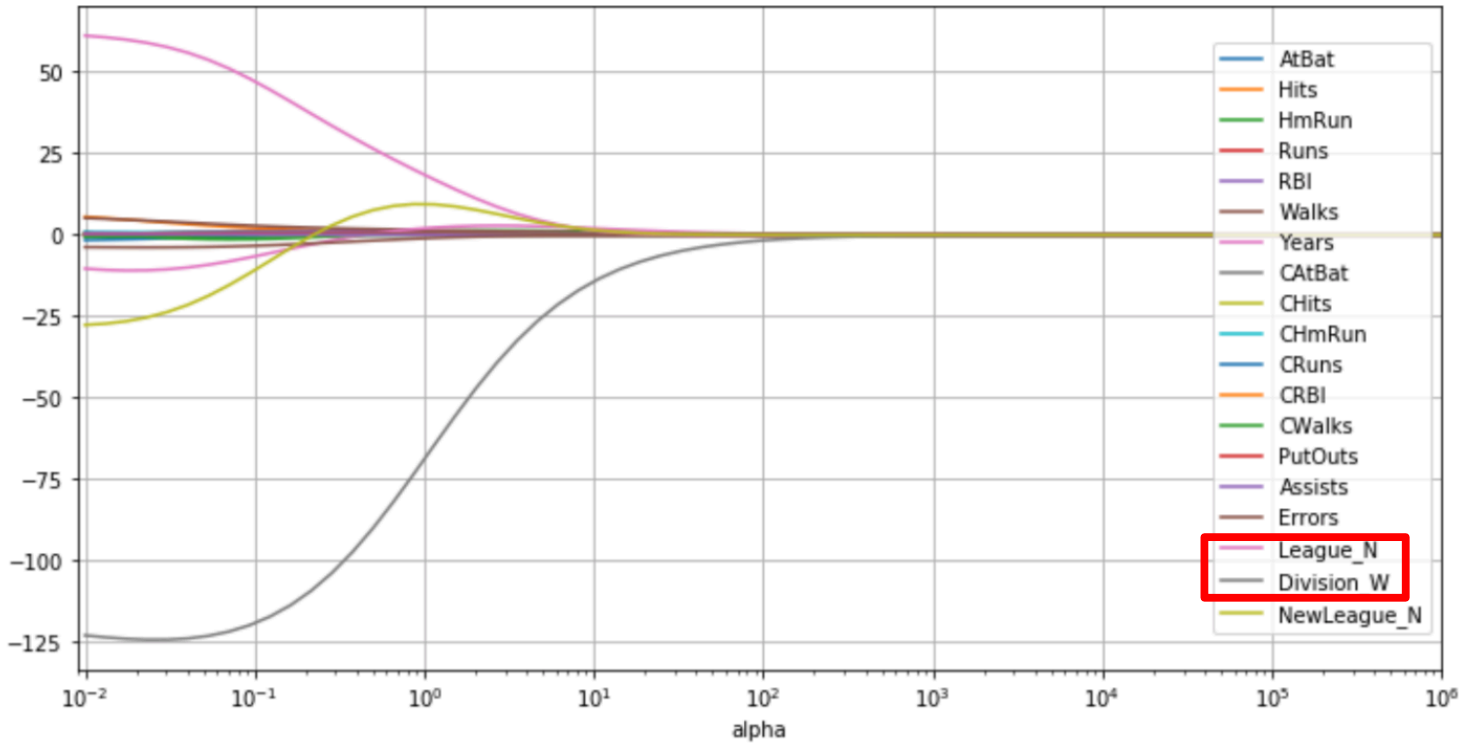


Ridge Regression

There are 19 curves one for each predictor

Each curve shows how the value of a ridge regression coefficient changes when α increases

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.009,10**6))
plt.legend(loc='right');
```



Ridge Regression – Validation Approach

Validation approach with fixed alpha = 4

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

fit model with alpha = 4

```
ridge2 = Ridge(alpha = 4, normalize = True)
ridge2.fit(X_train, y_train)
pred2 = ridge2.predict(X_test)
mspe = mean_squared_error(y_test, pred2)
mspe
```

```
106216.52238005563
```

```
np.sqrt(mspe)
```

```
325.90876388961317
```

Ridge Regression – Validation Approach

Validation approach with fixed alpha

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

fit model with huge alpha = 10^9 overshrinking

```
ridge3 = Ridge(alpha = 10**9, normalize = True)
ridge3.fit(X_train, y_train)
pred3 = ridge3.predict(X_test)
mean_squared_error(y_test, pred3)
```

```
172862.234750706
```

```
# Over-shrinking increases test MSE
# all coeffs very close to zero
```

Ridge Regression – Validation Approach

Validation approach with fixed alpha

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

Compare Ridge Regression with Linear Regression (alpha = 0)

```
ols_model = Ridge(alpha = 0, normalize = True)
ols_model.fit(X_train, y_train)
pred = ols_model.predict(X_test)
ols_mse = mean_squared_error(y_test, pred)
ols_mse
```

```
116690.46856660438
```

RR with alpha = 4 is better

```
np.sqrt(mean_squared_error(y_test, pred))
```

```
341.5998661688912
```

Ridge Regression – Validation Approach

```
# MSPE varies with alpha
```

loop to compute MSPE for all alpha values

```
model = Ridge(normalize = True)
mspes = []
```

list with mspe values

```
for a in alphas:
    model.set_params(alpha = a)
    model.fit(X_train, y_train)
    value = mean_squared_error(y_test, model.predict(X_test))
    mspes.append(value)
```

```
model.set_params(alpha = 0)
model.fit(X_train, y_train)
value = mean_squared_error(y_test, model.predict(X_test))
value
```

```
116690.46856660438
```

Ridge Regression – Create dataframe from *mspes* list

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

alpha	MSPE
1.000000e+10	172862.235804
7.564633e+09	172862.235766
5.722368e+09	172862.235716
4.328761e+09	172862.235651
3.274549e+09	172862.235563
3.053856e-02	102144.426231
2.310130e-02	102357.905350
1.747528e-02	102591.659514
1.321941e-02	102831.224821
1.000000e-02	103069.743901
0.000000	116690.468566

OLS

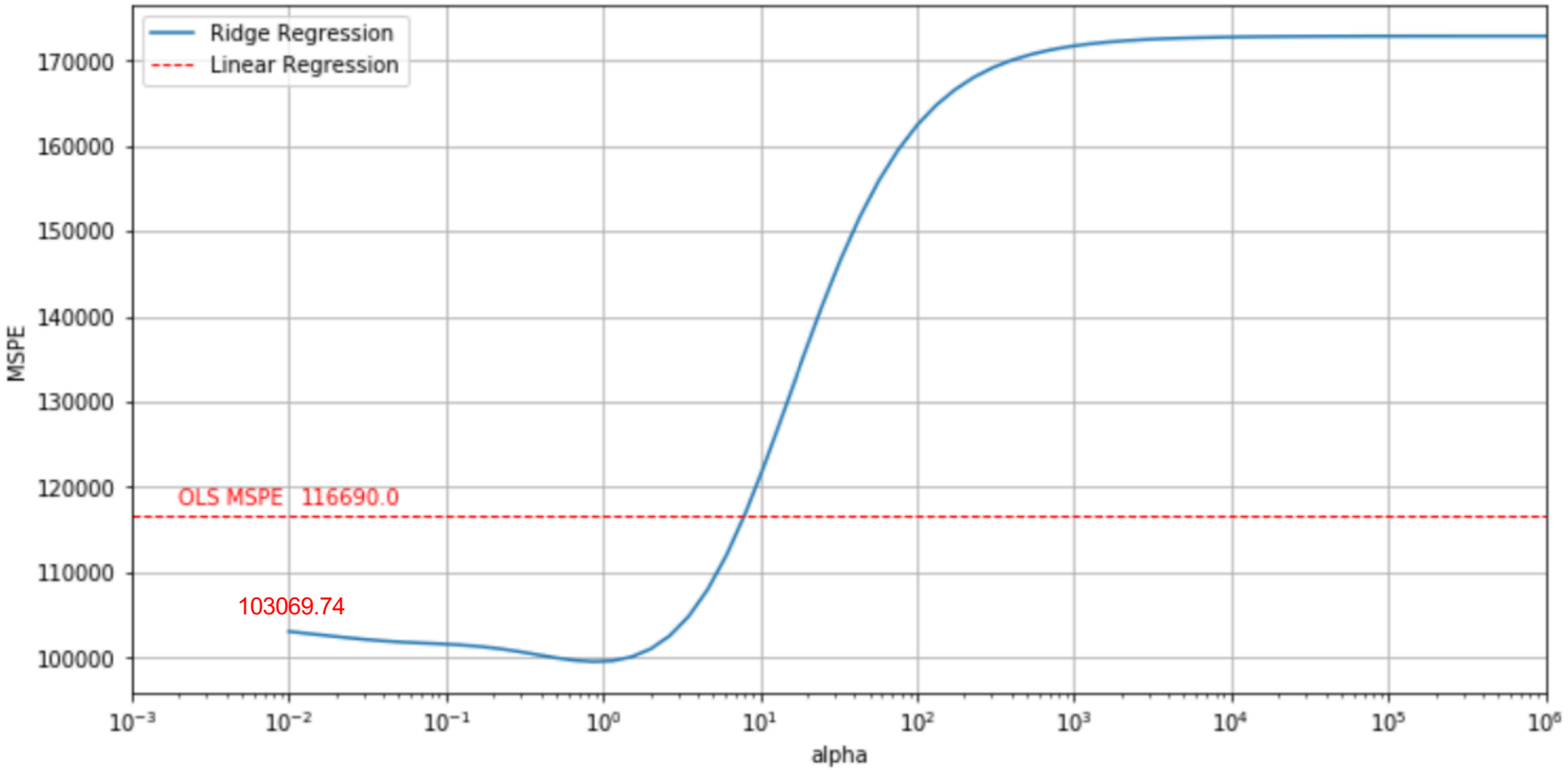
Ridge Regression – Validation Approach

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

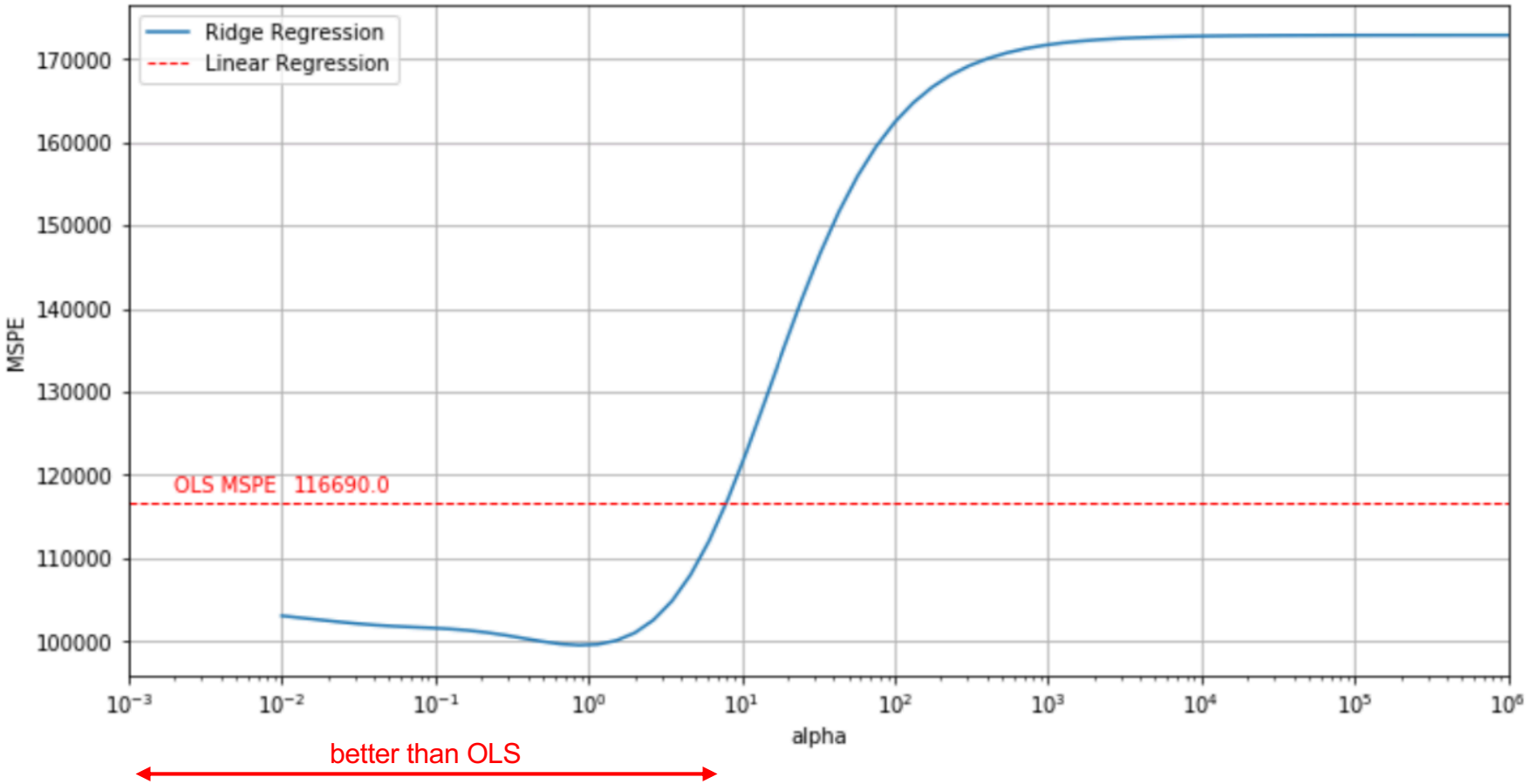
plot dataframe df

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.001,10**6))
# add horizontal line with OLS MSPE
plt.axhline(y = ols_mse,linestyle = '--',c='r',linewidth=1)
plt.annotate(round(ols_mse,0),xy=(0.012,1.01*ols_mse), c='r')
plt.annotate('OLS MSPE',xy=(0.002,1.01*ols_mse), c='r')
#
plt.ylabel('MSPE')
plt.legend(("Ridge Regression", "Linear Regression"));
```

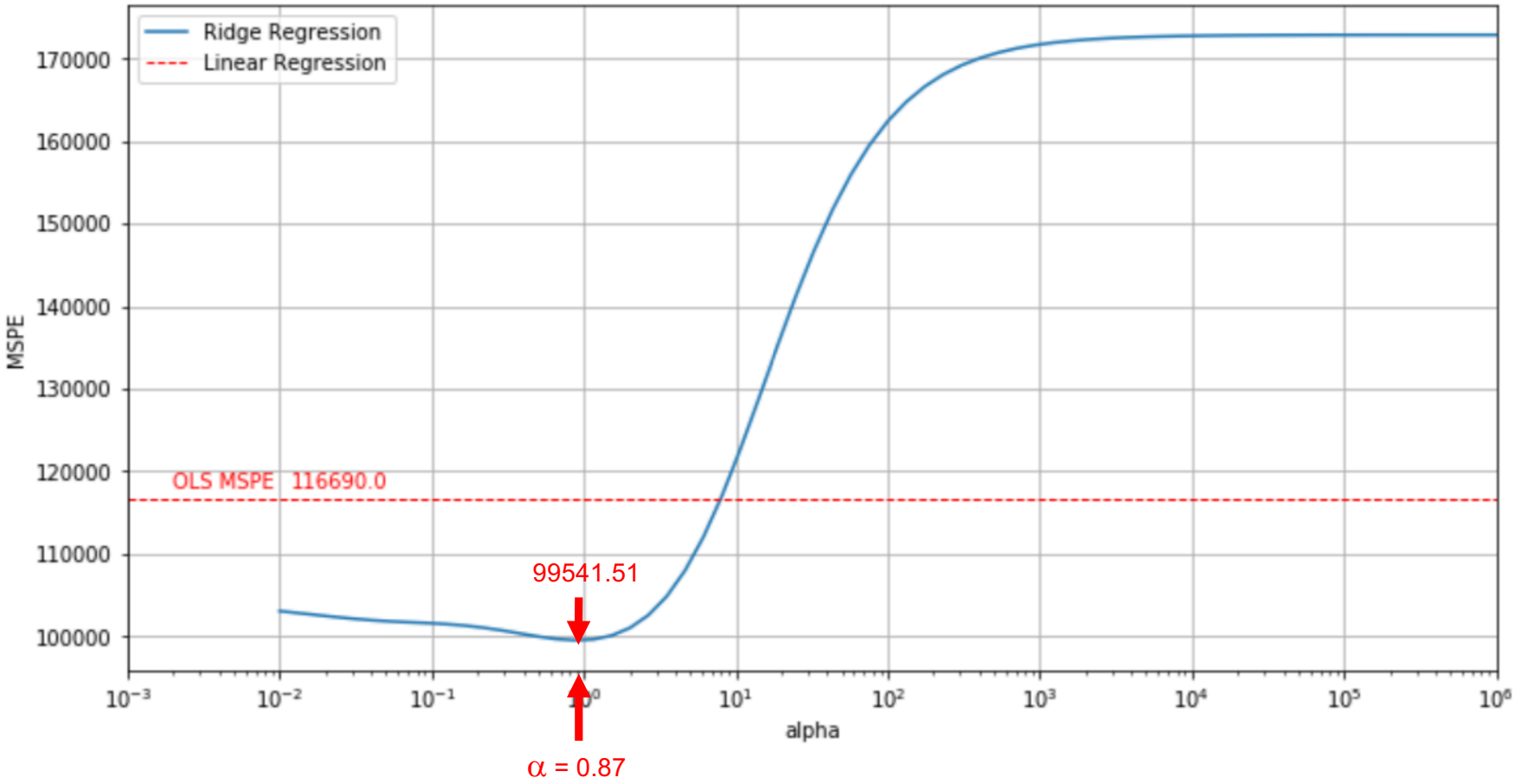
Ridge Regression – Validation Approach



Ridge Regression Models better than OLS



Ridge Regression best Model



Ridge Regression – Validation Approach – find best alpha minimizing test MSPE

```
mspes.index(min(mspes))
```

```
83
```

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

```
alphas[83]
```

```
0.8697490026177834
```

```
mspes[83]
```

```
99541.51776483622
```

Ridge Regression 10-fold cross validation to find best alpha

```
ridgecv = RidgeCV(alphas = alphas,cv = 10,
                  normalize = True,scoring = 'neg_mean_squared_error')
ridgecv.fit(X_train, y_train)
```

```
ridgecv.alpha_
```

```
0.6579332246575682
```

```
# test MSE of best alpha
```

```
yhatcv = ridgecv.predict(X_test)
best_mspe = mean_squared_error(y_test, yhatcv)
best_mspe
```

```
99691.75835893235
```

PREDICTION

```
ridgecv.alpha_
```

```
0.6579332246575682
```

```
# coefficients of best RR model (when fitting train set)
```

0.657933

```
ridge4 = Ridge(alpha = ridgecv.alpha_,normalize = True)
```

```
ridge4.fit(X_train,y_train)
df4 = pd.DataFrame(ridge4.coef_,index=X.columns,
                   columns=['ridge_coeff'])
df4
```

	ridge_coeff
AtBat	0.007004
Hits	0.820565
HmRun	-0.038859
Runs	0.738681
RBI	1.281625
Walks	2.006648
Years	1.490860
CAtBat	0.007236
CHits	0.050842

PREDICTION

```
ridge4.fit(X_train,y_train)
```

```
# Coefficients of best RR model (when fitting full dataset)
```

```
ridge4.fit(X,y)
pd.DataFrame(ridge4.coef_,index=X.columns,
                    columns=['ridge_coeff'])
```

	ridge_coeff
AtBat	0.069155
Hits	0.889574
HmRun	0.510983
Runs	1.076520
RBI	0.879851
Walks	1.662546
Years	1.147263
CAtBat	0.011367
CHits	0.058869

PREDICTION

```
newval = X_test[:1]
newval
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits
126	282.0	78.0	13.0	37.0	51.0	29.0	5.0	1649.0	453.0

```
ridge4.predict(newval)
```

```
array([417.01024776])
```

```
# predicted salary is 417,010 dollars
```

Shrinkage Methods

Lasso regression

Lasso Regression

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

```
# lasso may not converge unless large number of iterations are used
```

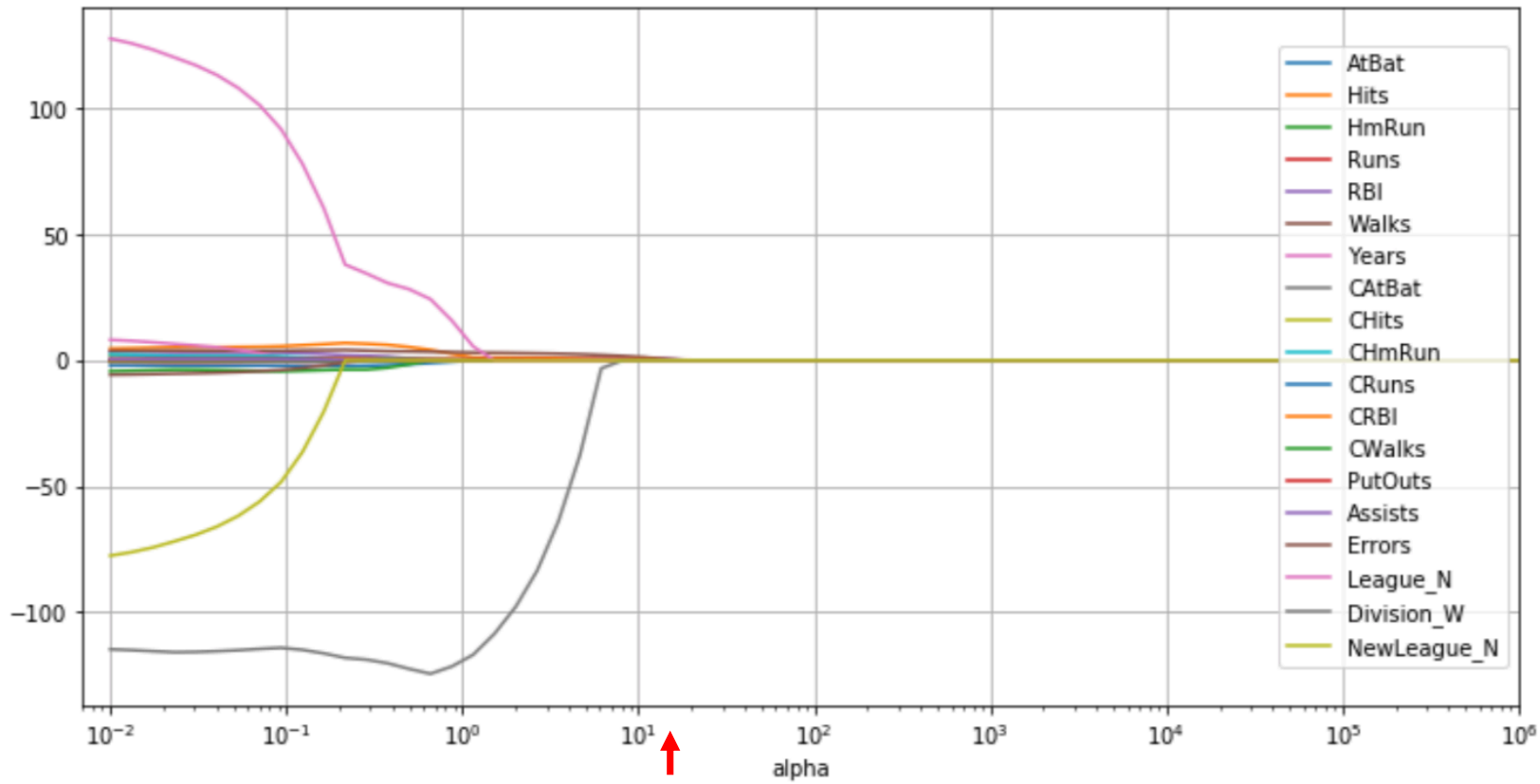
```
lasso_model = Lasso(max_iter = 10000, normalize = True)
```

```
coefs = []
for a in alphas:
    lasso_model.set_params(alpha=a)
    lasso_model.fit(X_train,y_train)
    coefs.append(lasso_model.coef_)
```

```
df = pd.DataFrame(coefs)
df.columns = x.columns
df.index = alphas
df.index.name = 'alpha'
```

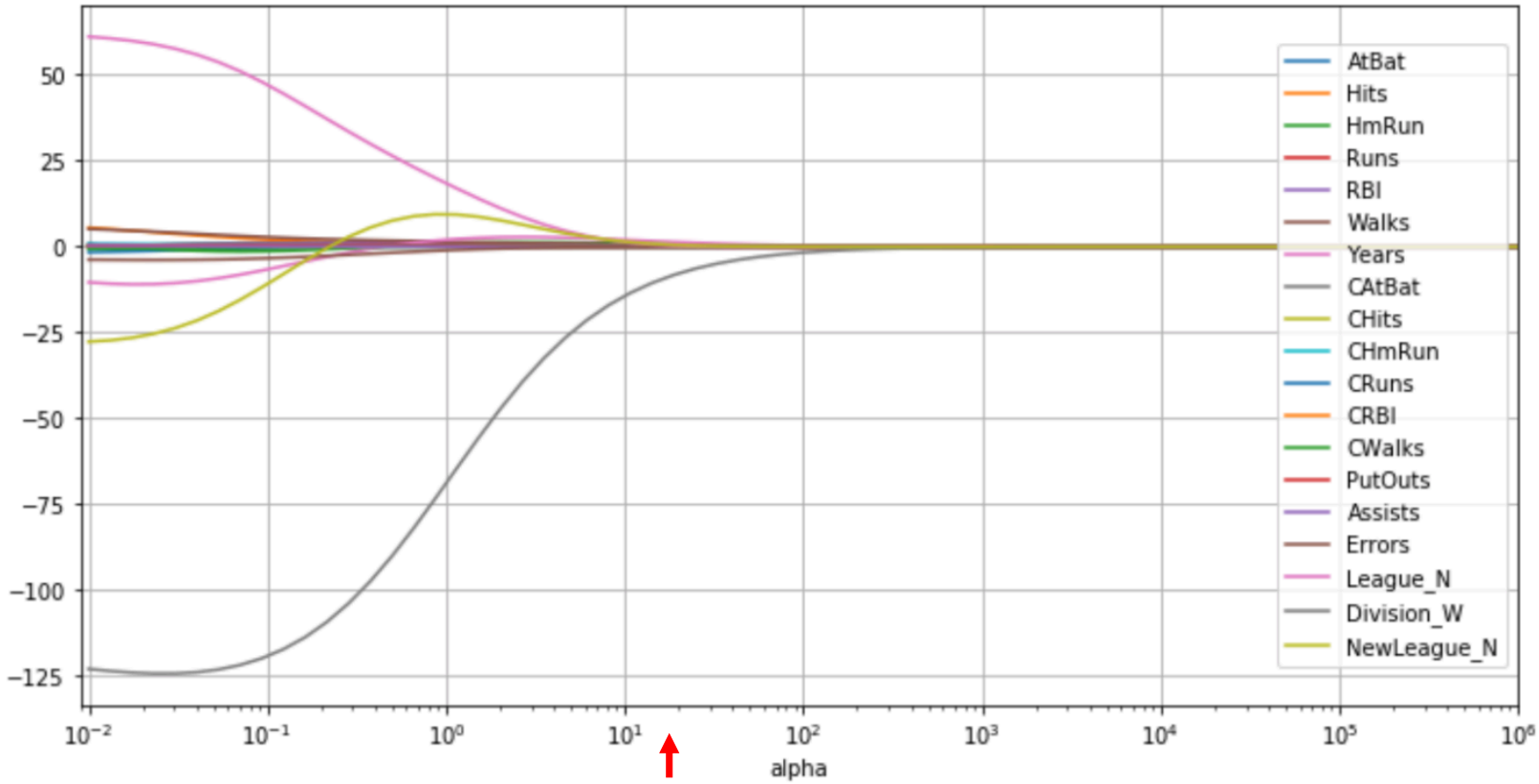
Lasso Regression

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.007,10**6))
plt.legend(loc='right');
```



Ridge Regression

```
df.plot(figsize=(12,6),grid=True,logx=True,xlim = (0.009,10**6))
plt.legend(loc='right');
```



Lasso Regression 10-fold cross validation to find best alpha

```
lassocv = LassoCV(alphas = alphas, cv=10, max_iter = 100000,
                  normalize = True)
lassocv.fit(X_train,y_train);
```

```
lassocv.alpha_
```

```
2.656087782946684
```

Lasso Regression 10-fold cross validation to find best alpha

```
lassocv = LassoCV(alphas = alphas, cv=10, max_iter = 100000,
                  normalize = True)
lassocv.fit(X_train,y_train);
```

```
lassocv.alpha_
```

```
2.656087782946684
```

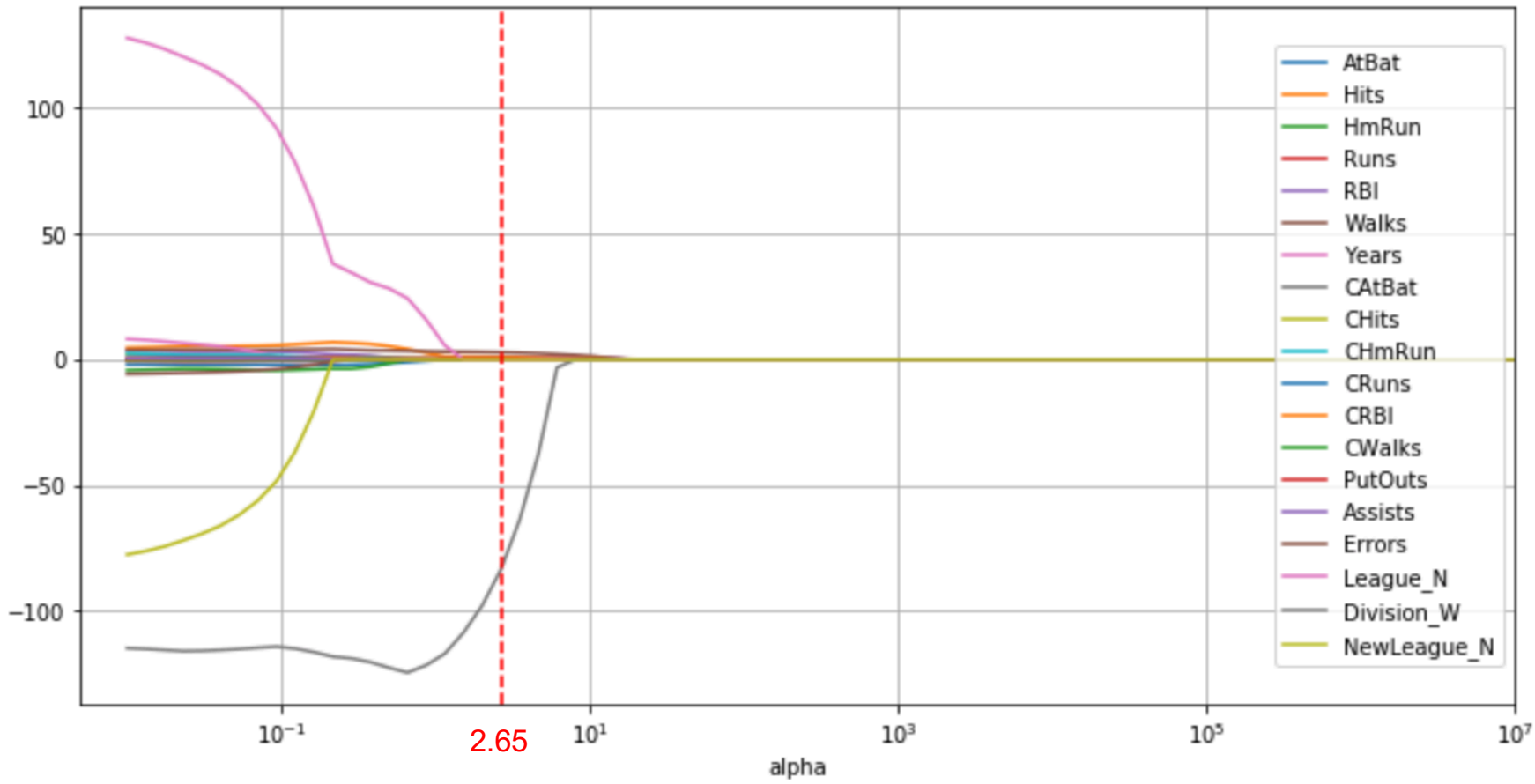
```
# test MSE of best alpha (creating new model)
```

2.656087

```
lasso4 = Lasso(alpha = lassocv.alpha_,normalize = True)
lasso4.fit(X_train, y_train)
mean_squared_error(y_test,lasso4.predict(X_test))
```

```
105162.14640546274
```

Lasso Regression



Lasso Regression Coefficients using best alpha

```
df4 = pd.DataFrame(lasso4.coef_,index=X.columns,columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	1.060001
HmRun	0.000000
Runs	0.000000
RBI	0.000000
Walks	2.859445
Years	0.000000
CAtBat	0.000000
CHits	0.000000

	lasso_coeff
CHmRun	0.206710
CRuns	0.000000
CRBI	0.510668
CWalks	0.000000
PutOuts	0.363681
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-83.474082
NewLeague_N	0.000000

Lasso Regression Coefficients using best alpha

```
df4 = pd.DataFrame(lasso4.coef_,index=X.columns,columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	1.060001
HmRun	0.000000
Runs	0.000000
RBI	0.000000
Walks	2.859445
Years	0.000000
CAtBat	0.000000
CHits	0.000000

	lasso_coeff
CHmRun	0.206710
CRuns	0.000000
CRBI	0.510668
CWalks	0.000000
PutOuts	0.363681
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-83.474082
NewLeague_N	0.000000

13 regression coeffs equal to zero

Lasso Regression Selected Features

```
df4 = pd.DataFrame(lasso4.coef_,index=X.columns,columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	1.060001
HmRun	0.000000
Runs	0.000000
RBI	0.000000
Walks	2.859445
Years	0.000000
CAtBat	0.000000
CHits	0.000000

	lasso_coeff
CHmRun	0.206710
CRuns	0.000000
CRBI	0.510668
CWalks	0.000000
PutOuts	0.363681
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-83.474082
NewLeague_N	0.000000

features selected by LASSO

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

	lasso_coeff
Hits	1.060001
Walks	2.859445
CHmRun	0.206710
CRBI	0.510668
PutOuts	0.363681
Division_W	-83.474082

Comparison of Lasso and Ridge Regression coefficients

```
df4 = pd.DataFrame(lasso4.coef_,index=X.columns,columns=['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	1.060001
HmRun	0.000000
Runs	0.000000
RBI	0.000000
Walks	2.859445
Years	0.000000
CAtBat	0.000000
CHits	0.000000
CHmRun	0.206710
CRuns	0.000000
CRBI	0.510668
CWalks	0.000000
PutOuts	0.363681
Assists	-0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-83.474082
NewLeague_N	0.000000

13 regression coeffs equal to zero

	ridge_coeff
AtBat	0.039089
Hits	0.987271
HmRun	0.210328
Runs	1.108092
RBI	0.875440
Walks	1.778644
Years	0.351680
CAtBat	0.011224
CHits	0.063656
CHmRun	0.443470
CRuns	0.126382
CRBI	0.135001
CWalks	0.033335
PutOuts	0.188256
Assists	0.040423
Errors	-1.751570
League_N	26.494555
Division_W	-90.056030
NewLeague_N	7.580892

No regression coeffs are equal to zero

Lasso Regression - Prediction

```
# coefficients for new predictions -use full data

lasso4.fit(X, y)
```

Lasso Regression - Prediction

```
# coefficients for new predictions -use full data

lasso4.fit(X, y)
df5 = pd.DataFrame(lasso4.coef_, index=X.columns, columns = ['lasso_coeff'])
```

	lasso_coeff		lasso_coeff
AtBat	0.000000	CHmRun	0.000000
Hits	1.645595	CRuns	0.181064
HmRun	0.000000	CRBI	0.377327
Runs	0.000000	CWalks	0.000000
RBI	0.000000	PutOuts	0.152331
Walks	1.903195	Assists	0.000000
Years	0.000000	Errors	-0.000000
CAtBat	0.000000	League_N	0.000000
CHits	0.000000	Division_W	-55.866189
CHmRun	0.000000	NewLeague_N	0.000000

regression coeffs slightly different

Lasso Regression - Prediction

```
# coefficients for new predictions -use full data
```

```
lasso4.fit(X, y)
df5 = pd.DataFrame(lasso4.coef_, index=X.columns, columns = ['lasso_coeff'])
```

	lasso_coeff
AtBat	0.000000
Hits	1.645595
HmRun	0.000000
Runs	0.000000
RBI	0.000000
Walks	1.903195
Years	0.000000
CAtBat	0.000000
CHits	0.000000
CHmRun	0.000000

	lasso_coeff
CHmRun	0.000000
CRuns	0.181064
CRBI	0.377327
CWalks	0.000000
PutOuts	0.152331
Assists	0.000000
Errors	-0.000000
League_N	0.000000
Division_W	-55.866189
NewLeague_N	0.000000

```
df5[df5.lasso_coeff != 0]
```

	lasso_coeff
Hits	1.645595
Walks	1.903195
CRuns	0.181064
CRBI	0.377327
PutOuts	0.152331
Division_W	-55.866189

Lasso Regression - Prediction

```
# predict salary of first player in test set
```

```
newval = X_test[:1]
newval
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	...
126	282.0	78.0	13.0	37.0	51.0	29.0	5.0	1649.0	453.0	73.0	

```
df3 = lasso4.predict(newval)
df3
```

```
array([447.93593068])
```

```
# predicted salary is 447,936 dollars
```

Shrinkage Methods

Logistic regression with Regularization

LOGISTIC REGRESSION LOSS FUNCTION – One predictor

If the likelihood function is

$$L(b_0, b_1) = \prod_{i=1}^{n} \frac{\left(e^{b_0+b_1 x_i}\right)^{y_i}}{1 + e^{b_0+b_1 x_i}}$$

the log-likelihood is

$$\log\ L(b_0, b_1) = \sum_{i=1}^{n} y_i\,(b_0 + b_1\,x_i) - \sum_{i=1}^{n} \log(1 + e^{b_0+b_1\,x_i})$$

this expression is the loss function for logistic regression

LOGISTIC REGRESSION LOSS FUNCTION – One predictor

The logistic regression loss function is

$$\underset{b_0,b_1}{\text{Max}} \quad \log\, L(b_0, b_1) \;=\; \sum_{i=1}^{n} y_i\,(b_0 + b_1\,x_i) \;-\; \sum_{i=1}^{n} \log(1 + e^{b_0 + b_1\,x_i})$$

The logistic regression loss function with regularization

$$\underset{b_0,b_1}{\text{Max}}\ \log\, L(b_0, b_1) \;=\; \sum_{i=1}^{n} y_i\,(b_0 + b_1\,x_i) \;-\; \sum_{i=1}^{n} \log(1 + e^{b_0 + b_1\,x_i}) + \alpha\, b_1^2$$

LOGISTIC REGRESSION LOSS FUNCTION – p predictors

The logistic regression loss function is

$$\underset{b_0,\ldots,b_p}{\text{Max}} \log\, L(b_0, \ldots, b_p) \;=\; \sum_{i=1}^{n} y_i\,(b_0 + b_1\,x_{1i} + \cdots + b_p\,x_{pi}) - \sum_{i=1}^{n} \log(1 + e^{b_0 + b_1\,x_{1i} + \cdots + b_p\,x_{pi}})$$

The logistic regression loss function with regularization

$$\underset{b_0,\ldots,b_p}{\text{Max}} \log\, L(b_0, \ldots, b_p) \;=\; \sum_{i=1}^{n} y_i\,(b_0 + b_1\,x_{1i} + \cdots + b_p\,x_{pi}) - \sum_{i=1}^{n} \log(1 + e^{b_0 + b_1\,x_{1i} + \cdots + b_p\,x_{pi}}) + \sum_{i=1}^{p} \alpha\, b_i^2$$

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)
```

Logistic regression

Example
Cancer data

Cancer Data - EXAMPLE

- The Cancer data from *sklearn* contains data from 569 patients.
- It includes 30 measurements associated with breast cancer tumors. These are the predictors.
- Each tumor is labeled as “benign” or 1 (for harmless tumors), or “malignant” or 0 (for cancerous tumors). This is the target.
- Build a Logistic Regression model to predict whether the tumor of a patient is malignant.
- Use K-fold 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)
```

```
print(len(X_train),
      len(X_test))
```

```
426 143
```

Logistic Regression – SCALE predictors

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

scale

```
print(len(X_train),
      len(X_test))
```

```
426 143
```

```
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)

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)

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)

model.score(X_test,y_test)
```

```
0.9440559440559441
```

α = 1/C = 1 is the default value

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000,C=1)
model.fit(X_train_scaled,y_train)

model.score(X_test_scaled,y_test)
```

```
0.972027972027972
```

this model was used with regularization and with scaled data

Logistic Regression – Holdout Cross Validation

.

not scaling the data

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000)
model.fit(X_train,y_train)

model.score(X_test,y_test)
```

```
0.9440559440559441
```

C = 1e20 means α = 1/C = 0 (no regularization)

```
model = LogisticRegression(solver = 'lbfgs',max_iter=10000,C=1e20)
model.fit(X_train_scaled,y_train)

model.score(X_test_scaled,y_test)
```

```
0.9440559440559441
```

thus, scaling the data did not help increasing the model performance

Logistic Regression – Holdout Cross Validation – Search best C value

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

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

```
1490
```

```
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)
```

Logistic Regression – Holdout Cross Validation – Search best C value

```
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[:9]
```

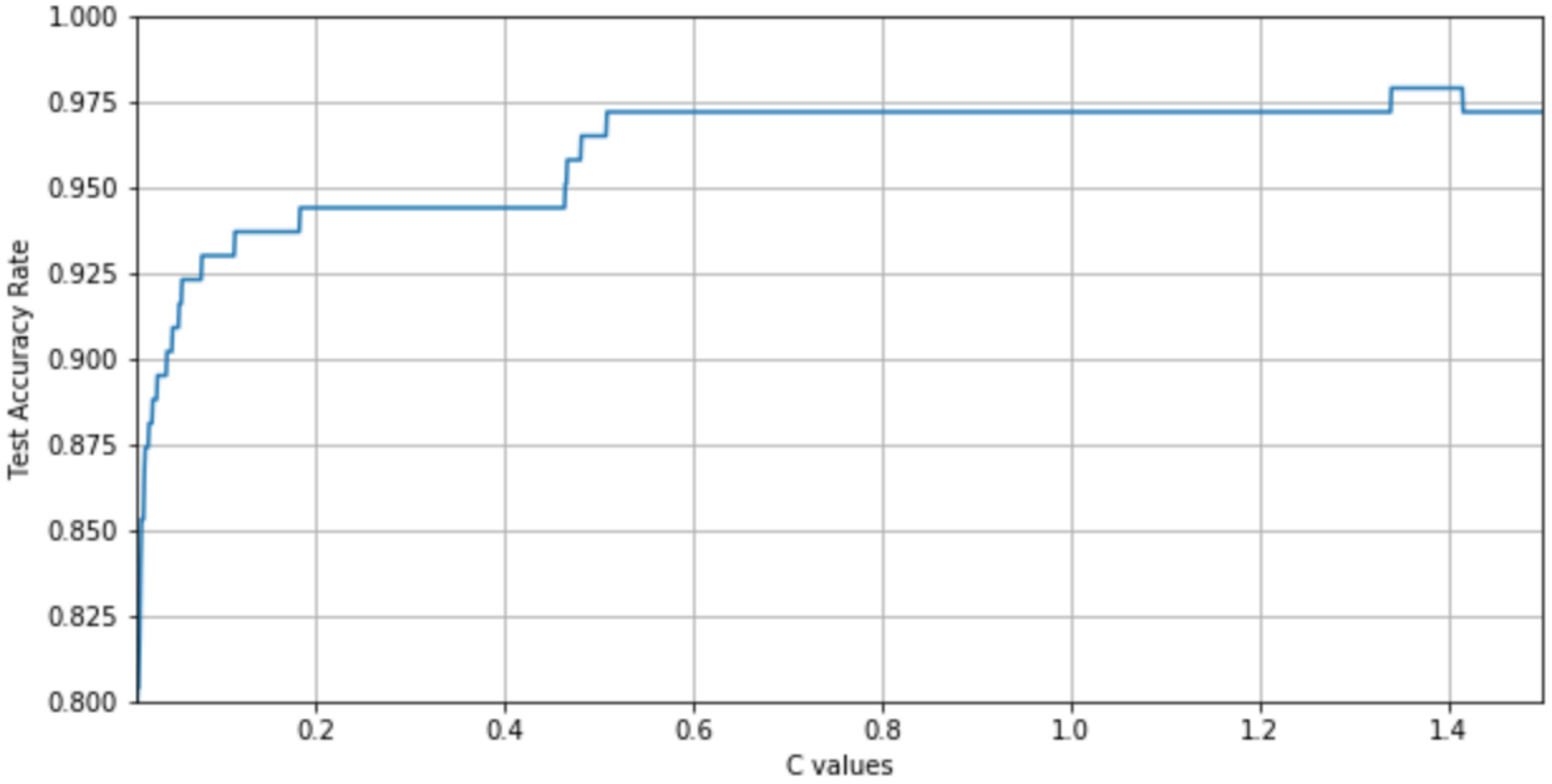
```
[0.7972027972027972,
 0.7972027972027972,
 0.8041958041958042,
 0.8041958041958042,
 0.8181818181818182,
 0.8391608391608392,
 0.8531468531468531,
 0.8531468531468531,
 0.8531468531468531]
```

```
df2 = pd.DataFrame(arates,
                   columns = ['Test accuracy'])
df2.index = Cvalues
df2.index.name = 'C values'
df2.head(9)
```

C values	Test accuracy
0.010	0.797203
0.011	0.797203
0.012	0.804196
0.013	0.804196
0.014	0.818182
0.015	0.839161
0.016	0.853147
0.017	0.853147
0.018	0.853147

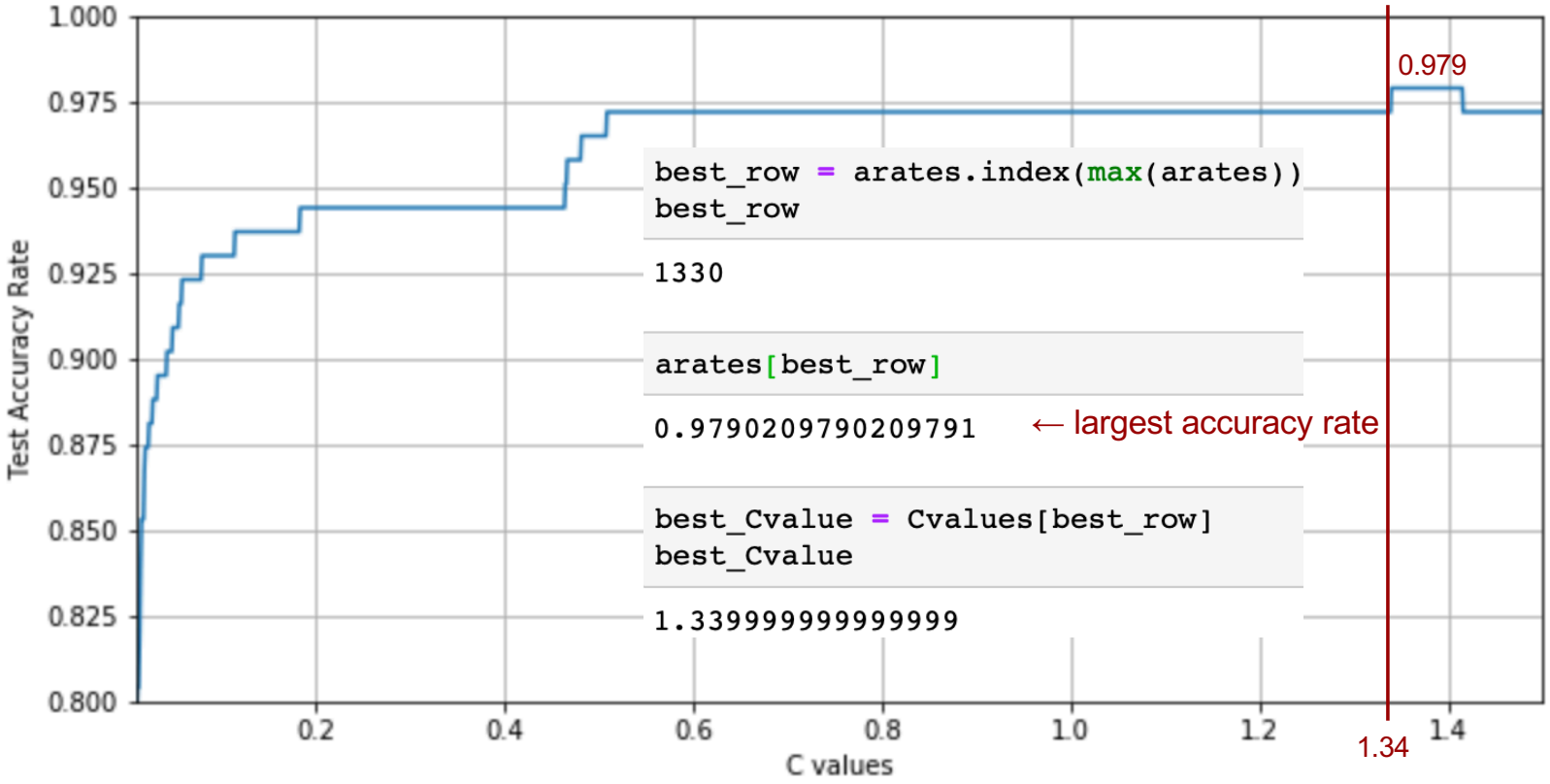
Logistic Regression – Holdout Cross Validation – Search best C value

```
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
```