Regression Analysis - Practice

Patrick C. Shih

Assistant Professor
Department of Informatics
Indiana University Bloomington

By the end of this lecture, you should be able to

- Understand and write code for regression using sklearn
- Understand the performance evaluation of models through Training and Test data
- Understand overfitting

Diabetes progress prediction example

Libraries used

```
import sklearn.datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import normalize
from sklearn.metrics import r2_score
```

Call Data & Split Training / Test Data

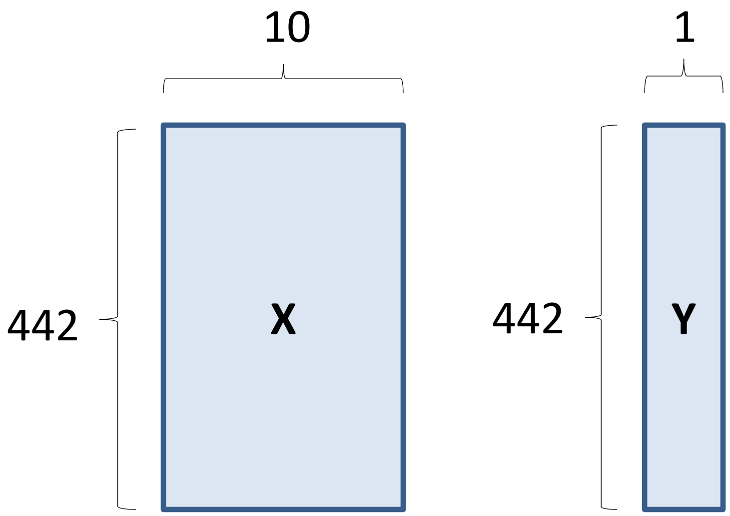
Leverage the data provided by sklearn

```
In [4]: diabetes = sklearn.datasets.load_diabetes()
        X, Y = normalize(diabetes['data']), diabetes['target']
```

Normalize data through normalize ()

See what your data looks like

```
In [29]: print(X.shape, Y.shape)

(442, 10) (442,)
```

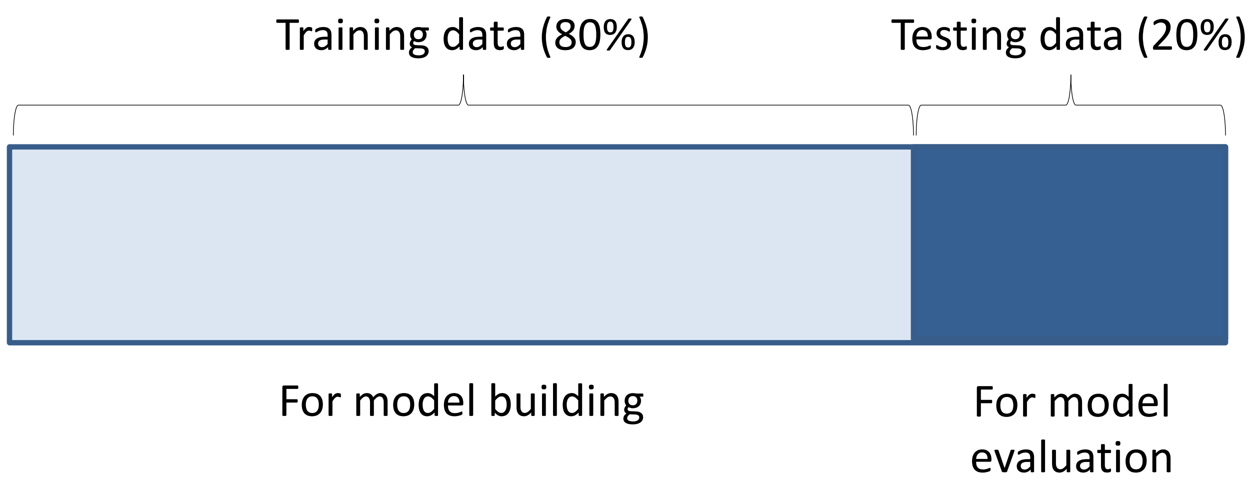


Split Training / Test Data

Data segmentation with Training and Test Datasets

In [22]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2)
```



Linear Regression Modeling

Input:

```
# linear regression
linear = LinearRegression()

# train data
linear.fit(X_train, Y_train)
```

Fit model based on X_train and Y_train values

Output:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Linear Regression Model Performance Evaluation

```
# test data
Y_predicted = linear.predict(X_test)
```

Get the predicted Y result (pred_linear) through the model based on the X_test value

```
# correlation
corr_linear = pd.Series(Y_predicted).corr(pd.Series(Y_test))

# r2_score
rsquared_linear = r2_score(Y_predicted, Y_test)
```

How much difference is there actually between the pred_linear value and the actual Y_test value?
Confirmed

- Correlation through corr()
- Performance comparison with r2_score ()

Check the result

```
print(linear.coef_)
```

Check the coefficients of the models of 10 features by checking the linear coefficients

```
[ -0.94354742 -35.53781008  81.5990139   48.22549766 -22.37633861
  12.28729454 -30.46534849   4.55101779  79.54184377  16.31606145]
```

```
plt.scatter(Y_predicted, Y_test)

plt.title("Linear Regression. Correlation=%f  $R^2$ score=%f" %
(corr_linear, rsquared_linear))

plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")

# plot a diagonal line
plt.plot(Y_test, Y_test, 'k--')
plt.show()
```

Graphical representation of results

Housing Data

Housing Data

- Information about houses outside Boston, 1987
- Free download from UCI Machine Learning Repository
- 506 samples, 14 features

Import data

```
import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data',header=None, sep='\s+')
```

Data Exploration

```
In [2]: df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
```

```
In [3]: df.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

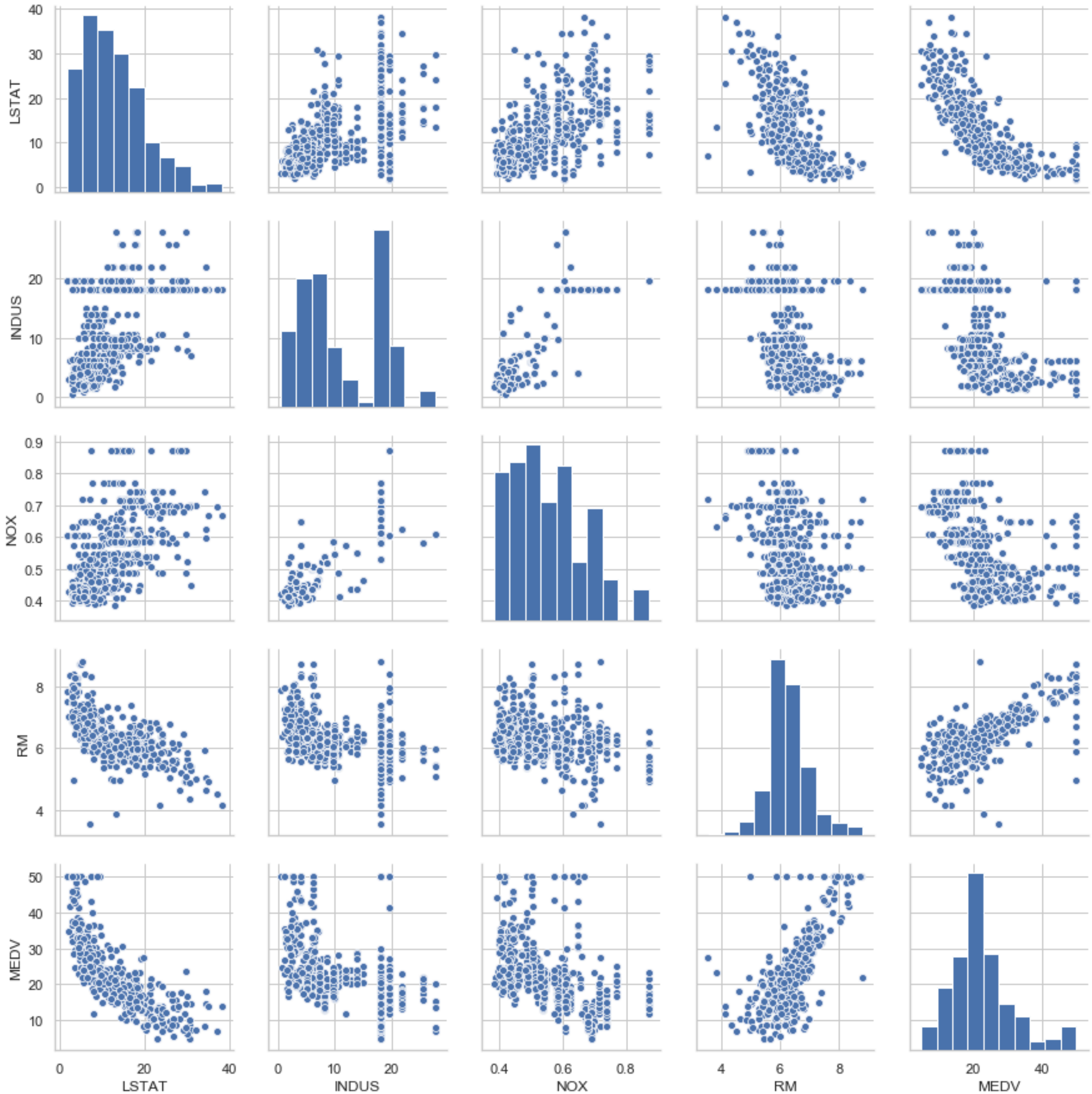
1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centers
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

Data Visualization

Various visualizations can be utilized with the Seaborn library (https://seaborn.pydata.org/)

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style='whitegrid', context='notebook')
cols = ['LSTAT', 'INDUS', 'NOX', 'RM', 'MEDV']

sns.pairplot(df[cols], height=2.5)
plt.show()
```

Data Visualization



Data Visualization

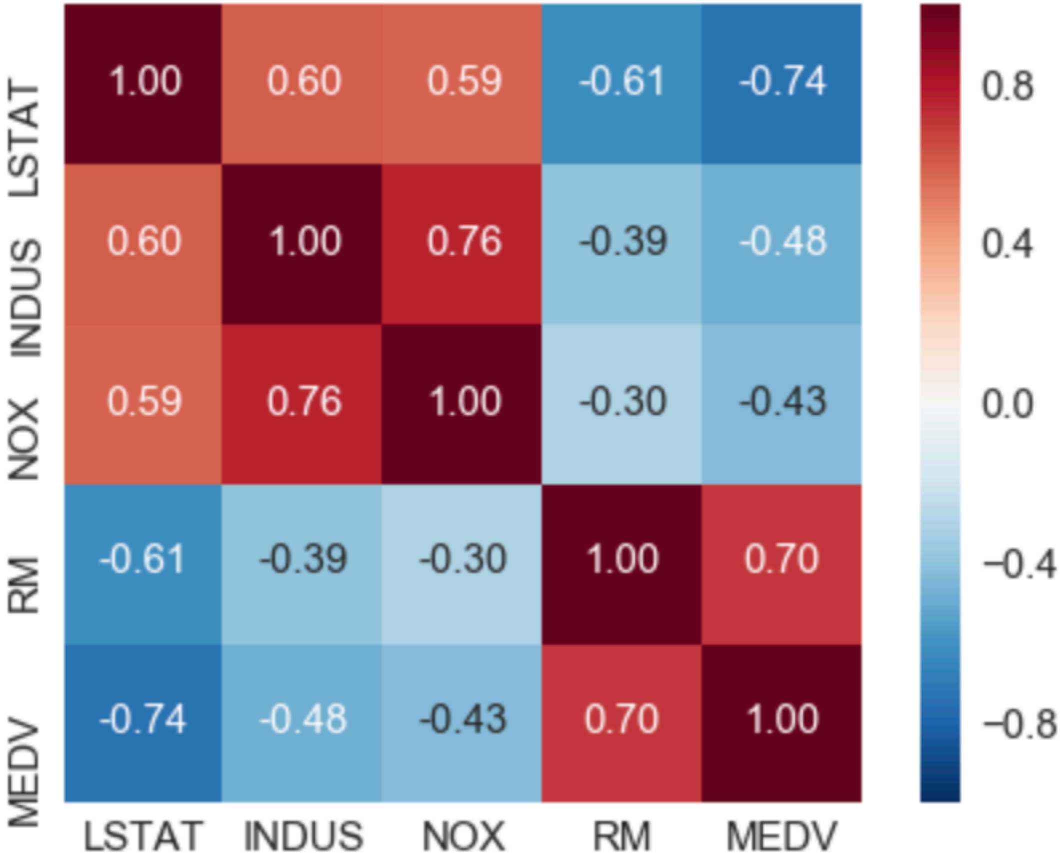
Various visualizations can be utilized with the Seaborn library (https://seaborn.pydata.org/)

Correlation between features through heatmap

```
In [10]: cm = np.corrcoef(df[cols].values.T)
```

```
In [11]: sns.set(font_scale=1.5)
         hm = sns.heatmap(cm,
                          cbar=True,
                          annot=True,
                          square=True,
                          fmt='.2f',
                          annot_kws={'size':15},
                          yticklabels=cols,
                          xticklabels=cols)
         plt.show()
```

Heatmap Results



Normalization (StandardScaler) & Linear Regression

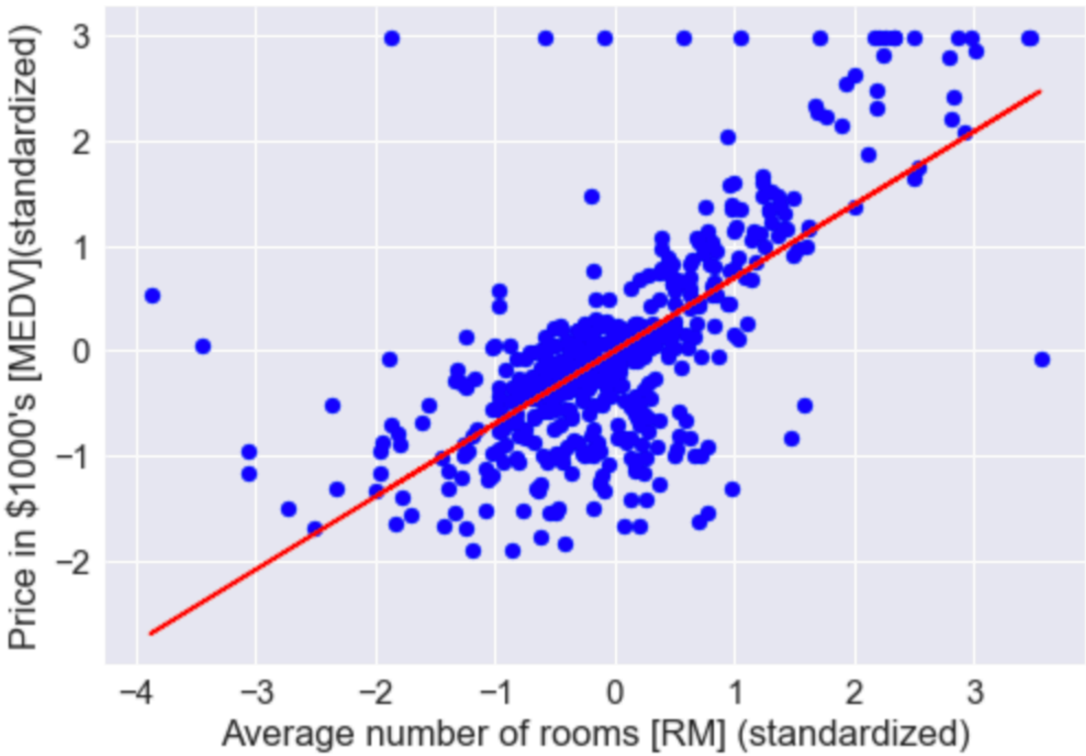
In [27]:
```
from sklearn.preprocessing import StandardScaler

X = df[['RM']].values
y = df[['MEDV']].values

sc = StandardScaler()

X_sc = sc.fit_transform(X)
y_sc = sc.fit_transform(y)
```

In [28]:
```
from sklearn.linear_model import LinearRegression

slr = LinearRegression()
slr.fit(X_sc, y_sc)
print('Slope: %.3f' % slr.coef_[0])
print('Intercept: %.3f' % slr.intercept_)
```

```
Slope: 0.695
Intercept: -0.000
```

Visualize relationships between variables

In [29]:
```
plt.scatter(X_sc, y_sc, c='blue')
plt.plot(X_sc, slr.predict(X_sc), color='red')
plt.xlabel('Average number of rooms [RM] (standardized)')
plt.ylabel('Price in $1000\'s [MEDV](standardized)')
plt.show()
```



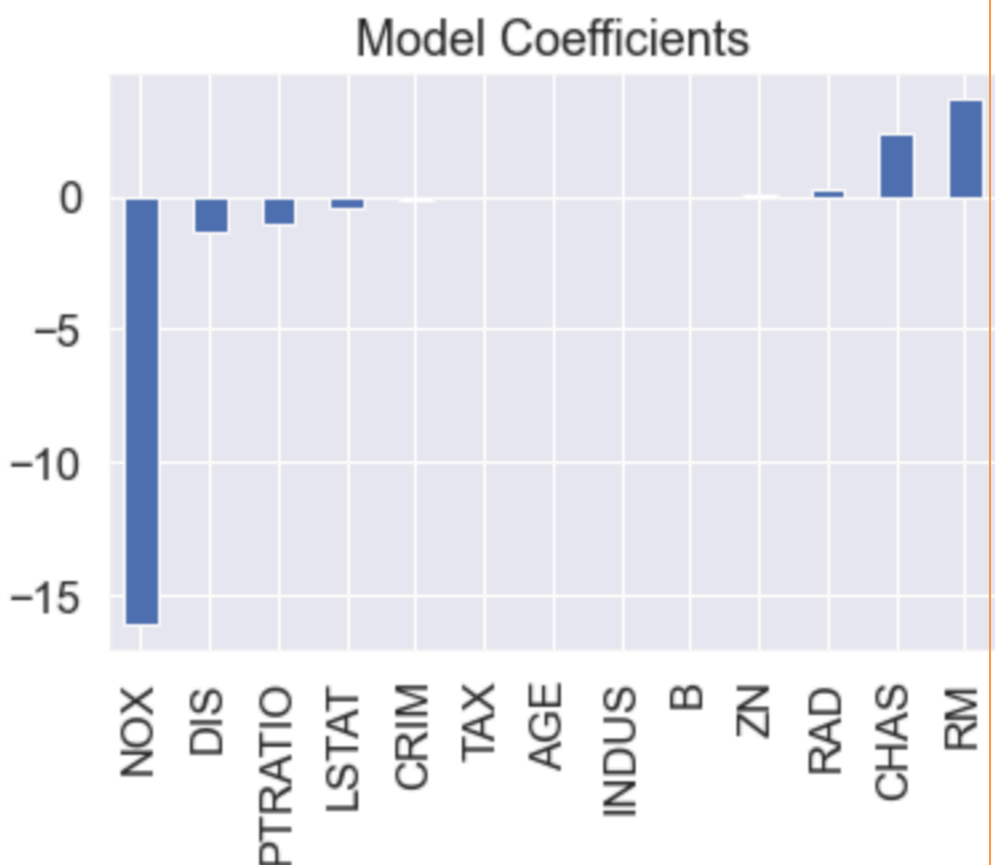
Model building and evaluation

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, :-1].values
y = df['MEDV'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

slr = LinearRegression()
slr.fit(X_train, y_train)

y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
```

Model building and evaluation

```
# checking the magnitude of coefficients
coef = pd.Series(slr.coef_, df.columns[:-1]).sort_values()

coef.plot(kind='bar', title='Model Coefficients')
```

```
<AxesSubplot:title={'center':'Model Coefficients'}>
```



1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centers
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

Visualize relationships between model results

In [33]:
```python
plt.scatter(y_train_pred, y_train_pred - y_train,
            c='blue',
            marker='o',
            label='Training data')
plt.scatter(y_test_pred, y_test_pred - y_test,
            c='lightgreen',
            marker='s',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, lw=2, color='red')
plt.xlim([-10, 50])
plt.show()
```

Model performance evaluation

```
from sklearn.metrics import r2_score

print('R^2 train: %.3f, test: %.3f' %
      (r2_score(y_train, y_train_pred),
      r2_score(y_test, y_test_pred)))
```

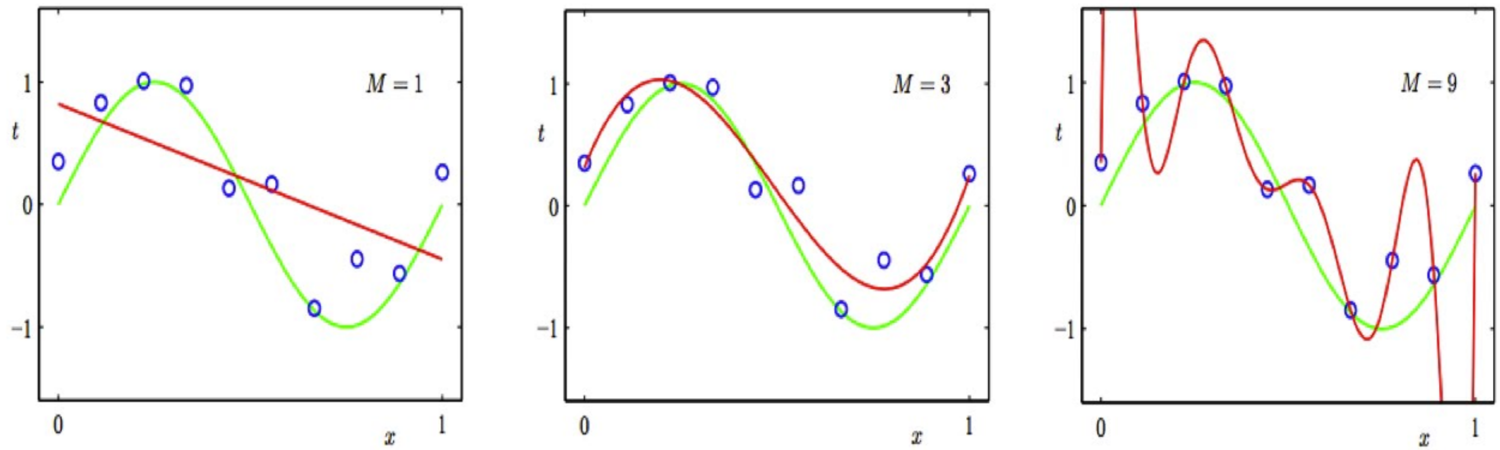
```
R^2 train: 0.773, test: 0.589
```

Training data based performance is higher than test data based performance
→ Overfitting

Overfitting

It is very important to build the model to avoid overfitting

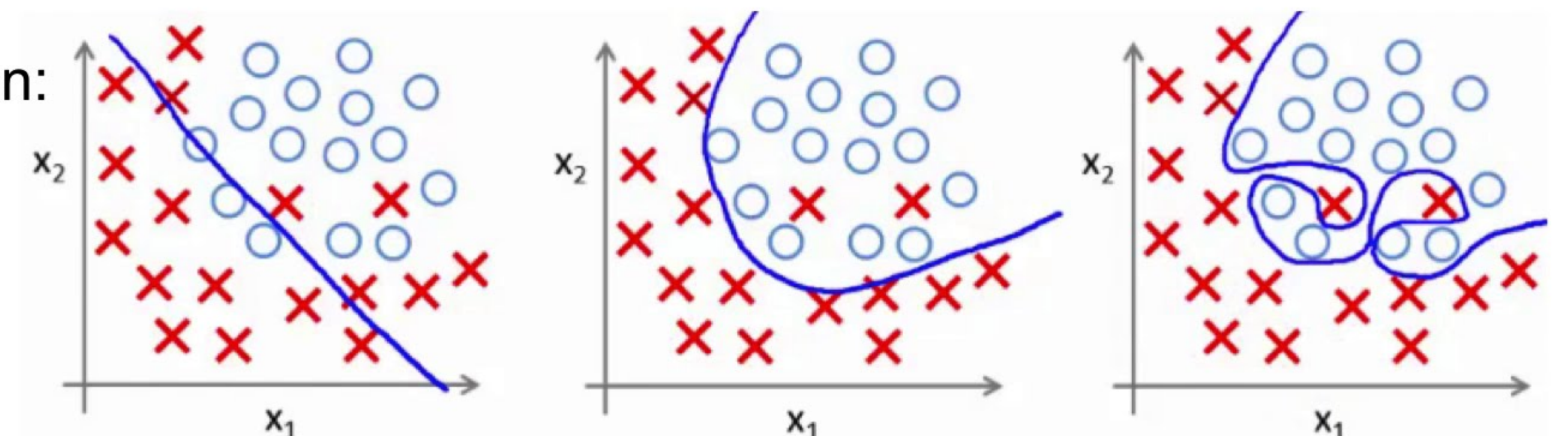
Regression:



predictor too inflexible:
cannot capture pattern

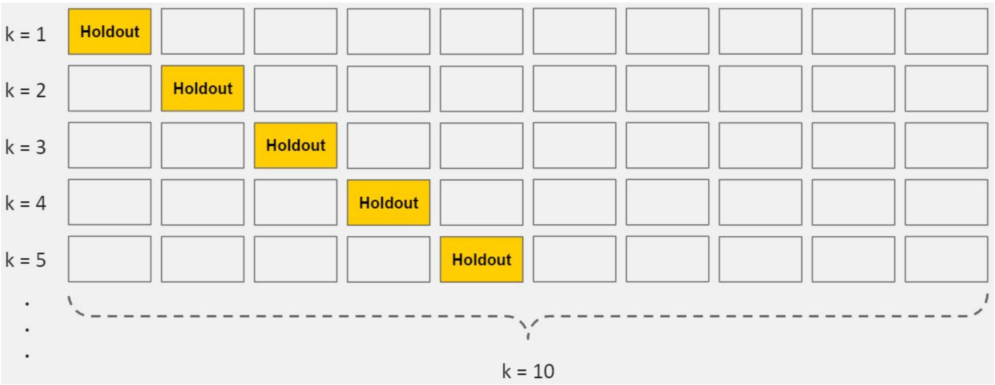
predictor too flexible:
fits noise in the data

Classification:

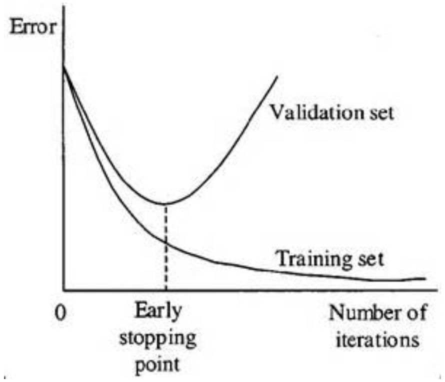


How to Avoid Overfitting

- Cross-validation
- Train with more data
- Remove features (Curse of Dimension)
- Early stopping
- Regularization
- Ensembling



Cross-validation



Early stopping

K-Fold Cross-Validation

```
# import cross validation library
from sklearn.model_selection import cross_val_score

# using slr to fit x_train, y_train with 10-fold cross-validation
scores = cross_val_score(slr, X_train, y_train, cv=10)

# average R^2 scores
print(scores.mean())
```

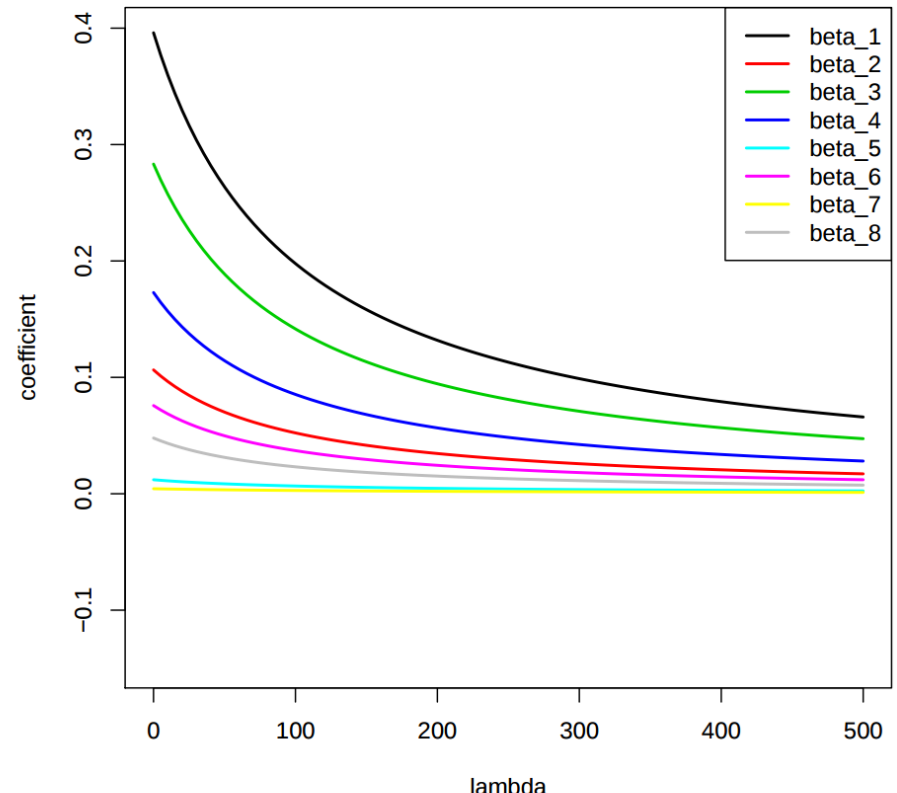
```
0.7310126345738197
```

Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

L1 (Ridge) Regularization

```
# import ridge regression library
from sklearn.linear_model import Ridge

# using ridge to fit X_train, y_train with 10-fold cross-validation over
alpha values from 0.1 to 0.9
for i in range (1,10):
    ridge = Ridge(alpha=i/10)
    scores = cross_val_score(ridge, X_train, y_train, cv=10)
    print(i/10, ": ", scores.mean())
```

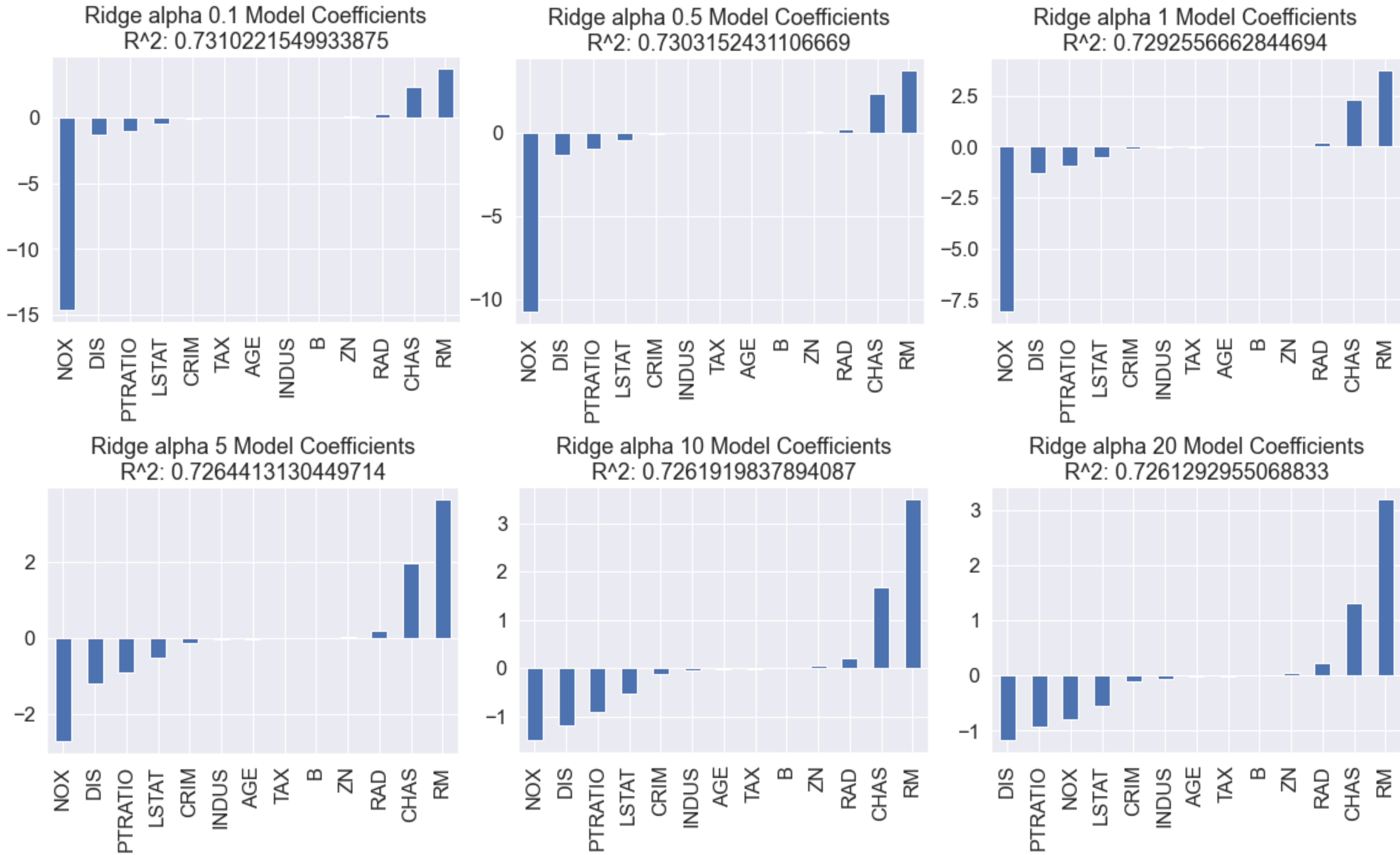
0.1 : 0.7310221549933875
0.2 : 0.7309139256060135
0.3 : 0.7307413740754638
0.4 : 0.730535650471918
0.5 : 0.7303152431106669
0.6 : 0.7300912148288151
0.7 : 0.729870164622047
0.8 : 0.7296559566028753
0.9 : 0.7294507564036607



L1 (Ridge) Regularization

```
# import ridge regression library
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_validate
import matplotlib.pyplot as plt

coefs = []

# using ridge to fit X_train, y_train with 10-fold cross-validation over alpha values
for i, alpha in enumerate([0.1, 0.5, 1, 5, 10, 20, 50, 100]):
    # adding new df for alpha results to list
    coefs.append(pd.DataFrame(columns=df.columns[:-1]))
    coefs_average = []
    scores = []

    ridge = Ridge(alpha=alpha)
    cv_results = cross_validate(ridge, X_train, y_train, cv=10, return_estimator=True)
    scores = cross_val_score(ridge, X_train, y_train, cv=10)

    # compute average coef value over k-fold per feature
    for j in cv_results['estimator']:
        coefs[i].loc[j]=j.coef_
    # create list of average coefs
    for column in coefs[i]:
        coefs_average.append(coefs[i][column].mean())
    # plot
    plt.figure()
    coef = pd.Series(coefs_average, df.columns[:-1]).sort_values()
    coef.plot(kind='bar', title='Ridge alpha ' + str(alpha) + ' Model Coefficients. R^2: ' + str(scores.mean()))
```

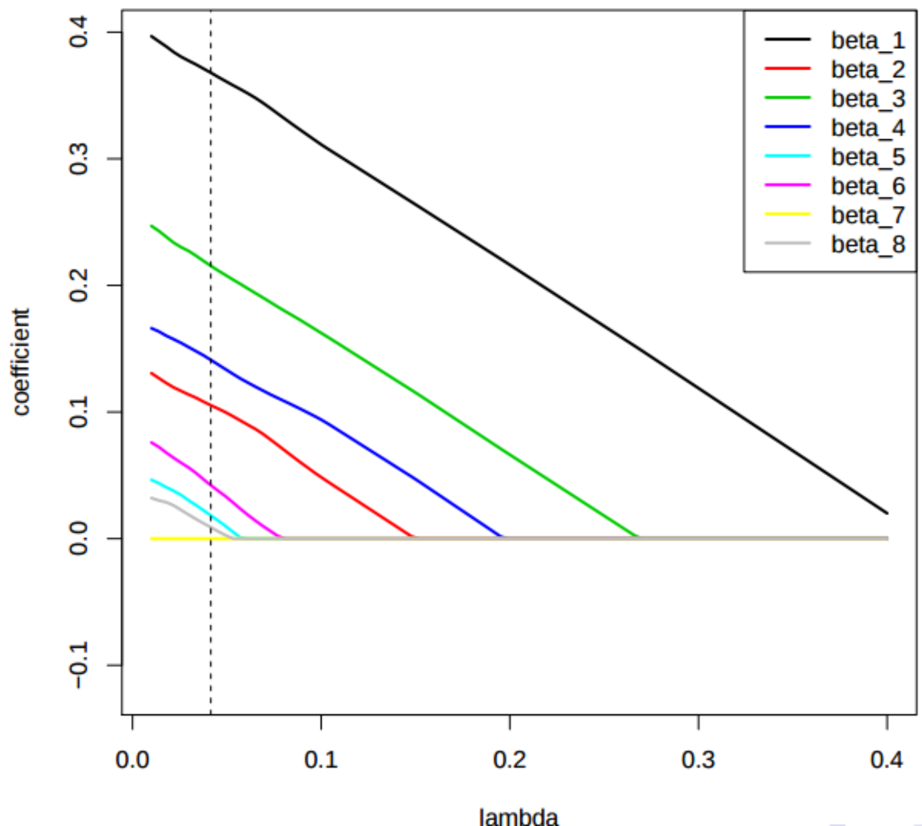
L1 (Ridge) Regularization



L2 (Lasso) Regularization

```
# import ridge regression library
from sklearn.linear_model import Lasso

# using lasso to fit X_train, y_train with 10-fold cross-validation over
alpha values from 0.1 to 0.9
for i in range (1,10):
    lasso = Lasso(alpha=i/10)
    scores = cross_val_score(lasso, X_train, y_train, cv=10)
    print(i/10, ": ", scores.mean())
```

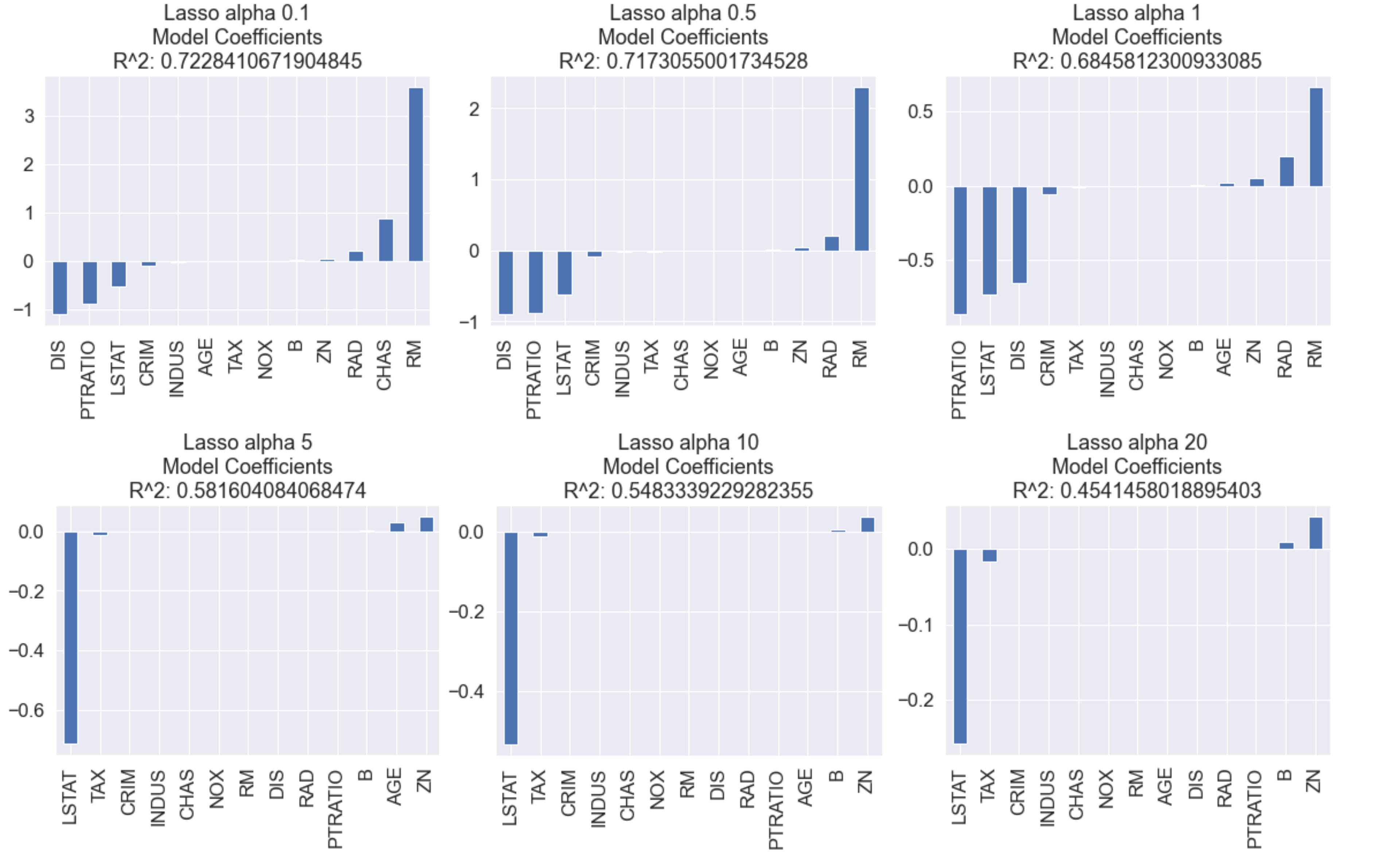
0.1 : 0.7228410671904845
0.2 : 0.7239171181407429
0.3 : 0.72270104511865
0.4 : 0.7204008221252061
0.5 : 0.7173055001734528
0.6 : 0.712842746405362 4
0.7 : 0.707271716728902 9
0.8 : 0.7007677899893462
0.9 : 0.6931990674107382



L2 (Lasso) Regularization

```
# import lasso regression library
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_validate
import matplotlib.pyplot as plt

coefs = []

# using lasso to fit X_train, y_train with 10-fold cross-validation over alpha values
for i, alpha in enumerate([0.1, 0.5, 1, 5, 10, 20, 50, 100]):
    # adding new df for alpha results to list
    coefs.append(pd.DataFrame(columns=df.columns[:-1]))
    coefs_average = []
    scores = []

    lasso = Lasso(alpha=alpha)
    cv_results = cross_validate(lasso, X_train, y_train, cv=10, return_estimator=True)
    scores = cross_val_score(lasso, X_train, y_train, cv=10)

    # compute average coef value over k-fold per feature
    for j in cv_results['estimator']:
        coefs[i].loc[j]=j.coef_
    # create list of average coefs
    for column in coefs[i]:
        coefs_average.append(coefs[i][column].mean())
    # plot
    plt.figure()
    coef = pd.Series(coefs_average, df.columns[:-1]).sort_values()
    coef.plot(kind='bar', title='Lasso alpha ' + str(alpha) + ' Model Coefficients. R^2: ' + str(scores.mean()))
```

L2 (Lasso) Regularization



Next Class

- Learn about Classification, a typical supervised learning

Thank you

Patrick C. Shih