S-109A Introduction to Data Science

Homework 4 - Regularization

Harvard University
Summer 2018
Instructors: Pavlos Protopapas, Kevin Rader

INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.

Names of people you have worked with goes here:

In [349]:
```
from IPython.core.display import HTML
def css_styling(): styles = open("cs109.css", "r").read(); return HTML(styles)
css_styling()
```

Out[349]:

import these libraries

In [350]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import KFold

#import statsmodels.api as sm

from pandas.core import datetools
%matplotlib inline
```

Continuing Bike Sharing Usage Data

In this homework, we will focus on regularization and cross validation. We will continue to build regression models for the Capital Bikeshare program in Washington D.C. See homework 3 for more information about the Capital Bikeshare data that we'll be using extensively.

Data Preparation

Question 1

In HW3 Questions 1-3, you preprocessed the data in preparation for your regression analysis. We ask you to repeat those steps (particularly those in Question 3) so that we can compare the analysis models in this HW with those you developed in HW3. In this HW we'll be using models from sklearn exclusively (as opposed to statsmodels)

1.1 [From HW3] Read `data/BSS_train.csv` and `data/BSS_test.csv` into dataframes `BSS_train` and `BSS_test`, respectively. Remove the `dteday` column from both train and test dataset. We do not need it, and its format cannot be used for analysis. Also remove the `casual` and `registered` columns for both training and test datasets as they make `count` trivial.

1.2 Since we'll be exploring Regularization and Polynomial Features, it will make sense to standardize our data. Standardize the numerical features. Store the dataframes for the processed training and test predictors into the variables `X_train` and `X_test`. Store the appropriately shaped numpy arrays for the corresponding train and test `count` columns into `y_train` and `y_test`.

1.3 Use the `LinearRegression` library from `sklearn` to fit a multiple linear regression model to the training set data in `X_train`. Store the fitted model in the variable `BikeOLSModel`.

1.4 What are the training and test set R^2 scores? Store the training and test R^2 scores of the `BikeOLSModel` in a dictionary `BikeOLS_r2scores` using the string 'training' and 'test' as keys.

1.5 We're going to use bootstrapped confidence intervals (use 500 bootstrap iterations) to determine which of the estimated coefficients for the `BikeOLSModel` are statistically significant at a significance level of 5% . We'll do so by creating 3 different functions:

1. `make_bootstrap_sample(dataset_X, dataset_y)` returns a bootstrap sample of `dataset_X` and `dataset_y`
2. `calculate_coefficients(dataset_X, dataset_y, model)` returns in the form of a dictionary regression coefficients calculated by your model on `dataset_X` and `dataset_y`. The keys for regression coefficients dictionary should be the names of the features. The values should be the coefficient values of that feature calculated on your model. An example would be {'hum': 12.3, 'windspeed': -1.2, 'Sunday': 0.6 ... }
3. `get_significant_predictors(regression_coefficients, significance_level)` takes as input a list of regression coefficient dictionaries (each one the output of `calculate_coefficients` and returns a python list of the feature names of the significant predictors e.g. ['Monday', 'hum', 'holiday', ...]

In the above functions `dataset_X` should always be a pandas dataframe with your features, `dataset_y` a numpy column vector with the values of the response variable and collectively they form the dataset upon which the operations take place. `model` is the `sklearn` regression model that will be used to generate the regression coefficients. `regression_coefficients` is a list of dictionaries of numpy arrays with each numpy array containing the regression coefficients (not including the intercept) calculated from one bootstrap sample. `significance_level` represents the significance level as a floating point number. So a 5% significance level should be represented as 0.05.

Store the feature names as a list of strings in the variable `BikeOLS_significant_bootstrap` and print them for your answer.

Answers

1.1 Read `data/BSS_train.csv` and `data/BSS_test.csv` into Pandas DataFrames

In [351]:
```
# your code here

BSS_train = pd.read_csv("data/BSS_train.csv")

BSS_train = BSS_train.drop(columns = ['dteday','casual','registered','Unnamed:
 0'])
#delete Unnamed:0 as well.
BSS_test = pd.read_csv("data/BSS_test.csv")

BSS_test = BSS_test.drop(columns = ['dteday','casual','registered','Unnamed:
 0'])
BSS_train.dtypes
```

Out[351]:
```
hour            int64
holiday         int64
year            int64
workingday      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
counts          int64
spring          int64
summer          int64
fall            int64
Feb             int64
Mar             int64
Apr             int64
May             int64
Jun             int64
Jul             int64
Aug             int64
Sept            int64
Oct             int64
Nov             int64
Dec             int64
Mon             int64
Tue             int64
Wed             int64
Thu             int64
Fri             int64
Sat             int64
Cloudy          int64
Snow            int64
Storm           int64
dtype: object
```

1.2 Standardizing our data

In [357]:
```
# your code here
#temp, atemp, hum, windspeed are numerical values.
#hour, year are ordinal values, and the others are binary.
#lets standardize
X_train = BSS_train.copy().drop(columns =['counts']) #make copy of BSS_train w
ithout y values
X_test = BSS_test.copy().drop(columns =['counts'])
y_train = BSS_train.counts
y_test = BSS_test.counts
#standardize data.
for i in ['temp','atemp','hum','windspeed']:
    X_train[i] = (BSS_train[i] - np.mean(BSS_train[i])) / np.std(BSS_train[i])
for i in ['temp','atemp','hum','windspeed']:
    X_test[i] = (BSS_test[i] - np.mean(BSS_test[i])) / np.std(BSS_test[i])
X_train.describe()
X_test.describe()
```

Out[357]:

	hour	holiday	year	workingday	temp	atem
count	3476.000000	3476.000000	3476.000000	3476.000000	3.476000e+03	3.476000e+0
mean	11.615938	0.026755	0.494534	0.691024	8.834258e-15	-1.373444e-1
std	6.901033	0.161389	0.500042	0.462138	1.000144e+00	1.000144e+0
min	0.000000	0.000000	0.000000	0.000000	-2.489906e+00	-2.606599e+0
25%	6.000000	0.000000	0.000000	0.000000	-8.230553e-01	-8.392069e-0
50%	12.000000	0.000000	0.000000	1.000000	1.036986e-02	4.448903e-02
75%	17.000000	0.000000	1.000000	1.000000	8.437950e-01	8.401070e-01
max	23.000000	1.000000	1.000000	1.000000	2.406467e+00	2.784238e+0

8 rows × 31 columns

1.3 Use the `LinearRegression` library from `sklearn` to fit a multiple linear regression.

In [311]:
```
# your code here

#make regression model by fit.
BikeOLSModel = LinearRegression().fit(X_train,y_train)
BikeOLSModel_test = LinearRegression().fit(X_test,y_test)
#test model as well.

print('train set R score')
print(BikeOLSModel.score(X_train,y_train))
print('test set R score')
print(BikeOLSModel_test.score(X_test,y_test))
```

```
train set R score
0.406538782797
test set R score
0.414481133552
```

1.4 What are the training and test set R^2 scores? Store the R^2 scores of the `BikeOLSModel` on the training and test sets in a dictionary `BikeOLS_r2scores`.

Your answer here

---

train set R score 0.406538782797

test set R score 0.414481133552

It is not that low but also not that high.

we do not have no clue yet.

(Before comparing with others)

In [354]:
```
# your code here

#store r2scores.
BikeOLS_r2scores ={'training' : BikeOLSModel.score(X_train,y_train) ,
                   'test' : BikeOLSModel_test.score(X_test,y_test)}
BikeOLS_r2scores
```

Out[354]:
```
{'test': 0.41448113355225891, 'training': 0.40653878279690858}
```

1.5 We're going to use bootstrapped confidence intervals (use 500 bootstrap iterations) to determine which of the estimated coefficients for the `BikeOLSModel` are statistically significant at a significance level of 5% . We'll do so by creating 3 different functions:

1. `make_bootstrap_sample(dataset_X, dataset_y)` returns a bootstrap sample of `dataset_X` and `dataset_y`
2. `calculate_coefficients(dataset_X, dataset_y, model)` returns in the form of a dictionary regression coefficients calculated by your model on `dataset_X` and `dataset_y`. The keys for regression coefficients dictionary should be the names of the features. The values should be the coefficient values of that feature calculated on your model. An example would be {'hum': 12.3, 'windspeed': -1.2, 'Sunday': 0.6 ... }
3. `get_significant_predictors(regression_coefficients, significance_level)` takes as input a list of regression coefficient dictionaries (each one the output of `calculate_coefficients` and returns a python list of the feature names of the significant predictors e.g. ['Monday', 'hum', 'holiday', ...]

In the above functions `dataset_X` should always be a pandas dataframe with your features, `dataset_y` a numpy column vector with the values of the response variable and collectively they form the dataset upon which the operations take place. `model` is the `sklearn` regression model that will be used to generate the regression coefficients. `regression_coefficients` is a list of dictionaries of numpy arrays with each numpy array containing the regression coefficients (not including the intercept) calculated from one bootstrap sample. `significance_level` represents the significance level as a floating point number. So a 5% significance level should be represented as 0.05.

Store the feature names as a list of strings in the variable `BikeOLS_significant_bootstrap` and print them for your answer.

In [196]:
```
def make_bootstrap_sample(dataset_X, dataset_y, size = None):

    # by default return a bootstrap sample of the same size as the original da
taset
    if not size: size = len(dataset_X)

    # if the X and y datasets aren't the same size, raise an exception
    if len(dataset_X) != len(dataset_y):
        raise Exception("Data size must match between dataset_X and dataset_y"
)

    #your code here

    #I changed a location of code because i want to use if condition above.
    #use .sample method to make bootstrap. n = size , with replace.
    bootstrap_dataset_X = dataset_X.sample(n=size, replace=True)
    bootstrap_dataset_y = dataset_y.loc[bootstrap_dataset_X.index]
    #use loc to find same index's response values.

    return (bootstrap_dataset_X, bootstrap_dataset_y)


def calculate_coefficients(dataset_X, dataset_y, model):
    # your code here
    coef = model.coef_ #find coef
    coefficients_dictionary = {}  #make empty dictionary; i will store the res
ults.

    #use for loop to make dictionary, each predictior : coef
    for i_column,i_coef in zip(dataset_X.columns,coef):
        pair = {i_column : i_coef}
        coefficients_dictionary.update(pair)  #store in dictionary

    return coefficients_dictionary


def get_significant_predictors(regression_coefficients, significance_level):
    # your code here
    coeflist = {k:[d[k] for d in regression_coefficients] for k in regression_
coefficients[0]}
    #arrange dictionary. it makes every key has all trial's coef.
    #Ex) 'Apr' : 1st coef, 2st coef .... 'hour' : 1st coef, 2st coef ...

    significant_coefficients = [] #make list to store results
    for i in X_train.columns:   #for loop at each predictor.
        percent = [significance_level/2,100-significance_level/2]
        #define percentage of confidence interval
        confidence = np.percentile(coeflist[i],percent)
        #find which values are in the interval
        if np.all([confidence>0]) == True: #if it is True, appen in the list.
            significant_coefficients.append(i)

    # return the significant coefficients as a list of strings
    return significant_coefficients
```

In [355]:
```
# Find coefficients by running my above code.

coef_list=[] #define empty list, it will be whole list of dictionaries that fr
om 'calculate_coeffiecent'

for i in range(500): #500 iterations

    X,Y =make_bootstrap_sample(X_train,y_train) #make bootstrap with X-train,
 y_train
    model = LinearRegression().fit(X,Y) #define model
    coef_list.append(calculate_coefficients(X,Y,model)) #store every each coef
 dictionary

print(get_significant_predictors(coef_list,5)) #5% significance level
```

```
['hour', 'year', 'workingday', 'temp', 'windspeed', 'spring', 'summer', 'fal
l', 'Sat', 'Cloudy']
```

Penalization Methods

In HW 3 Question 5 we explored using subset selection to find a significant subset of features. We then fit a regression model just on that subset of features instead of on the full dataset (including all features). As an alternative to selecting a subset of predictors and fitting a regression model on the subset, one can fit a linear regression model on all predictors, but shrink or regularize the coefficient estimates to make sure that the model does not "overfit" the training set.

Question 2

We're going to use Ridge and Lasso regression regularization techniques to fit linear models to the training set. We'll use cross-validation and shrinkage parameters λ from the set $\{.001, .005, 1, 5, 10, 50, 100, 500, 1000\}$ to pick the best model for each regularization technique.

2.1 Use 5-fold cross-validation to pick the best shrinkage parameter from the set $\{.001, .005, 1, 5, 10, 50, 100, 500, 1000\}$ for your Ridge Regression model on the training data. Fit a Ridge Regression model on the training set with the selected shrinkage parameter and store your fitted model in the variable `BikeRRModel`. Store the selected shrinkage parameter in the variable `BikeRR_shrinkage_parameter`.

2.2 Use 5-fold cross-validation to pick the best shrinkage parameter from the set $\{.001, .005, 1, 5, 10, 50, 100, 500, 1000\}$ for your Lasso Regression model on the training data. Fit a Lasso Regression model on the training set with the selected shrinkage parameter and store your fitted model in the variable `BikeLRModel`. Store the selected shrinkage parameter in the variable `BikeLR_shrinkage_parameter`.

2.3 Create three dictionaries `BikeOLSparams`, `BikeLRparams`, and `BikeRRparams`. Store in each the corresponding regression coefficients for each of the regression models indexed by the string feature name.

2.4 For the Lasso and Ridge Regression models list the features that are assigned a coefficient value close to 0 (i.e. the absolute value of the coefficient is less than 0.1). How closely do they match the redundant predictors found (if any) in HW 3, Question 5?

2.5 To get a visual sense of how the features different regression models (Multiple Linear Regression, Ridge Regression, Lasso Regression) estimate coefficients, order the features by magnitude of the estimated coefficients in the Multiple Linear Regression Model (no shrinkage). Plot a bar graph of the magnitude (absolute value) of the estimated coefficients from Multiple Linear Regression in order from greatest to least. Using a different color (and alpha values) overlay bar graphs of the magnitude of the estimated coefficients (in the same order as the Multiple Linear Regression coefficients) from Ridge and Lasso Regression.

2.6 Let's examine a pair of features we believe to be related. Is there a difference in the way Ridge and Lasso regression assign coefficients to the predictors `temp` and `atemp`? If so, explain the reason for the difference.

2.7 Discuss the Results:

1. How do the estimated coefficients compare to or differ from the coefficients estimated by a plain linear regression (without shrinkage penalty) in Question 1?
2. Is there a difference between coefficients estimated by the two shrinkage methods? If so, give an explantion for the difference.
3. Is the significance related to the shrinkage in some way?

Hint: You may use `sklearn`'s `RidgeCV` and `LassoCV` classes to implement Ridge and Lasso regression. These classes automatically perform cross-validation to tune the parameter λ from a given range of values.

Answers

In [199]:
```
lambdas = [.001, .005, 1, 5, 10, 50, 100, 500, 1000]
```

2.1 Use 5-fold cross-validation to pick the best shrinkage parameter from the set $\{.001, .005, 1, 5, 10, 50, 100, 500, 1000\}$ for your Ridge Regression model.

In [360]:
```
# your code here

# make splitter with KFold function.

# splitter = KFold(5,random_state=22, shuffle=True)

#but.. TA advice was not shuffle it.

ridgeCV_object = RidgeCV(alphas=lambdas, cv=5)
#use RidegeCV to kflod.

ridgeCV_object.fit(X_train.values, y_train)
#fit data in CVmodel


BikeRR_shrinkage_parameter = ridgeCV_object.alpha_
#store best alpha_.

#use that to fit new model.
BikeRRModel = Ridge(alpha=BikeRR_shrinkage_parameter).fit(X_train,y_train)
#define that as a BikeRRModel
BikeRR_shrinkage_parameter
```

Out[360]: 500

2.2 Use 5-fold cross-validation to pick the best shrinkage parameter from the set $\{.001, .005, 1, 5, 10, 50, 100, 500, 1000\}$ for your Lasso Regression model.

In [362]:
```
# your code here

LassoCVobject = LassoCV(alphas=lambdas, cv=5)
#use LassoCV this time.
LassoCVobject.fit(X_train,y_train)

BikeLR_shrinkage_parameter = LassoCVobject.alpha_
#same thing.

BikeLRModel = Lasso(alpha=BikeLR_shrinkage_parameter).fit(X_train,y_train)

BikeLR_shrinkage_parameter
```

Out[362]: 0.0050000000000000001

2.3 Create three dictionaries `BikeOLSparams`, `BikeLRparams`, and `BikeRRparams`. Store in each the corresponding regression coefficients.

In [422]:
```
# your code here

BikeOLSparams = calculate_coefficients(X_train,y_train,BikeOLSModel)
BikeLRparams = calculate_coefficients(X_train,y_train,BikeLRModel)
BikeRRparams = calculate_coefficients(X_train,y_train,BikeRRModel)
#store every params by using function that we made.
```

2.4 For the Lasso and Ridge Regression models list the features that are assigned a coefficient value close to 0 (i.e. the absolute value of the coefficient is less than 0.1). How closely do they match the redundant predictors found (if any) in HW 3, Question 5?

In [412]:
```
# your code here

#find which coefficient is less than 0.1
for i in X_train.columns:
    if abs(BikeRRparams[i]) < 0.1:
        print(i)

for i in X_train.columns:
    if abs(BikeLRparams[i]) < 0.1:
        print(i)
```

```
Mon
```

**Your answer here

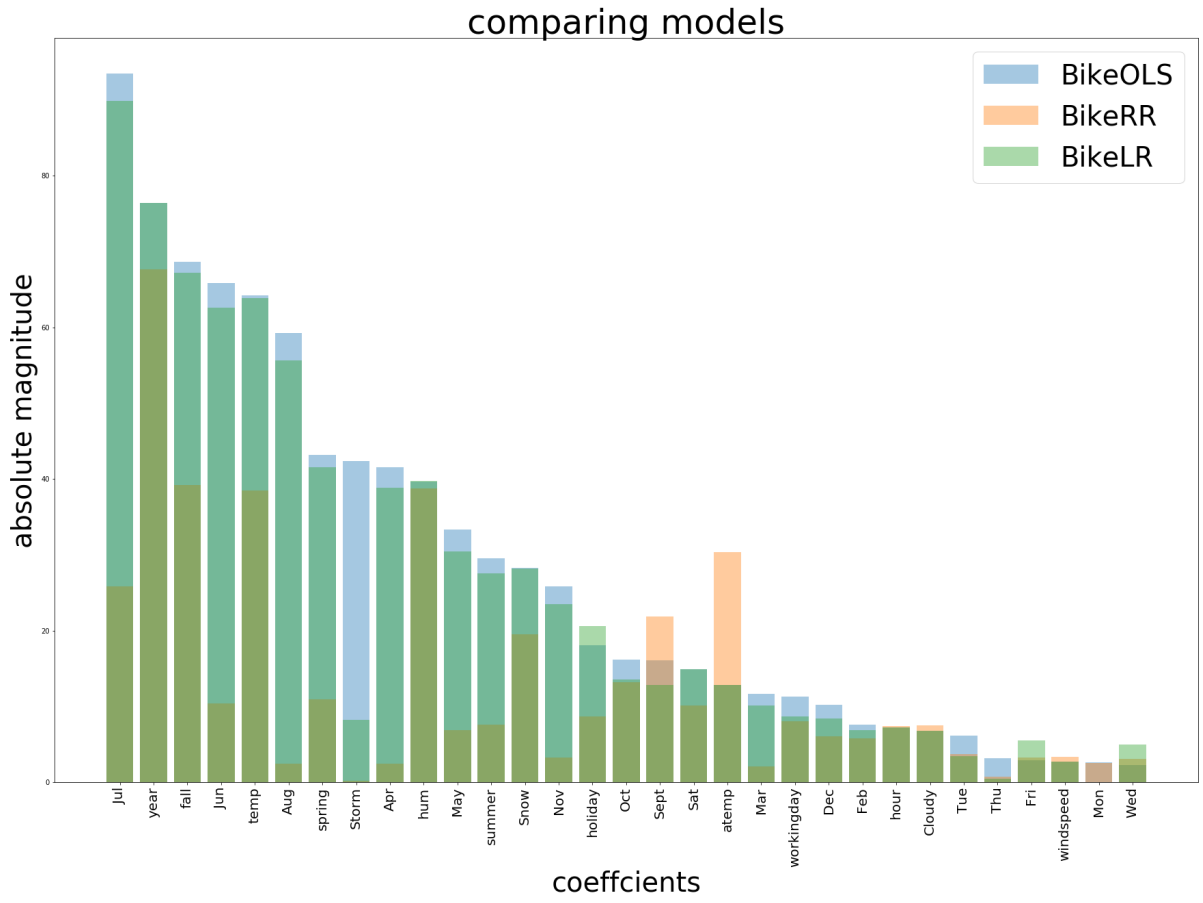
1. very weirdly, almost nothing is matching the redundant predictors in HW3, Question 5. except Lasso model sometimes. (1~3 predictors shrink sometimes) Because It is depending on kfold's random shuffle, so every time it will be a little changed.
2. First of all, why Lasso has shrunk coefficients unlike Ridge is about penalty term in the formula. Ridge has that the shape of the constraint region is circle unlike Lasso is diamond. So when Lasso's coefficients have some collinear, it would be shrunk. Therefore params < 0.1
3. And RidgeCV and LassoCV just found the best lamdas(alpha) when the cross validation error is the lowest. So the coefficients can be not shrunk (not dominated by penalty term).

2.5 To get a visual sense of how the features different regression models (Multiple Linear Regression, Ridge Regression, Lasso Regression) estimate coefficients, order the features by magnitude of the estimated coefficients in the Multiple Linear Regression Model (no shrinkage). Plot a bar graph of the magnitude (absolute value) of the estimated coefficients from Multiple Linear Regression in order from greatest to least. Using a different color (and alpha values) overlay bar graphs of the magnitude of the estimated coefficients (in the same order as the Multiple Linear Regression coefficients) from Ridge and Lasso Regression.

In [402]:
```
data = sorted(BikeOLSparams.items(),key=lambda x:x[1], reverse=True) #sorted f
rom greatest to least
order, y = zip(*data)
index = np.arange(len(order)) #make index in plot
#make order

#extract values by function
def param_values(params): #make function that making polt, 'name' = legend
    for i in X_train.columns:
        params[i] = abs(params[i])
#make data to absolute values
    data = sorted(params.items(),key=lambda x:order.index(x[0])) #sorted from
 greatest to least
    x, y = zip(*data) #seperate it to use X,Y
    return y

OLSvalue = param_values(BikeOLSparams)
LRvalue = param_values(BikeLRparams)
RRvalue = param_values(BikeRRparams)
```

In [411]:
```
# your code here
fig,ax = plt.subplots(1,1, figsize=(30,20))

ax.bar(index,OLSvalue, label="BikeOLS" ,alpha=0.4)
ax.bar(index,RRvalue, label="BikeRR" ,alpha=0.4)
ax.bar(index,LRvalue, label="BikeLR", alpha=0.4 )


ax.set_xticks(index)
ax.set_xticklabels(order,fontsize=20, rotation='vertical')
ax.set_title("comparing models",fontsize=50)
ax.set_xlabel("coeffcients" ,fontsize=40)
ax.set_ylabel("absolute magnitude" ,fontsize=40)
ax.legend(fontsize=40)
```

Out[411]: <matplotlib.legend.Legend at 0x1f982061828>



2.6 Let's examine a pair of features we believe to be related. Is there a difference in the way Ridge and Lasso regression assign coefficients to the predictors `temp` and `atemp`? If so, explain the reason for the difference.

Your answer here

1. As for the difference in teap and atemp predictors,

first of all, both Ridge and Lasso regression show that temp has bigger magnitude than atemp.

But Ridge's magnitude in atemp is higher than Lasso's.

I guess it is because in the Lasso's regression,

it tend to be more shrink when value has collinearity.

For example, any fixed value of the coefficient β1+2β2,

the penalty |β1|+|β2| is minimized when β1=0.

This is because the penalty on β1 is twice as weighted.

2.7.1. How do the estimated coefficients compare to or differ from the coefficients estimated by a plain linear regression (without shrinkage penalty) in Question 1?

Your answer here

---

1. Ridge and Lasso regression have smaller coefficients' magnitudes than OLS models.

Because Ridge and Lasso regression model has a shrinkage penalty,

which affects other coefficients.

Ridge and Lasso's magnitude are affected by penalty, and then get smaller values.

2.7.2. Is there a difference between coefficients estimated by the two shrinkage methods? If so, give an explantion for the difference.

Your answer here

1.First of all, Lasso regression sometimes(because based on random shuffle) has

shrunk value to 0 unlike Ridge. Basically, it is because of their penalty formula.

when we see region of that affection,

Ridge has that the shape of the constraint region is circle unlike Lasso is diamond.

So 0 values happen often in Lasso.

and also Lasso is more sensitive in collinearity.

Each coefficient seems to have less value when it has some collinear relationship.

2.7.3 Is the significance related to the shrinkage in some way?

Your answer here

Yes. Because technically they have very similiar formular.

So when the shrinkage goes up, their coefficients are dominated by the penalty,

every values is closer to 0. They will be not significant in high level of shrinkage, lamda.

It's underfitted with too high level of lamda.

And if that goes down, it is going to be overfitted. And every coefficients will be

significant.

Question 3: Polynomial Features, Interaction Terms, and Cross Validation

We would like to fit a model to include all main effects and polynomial terms for numerical predictors up to the 4^{th} order. More precisely use the following terms:

- predictors in `X_train` and `X_test`
- X_j^1, X_j^2, X_j^3, and X_j^4 for each numerical predictor X_j

3.1 Create an expanded training set including all the desired terms mentioned above. Store that training set (as a pandas dataframe) in the variable `X_train_poly`. Create the corresponding test set and store it as a pandas dataframe in `X_test_poly`.

3.2 Discuss the following:

1. What are the dimensions of this 'design matrix' of all the predictor variables in 3.1?
2. What issues may we run into attempting to fit a regression model using all of these predictors?

3.3 Let's try fitting a regression model on all the predictors anyway. Use the `LinearRegression` library from `sklearn` to fit a multiple linear regression model to the training set data in `X_train_poly`. Store the fitted model in the variable `BikeOLSPolyModel`.

3.4 Discuss the following:

1. What are the training and test R^2 scores?
2. How does the model performance compare with the OLS model on the original set of features in Question 1?

3.5 The training set R^2 score we generated for our model with polynomial and interaction terms doesn't have any error bars. Let's use cross-validation to generate sample sets of R^2 for our model. Use 5-fold cross-validation to generate R^2 scores for the multiple linear regression model with polynomial terms. What are the mean and standard deviation of the R^2 scores for your model.

3.6 Visualize the R^2 scores generated from the 5-fold cross validation as a box and whisker plot.

3.7 We've used cross-validation to generate error bars around our R^2 scores, but another use of cross-validation is as a way of model selection. Let's construct the following model alternatives:

1. Multiple linear regression model generated based upon the feature set in Question 1 (let's call these the base features.
2. base features plus polynomial features to order 2
3. base features plus polynomial features to order 4

Use 5-fold cross validation on the training set to select the best model. Make sure to evaluate all the models as much as possible on the same folds. For each model generate a mean and standard deviation for the R^2 score.

3.8 Visualize the R^2 scores generated for each model from 5-fold cross validation in box and whiskers plots. Do the box and whisker plots influence your view of which model was best?

3.9 Evaluate each of the model alternatives on the test set. How do the results compare with the results from cross-validation?

Answers

3.1 Create an expanded training set including all the desired terms mentioned above. Store that training set (as a numpy array) in the variable `X_train_poly`....

In [413]:
```
# # your code here
numerical = ['temp','atemp','hum','windspeed']
#make a numerical list.

def get_poly_dataset(train,test,number=int):
   #make function will use below cell.

    X_train_poly = train.drop(numerical,axis=1).copy()# make base data frame w
ithout numerical
    #I will add numerical after.
    X_test_poly = test.drop(numerical,axis=1).copy()

    feature =  PolynomialFeatures(number, include_bias=False)
    #make polynomail feature
    for i in numerical: #use for loop to put every each numerical's polynomial
 features.
        data = feature.fit_transform(train[i].values.reshape(-1,1))


    #transform data
        columns = feature.get_feature_names([i])
    #get columns name
        X_poly = pd.DataFrame(data=data, columns=columns, index = train.index)

#The newly created polynomial features would be on different scales
#so it would make sense to standardize them again.
        scaler = StandardScaler().fit(X_poly)
        X_poly[columns]=scaler.transform(X_poly)

    #make dataframe
        X_train_poly = pd.concat([X_poly,X_train_poly],axis=1)

    #merge

    #same thing in test data.
        testdata = feature.fit_transform(test[i].values.reshape(-1,1))
        testcolumns = feature.get_feature_names([i])
        test_poly = pd.DataFrame(data=testdata, columns=testcolumns)
        scaler = StandardScaler().fit(test_poly)
        test_poly[columns]=scaler.transform(test_poly)
        X_test_poly = pd.concat([test_poly,X_test_poly],axis=1)


    return X_train_poly, X_test_poly
```

In [414]:
```
X_train_poly, X_test_poly = get_poly_dataset(X_train, X_test, 4)
len(X_train_poly.columns)

X_train_poly.head()
```

Out[414]:

	windspeed	windspeed^2	windspeed^3	windspeed^4	hum	hum^2	hum^3
0	-1.557515	0.88456	-0.810236	0.113435	0.937108	-0.111969	0.349869
1	-1.557515	0.88456	-0.810236	0.113435	0.885320	-0.198713	0.301867
2	-1.557515	0.88456	-0.810236	0.113435	0.885320	-0.198713	0.301867
3	-1.557515	0.88456	-0.810236	0.113435	0.626377	-0.558479	0.135155
4	-1.557515	0.88456	-0.810236	0.113435	0.626377	-0.558479	0.135155

5 rows × 43 columns

3.2 Discuss the following:

1. What are the dimensions of this 'design matrix' of all the predictor variables in 3.1?
2. What issues may we run into attempting to fit a regression model using all of these predictors? #### 3.2.1 What are the dimensions of this 'design matrix'...**

Your answer here

1. As you can see, 43 dimensions is this design matrix.

The dimensions are same as columns' length

Previous X_train's dimension was 31.

PolynomialFeatures are not including interactionterms. (As TA mentioned)

Only X^1,X^2,X^3,X^4 so temp, atemp, hum, windspeed -> 12 more predictor.

31 + 12 = 43

3.2.2 What issues may we run into attempting to fit a regression model using all of these predictors? ...**

Your answer here

1. If we run this model with all of these predictors,

it might be litte bit heavier, takes long time to run.

Also overfitting is possible due to too many predictors.

And then definitely not that interpretable in terms of Visualization, coefficient, etc..

3.3 Let's try fitting a regression model on all the predictors anyway. Use the `LinearRegression` library from `sklearn` to fit a multiple linear regression model ....

In [415]:
```
# your code here

BikeOLSPolyModel = LinearRegression().fit(X_train_poly,y_train)
```

3.4.1 What are the training and test R^2 scores?

In [416]:
```
# your code here
print("train data's r2 scores")
print(r2_score(y_train,BikeOLSPolyModel.predict(X_train_poly)))
print("test data's r2 scores")
print(r2_score(y_test,BikeOLSPolyModel.predict(X_test_poly)))
```

```
train data's r2 scores
0.422308051666
test data's r2 scores
0.419280625032
```

Your answer here

train data's r2 scores

0.422308051666

test data's r2 scores

0.419423223832

They are not that high than I thought,

I thought it was high due to overfit with many predictors.

3.4.2 How does the model performance compare with the OLS model on the original set of features in Question 1?

Your answer here

QW1's OLS model's R2 scores:

train set R score 0.406538782797

test set R score 0.414481133552

They are of course less than polynomial models.

Because, new polynomial model predictiors = OLS model's + new poly features.

When model has new predictor, its R score will go up.

3.5 The training set R^2 score we generated for our model with polynomial and interaction terms doesn't have any error bars. Let's use cross-validation to generate sample sets of R^2 for our model. Use 5-fold cross-validation to generate R^2 scores for the multiple linear regression model with polynomial terms. What are the mean and standard deviation of the R^2 scores for your model.

In [417]:
```
# your code here
splitter = KFold(5,random_state=22, shuffle=True)
multiple_model = LinearRegression()
score = cross_val_score(multiple_model, X_train_poly,y_train, cv=splitter)
testscore = cross_val_score(multiple_model, X_test_poly,y_test, cv=splitter)
print("train_set_mean: {0}".format(np.mean(score)))
print("train_set_standard deviation : {0}".format(np.std(score)))

print("test_set_mean: {0}".format(np.mean(testscore)))
print("test_set_standard deviation : {0}".format(np.std(testscore)))
```

```
train_set_mean: 0.41880415931168347
train_set_standard deviation : 0.017342308092691168
test_set_mean: 0.4148443620163892
test_set_standard deviation : 0.016338610322228324
```

Your answer here

---

train_set_mean: 0.41880415931168347

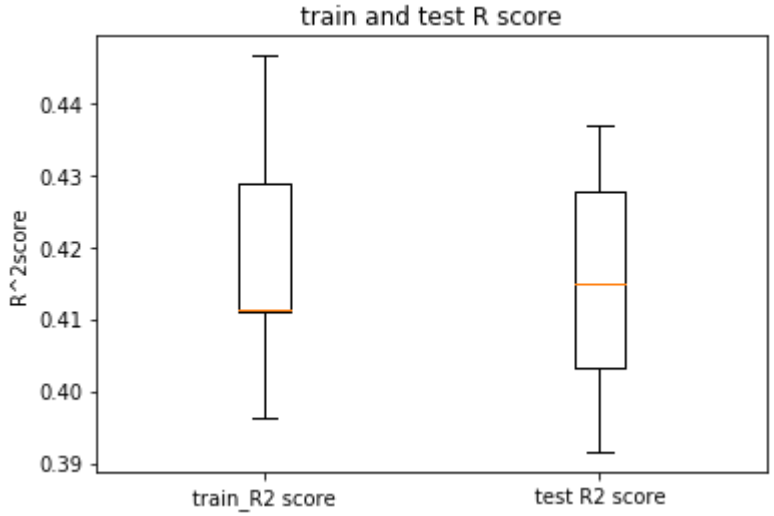
train_set_standard deviation : 0.017342308092691168

test_set_mean: 0.4148443620163892

test_set_standard deviation : 0.016338610322228324

3.6 Visualize the R^2 scores generated from the 5-fold cross validation as a box and whisker plot.

In [418]:
```
# your code here
data= [score,testscore] #make train R^2 score and test R^2scroe
plt.boxplot(data)
plt.xticks([1,2], ["train_R2 score","test R2 score"])
plt.ylabel("R^2score")
plt.title("train and test R score");
```



3.7 We've used cross-validation to generate error bars around our R^2 scores, but another use of cross-validation is as a way of model selection. Let's construct the following model alternatives:

1. Multiple linear regression model generated based upon the feature set in Question 1 (let's call these the base features.
2. base features plus polynomial features to order 2
3. base features plus polynomial features to order 4

Use 5-fold cross validation on the training set to select the best model. Make sure to evaluate all the models as much as possible on the same folds. For each model generate a mean and standard deviation for the R^2 score.

In [419]:
```
# your code here

#use 5 Kfold
five_fold= KFold(5,random_state=20,shuffle=True)
#split X_train's index -> get each index.
newindex = five_fold.split(X_train.index)
#this list will be R2score list.
Linearscore = []
poly2score = []
poly4score = []
#using for loop to fit each fold model.
for train,valid in newindex: #train -> index.
    #find Multiple linear regression model.
    #use iloc in
    Multilinear_model = LinearRegression().fit(X_train.iloc[train],y_train.ilo
c[train])
    Linearscore.append(r2_score(y_train.iloc[train],Multilinear_model.predict(
X_train.iloc[train])))

    #use function i made
    X_poly2, X_poly2_test= get_poly_dataset(X_train.iloc[train],X_test,2)
    poly2_model = LinearRegression().fit(X_poly2, y_train.iloc[train])
    poly2score.append(r2_score(y_train.iloc[train],poly2_model.predict(X_poly2
)))

    X_poly4, X_poly4_test = get_poly_dataset(X_train.iloc[train],X_test,4)
    poly4_model =LinearRegression().fit(X_poly4, y_train.iloc[train])
    poly4score.append(r2_score(y_train.iloc[train],poly4_model.predict(X_poly4
)))

print("Linear model R2 score mean : {0} \n Linear model R2 score std: {1} "
      .format(np.mean(Linearscore),np.std(Linearscore)))

print("poly2 model R2 score mean : {0} \n poly2 model R2 score std: {1} "
      .format(np.mean(poly2score),np.std(poly2score)))

print("poly4 model R2 score mean : {0} \n poly4 model R2 score std: {1} "
      .format(np.mean(poly4score),np.std(poly4score)))
```

```
Linear model R2 score mean : 0.40688755712034574
 Linear model R2 score std: 0.0025112108981688113
poly2 model R2 score mean : 0.412247334148676
 poly2 model R2 score std: 0.0022035014179986666
poly4 model R2 score mean : 0.42274737386322647
 poly4 model R2 score std: 0.002466348629530571
```

Your answer here

Linear model R2 score mean : 0.40688755712034574

Linear model R2 score std: 0.0025112108981688113

poly2 model R2 score mean : 0.412247334148676

poly2 model R2 score std: 0.0022035014179986666

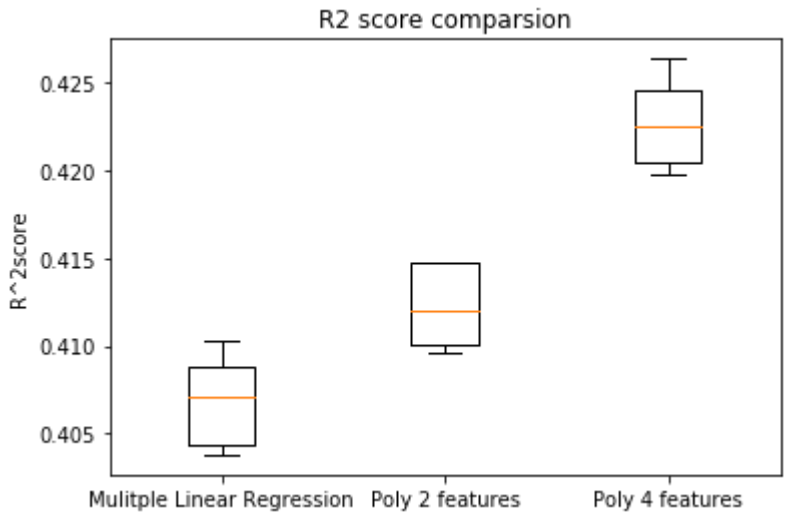
poly4 model R2 score mean : 0.42274737386322647

poly4 model R2 score std: 0.0024663486295305875

**More predictor (Linear < poly2 < poly4), higher R2 score

3.8 Visualize the R^2 scores generated for each model from 5-fold cross validation in box and whiskers plots. Do the box and whisker plots influence your view of which model was best? ...

In [420]:
```
# your code here
data= [Linearscore,poly2score,poly4score] #make train R^2 score and test R^2sc
roe
plt.boxplot(data)
plt.xticks([1,2,3], ["Mulitple Linear Regression","Poly 2 features",'Poly 4 fe
atures'])
plt.ylabel("R^2score")
plt.title("R2 score comparsion");
```



It shows that Poly 4 features are going to be best model,

But not change my view because we already saw that with comparing R2 score

3.9 Evaluate each of the model alternatives on the test set. How do the results compare with the results from cross-validation?

In [421]:
```
# your code here

X_poly2_train, X_poly2_test = get_poly_dataset(X_train,X_test,2)
X_poly4_train, X_poly4_test =get_poly_dataset(X_train,X_test,4)

Multilinear_model = LinearRegression().fit(X_train,y_train)
poly2_model = LinearRegression().fit(X_poly2_train,y_train)
poly4_model = LinearRegression().fit(X_poly4_train,y_train)

print("Multi regression on test set's R2_score : {0}".format(r2_score(y_test,M
ultilinear_model.predict(X_test))))
print("Poly2 model on test set's R2_score : {0}".format(r2_score(y_test,poly2_
model.predict(X_poly2_test))))
print("Poly4 model on test set's R2_score : {0}".format(r2_score(y_test,poly4_
model.predict(X_poly4_test))))
```

```
Multi regression on test set's R2_score : 0.40540416900870235
Poly2 model on test set's R2_score : 0.4098664406584346
Poly4 model on test set's R2_score : 0.4192806250317286
```

R2 scores are a little bit lower than train set.

But intuitively, we can see same pattern that more predictors,

higher R2 score.