S-109A Introduction to Data Science

Homework 4 - Regularization

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 [1]:

```
## RUN THIS CELL TO GET THE RIGHT FORMATTING
from IPython.core.display import HTML
def css_styling():
    styles = open("style/cs109.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]:

In [2]:

```
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 [3]:

```
BSS_train = pd.read_csv('data/BSS_train.csv', index_col=0)
BSS_train.drop(columns=['dteday', 'casual', 'registered'], inplace=True)
BSS_test = pd.read_csv('data/BSS_test.csv', index_col=0)
BSS_test.drop(columns=['dteday', 'casual', 'registered'], inplace=True)
```

This is our data:

In [4]:

```
BSS_train.sample(3)
```

Out[4]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed	counts	spring	...	D
8174	6	0	0	1	0.16	0.1818	0.80	0.1045	64	0	...	
10926	10	0	1	1	0.42	0.4242	0.35	0.2836	246	1	...	
12428	1	0	1	1	0.52	0.5000	0.77	0.1642	22	1	...	

3 rows × 32 columns

1.2 Standardizing our data

The numerical features are:

In [5]:

```
numerical_features = [
    'hour', 'holiday', 'year', 'workingday',
    'temp', 'atemp', 'hum', 'windspeed', 'counts'
]
```

Create the standard scaler:

In [6]:

```
scaler = StandardScaler()
scaler.fit(BSS_train[numerical_features])
BSS_train[numerical_features] = scaler.transform(BSS_train[numerical_features])
BSS_train.sample(3)
```

Out[6]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed
5508	-0.221095	-0.173658	-1.009177	0.684979	1.054885	1.024016	-0.771916	0.766979
3435	-1.522118	-0.173658	-1.009177	-1.459899	0.432131	0.407540	1.299629	0.032583
918	1.079928	-0.173658	-1.009177	0.684979	-1.851301	-1.795492	-1.807688	-0.700993

3 rows × 32 columns

Apply the same scaler to test dataset:

In [7]:

```
BSS_test[numerical_features] = scaler.transform(BSS_test[numerical_features])
BSS_test.sample(3)
```

Out[7]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed
11840	0.212579	-0.173658	0.990907	-1.459899	0.951093	0.936197	-0.461184	0.521907
12944	0.212579	-0.173658	0.990907	0.684979	1.885224	1.552674	-1.911265	0.277655
5171	-0.365653	-0.173658	-1.009177	0.684979	1.366262	1.288636	-0.409396	1.623501

3 rows × 32 columns

Now we organize the variables:

In [8]:

```
X_train = BSS_train.drop(columns='counts')
X_test = BSS_test.drop(columns='counts')

X_train.sample()
```

Out[8]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed	sp
4828	-1.377559	-0.173658	-1.009177	0.684979	1.26247	1.376455	0.833531	-0.823119	

1 rows × 31 columns

In [9]:

```
y_train = BSS_train.counts.to_numpy()[..., np.newaxis]
y_test = BSS_test.counts.to_numpy()[..., np.newaxis]

y_train
```

Out[9]:

```
array([[-0.9543521 ],
       [-0.82109798],
       [-0.86551602],
       ...,
       [-0.54903748],
       [-0.70450062],
       [-0.77112768]])
```

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

In [10]:

```
BikeOLSModel = LinearRegression().fit(X_train, y_train)
```

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` .

In [11]:

```
BikeOLS_r2scores = {
    'training': BikeOLSModel.score(X_train, y_train),
    'test': BikeOLSModel.score(X_test, y_test)
}
print(BikeOLS_r2scores)
```

```
{'training': 0.4065387827969087, 'test': 0.40638554757102274}
```

Training set R^2 score: 0.4065.

Test set R^2 score: 0.4064.

1.5 We're going to use bootstrapped confidence intervals to determine which of the estimated coefficients ...

In [12]:

```
# dataset_x should be a pandas dataframe

## accepts dataset inputs as numpy arrays
def make_bootstrap_sample(dataset_X, dataset_y, size = None):
    dataset_X = dataset_X.copy()
    dataset_y = dataset_y.copy()

    # by default return a bootstrap sample of the same size as the original dataset
    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")

    samples_indices = np.random.randint(0, len(dataset_X), len(dataset_X))

    bootstrap_dataset_X = dataset_X.iloc[samples_indices]
    bootstrap_dataset_y = dataset_y[samples_indices, :]

    # return as a tuple your bootstrap samples of dataset_X as a pandas dataframe
    # and your bootstrap samples of dataset y as a numpy column vector

    return (bootstrap_dataset_X, bootstrap_dataset_y)


def calculate_coefficients(dataset_X, dataset_y, model):

    features = list(dataset_X.columns)
    coef_ = list(model.coef_[0])
    coefficients_dictionary = dict(zip(features, coef_))

    # return coefficients  in the variable  coefficients_dictionary as a dictionary
    # with the key being the name of the feature as a string
    # the value being the value of the coefficients
    # do not return the intercept as part of this
    return coefficients_dictionary


def get_significant_predictors(regression_coefficients, significance_level):
    features = list(regression_coefficients[0].keys())
    significant_coefficients = []
    for feature in features:
        values = [coefficients[feature] for coefficients in regression_coefficients
        if np.quantile(values, significance_level/2) > 0:
            significant_coefficients.append(feature)
        else:
            if np.quantile(values, 1 - significance_level/2) < 0:
                significant_coefficients.append(feature)

    # regression_coefficients is a list of dictionaries
    # with the key being the name of the feature as a string
    # the value being the value of the coefficients
    # each dictionary in the list should be the output of calculate_coefficients

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

In [13]:

```
regression_coefficients = []
for i in range(500):
    bootstrap_sample = make_bootstrap_sample(X_train, y_train)
    model = LinearRegression().fit(bootstrap_sample[0], bootstrap_sample[1])
    coefficients = calculate_coefficients(bootstrap_sample[0], bootstrap_sample[1],
    regression_coefficients.append(coefficients)
BikeOLS_significant_bootstrap = get_significant_predictors(regression_coefficients,
```

In [14]:

```
print(BikeOLS_significant_bootstrap)
```

```
['hour', 'holiday', 'year', 'workingday', 'temp', 'hum', 'windspeed',
'spring', 'summer', 'fall', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Oct', 'Nov', 'Dec', 'Sat', 'Cloudy', 'Snow']
```

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 [15]:

```
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 [16]:

```
BikeRRModel = RidgeCV(alphas=lambdas, cv=5).fit(X_train, y_train)
BikeRR_shrinkage_parameter = BikeRRModel.alpha_
```

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 [17]:

```
BikeLRModel = LassoCV(alphas=lambdas, cv=5).fit(X_train, y_train)
BikeLR_shrinkage_parameter = BikeLRModel.alpha_
```

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

In [18]:

```
BikeOLSparams = dict(zip(X_train.columns, BikeOLSModel.coef_[0]))
BikeLRparams = dict(zip(X_train.columns, BikeLRModel.coef_))
BikeRRparams = dict(zip(X_train.columns, BikeRRModel.coef_[0]))
```

2.4 For the Lasso and Ridge Regression models list the features that are assigned a coefficient value close to 0 ...

We calculate the features with little coefficients:

In [19]:

```
lasso_0 = [key for key, values in BikeLRparams.items() if np.abs(values) < 0.1]
ridge_0 = [key for key, values in BikeRRparams.items() if np.abs(values) < 0.1]
```

In [20]:

```
print(lasso_0)
```

```
['holiday', 'workingday', 'atemp', 'windspeed', 'summer', 'Feb', 'Ma
r', 'Apr', 'May', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon', 'Tue', 'W
ed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Storm']
```

In [21]:

```
print(ridge_0)
```

```
['holiday', 'workingday', 'windspeed', 'spring', 'summer', 'Feb', 'Ma
r', 'Apr', 'May', 'Jun', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon', 'T
ue', 'Wed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm']
```

We verify the features removed in Homework 3, question 5:

In [22]:

```
hw_3 = ['temp',
  'hour',
  'year',
  'hum',
  'fall',
  'Jul',
  'Snow',
  'Aug',
  'Jun',
  'holiday',
  'spring']
print([feature for feature in X_train.columns if feature not in hw_3])
```

```
['workingday', 'atemp', 'windspeed', 'summer', 'Feb', 'Mar', 'Apr', 'M
ay', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri',
'Sat', 'Cloudy', 'Storm']
```

The lists are almost the same. It means that the features with little coefficients calculated with Lasso and Ridge are almost the same that those removed in Question 5 from Homework 3.

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).

In [23]:

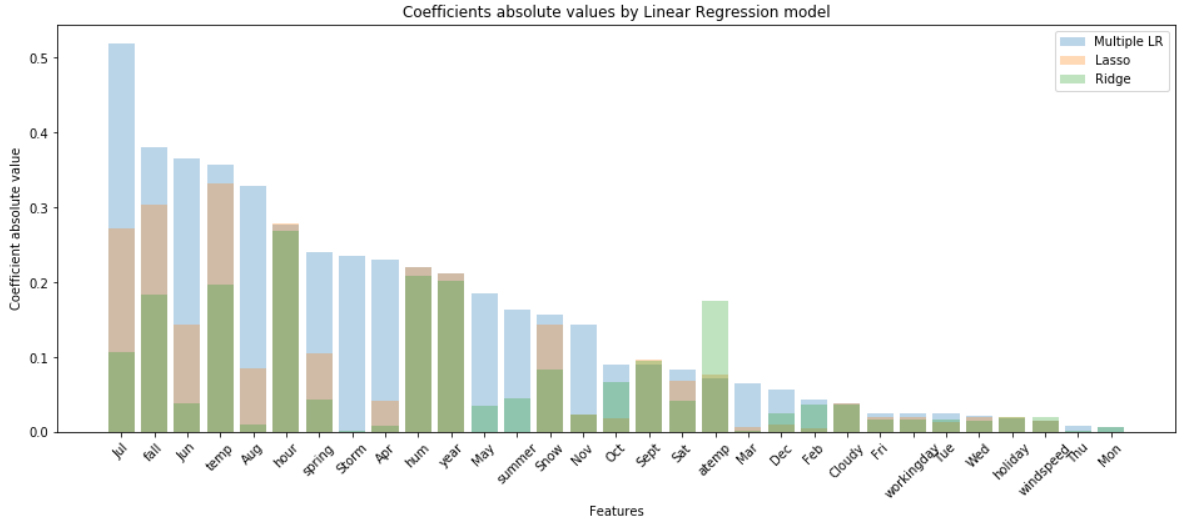
```
features = list(X_train.columns)
```

In [24]:

```
sorted_features = sorted(features, key=lambda x: np.abs(BikeOLSparams[x]), reverse=
```

In [25]:

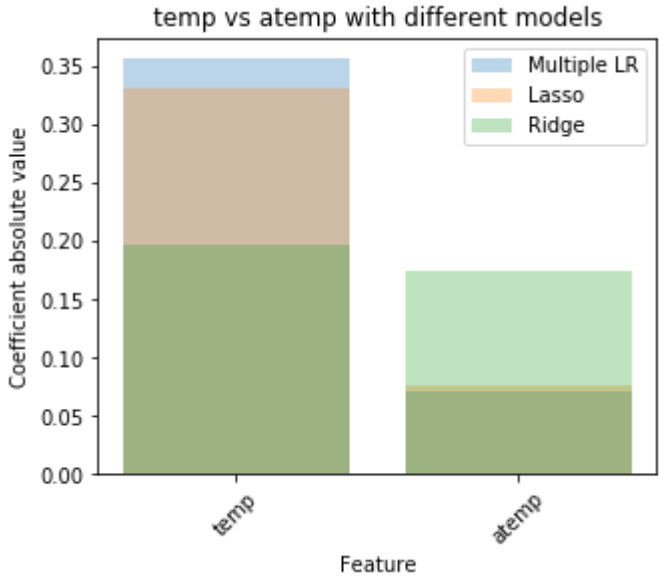
```
fig, ax = plt.subplots(figsize=(16, 6))
plt.bar(sorted_features, [np.abs(BikeOLSparams[feature]) for feature in sorted_feat
plt.bar(sorted_features, [np.abs(BikeLRparams[feature]) for feature in sorted_featu
plt.bar(sorted_features, [np.abs(BikeRRparams[feature]) for feature in sorted_featu
plt.xticks(rotation=45)
plt.legend()
plt.title('Coefficients absolute values by Linear Regression model')
plt.xlabel('Features')
plt.ylabel('Coefficient absolute value')
plt.show()
```



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 ...v

In [26]:

```
fig, ax = plt.subplots(figsize=(5, 4))
plt.bar(['temp', 'atemp'], [np.abs(BikeOLSparams[feature]) for feature in ['temp',
plt.bar(['temp', 'atemp'], [np.abs(BikeLRparams[feature]) for feature in ['temp', '
plt.bar(['temp', 'atemp'], [np.abs(BikeRRparams[feature]) for feature in ['temp', '
plt.xticks(rotation=45)
plt.legend()
plt.title('temp vs atemp with different models')
plt.xlabel('Feature')
plt.ylabel('Coefficient absolute value')
plt.show()
```



Although Lasso performs the same as Multiple Linear Regression for the features `temp` and `atemp`, Ridge decreases `temp`'s coefficient and increases `atemp`'s. In fact, for the Ridge, they are very close to each other. Well, this difference between Lasso and Ridge is because Lasso tends to produce sparse solutions, that is, tends to produce zero estimates, differently from Ridge. Lasso coefficients are nullified fast, when lambda increases.

2.7.1 How do the estimated coefficients compare to or differ from ...

The estimated coefficients for Lasso and Ridge do not follow the same order of magnitude of plain linear regression. Also, many of them are zero, when dealing with Lasso. In general, these coefficients are smaller than plain linear regression's.

2.7.2 Is there a difference between coefficients estimated by the two shrinkage methods ...

They are very different too: not same order of magnitude, many of Lasso coefficients are zero, in general Lasso coefficients are greater.

The reason for this is because they have different loss functions. Lasso tend to produce zero estimates, sparse solutions.

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

It appears not to have relation between significance (original bar height) and shrinkage. That is, the proportion of increase/decrease of the coefficient magnitude appears not to be related to its original magnitude.

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 [27]:

```
predictors = list(X_train.columns)
to_be_transformed = [predictor for predictor in predictors if predictor in numerica
transform = PolynomialFeatures(4, include_bias=False)
X_train_poly = X_train.copy()
X_test_poly = X_test.copy()

for predictor in to_be_transformed:
    X_train_poly[[
        predictor,
        predictor+'^2',
        predictor+'^3',
        predictor+'^4'
    ]] = pd.DataFrame(
        transform.fit_transform(X_train[[predictor]]),
        index=X_train_poly.index
    )
    X_test_poly[[
        predictor,
        predictor+'^2',
        predictor+'^3',
        predictor+'^4'
    ]] = pd.DataFrame(
        transform.fit_transform(X_test[[predictor]]),
        index=X_test_poly.index
    )
```

3.2.1 What are the dimensions of this 'design matrix'...**

In [28]:

```
print('The dimensions of X_train_poly is {}.'.format(X_train_poly.shape))
```

```
The dimensions of X_train_poly is (13903, 55).
```

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

We may overfit the training dataset, because of the high order.

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 [29]:

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

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

In [30]:

```
print(r2_score(y_train, BikeOLSPolyModel.predict(X_train_poly)),
r2_score(y_test, BikeOLSPolyModel.predict(X_test_poly)))
```

```
0.5553131294038416 0.5518725502376847
```

The training R^2 score is 0.5553, and the test is 0.5519.

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

The training performance of this new model is obviously better. But we see that the test performance is better too. So we conclude it performs better than the OLS model with 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...

In [31]:

```
# Note that's important to set shuffle=True
# in order to shuffle the data before splitting
# But it's also important to set random_state=something
# in order to allow reproducibility
r2 = cross_val_score(
    LinearRegression(),
    X_train_poly,
    y_train,
    cv=KFold(n_splits=5, shuffle=True, random_state=0),
    scoring='r2'
)
```

In [32]:

```
print(np.mean(r2), np.std(r2))
```

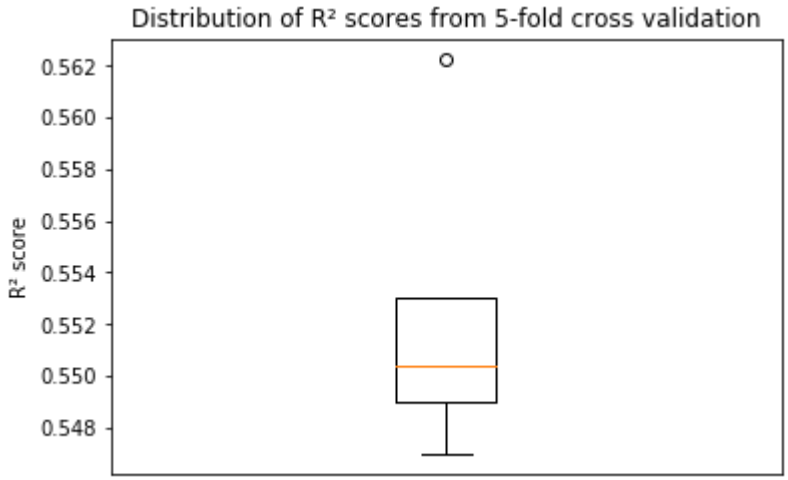
```
0.5523235805837091 0.005348270790672392
```

The mean of R^2 score is 0.5523, and the standard deviation is 0.0053.

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

In [33]:

```
plt.boxplot(r2)
plt.title('Distribution of R² scores from 5-fold cross validation')
plt.ylabel('R² score')
plt.xticks([])
plt.show()
```



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 ...

In [34]:

```
r2_1 = cross_val_score(
    LinearRegression(),
    X_train[BikeOLS_significant_bootstrap],
    y_train,
    cv=KFold(n_splits=5, shuffle=True, random_state=0),
    scoring='r2'
)
print(np.mean(r2_1), np.std(r2_1))
```

```
0.4040540961168496 0.012521467036258315
```

In [35]:

```
r2_2 = cross_val_score(
    LinearRegression(),
    X_train_poly.drop(
        columns=[column for column in X_train_poly.columns if '^3' in column or '^4
    ),
    y_train,
    cv=KFold(n_splits=5, shuffle=True, random_state=0),
    scoring='r2'
)
print(np.mean(r2_2), np.std(r2_2))
```

```
0.499570427331628 0.0065362630118968
```

In [36]:

```
r2_3 = cross_val_score(
    LinearRegression(),
    X_train_poly.drop(
        columns=[column for column in X_train_poly.columns if column.split('^')[0]
    ),
    y_train,
    cv=KFold(n_splits=5, shuffle=True, random_state=0),
    scoring='r2'
)
print(np.mean(r2_3), np.std(r2_3))
```

```
0.5501765095089214 0.0064477801687256335
```

Model 1: mean $= 0.4041$, std $= 0.0125$

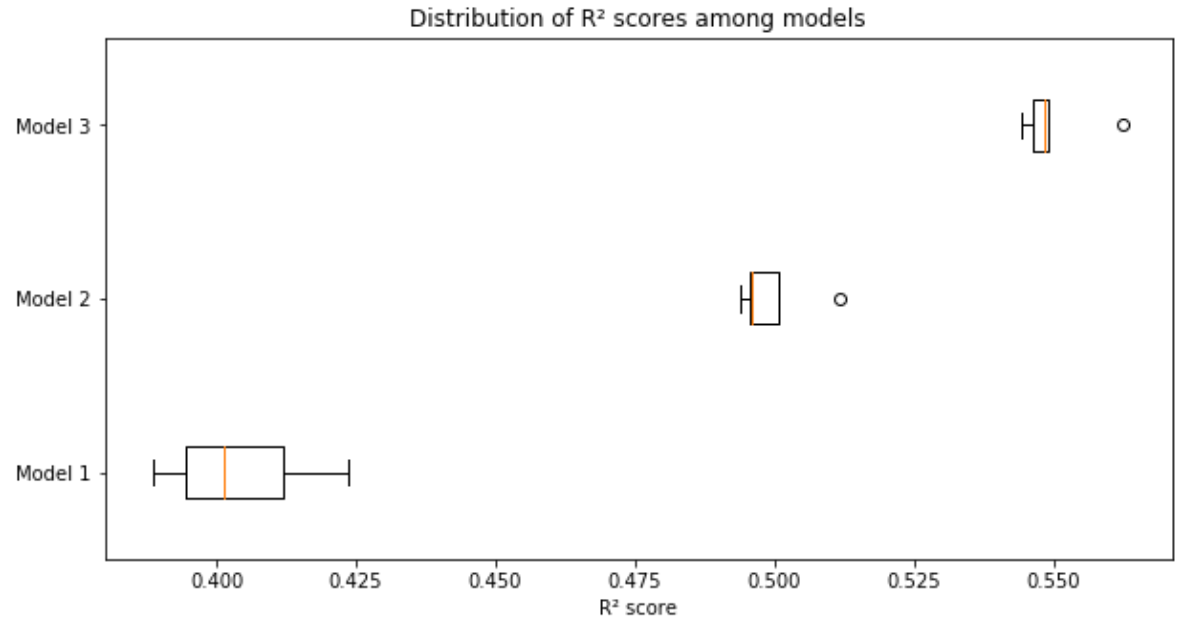
Model 2: mean $= 0.4996$, std $= 0.0065$

Model 3: mean $= 0.5502$, std $= 0.0064$

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 [37]:

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.boxplot([r2_1, r2_2, r2_3], vert=False)
ax.set_title('Distribution of R² scores among models')
ax.set_xlabel('R² score')
# ax.set_ylabel('Model')
ax.set_yticklabels(['Model 1', 'Model 2', 'Model 3'])
plt.show()
```



We see that the best models were those with polynomial features, specially the one with 4th order polynomial features.

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

In [38]:

```
model_1 = LinearRegression().fit(X_train[BikeOLS_significant_bootstrap], y_train)
r2_score(y_test, model_1.predict(X_test[BikeOLS_significant_bootstrap]))
```

Out[38]:

```
0.40617652723763764
```

In [39]:

```
model_2 = LinearRegression().fit(
    X_train_poly.drop(
        columns=[column for column in X_train_poly.columns if '^3' in column or '^4
    ),
    y_train
)
r2_score(
    y_test,
    model_2.predict(
        X_test_poly.drop(
            columns=[column for column in X_test_poly.columns if '^3' in column or
        )
    )
)
```

Out[39]:

```
0.49665337140763344
```

In [40]:

```
model_3 = LinearRegression().fit(
    X_train_poly.drop(
        columns=[column for column in X_train_poly.columns if column.split('^')[0]
    ),
    y_train
)
r2_score(
    y_test,
    model_3.predict(
        X_test_poly.drop(
            columns=[column for column in X_test_poly.columns if column.split('^')[
        )
    )
)
```

Out[40]:

```
0.548734718886861
```

The results are very close to the medians of the results from cross-validation.
