# index

March 3, 2022

# 1 Extensions to Linear Models - Lab

## 1.1 Introduction

In this lab, you'll practice many concepts you have learned so far, from adding interactions and polynomials to your model to AIC and BIC!

## 1.2 Summary

You will be able to: - Build a linear regression model with interactions and polynomial features - Use AIC and BIC to select the best value for the regularization parameter

## 1.3 Let's get started!

Import all the necessary packages.

[392]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from itertools import combinations

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import scale
from sklearn.preprocessing import PolynomialFeatures
```

Load the data.

[393]:
```
df = pd.read_csv("ames.csv")
df.columns
```

[393]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
```

```
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

[394]:

```
df = df[['LotArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF',
         '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'TotRmsAbvGrd',
         'GarageArea', 'Fireplaces', 'SalePrice']]
```

### 1.4 Look at a baseline housing data model

Above, we imported the Ames housing data and grabbed a subset of the data to use in this analysis.

Next steps:

- Split the data into target (y) and predictors (X) – ensure these both are DataFrames
- Scale all the predictors using `scale`. Convert these scaled features into a DataFrame
- Build at a baseline model using *scaled variables* as predictors. Use 5-fold cross-validation (set `random_state` to 1) and use the $R^2$ score to evaluate the model

[395]:

```
# Your code here
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

base_model = LinearRegression()

y = df[['SalePrice']]
X = df.drop(columns = ['SalePrice'])
X_scaled = pd.DataFrame(scale(X),columns = X.columns)

## For MySelf
print(cross_val_score(base_model, X_scaled, y, cv=5))
```

```
[0.81590682 0.78658346 0.77744573 0.77448228 0.63013673]
```

[396]:

```
from sklearn.model_selection import KFold

crossval = KFold(n_splits=5, random_state=1, shuffle=True)
```

```
base_model_scores = cross_val_score(base_model , X_scaled, y, scoring="r2",
                        cv = crossval)
base_model_r2 = np.mean(base_model_scores)
base_model_r2
```

[396]: 0.7524751004088885

[397]:
```
### From GitHub

# y = df[['SalePrice']]
# X = df.drop(columns='SalePrice')

# X_scaled = scale(X)
# X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# all_data = pd.concat([y, X_scaled], axis=1)
# regression = LinearRegression()

# crossvalidation = KFold(n_splits=5, shuffle=True, random_state=1)
# baseline = np.mean(cross_val_score(regression, X_scaled, y, scoring='r2',
#             cv=crossvalidation))
# baseline
```

## 1.5 Include interactions

Look at all the possible combinations of variables for interactions by adding interactions one by one to the baseline model. Next, evaluate that model using 5-fold cross-validation and store the $R^2$ to compare it with the baseline model.

Print the 7 most important interactions.

[398]:
```
# Your code here
from itertools import combinations

combo = list(combinations(X_scaled.columns,2))
X_inter = X_scaled.copy()

combo_r2 = pd.DataFrame()
for i,item in enumerate(combo):

    X_inter["interactions"] = X_scaled[item[0]]*X_scaled[item[1]]
    inter_model_r2 = np.mean(cross_val_score(base_model, X_inter, y,
                                            scoring="r2", cv = crossval))

#     combo_r2.loc[i, "combo"] = item[0] + "*" + item[1]
    combo_r2.loc[i, "r2"] = np.round(inter_model_r2,3)
    if inter_model_r2 > base_model_r2:
        combo_r2.loc[i, "r2"] = np.round(inter_model_r2,3)
```

```
        combo_r2.loc[i, "col_1"] = item[0]
        combo_r2.loc[i, "col_2"] = item[1]
```

Write code to include the 7 most important interactions in your data set by adding 7 columns. Name the columns "var1_var2", where var1 and var2 are the two variables in the interaction.

[399]:
```
# Your code here
best_inter_score = combo_r2.loc[combo_r2["r2"]>base_model_r2].sort_values(
                                            by = "r2", ascending = False
                                                                     )


best_inter = best_inter_score.iloc[0:7]
best_inter.reset_index(inplace = True, drop = True)
best_inter
```

[399]:

| | r2 | col_1 | col_2 |
|---|---|---|---|
| 0 | 0.770 | OverallQual | TotRmsAbvGrd |
| 1 | 0.764 | OverallQual | GarageArea |
| 2 | 0.758 | OverallQual | 2ndFlrSF |
| 3 | 0.756 | 2ndFlrSF | GrLivArea |
| 4 | 0.756 | 2ndFlrSF | TotRmsAbvGrd |
| 5 | 0.754 | OverallQual | Fireplaces |
| 6 | 0.754 | OverallCond | TotalBsmtSF |

[400]:
```
df_inter = X_scaled.copy()
for i in range(len(best_inter)):
    col1 = best_inter.loc[int(i), "col_1"]
    col2 = best_inter.loc[int(i), "col_2"]
    df_inter[col1+"_"+col2] = df_inter[col1]*df_inter[col2]
df_inter.columns
```

[400]:
```
Index(['LotArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF', '1stFlrSF',
       '2ndFlrSF', 'GrLivArea', 'TotRmsAbvGrd', 'GarageArea', 'Fireplaces',
       'OverallQual_TotRmsAbvGrd', 'OverallQual_GarageArea',
       'OverallQual_2ndFlrSF', '2ndFlrSF_GrLivArea', '2ndFlrSF_TotRmsAbvGrd',
       'OverallQual_Fireplaces', 'OverallCond_TotalBsmtSF'],
      dtype='object')
```

[401]:
```
### From GitHub
from itertools import combinations
X = df.drop(columns = ['SalePrice'])
combo = list(combinations(X.columns, 2))
baseline = base_model_r2
interactions = []
```

```
data = X_scaled.copy()
for comb in combo:
    data['interaction'] = data[comb[0]] * data[comb[1]]
    score = np.mean(cross_val_score(base_model, data, y, scoring='r2',␣
 ↪cv=crossval))
    if score > baseline: interactions.append((comb[0], comb[1], round(score,␣
 ↪3)))

print('Top 7 interactions: %s' %sorted(interactions,
                                       key=lambda inter: inter[2],
                                       reverse=True)[:7])
```

```
Top 7 interactions: [('OverallQual', 'TotRmsAbvGrd', 0.77), ('OverallQual',
'GarageArea', 0.764), ('OverallQual', '2ndFlrSF', 0.758), ('2ndFlrSF',
'GrLivArea', 0.756), ('2ndFlrSF', 'TotRmsAbvGrd', 0.756), ('OverallQual',
'Fireplaces', 0.754), ('OverallCond', 'TotalBsmtSF', 0.754)]
```

### 1.6 Include polynomials

Try polynomials of degrees 2, 3, and 4 for each variable, in a similar way you did for interactions (by looking at your baseline model and seeing how $R^2$ increases). Do understand that when going for a polynomial of 4, the particular column is raised to the power of 2 and 3 as well in other terms. We only want to include "pure" polynomials, so make sure no interactions are included. We want the result to return a list that contain tuples of the form:

`(var_name, degree, R2)`, so eg. `('OverallQual', 2, 0.781)`

[402]:
```
# Your code here
from sklearn.preprocessing import PolynomialFeatures



degree = [2, 3, 4]
poly_r2 = pd.DataFrame()
i = 0
for col in X_scaled.columns:

    for item in degree:

        X_poly = X_scaled.copy()

        poly = PolynomialFeatures(degree=item,
                                  interaction_only=False,
                                  include_bias=False)
        X_col = poly.fit_transform(X_scaled[[col]])

        X_poly.drop(columns = col, axis = 1, inplace = True)
        X_poly_f = pd.concat([X_poly, pd.DataFrame(X_col)], axis = 1)
```

```
        ploy_model_r2 = np.mean(cross_val_score(base_model, X_poly_f, y,
                                                scoring="r2", cv = crossval))

        poly_r2.loc[i, "col"] = col
        poly_r2.loc[i, "degree"] = item
        poly_r2.loc[i, "r2"] = np.round(ploy_model_r2,3)

        i += 1
```

[403]:

```
# poly_r2
```

[404]:

```
better_score = poly_r2.loc[poly_r2["r2"]>base_model_r2].sort_values(
                                            by = "r2", ascending = False
                                                                    )

top_7_poly = better_score.iloc[0:7]
```

For each variable, print out the maximum $R^2$ possible when including Polynomials.

[405]:

```
# Your code here
best_poly = better_score.groupby(["col"])["degree","r2"].max().sort_values(
    by = "r2",
    ascending = False)

best_poly.reset_index(inplace = True)
```

[406]:

```
best_poly
```

[406]:

| | col | degree | r2 |
|---|---|---|---|
| 0 | GrLivArea | 4.0 | 0.807 |
| 1 | OverallQual | 4.0 | 0.781 |
| 2 | 2ndFlrSF | 4.0 | 0.775 |
| 3 | GarageArea | 4.0 | 0.767 |
| 4 | OverallCond | 4.0 | 0.753 |
| 5 | TotRmsAbvGrd | 3.0 | 0.753 |

[407]:

```
### From GitHub

polynomials = []
for col in X.columns:
    for degree in [2, 3, 4]:
        data = X_scaled.copy()
        poly = PolynomialFeatures(degree, include_bias=False)
        X_transformed = poly.fit_transform(X[[col]])
```

```
        data = pd.concat([data.drop(col, axis=1),pd.DataFrame(X_transformed)],␣
↪axis=1)
        score = np.mean(cross_val_score(base_model, data, y,
                                        scoring='r2', cv=crossval))
        if score > baseline: polynomials.append((col, degree, round(score, 3)))
print('Top 10 polynomials: %s' %sorted(polynomials, key=lambda poly: poly[2],
                                      reverse=True)[:10])
```

```
Top 10 polynomials: [('GrLivArea', 4, 0.807), ('GrLivArea', 3, 0.788),
('OverallQual', 2, 0.781), ('OverallQual', 3, 0.779), ('OverallQual', 4, 0.779),
('2ndFlrSF', 3, 0.775), ('2ndFlrSF', 2, 0.771), ('2ndFlrSF', 4, 0.771),
('GarageArea', 4, 0.767), ('GarageArea', 3, 0.758)]
```

[408]:
```
### From GitHub

polynom = pd.DataFrame(polynomials)
polynom.groupby([0], sort=False)[2].max()
```

[408]:
```
0
OverallQual     0.781
OverallCond     0.753
2ndFlrSF        0.775
GrLivArea       0.807
TotRmsAbvGrd    0.753
GarageArea      0.767
Name: 2, dtype: float64
```

Which two variables seem to benefit most from adding polynomial terms?

Add Polynomials for the two features that seem to benefit the most, as in have the best $R^2$ compared to the baseline model. For each of the two features, raise to the Polynomial that generates the best result. Make sure to start from the data set `df_inter` so the final data set has both interactions and polynomials in the model.

[409]:
```
# # Your code here
# from sklearn.preprocessing import PolynomialFeatures



# features_to_pick = list(best_poly.iloc[0:2,0])
# degree_to_pick  = list(best_poly.iloc[0:2,1])
# feature_degree = list(zip(features_to_pick, degree_to_pick))

# # X_features = X_scaled.copy()
# # X_poly_final = X_scaled.drop(columns = features_to_pick, axis = 1)

# for (col, deg) in feature_degree:

#     poly = PolynomialFeatures(degree = int(deg),
```

```
#                           interaction_only=False,
#                           include_bias=False)

#     X_features = poly.fit_transform(X[[col]])
#     col_names = [col + f"**{i}" for i in range(int(deg))]
#     X_poly = pd.DataFrame(X_features, columns = col_names)

#     df_inter = pd.concat([df_inter.drop(col,axis = 1), pd.
↪DataFrame(X_features, columns = col_names)], axis = 1)

## From GitHub

for col in ['OverallQual', 'GrLivArea']:
    poly = PolynomialFeatures(4, include_bias=False)
    X_transformed = poly.fit_transform(X[[col]])
    colnames= [col, col + '_' + '2',  col + '_' + '3', col + '_' + '4']
    df_inter = pd.concat([df_inter.drop(col, axis=1), pd.
↪DataFrame(X_transformed, columns=colnames)], axis=1)
```

Check out your final data set and make sure that your interaction terms as well as your polynomial terms are included.

[410]:
```
# Your code here
df_inter.head()
```

[410]:
```
    LotArea  OverallCond  TotalBsmtSF  1stFlrSF  2ndFlrSF  TotRmsAbvGrd  \
0 -0.207142    -0.517200    -0.459303 -0.793434  1.161852      0.912210
1 -0.091886     2.179628     0.466465  0.257140 -0.795163     -0.318683
2  0.073480    -0.517200    -0.313369 -0.627826  1.189351     -0.318683
3 -0.096897    -0.517200    -0.687324 -0.521734  0.937276      0.296763
4  0.375148    -0.517200     0.199680 -0.045611  1.617877      1.527656

   GarageArea  Fireplaces  OverallQual_TotRmsAbvGrd  OverallQual_GarageArea  \
0    0.351000   -0.951226                  0.594286                0.228669
1   -0.060731    0.600495                  0.022893                0.004363
2    0.631726    0.600495                 -0.207616                0.411557
3    0.790804    0.600495                  0.193335                0.515193
4    1.698485    0.600495                  2.100214                2.335068

   …  OverallQual_Fireplaces  OverallCond_TotalBsmtSF  OverallQual  \
0  …               -0.619704                 0.237551          7.0
1  …               -0.043137                 1.016720          6.0
2  …                0.391210                 0.162074          7.0
3  …                0.391210                 0.355484          7.0
4  …                0.825557                -0.103274          8.0

   OverallQual_2  OverallQual_3  OverallQual_4  GrLivArea  GrLivArea_2  \
```

```
0          49.0          343.0          2401.0      1710.0     2924100.0
1          36.0          216.0          1296.0      1262.0     1592644.0
2          49.0          343.0          2401.0      1786.0     3189796.0
3          49.0          343.0          2401.0      1717.0     2948089.0
4          64.0          512.0          4096.0      2198.0     4831204.0

    GrLivArea_3   GrLivArea_4
0  5.000211e+09  8.550361e+12
1  2.009917e+09  2.536515e+12
2  5.696976e+09  1.017480e+13
3  5.061869e+09  8.691229e+12
4  1.061899e+10  2.334053e+13

[5 rows x 23 columns]
```

### 1.7 Full model R-squared

Check out the $R^2$ of the full model.

[411]:
```python
# Your code here
ploy_final_model_r2 = np.mean(cross_val_score(base_model,
                                              df_inter,
                                              y,
                                              scoring="r2",
                                              cv = crossval))
ploy_final_model_r2
```

[411]:
```
0.8245194818705173
```

### 1.8 Find the best Lasso regularization parameter

You learned that when using Lasso regularization, your coefficients shrink to 0 when using a higher regularization parameter. Now the question is which value we should choose for the regularization parameter.

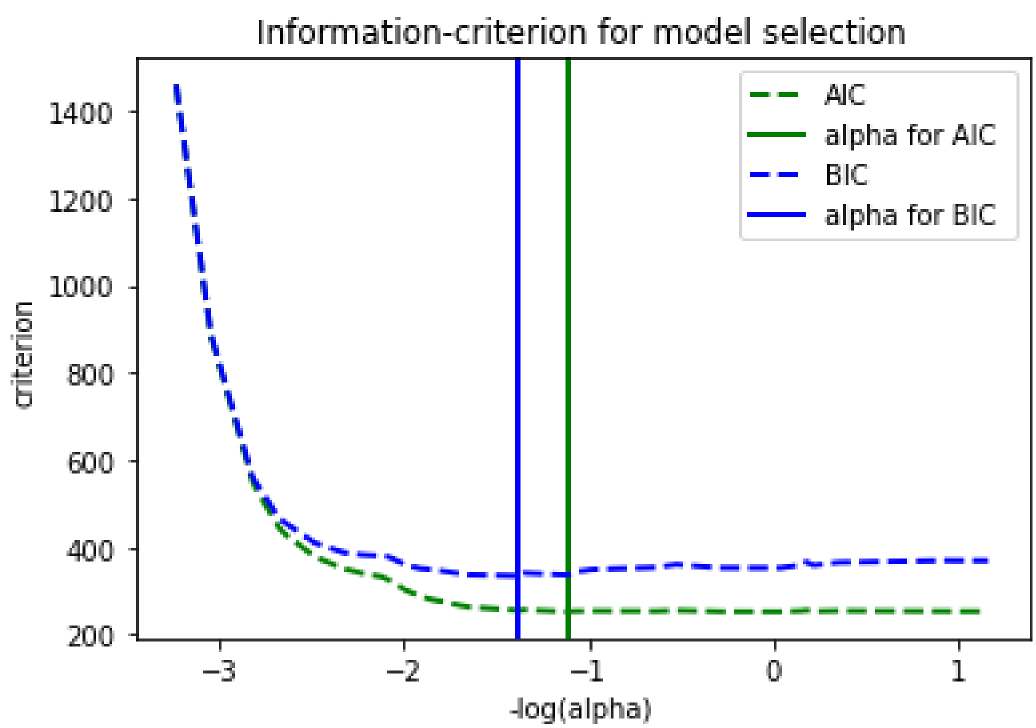
This is where the AIC and BIC come in handy! We'll use both criteria in what follows and perform cross-validation to select an optimal value of the regularization parameter *alpha* of the Lasso estimator.

Read the page here: https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.ht and create a similar plot as the first one listed on the page.

[412]:
```python
from sklearn.linear_model import Lasso, LassoCV, LassoLarsCV, LassoLarsIC
```

[413]:
```python
# Your code here

### From GitHub

model_bic = LassoLarsIC(criterion='bic')
```

```
model_bic.fit(df_inter, y)
alpha_bic_ = model_bic.alpha_

model_aic = LassoLarsIC(criterion='aic')
model_aic.fit(df_inter, y)
alpha_aic_ = model_aic.alpha_


def plot_ic_criterion(model, name, color):
    alpha_ = model.alpha_
    alphas_ = model.alphas_
    criterion_ = model.criterion_
    plt.plot(-np.log10(alphas_), criterion_, '--', color=color, linewidth=2,␣
 ↪label= name)
    plt.axvline(-np.log10(alpha_), color=color, linewidth=2,
                label='alpha for %s ' % name)
    plt.xlabel('-log(alpha)')
    plt.ylabel('criterion')

plt.figure()
plot_ic_criterion(model_aic, 'AIC', 'green')
plot_ic_criterion(model_bic, 'BIC', 'blue')
plt.legend()
plt.title('Information-criterion for model selection');
```



### 1.9 Analyze the final result

Finally, use the best value for the regularization parameter according to AIC and BIC, and compare $R^2$ and RMSE using train-test split. Compare with the baseline model.

Remember, you can find the Root Mean Squared Error (RMSE) by setting `squared=False` inside the function (see the documentation), and the RMSE returns values that are in the same units as our target - so we can see how far off our predicted sale prices are in dollars.

[414]:
```
from sklearn.metrics import mean_squared_error, mean_squared_log_error, r2_score
from sklearn.model_selection import train_test_split
```

[415]:
```
# Split X_scaled and y into training and test sets
# Set random_state to 1



X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, random_state=1)



# Code for baseline model
linreg_all = LinearRegression()
linreg_all.fit(X_train, y_train)
y_train_predict = linreg_all.predict(X_train)

# Print R-Squared and RMSE
# print('Training R-Squared:', linreg_all.score(X_train, y_train))
print('Training R-Squared:', linreg_all.score(X_train, y_train))
print('Test R-Squared:', linreg_all.score(X_test, y_test))
print('Training RMSE:', mean_squared_error(y_train,
                                           linreg_all.predict(X_train),
                                           squared=False))
print('Test RMSE:', mean_squared_error(y_test,
                                       linreg_all.predict(X_test),
                                       squared=False))
```

```
Training R-Squared: 0.7478270652928448
Test R-Squared: 0.8120708166668685
Training RMSE: 39424.15590381302
Test RMSE: 35519.17035590487
```

[419]:
```
# Split df_inter and y into training and test sets
# Set random_state to 1
X_train, X_test, y_train, y_test = train_test_split(df_inter, y, random_state=1)

# Code for lasso with alpha from AIC
lasso = Lasso(alpha = model_aic.alpha_)
```

```
lasso.fit(X_train, y_train)

# Print R-Squared and RMSE
print('Training R-Squared:', lasso.score(X_train, y_train))
print('Test R-Squared:', lasso.score(X_test, y_test))
print('Training RMSE:', mean_squared_error(y_train,
                                           lasso.predict(X_train),
                                           squared=False))
print('Test RMSE:', mean_squared_error(y_test,
                                       lasso.predict(X_test),
                                       squared=False))
```

```
Training R-Squared: 0.8446321043844025
Test R-Squared: 0.8652881796740386
Training RMSE: 30945.23670210737
Test RMSE: 30072.432048410283
```

[420]:
```
# Code for lasso with alpha from BIC
lasso = Lasso(alpha = model_bic.alpha_)
lasso.fit(X_train, y_train)

# Print R-Squared and RMSE
print('Training R-Squared:', lasso.score(X_train, y_train))
print('Test R-Squared:', lasso.score(X_test, y_test))
print('Training RMSE:', mean_squared_error(y_train,
                                           lasso.predict(X_train),
                                           squared=False))
print('Test RMSE:', mean_squared_error(y_test,
                                       lasso.predict(X_test),
                                       squared=False))



# Print R-Squared and RMSE
```

```
Training R-Squared: 0.8445833071362633
Test R-Squared: 0.865202119898564
Training RMSE: 30950.09589080876
Test RMSE: 30082.036304145742
```

## 1.10 Level up (Optional)

### 1.10.1 Create a Lasso path

From this section, you know that when using Lasso, more parameters shrink to zero as your regularization parameter goes up. In Scikit-learn there is a function `lasso_path()` which visualizes the shrinkage of the coefficients while *alpha* changes. Try this out yourself!

https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_coordinate_descent_path.html#sphx-glr-auto-examples-linear-model-plot-lasso-coordinate-descent-path-py

### 1.10.2 AIC and BIC for subset selection

This notebook shows how you can use AIC and BIC purely for feature selection. Try this code out on our Ames housing data!

https://xavierbourretsicotte.github.io/subset_selection.html

## 1.11 Summary

Congratulations! You now know how to create better linear models and how to use AIC and BIC for both feature selection and to optimize your regularization parameter when performing Ridge and Lasso.