5 Linear Regression

Lab Objective: *This section will introduce the basics of Linear Regression, feature selection methods, and regularization.*

Introduction to Linear Regression

One of the first skills taught in basic algebra is to effectively plot the line $y = mx + b$ which can be done with two points. But what if we want to find the line that best fits a set of points?

In this case, we can use the simplest form of linear regression: *Ordinary Least Squares* (OLS). Given data as a set of points $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ we wish to find the line that best fits the data. The line is given by $y = mx + b$ where m and b are unknown constants and x and y are the independent and dependent variables respectively. Using OLS, let

$$y_i = mx_i + b + \varepsilon_i$$

describe the *ith* point in D for each $i \in \{1, \dots, n\}$. Note that ε_i is the vertical distance from the *ith* point to the line given by $y = mx + b$ and is often called the *residual* or the *error*.

The n equations for each point in D can be written in vector notation. Let the x and y coordinates of D be represented by column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. In statistical science, the intercept (b) and slope (m) are denoted as β_0 and β_1 respectively and

$$\begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \boldsymbol{\beta}.$$

Additionally, the residuals are represented by a column vector $\boldsymbol{\varepsilon}$ and $\mathbf{1}$ is a column vector of ones. So we have

$$\boldsymbol{y} = m\boldsymbol{x} + \mathbf{1}b + \boldsymbol{\varepsilon} = [\mathbf{1}, \boldsymbol{x}] \cdot \begin{bmatrix} b \\ m \end{bmatrix} + \boldsymbol{\varepsilon}.$$

Denoting $X = [\mathbf{1}, \boldsymbol{x}]$, we have our final equation given as

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

This notation may seem excessive, but suppose we wanted to fit a model of the form $y = ax^3 + bx^2 + cx + d$ A little work can show that $X = [\mathbf{1}, \boldsymbol{x}, \boldsymbol{x}^2, \boldsymbol{x}^3]$ and $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3]^T$, which is very easy to work with. Thus, this notation is actually the ideal way to generalize linear regression, especially when working with higher degree polynomials.

The solution to OLS is straight forward with some important assumptions. Sparing you the algebraic details and assuming that $\boldsymbol{y} \sim \mathcal{N}(X\boldsymbol{\beta}, \sigma^2\boldsymbol{I})$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I})$ and $\boldsymbol{I}$ is the identity matrix, the least squares estimator for $\boldsymbol{\beta}$ is given as

$$\widehat{\boldsymbol{\beta}} = (X^TX)^{-1}X^T\boldsymbol{y}. \tag{5.1}$$

Problem 1. Write a function that takes as input `X` and `y`. In your function, add a column of ones to `X` to account for β_0. Call this function `ols`. This function should return the least squares estimator for $\boldsymbol{\beta}$ as a numpy array.

Hint: Use functions from `numpy` or `scipy` to calculate a matrix inverse

Problem 2. Use the following code to generate random data.

```
n = 100 # Number of points to generate
X = np.arange(100) # The input X for the function ols
eps = np.random.uniform(-10,10, size=(100,)) # Noise to generate random y ↩
    coordinates
y = .3*X + 3 + eps # The input y for the function ols
```

Find the least squares estimator for $\boldsymbol{\beta}$ using this random data. Produce a plot showing the random data and the line of best fit determined by the least squares estimator for $\boldsymbol{\beta}$. Your plot should include a title, axis labels, and a legend.

Hint: Since `ols` takes `X` without a column of ones, slice `X` when you call `ols`.

Rank-Deficient Models

Notice that in order to find the least squares estimator $\widehat{\boldsymbol{\beta}}$, we need X^TX to be invertible. However, when X does not have full rank, the product X^TX is singular and not invertible. We can no longer use the previous solution for the least squares estimator, but we can use the SVD and still compute a solution.

Recall that if $X \in M_{n\times d}$ has rank r, then the compact form of the SVD of X is

$$X = U\Sigma V^H$$

where $U \in M_{n\times r}$ and $V \in M_{r\times d}$ have orthonormal columns and $\Sigma \in M_{r\times r}$ is diagonal. In addition, if X is real, then the factors U, Σ, and V^H are also real. In this lab we assume X is real. As described in Volume 1, there is a unique solution for the least squares estimator given by

$$\widehat{\boldsymbol{\beta}} = V\Sigma^{-1}U^T\boldsymbol{y}. \tag{5.2}$$

Problem 3. Write a function that finds the least squares estimator for rank-deficient models using the SVD. The function should still take `X` and `y` as inputs. In your function, add a column of ones to `X` to account for β_0. Call the function `svd_ols` and return the least squares estimator for $\boldsymbol{\beta}$ as a numpy array.

Hint: Use `np.linalg.svd` to factor `X` and use the argument `full_matrices=False`.

Problem 4. Use the following code to generate random data:

```
x = np.linspace(-4, 2, 500)
y = x**3 + 3*x**2 - x - 3.5
eps = np.random.normal(0, 3, len(y)) # Create noise
y += eps # Add noise to randomize data
```

Now use your function `svd_ols` to find the least squares estimator for a cubic polynomial. Create a plot that shows a scatter plot of the data and a curve using the least squares estimator. Your plot should include a title, axis labels, and a legend.

Model Accuracy

Residual Sum of Squares

The *Residual Sum of Squares* (RSS) is a common choice of measure for the quality of a model. The formula for RSS is given by

$$RSS = ||\boldsymbol{y} - X\widehat{\boldsymbol{\beta}}||_2^2.$$

Notice that the RSS measures the variance in the error of the model. So relative to other models, a smaller RSS value indicates a more accurate model.

Coefficient of Determination

Another method of model accuracy is the *Coefficient of Determination*, denoted R^2. In the case of linear regression,

$$R^2 = 1 - \frac{RSS}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

and $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$ is the sample mean of $\boldsymbol{y}$. The intuition of R^2 is that the ratio of the average residual and biased sample variance of $\boldsymbol{y}$ is approximately the total variance explained by the model. A larger R^2 corresponds to a model that fits better. However, R^2 comes with flaws such as being able to take negative values, rewarding overfitting, and punishng under-fit models. Because of this, we typically want to use other methods for model accuracy.

Python Example

There are various python packages that can be used to calculate R^2, but we will use `statsmodels` in this lab. Below is an example of how to build a model and extract R^2 using `statsmodels`.

```
import statsmodels.api as sm
data = pd.read_csv("/filepath") # Read in data as pandas dataframe
y = data["dependent_variable"] # Extract dependent variable
temp_X = data[["var_1", ..., "var_n"]] # Extract independent variables
```

```python
X = sm.add_constant(temp_X) # Add column of 1's
model = sm.OLS(y, X).fit() # Fit the linear regression model
print(model.rsquared) # Print the R squared value
```

Problem 5. The file `realestate.csv` contains transaction data from 2012-2013. It has columns for transaction data, house age, distance to nearest MRT station, number of convenience stores, latitude, longitude, and house price of unit area. [a] Each row in the array is a separate measurement.

Find the combination of variables that builds the model with the best R^2 value when predicting `house price of unit area`. Use `statsmodels` to build each model and calculate R^2. Using the same combination of variables, time the methods `ols`, `svd_ols`, and `statsmodels` . Return a list with the first element being a tuple of times for each method and the second element being the best R^2 value from the first part of the problem.

Hint: The `combinations` method from the `itertools` package will be very helpful for finding all feature combinations.

[a]See `https://www.kaggle.com/datasets/quantbruce/real-estate-price-prediction?resource=download`.

Feature Selection

Every regression model consists of features or variables used to predict a dependent variable or result. An important question to ask when building regression models is, which features are the most important in predicting the dependent variable? In addition to being used for model accuracy, R^2 can also be used in feature selection, as it was in Problem 5. It still has the same pitfalls of rewarding overfitting and punishing under-fit models, but it can be a useful tool used in conjunction with the following tools for feature selection. While there are other methods for implementing feature selection, most incorporate the p-value and are not included in this lab.

Akaike's Information Criterion (AIC)

A simple motivation for AIC is based on balancing goodness of fit and prescribing a penalty for model complexity. A more rigorous motivation for AIC is given in Volume 3 using the *Kullback-Leibler* (KL) divergence. Given two models, f and g, the KL divergence is given by

$$KL(f,g) = \int f(z) \log\left(\frac{f(z)}{g(z)}\right) dz$$

and it measures the amount of information lost when g is used to model f. Thus, a lower AIC value indicates a better model. Additionally, AIC penalizes the size of the parameter space with a coefficient of 2 which allows for slightly more complex models.

Bayesian Information Criterion (BIC)

Instead of estimating the KL-divergence between the model in question and the true model, BIC has the property of being minimized precisely when the posterior probability of a model, given the data, is maximized. The equations for AIC and BIC only differ with one term: the coefficient weighting the size of the parameter space. The coefficient for BIC is $log(n)$ which is generally much larger than 2. As a result, BIC penalizes complex models more than AIC. The difference in AIC and BIC values will grow from having more data points.

When using AIC or BIC for feature selection, you need to consider how you want to penalize features in your model. If you want to exclude irrelevant features, then use BIC. If you want to keep all features that are relevant, then use AIC. In other words, BIC is more likely to choose too small a model, and AIC is more likely to choose too large a model.

Python Example

There are multiple ways to calculate AIC and BIC with various python packages. We will use the package `statsmodels` for the following problem. When constructing X for `statsmodels`, do not add the column of 1's manually because `statsmodels` has a method that will do this for us.

```python
import statsmodels.api as sm
data = pd.read_csv("/filepath") # Read in data as pandas dataframe
y = data["dependent_variable"] # Extract dependent variable
temp_X = data[["var_1", ..., "var_n"]] # Extract independent variables
X = sm.add_constant(temp_X) # Add column of 1's
model = sm.OLS(y, X).fit() # Fit the linear regression model
print(model.aic) # or print(model.bic)
```

Problem 6. Use the file `realestate.csv` and the Python Example above as a template for constructing `y` and `X` and calculating model AIC and BIC. For the dependent variable, use `house price of unit area`. For the independent variables, use `distance to the nearest MRT station`, `number of convenience stores`, `latitude`, and `longitude`.

Find the model that has the lowest AIC and the model that has the lowest BIC. Are they the same model? Print the features of the model with the lowest AIC as a list.

Hint: The `combinations` method from the `itertools` package will be very helpful for finding all feature combinations.

Regularization

Up to this point, we have been solving the problem

$$\min_{\boldsymbol{\beta}} ||X\boldsymbol{\beta} - \boldsymbol{y}||_2^2.$$

However, we have also assumed independence among the features used to predict the dependent variable. The pitfall of multicollinearity arises when the features of X have dependence and X becomes nearly singular. As a result, the least squares estimator is susceptible to random noise or error. Multicollinearity typically occurs when data is collected with poor experimental design. It is important to have good experimental design, but regularization can be used to mitigate poor design. Another issue OLS faces is feature selection. While there are feature selection methods available, regularization can be used to minimize non-zero coefficients.

Ridge Regularization Regression

The problem posed by *Ridge Regularization* is

$$\min_{\boldsymbol{\beta}} ||X\boldsymbol{\beta} - \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_2^2$$

where $\alpha \geq 0$. This essentially penalizes the size of the coefficients. The larger α is, the more the model resists multicollinearity.

Lasso Regularization Regression

The problem posed by *Lasso Regularization* is

$$\min_{\boldsymbol{\beta}} \frac{1}{n}||X\boldsymbol{\beta} - \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_1.$$

Note that α provides the same functionality here as it does in Ridge Regularization. However, the use of the 1-norm often results in sparse solutions. As a result, Lasso Regularization can be used for feature selection since it only includes the most important features.

Python Example

Since α is not a fixed value in Ridge and Lasso Regularization, it is best practice to perform a Grid-Search to find the best parameter value. The example below goes over the syntax for implementing Ridge Regularization. Note that the syntax for Lasso Regularization is similar.

```python
>>> from sklearn import linear_model
>>> y = # dependent variable data
>>> X = # independent variable data with no column of ones
>>> reg = linear_model.RidgeCV(alphas=np.logspace(-6, 6, 13)) # Range for grid ↩
    search
>>> reg.fit(X, y) # Fit the model
>>> reg.alpha_ # Best parameter value
```

Problem 7. Use Ridge and Lasso Regression to model `house price of unit area` from the file `realestate.csv`. First, do a grid search for the model parameter. Then use the grid search result to fit the model. Once you have fit the model, you can use the `score` method to get R^2. Print R^2 for each model as a tuple. How do these models compare to the models in problem 6?