# jgiebas_HW1

January 22, 2018

## 1 46-926, Statistical Machine Learning 1: Homework 1

*Author* : Jordan Giebas *Due Date*: January 22nd, 2018

### 1.1 Question 1:

I previously had all of the required packages downloaded. The below input/output cell gives proof,

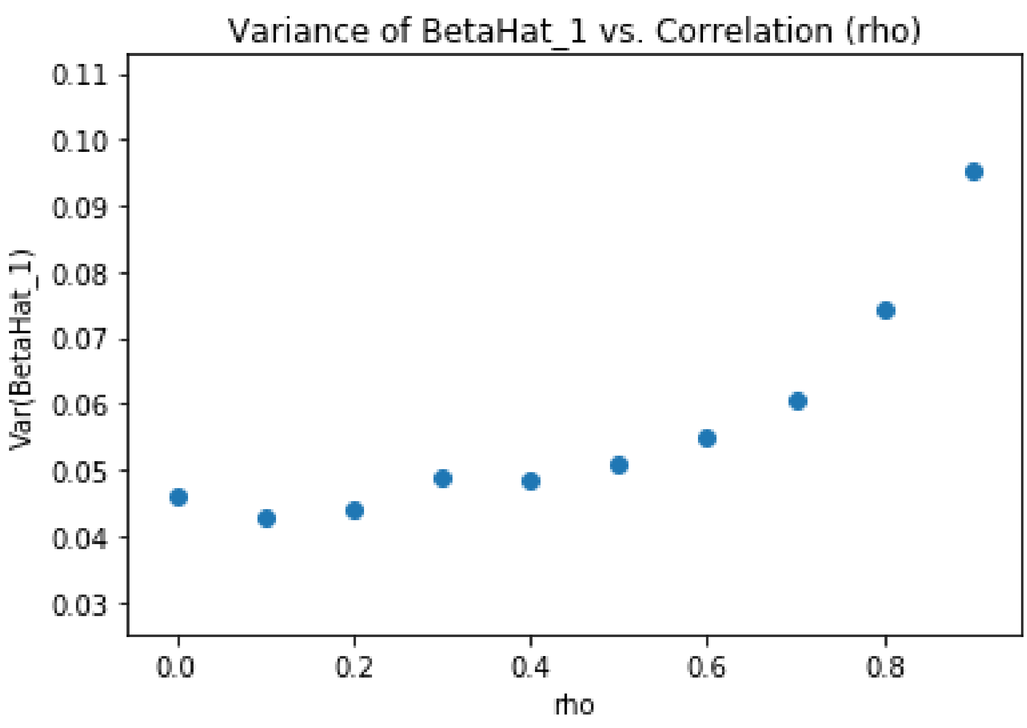
```
In [4]: !pip3 install scipy numpy sklearn

Requirement already satisfied: scipy in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: sklearn in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/site-packages (from sk
```

### 1.2 Question 2(i):

In question two, we explore the relationship between the correlation of linear regression feature variables and the variance of the estimated parameters for one on the features.

```
In [5]: import pandas as pd
        import numpy as np
        import statsmodels.formula.api as sm
        import matplotlib.pyplot as plt

        def build_df( num_obs, rho ):

            n = num_obs
            x_means = [0,0]
            x_cov = [[1,rho],[rho,1]]
            feature_list = np.random.multivariate_normal(x_means,x_cov,size=n)

            x1_col = pd.Series([x[0] for x in feature_list])
            x2_col = pd.Series([x[1] for x in feature_list])
            y_col = pd.Series(np.array([x[0]+x[1]+np.random.normal() for x in feature_list]))
```

```
    df = pd.concat([x1_col,x2_col,y_col], axis=1)
    df.columns = ['X_1', 'X_2','Y']

    return df

def get_b1h( df ):

    # create a fitted model with all three features
    lm = sm.ols(formula='Y ~ X_1 + X_2', data=df).fit()
    return lm.bse['X_1']

def driver_2a():

    r_ = np.arange(0.0,1.0,0.1)
    bh1_ = [get_b1h(build_df(500, rho)) for rho in r_]

    %matplotlib inline
    plt.ylabel('Var(BetaHat_1)')
    plt.xlabel('rho')
    plt.title('Variance of BetaHat_1 vs. Correlation (rho)')
    plt.scatter(r_,bh1_)

driver_2a()
```

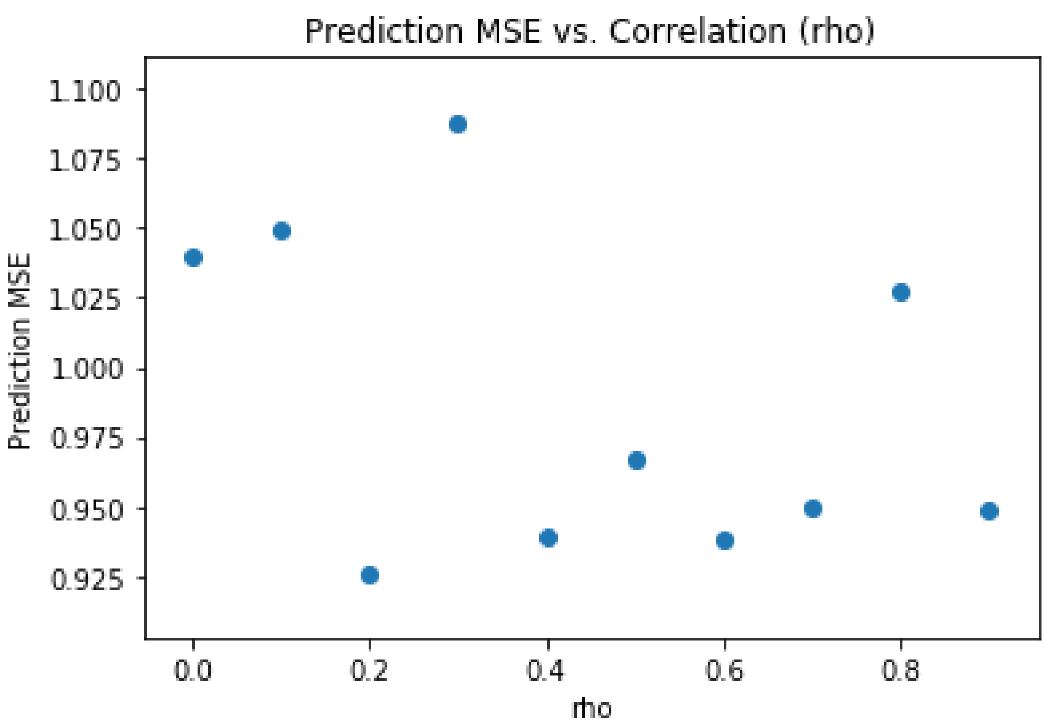


There appears to be a strong, nearly quadratic, relationship between $\rho$ and $\hat{\beta}_1$. The only hiccup is near $\rho = 0.2$, when the relationship hits a local maximum.

### 1.3 Question 2(ii):

```
In [10]: def build_test_df( num_obs, rho ):

             n = num_obs
             x_means = [0,0]
             x_cov = [[1,rho],[rho,1]]
             feature_list = [np.random.multivariate_normal(x_means,x_cov) for i in range(n)]

             x1_list = pd.Series([x[0] for x in feature_list])
             x2_list = pd.Series([x[1] for x in feature_list])
             eps_list = pd.Series([np.random.normal() for i in range(n)])

             y_list = pd.Series(np.array([ x[0] + x[1] + e for x,e in zip(feature_list, eps_li

             df = pd.concat([x1_list,x2_list,y_list], axis=1)
             df.columns = ['X_1', 'X_2','Y_act']

             return df

         def get_linear_model( df ):

             lm = sm.ols(formula='Y ~ X_1 + X_2', data=df).fit()
             return lm


         def get_mse( lm, rho ):

             # Build the test dataframe with the rho passed in
             test_df = build_test_df(500,rho)

             # Generate the predicted Y values using the linear model passed in,
             # and determine the squared error
             test_df['Y_pred'] = lm.predict(test_df)
             test_df['Y_err'] = (test_df.Y_act - test_df.Y_pred)**2

             # Return the mean squared error
             return test_df.Y_err.mean()

         def driver_2b_helper(rho):

             train_df = build_df(500, rho)
             lm = get_linear_model( train_df )

             return get_mse(lm, rho)
```

```
def driver_2b():

    rho_list = np.arange(0.0,1.0,0.1)
    avgerr_list = [driver_2b_helper(rho) for rho in rho_list]

    # Pass the linear model and the rho into the get_avgerr function
    #avgerr_list = [get_mse(lm, rho) for rho in rho_list]

    %matplotlib inline
    plt.ylabel('Prediction MSE')
    plt.xlabel('rho')
    plt.title('Prediction MSE vs. Correlation (rho)')
    plt.scatter(rho_list, avgerr_list)


driver_2b()
```




There appears to be no significant relationship between the correlation of the feature variables and the Mean Squared Error (MSE) of the predicted values.

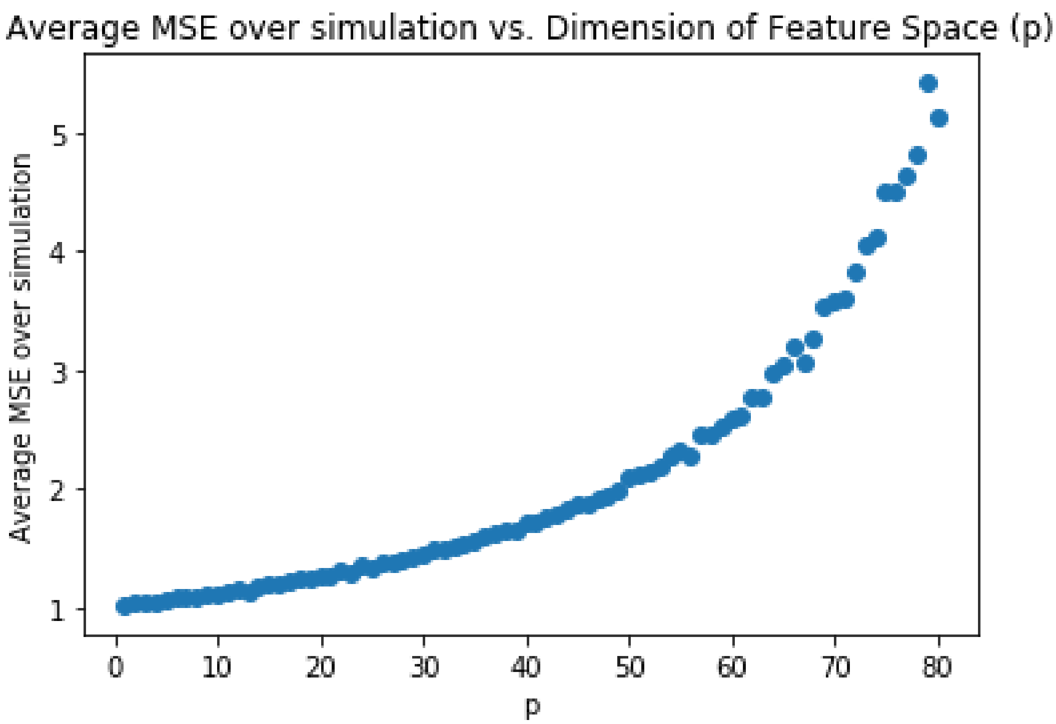
### 1.4 Question 3:

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error
        import matplotlib.pyplot as plt

        p_list = list()
        avg_mse_list = list()
        for p in range(1,81):

            p_list.append(p)
            mse_list = list()
            for i in range(100):

                # Build a train dataframe (100 obs)
                train_df = pd.DataFrame(np.random.normal(size=(100,p)), columns=pd.RangeIndex(s
                train_df['Y'] = 4*train_df[1] + np.random.normal(size=100)

                # Fit the linear model
                X = train_df[train_df.columns.tolist()[:-1]]
                y = train_df['Y']
                lm = LinearRegression()
                lm.fit(X,y)

                # Build the test dataframe (1000 obs)
                test_df = pd.DataFrame(np.random.normal(size=(1000,p)), columns=pd.RangeIndex(s
                test_df['Y'] = 4*test_df[1] + np.random.normal(size=1000)

                # Use the linear model to predict
                X_new = test_df[test_df.columns.tolist()[:-1]]
                test_df['Y_pred'] = lm.predict(X_new)
                test_df['Y_sqerr'] = (test_df.Y - test_df.Y_pred)**2

                # mse is average of Y_sqerr column
                mse = test_df.Y_sqerr.mean()

                # append to mse_list
                mse_list.append(mse)


            # get average mse over simulations
            avg_mse = np.array(mse_list).mean()

            # append to avg_mse_list
            avg_mse_list.append(avg_mse)
```

```
%matplotlib inline
plt.ylabel('Average MSE over simulation')
plt.xlabel('p')
plt.title('Average MSE over simulation vs. Dimension of Feature Space (p)')
plt.scatter(p_list, avg_mse_list)
```

```
/usr/local/lib/python3.6/site-packages/scipy/linalg/basic.py:1226: RuntimeWarning: internal gel
  warnings.warn(mesg, RuntimeWarning)
```

```
Out[1]: <matplotlib.collections.PathCollection at 0x10e9fe0b8>
```




The relationship between the average MSE (over the simulations) and the dimension of the feature space (p) appears to be quite exponential.

### 1.5 Question 4(a):

```
In [8]: def q4_a():

            # Estimator I: Using the mean
            def est1_mse():

                sqerr_list = [(np.random.exponential(1) - 1)**2 for i in range(1000)]
                mse_e1 = np.mean(sqerr_list)
```

```
            return mse_e1

        # Estimator II: Using the median
        def est2_mse():

            sqerr_list = [(np.random.exponential(1) - np.log(2))**2 for i in range(1000)]
            mse_e2 = np.mean(sqerr_list)

            return mse_e2

        def print_helper( s ):

            print("Estimator %s is better, since it has a lower MSE" % s)


        print("The MSE for estimator I: ", est1_mse())
        print("The MSE for estimator II: ", est2_mse())

        print_helper("I") if (est1_mse() < est2_mse()) else print_helper("II")

    q4_a()
```

```
The MSE for estimator I:  0.899901169516
The MSE for estimator II:  1.03740777093
Estimator I is better, since it has a lower MSE
```

### 1.6 Question 4(b):

```
In [9]: from math import fabs

    def q4_b():

        # Estimator I: Using the mean
        def est1_mse():

            sqerr_list = [fabs(np.random.exponential(1) - 1) for i in range(1000)]
            mse_e1 = np.mean(sqerr_list)

            return mse_e1

        # Estimator II: Using the median
        def est2_mse():

            sqerr_list = [fabs(np.random.exponential(1) - np.log(2)) for i in range(1000)]
            mse_e2 = np.mean(sqerr_list)

            return mse_e2
```

```
    def print_helper( s ):

        print("Estimator %s is better, since it has a lower MSE" % s)


    print("The MSE for estimator I: ", est1_mse())
    print("The MSE for estimator II: ", est2_mse())

    print_helper("I") if (est1_mse() < est2_mse()) else print_helper("II")

q4_b()
```

```
The MSE for estimator I:  0.712937473311
The MSE for estimator II:  0.700672038384
Estimator II is better, since it has a lower MSE
```

This is interesting. By the above two cells, we see that the estimator II, the estimator using the median, is better when using the median squared prediction error but it's worse when using the mean squared error per usual.