LinearRegression

November 2, 2017

```
In [1]: import _pickle as cp
        import numpy as np
```

0.0.1 Dataset Loader

```
In [2]: def load_dataset(name, get_xy = False):
            X,y = cp.load(open(name,'rb'))
            if get_xy:
                return X, y
            N, D = X.shape
            N_train = int(.8 * N)
            N_test = N - N_train

            X_train = X[:N_train]
            y_train = y[:N_train]
            X_test = X[N_train:]
            y_test = y[N_train:]

            return X_train, y_train, X_test, y_test
```

0.0.2 Task 1 - Visualisation

```
In [3]: import matplotlib.pyplot as plt
```

```
In [4]: def visualise(name):
            X, y = load_dataset(name, True)
            unique, counts = np.unique(y.astype(int), return_counts=True)
            print(unique)
            print(counts)
            plt.bar(range(min(unique),max(unique)+1), counts)
            plt.show()
```

0.0.3 Task 2 - Predict average as solution and find MSE

```
In [5]: def mean_predictor(y_train, y_test):
            y_avg = np.mean(y_train)
            mse = np.mean(np.power(y_test - y_avg, 2))
            print('The average is '+str(y_avg))
            print('The mse is '+str(mse))
```

0.0.4 Task 3 - Linear Regression & Task 4 - Plot Effect of varying train size on MSE

In [6]:
```
import os

#Scaler function, X - Input, reset - Reset the saved musigma, verbose - Do you want co
def scaler(X, reset = False, verbose = False):
    if os.path.exists('musigma.pickle') and not reset:
        if verbose:
            print('musigma exists!, using values')
        mean, std = cp.load(open('musigma.pickle','rb'))
        X = (X-mean)/std #scaler
    else:
        if verbose:
            print('musigma doesn\'t exist!, creating values')
        mean = np.mean(X)
        std = np.std(X)
        musigma = (mean, std)
        cp.dump(musigma,open('musigma.pickle', 'wb'))
        X = (X-mean)/std #scaler

    return X

#Mean Squared error
def mean_squared_error(y, y_pred):
    return np.mean(np.power(y-y_pred, 2)) #This is mean((y-ypred)^2)

#Split into train, validation, split
def train_validation_split(X_train, y_train):
    x_t = X_train[:int(len(X_train)*.8)]
    x_v = X_train[int(len(X_train)*.8):]
    y_t = y_train[:int(len(X_train)*.8)]
    y_v = y_train[int(len(X_train)*.8):]

    return x_t, x_v, y_t, y_v

#Do linear regression
def linear_regression(X_train, y_train, X_test, y_test, train_size=None, train_validati
    if train_validation:
        x_t, x_v, y_t, y_v = train_validation_split(X_train, y_train) #get train, vali
    else:
        x_t = X_train[:train_size]
        y_t = y_train[:train_size]
        x_v = X_test #use x_v as x_test
        y_v = y_test #use y_v as y_test

    x_train_scaled = scaler(x_t, True)
    w = np.matmul(np.matmul(np.linalg.pinv(np.matmul(x_train_scaled.T, x_train_scaled))
```

```
        y_pred = np.matmul(x_train_scaled, w)
        mse_t = mean_squared_error(y_t, y_pred)
        if train_validation:
            print('The mean squared error on train is '+str(mse_t))


        x_validation_scaled = scaler(x_v)
        y_pred = np.matmul(x_validation_scaled, w)
        mse_v = mean_squared_error(y_v, y_pred)
        if train_validation:
            print('The mean squared error on validation is '+str(mse_v))
        np.save('weights.npy', w) #Numpy saves weights faster
        if verbose:
            print('Weights saved')
        return (mse_t, mse_v)
```

In [7]:
```
#Function for prediction
def linear_regression_predict(x):
    w = np.load('weights.npy')
    y_pred = np.matmul(x,w)
    return y_pred
```

In [8]:
```
#Function with mse
def linear_regression_predict_with_mse(x, y):
    y_pred = linear_regression_predict(x)
    mse = mean_squared_error(y_pred, y)
    print('The mean squared error is '+str(mse))
    return y_pred
```

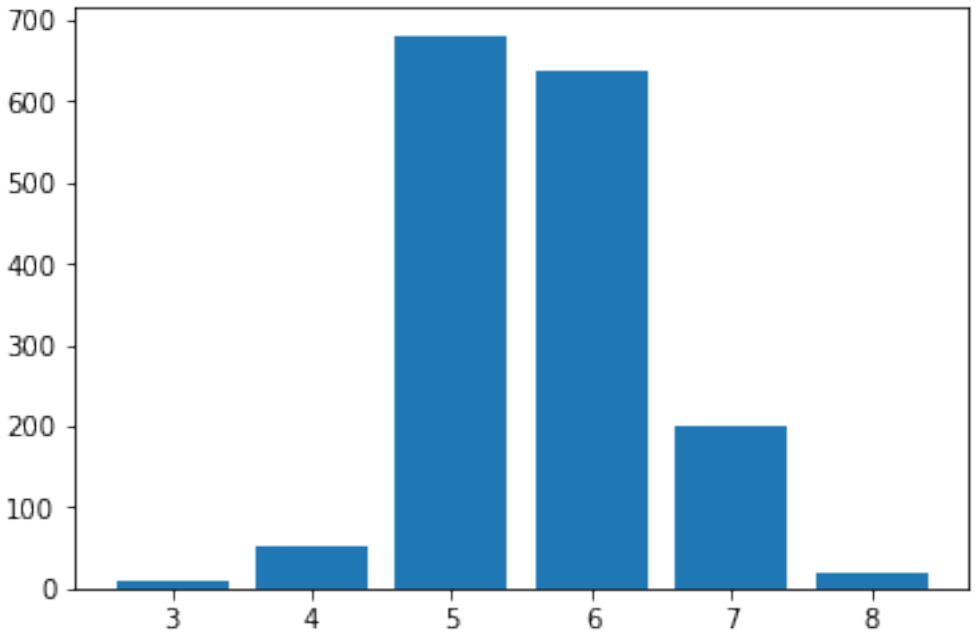
In [9]:
```
#The function for various sizes
def size_varying_train(X_train, y_train, X_test, y_test):
    mse_train = []
    mse_test = []
    for i in range(20, X_train.shape[0]+1, 20):
        mse_pair = linear_regression(X_train, y_train, X_test, y_test, i, train_validat
        if i%100 == 0:
            print('Train size:'+str(i)+' train error:'+str(mse_pair[0])+' test error:'
        mse_train.append(mse_pair[0])
        mse_test.append(mse_pair[1])

    plt.plot(range(20,X_train.shape[0]+1, 20), mse_train, 'r', label = 'Train')
    plt.plot(range(20,X_train.shape[0]+1, 20), mse_test, 'b', label = 'Test')
    plt.xlabel('Train size')
    plt.ylabel('Mean Squared Error')
    plt.title('Train Size vs Mean Squared Error')
    plt.show()
```

In [10]:
```
def main():
     i = input('Enter 1 for red wine and 2 for white wine')
```

```
    name = None
    if int(i) == 1:
        name = 'winequality-red.pickle'
    elif int(i) == 2:
        name = 'winequality-white.pickle'
    else:
        print('Wrong choice!')
        return
    visualise(name)
    X_train, y_train, X_test, y_test = load_dataset(name)
    mean_predictor(y_train, y_test)
    linear_regression(X_train, y_train, X_test, y_test)
    linear_regression_predict_with_mse(X_test, y_test)
    size_varying_train(X_train, y_train, X_test, y_test)
```

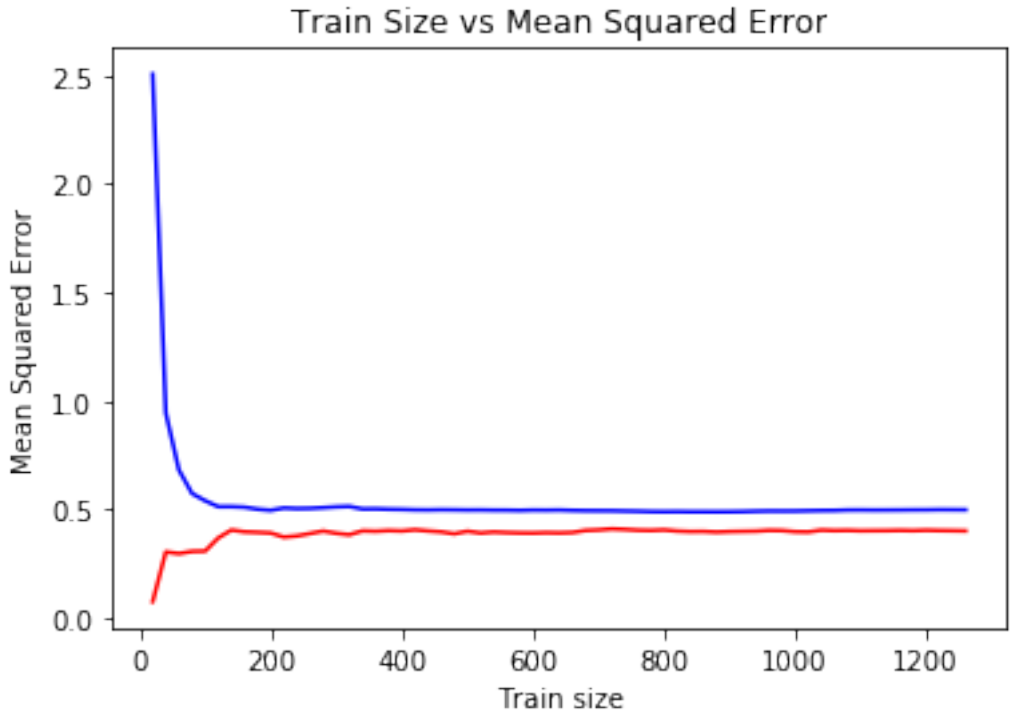
```
In [11]: main()
```

```
Enter 1 for red wine and 2 for white wine1
[3 4 5 6 7 8]
[ 10  53 681 638 199  18]
```

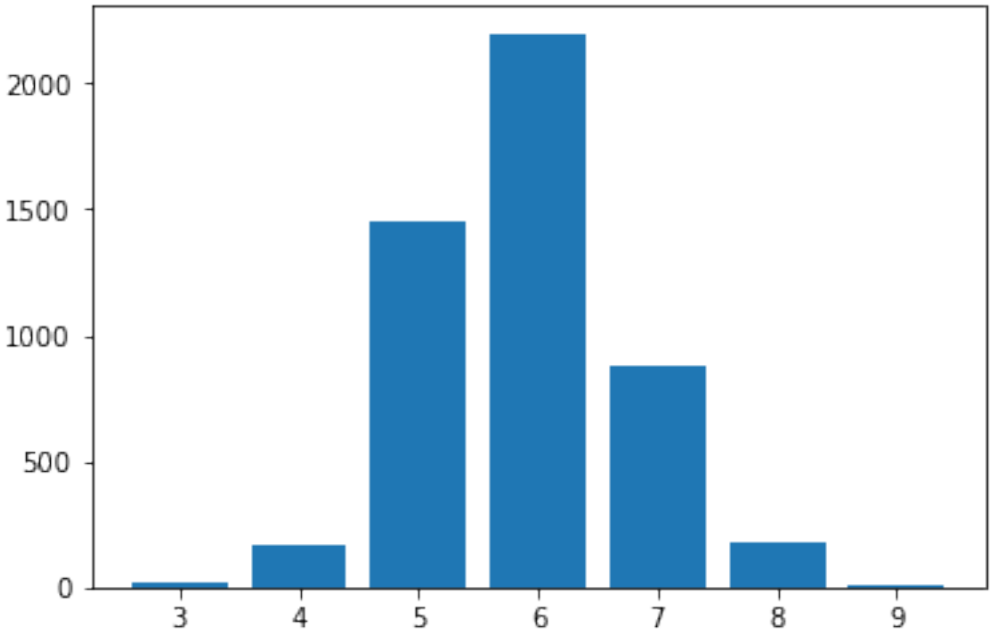



```
The average is 5.62470680219
The mse is 0.745345570719
The mean squared error on train is 0.401942011069
The mean squared error on validation is 0.385870656968
```

```
The mean squared error is 1633.41931452
Train size:100 train error:0.307766735441 test error:0.539769507304
Train size:200 train error:0.391175869666 test error:0.495576959332
Train size:300 train error:0.389793694866 test error:0.511555092207
Train size:400 train error:0.400502470447 test error:0.499559446827
Train size:500 train error:0.399399739568 test error:0.496930937861
Train size:600 train error:0.390955755964 test error:0.496560654966
Train size:700 train error:0.405293478835 test error:0.493821722846
Train size:800 train error:0.405365954412 test error:0.489294419784
Train size:900 train error:0.397806876642 test error:0.489092544483
Train size:1000 train error:0.397121016083 test error:0.492446297905
Train size:1100 train error:0.401402074825 test error:0.49712313109
Train size:1200 train error:0.403850844812 test error:0.497732734477
```

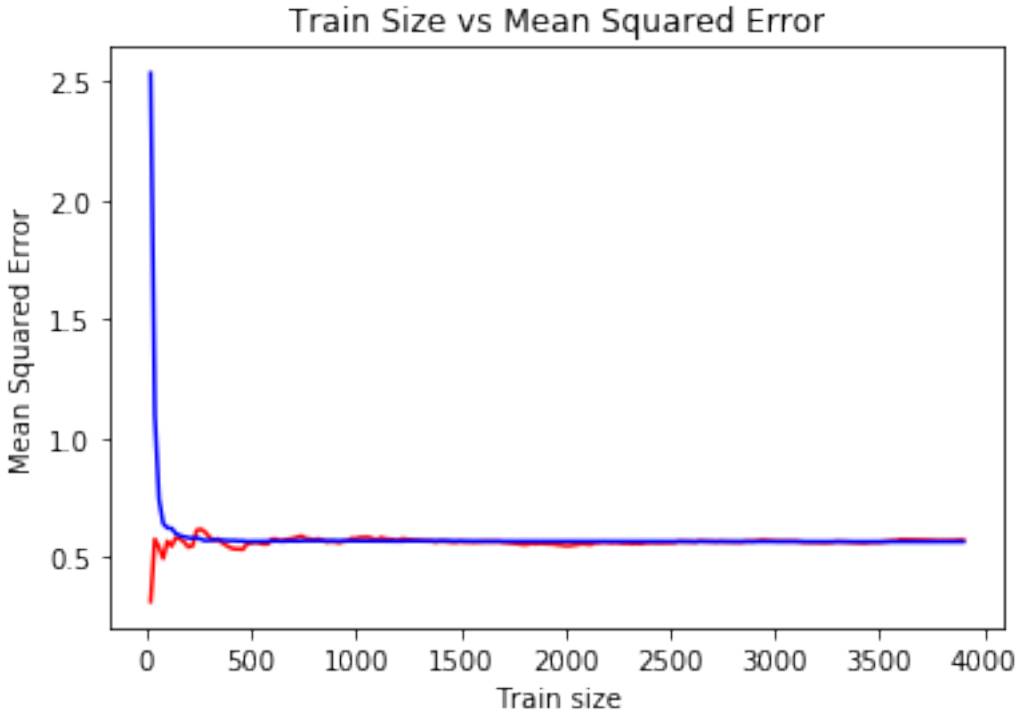



```
In [12]: main()

Enter 1 for red wine and 2 for white wine2
[3 4 5 6 7 8 9]
[  20  163 1457 2198  880  175    5]
```




```
The average is 5.87876467586
The mse is 0.813857300005
The mean squared error on train is 0.564998645178
The mean squared error on validation is 0.600466228909
The mean squared error is 53746.8840465
Train size:100 train error:0.568629863107 test error:0.624914303906
Train size:200 train error:0.54454628833 test error:0.583364251312
Train size:300 train error:0.586959459106 test error:0.572453649901
Train size:400 train error:0.54152474435 test error:0.571053285017
Train size:500 train error:0.561051583921 test error:0.567664473548
Train size:600 train error:0.577928015179 test error:0.570519633683
Train size:700 train error:0.580300526519 test error:0.569945274568
Train size:800 train error:0.573702419688 test error:0.571656202184
Train size:900 train error:0.566797183793 test error:0.5710509755
Train size:1000 train error:0.580049383627 test error:0.570839075785
Train size:1100 train error:0.574282132323 test error:0.569848001289
Train size:1200 train error:0.571257838003 test error:0.569091982921
Train size:1300 train error:0.572336253232 test error:0.570652838067
Train size:1400 train error:0.568603298874 test error:0.571147191211
Train size:1500 train error:0.567961228686 test error:0.57005697231
Train size:1600 train error:0.566958319196 test error:0.568843289752
Train size:1700 train error:0.564585099535 test error:0.56820039947
Train size:1800 train error:0.55492891614 test error:0.568357790143
Train size:1900 train error:0.558660858112 test error:0.568465268619
Train size:2000 train error:0.550093983709 test error:0.568903146152
Train size:2100 train error:0.556157189469 test error:0.567880548243
```

```
Train size:2200 train error:0.564330546059 test error:0.567413466063
Train size:2300 train error:0.560691643342 test error:0.567651234362
Train size:2400 train error:0.562973197332 test error:0.567391597316
Train size:2500 train error:0.563154490903 test error:0.567187507572
Train size:2600 train error:0.566570447443 test error:0.567068956201
Train size:2700 train error:0.567602272453 test error:0.567176383037
Train size:2800 train error:0.566101007242 test error:0.567095711121
Train size:2900 train error:0.56814437745 test error:0.566662135322
Train size:3000 train error:0.568999398097 test error:0.566433334558
Train size:3100 train error:0.568767960417 test error:0.566170844428
Train size:3200 train error:0.56397982893 test error:0.566303073563
Train size:3300 train error:0.567797134779 test error:0.566276194783
Train size:3400 train error:0.563058266111 test error:0.566407067265
Train size:3500 train error:0.564747377831 test error:0.565920029629
Train size:3600 train error:0.576077399252 test error:0.566322075545
Train size:3700 train error:0.573682517498 test error:0.566761398113
Train size:3800 train error:0.571098018476 test error:0.567019762691
Train size:3900 train error:0.573165980286 test error:0.567032324169
```



From the above runs, we can see that as the train size increases, the train error increases, i.e.- overfitting decreases, and the test error decreases. Then there is an equilibrium point where the train, test errors don't improve, meaning the model has reached its threshold accuracy. Using this given model and data split in this exact way, this is the optimal solution that can be reached.

0.0.5 For optional part, building the pipeline

In [13]:
```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

#Setup the pipeline, alpha is lambda and type is Ridge or Lasso
def set_up_pipeline(alpha, type_of_model = 'Ridge'):
    scaler = StandardScaler()
    polynomial_features = PolynomialFeatures()
    model = None
    if type_of_model == 'Ridge':
        model = Ridge(alpha=alpha)
    else:
        model = Lasso(alpha=alpha)
    pipeline = Pipeline([('scaler',scaler),('polynomial_features', polynomial_features
    return pipeline
```

In [14]:
```
def pipeline_train(X_train, y_train, type_of_model = 'Ridge'):
    x_t,x_v, y_t, y_v = train_validation_split(X_train, y_train)
    min_i = -2
    min_score = 100 #Should be a bad model if mse is 100 on values ranging from 3-9
    best_model = None
    for i in range(-2,3): #Train on various powers of 10 from -2 to 2
        pipeline = set_up_pipeline(10^i, type_of_model)#setup pipeline
        pipeline.fit(x_t, y_t) #Fit the model
        y_p = pipeline.predict(x_v) #Predict the model
        score = mean_squared_error(y_p, y_v) #Find MSE
        if score < min_score: #Save one with best mse
            min_i = i
            min_score = score
            best_model = pipeline
        print('Score for 10^'+str(i)+': '+str(score))
    print('Saving best model')
    cp.dump(pipeline, open('best_model_'+type_of_model+'.pickle','wb')) #saving best
    print('Saved best model at 10^'+str(min_i))
    return
```

In [15]:
```
#Test the pipeline, similar to testing linear regression model.
def pipeline_test(X_test, y_test, type_of_model = 'Ridge'):
    pipeline = cp.load(open('best_model_'+type_of_model+'.pickle','rb'))
    y_p = pipeline.predict(X_test)
    score = mean_squared_error(y_p, y_test)
    print('Test score is '+str(score))
    return
```

In [16]:
```
#this is the driver function
def main_t():
```

```
    x = input('Enter 1 for Red Wine and 2 for White ')
    i = input('Enter 1 for Ridge and 2 for Lasso ')
    type_of_model = None

    if int(x) == 1:
        name = 'winequality-red.pickle'
    elif int(x) == 2:
        name = 'winequality-white.pickle'
    else:
        print('Wrong choice!')
        return

    if int(i) == 1:
        type_of_model = 'Ridge'
    elif int(i) == 2:
        type_of_model = 'Lasso'
    else:
        print('Wrong choice!')
        return
    X_train, y_train, X_test, y_test = load_dataset(name)
    pipeline_train(X_train, y_train, type_of_model)
    pipeline_test(X_test, y_test, type_of_model)
    return
```

```
In [17]: main_t()
```

```
Enter 1 for Red Wine and 2 for White 1
Enter 1 for Ridge and 2 for Lasso 1
Score for 10^-2: 0.491010793359
Score for 10^-1: 0.605578694285
Score for 10^0: 0.403309700704
Score for 10^1: 0.402993527374
Score for 10^2: 0.403983765373
Saving best model
Saved best model at 10^1
Test score is 0.507834779059
```

This shows that of all the models built, the model where lambda is 10 is the best, with a test score of .5

```
In [18]: main_t()
```

```
Enter 1 for Red Wine and 2 for White 2
Enter 1 for Ridge and 2 for Lasso 1
Score for 10^-2: 4.42234498251
Score for 10^-1: 0.60814220103
Score for 10^0: 0.52783399967
Score for 10^1: 0.527924669111
```

```
Score for 10^2: 0.527684571793
Saving best model
Saved best model at 10^2
Test score is 0.518541700791
```

This shows that of all the models built, the model where lambda is 10^2 is the best, with a test score of .51

```
In [19]: main_t()

Enter 1 for Red Wine and 2 for White 2
Enter 1 for Ridge and 2 for Lasso 2
Score for 10^-2: 654589.738395
Score for 10^-1: 550032.318858
Score for 10^0: 0.790633883341
Score for 10^1: 0.790633883341
Score for 10^2: 0.790633883341
Saving best model
Saved best model at 10^0
Test score is 0.813886751815
```

This shows that of all the models built, the model where lambda is 1 is the best, with a test score of .81, this is due to the nature of the data, which is distinctly gaussian and fitting a laplacian model on it will not work too well.

```
In [20]: main_t()

Enter 1 for Red Wine and 2 for White 1
Enter 1 for Ridge and 2 for Lasso 2
Score for 10^-2: 1224884.67871
Score for 10^-1: 1029246.11243
Score for 10^0: 0.630079854303
Score for 10^1: 0.630079854303
Score for 10^2: 0.630079854303
Saving best model
Saved best model at 10^0
Test score is 0.744824325222
```

This shows that of all the models built, the model where lambda is 1 is the best, with a test score of .74, this is due to the nature of the data, which is distinctly gaussian and fitting a laplacian model on it will not work too well.