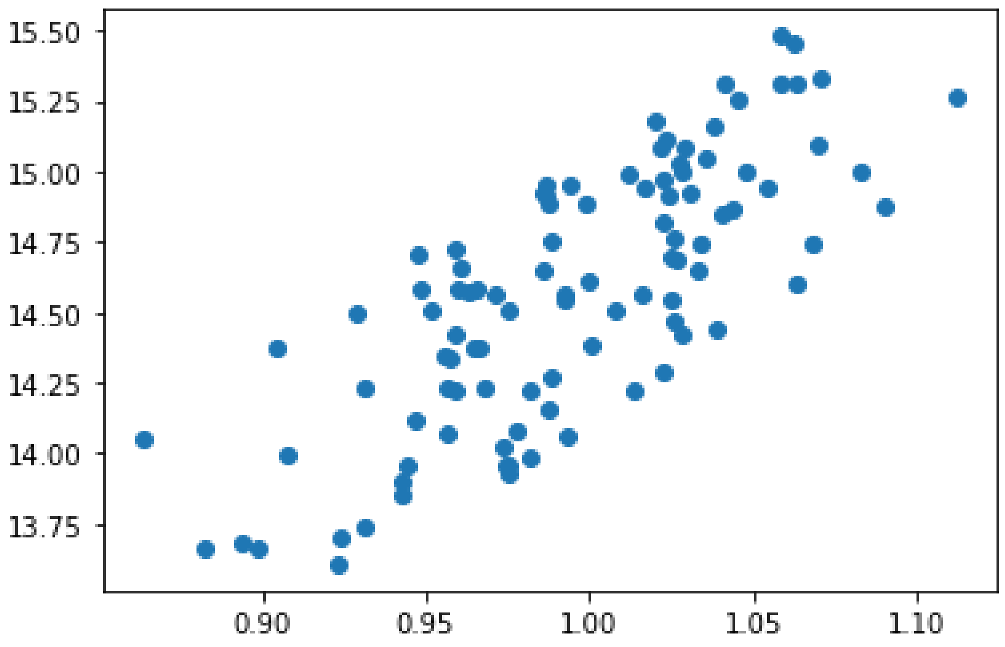
Lab 6 - ML Programming

December 17, 2021

1 EXERCISE 0

1.1 Data pre-processing

[67]:
```
## Generate a Sample dataset called D1 :
## Initialize matrix x   R100×1 using Uniform distribution with μ = 1 and   = 0.
↪05
## Generate target y   R100×1 using y = 1.3x^2 + 4.8x + 8 +  , where    R100×1␣
↪randomly initialized
## Wine Quality called D2: (use winequality-red.csv)

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge


D1_x = np.random.normal(loc=1, scale=0.05, size=(100,1))
random = np.random.rand(100,1)
D1_y = (1.3*(D1_x**2)) + (4.8*D1_x) + 8 + random

D2 = pd.read_csv('winequality-red.csv',delimiter=';')

plt.scatter(D1_x,D1_y)
plt.show()
```



2 EXERCISE 1

2.1 Generalized Linear Models with Scikit Learn

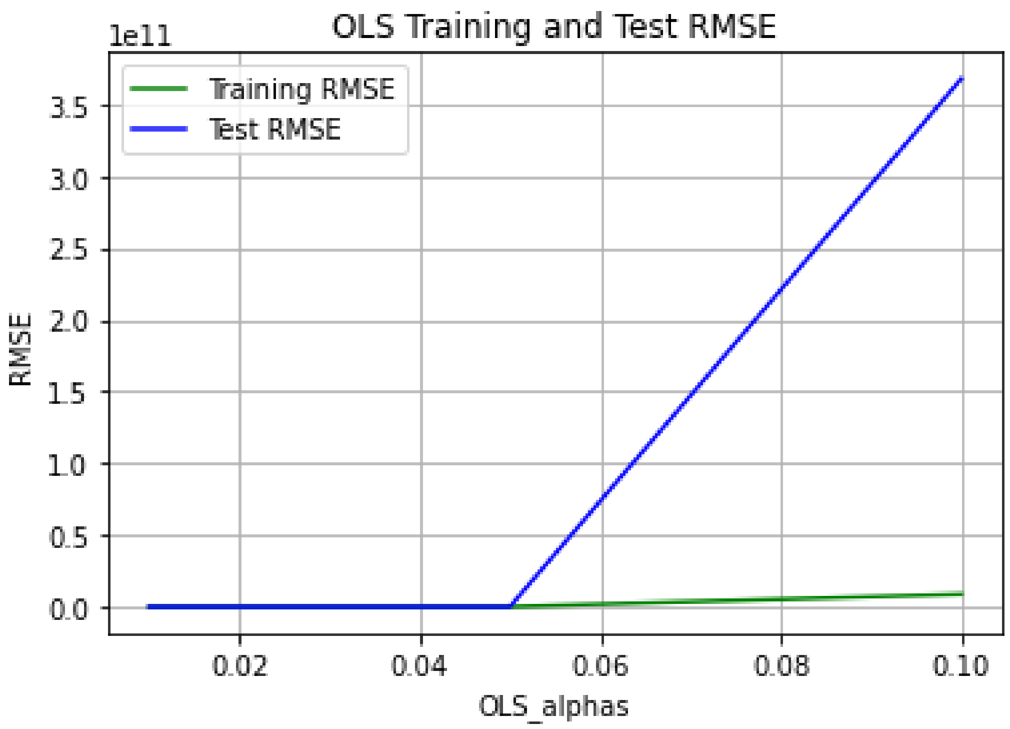
[12]:
```
## Split wine data into test and train using 80%-20% split
D2_train = D2.sample(frac=0.8,random_state=42) ## random state is just a seed␣
↪value
D2_test = D2.drop(D2_train.index)
```

[13]:
```
## Normalize the data with xi-μ/
## We should first normalize the training data
for column in D2_train.columns:
    D2_train[column] = (D2_train[column]-D2_train[column].mean())/
↪D2_train[column].std()

## To normalize test set, apply normalization parameters obtained from training␣
↪set
for column in D2_test.columns:
    D2_test[column] = (D2_test[column]-D2_train[column].mean())/
↪D2_train[column].std()
```

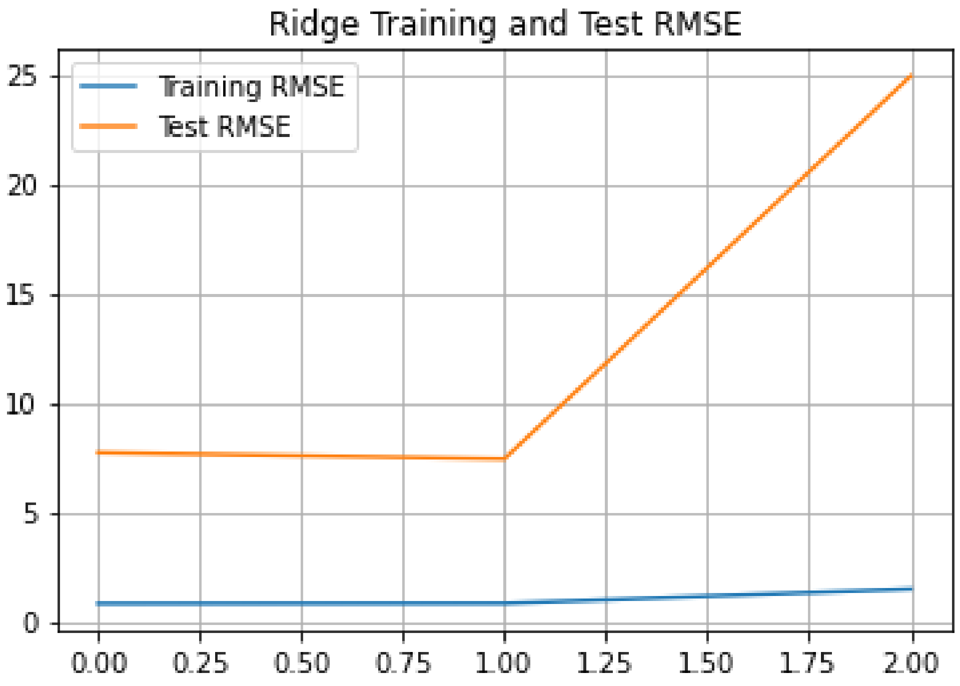
[14]:
```
## Split data into features and targets
x_train = D2_train.iloc[:,:-1].values
y_train = D2_train.iloc[:,-1].values
x_test = D2_test.iloc[:,:-1].values
```

```
y_test = D2_test.iloc[:,-1].values
```

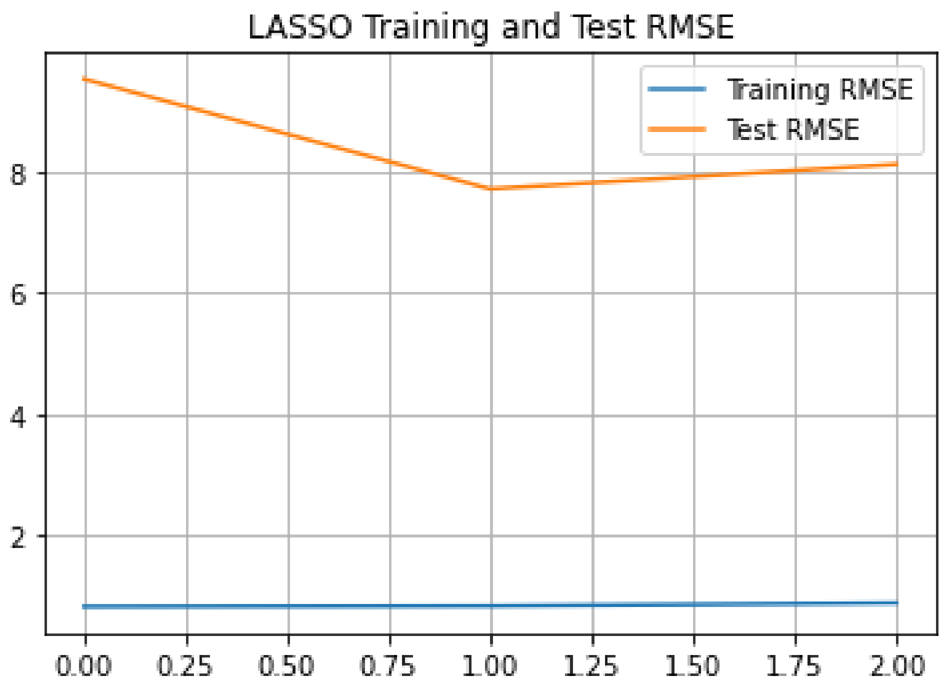
[16]:
```
## ORDINARY LEAST SQUARES
## Pick 3 sets of hyperparameters and learn each model (without cross␣
↪validation)
## Measure Train and Test RMSE and plot it on one plot
## Explain the plots and relate it to the theory studied in lectures

OLS_alphas = [0.01,0.05,0.1]
RMSE_train = []
RMSE_test = []

def calc_rmse(y, predictions):
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mse)
    return rmse

for i in OLS_alphas:
    model = linear_model.SGDRegressor(loss='squared_loss', penalty='none',␣
↪alpha=0, \
                                 shuffle=True, learning_rate='constant', eta0=i)
    model.fit(x_train,y_train)
    y_pred = model.predict(x_train)
    y_testpred = model.predict(x_test)
    RMSE_train.append(calc_rmse(y_train,y_pred))
    RMSE_test.append(calc_rmse(y_test,y_testpred))

plt.plot(OLS_alphas, RMSE_train, 'green', label='Training RMSE')
plt.plot(OLS_alphas, RMSE_test, 'blue', label='Test RMSE')
plt.title('OLS Training and Test RMSE')
plt.xlabel('OLS_alphas')
plt.ylabel('RMSE')
plt.legend()
plt.grid()
plt.show()

## Plot shows that as the learning rate increases, the RMSE increases as well
## The dramatic increase on test but not on training RMSE suggests the model is␣
↪overfitting for high learning rates
```



```
[18]: ## RIDGE REGRESSION
      ## Pick 3 sets of hyperparameters and learn each model (without cross␣
       ↪validation)
      ## Measure Train and Test RMSE and plot it on one plot
      ## Explain the plots and relate it to the theory studied in lectures

      ridge_alphas = [[0.001,0.001],[0.01,0.01],[0.05,0.05]]
      ridge_RMSE_train = []
      ridge_RMSE_test = []

      for rows,cols in ridge_alphas:
          ridge_model = linear_model.SGDRegressor(loss='squared_loss', penalty='l2',␣
       ↪alpha=cols, \
                                             shuffle=True, learning_rate='constant',␣
       ↪eta0=rows)
          ridge_model.fit(x_train,y_train)
          y_pred_ridge = ridge_model.predict(x_train)
          y_testpred_ridge = ridge_model.predict(x_test)
          ridge_RMSE_train.append(calc_rmse(y_train,y_pred_ridge))
          ridge_RMSE_test.append(calc_rmse(y_test,y_testpred_ridge))

      plt.plot(ridge_RMSE_train, label = 'Training RMSE')
      plt.plot(ridge_RMSE_test, label = 'Test RMSE')
      plt.title('Ridge Training and Test RMSE')
      plt.legend()
      plt.grid()
      plt.show()

      ## Here we can see that the gap between the training and test RMSE is much␣
       ↪larger
      ## This suggests that the regularization term in ridge regression contributes␣
       ↪more to overfitting
```

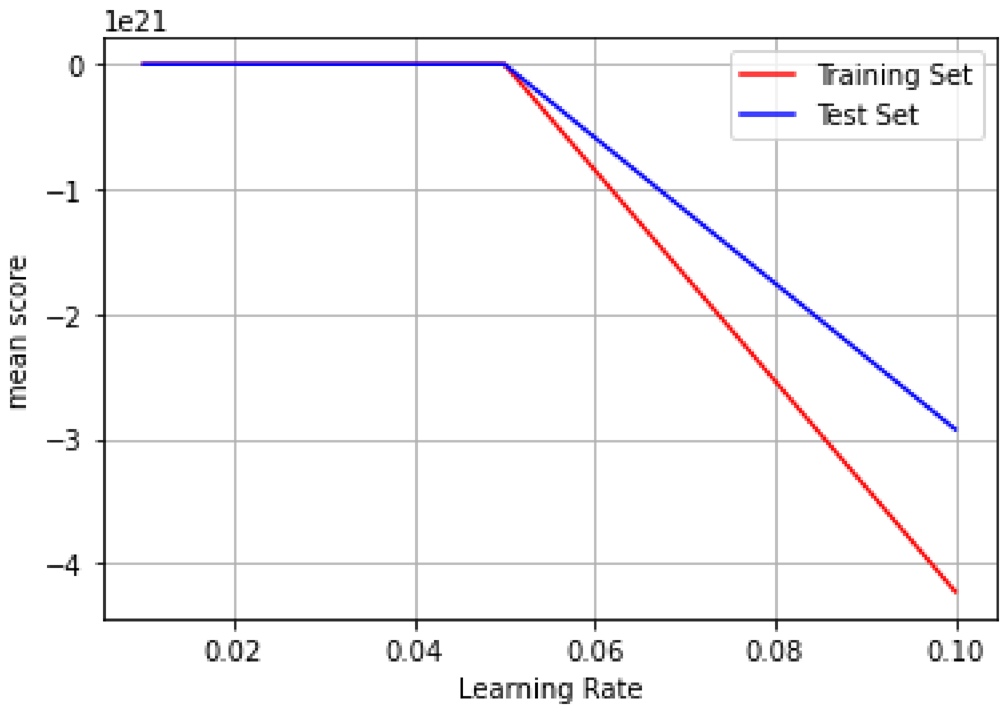


[19]:
```
## LASSO
## Pick 3 sets of hyperparameters and learn each model (without cross␣
 ↪validation)
## Measure Train and Test RMSE and plot it on one plot
## Explain the plots and relate it to the theory studied in lectures

lasso_alphas = [[0.001,0.001],[0.01,0.01],[0.05,0.05]]
lasso_RMSE_train = []
lasso_RMSE_test = []

for rows,cols in lasso_alphas:
    lasso_model = linear_model.SGDRegressor(loss='squared_loss', penalty='l1',␣
 ↪alpha=cols, \
                                 shuffle=True, learning_rate='constant',␣
↪eta0=rows)
    lasso_model.fit(x_train,y_train)
    y_pred_lasso = lasso_model.predict(x_train)
    y_testpred_lasso = lasso_model.predict(x_test)
    lasso_RMSE_train.append(calc_rmse(y_train,y_pred_lasso))
    lasso_RMSE_test.append(calc_rmse(y_test,y_testpred_lasso))

plt.plot(lasso_RMSE_train, label = 'Training RMSE')
plt.plot(lasso_RMSE_test, label = 'Test RMSE')
plt.title('LASSO Training and Test RMSE')
plt.legend()
plt.grid()
plt.show()

## Plot shows us something similar to the ridge regression
## Test RMSE is much higher due to the penalty fitting to the training data
```



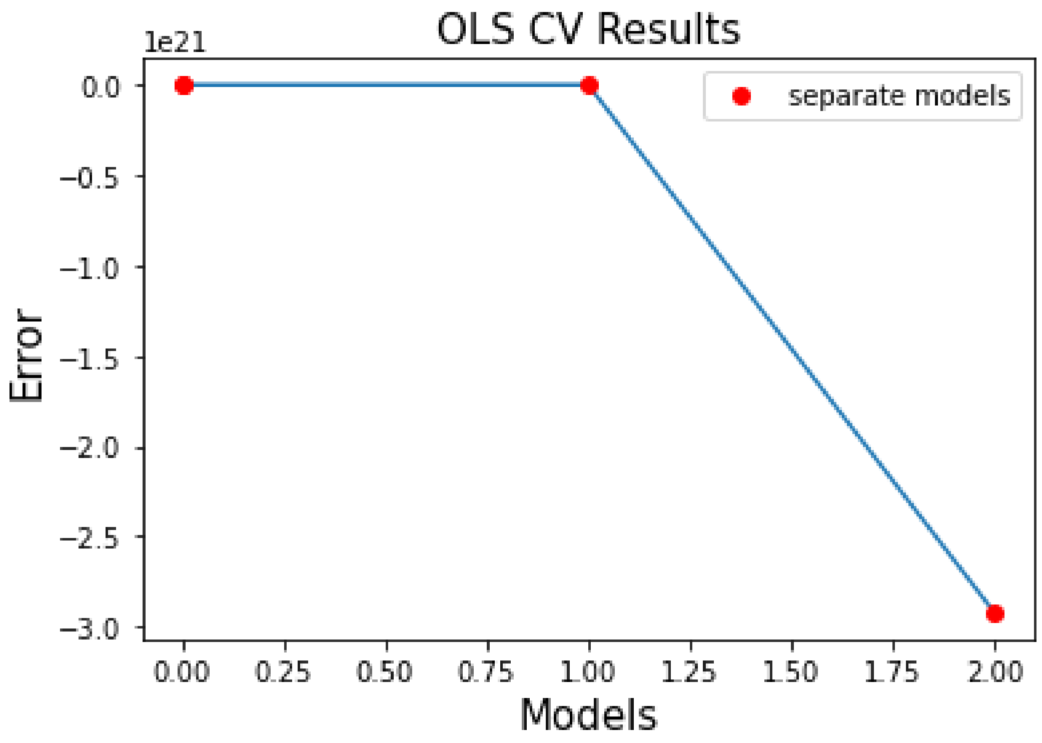
[20]:
```
## ORDINARY LEAST SQUARES
## Tune the hyperparameters using scikit learn GridSearchCV
## Plot the results of cross validation

hp = {'eta0':[0.01,0.05,0.1]}
```

```
model = linear_model.SGDRegressor(loss='squared_loss', penalty='none', alpha=0,␣
↪\
                                  shuffle=True, learning_rate='constant')
model_gridsearch = GridSearchCV(model, hp, cv=5, return_train_score=True)
model_gridsearch.fit(x_train,y_train)

plt.plot(hp['eta0'], model_gridsearch.cv_results_['mean_train_score'], 'red',␣
↪label='Training Set')
plt.plot(hp['eta0'], model_gridsearch.cv_results_['mean_test_score'],'blue',␣
↪label='Test Set')
plt.xlabel('Learning Rate')
plt.ylabel('mean score')
plt.legend()
plt.grid()
plt.show()
```



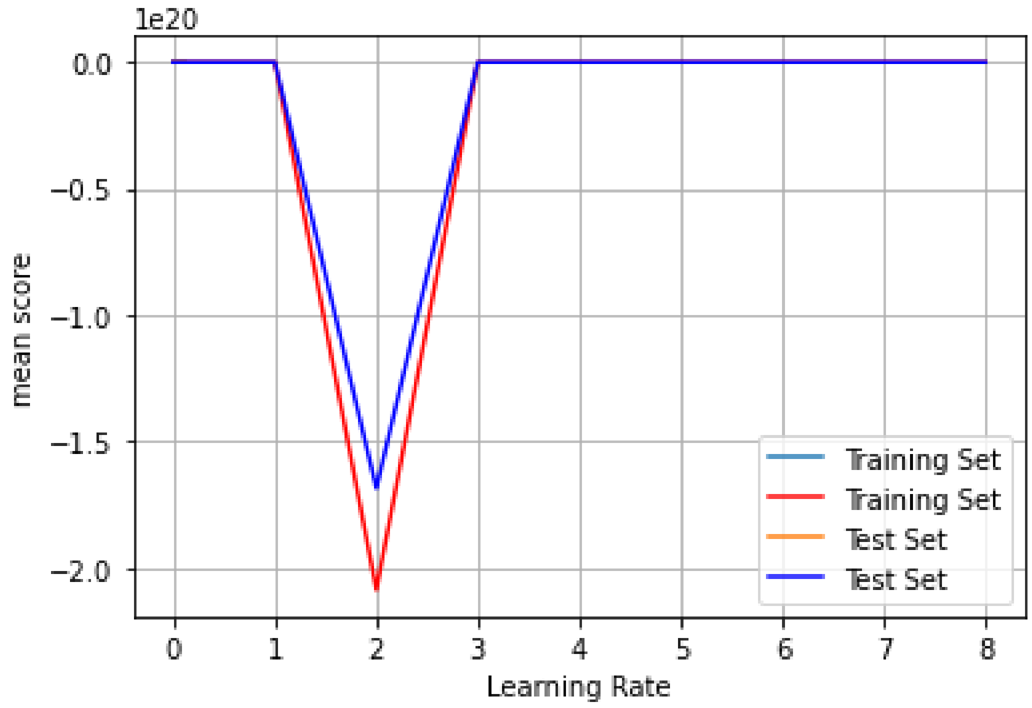
[21]:
```
## ORDINARY LEAST SQUARES
## Cross validation

model = linear_model.SGDRegressor(loss='squared_loss', penalty='none', alpha=0,␣
↪\
                                  shuffle=True, learning_rate='constant',␣
↪eta0=model_gridsearch.best_params_['eta0'])
```

```
model.fit(x_train,y_train)
cv_train = cross_val_score(model,x_train,y_train,cv=4)
cv_test = cross_val_score(model,x_test,y_test,cv=4)

plt.title("OLS CV Results", fontsize = 15)
plt.plot(model_gridsearch.cv_results_["mean_test_score"])
plt.plot(model_gridsearch.cv_results_["mean_test_score"], "ro", label =␣
 ↪"separate models")
plt.plot(model_gridsearch.best_score_, "ro")
plt.xlabel('Models', fontsize = 15)
plt.ylabel('Error', fontsize = 15)
plt.legend()
plt.show()
```



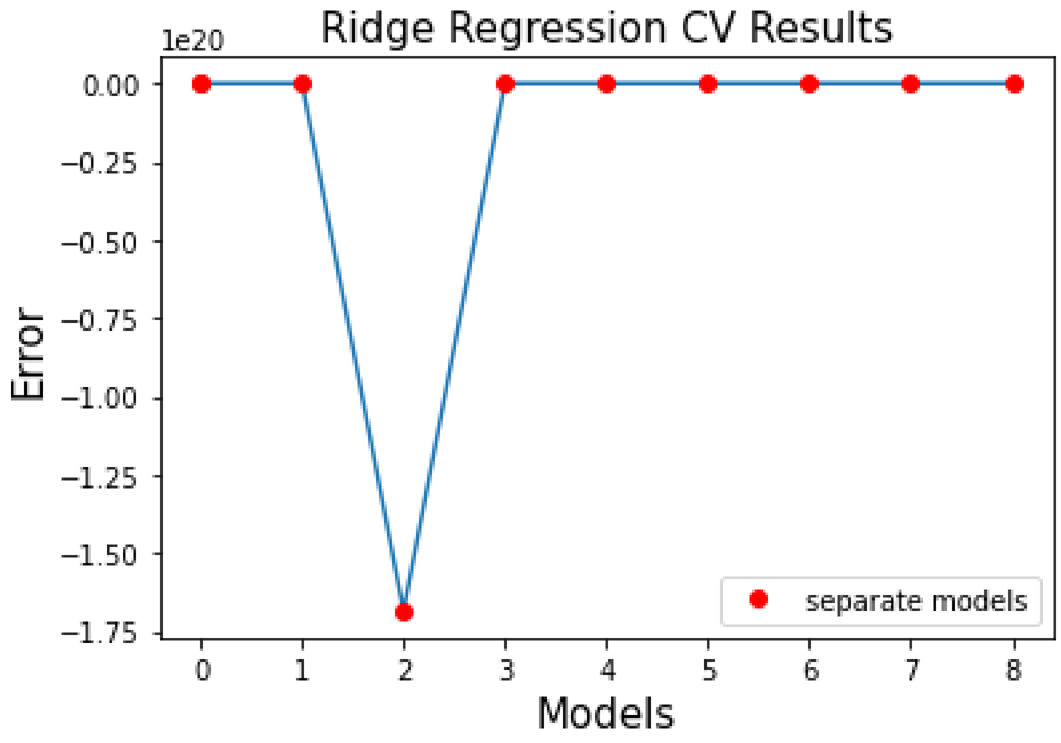
[22]:
```
## RIDGE REGRESSION
## Tune the hyperparameters using scikit learn GridSearchCV
## Plot the results of cross validation for each model

hp = {'eta0':[0.01,0.05,0.1],'alpha':[0.01,0.05,0.1]}
ridge = linear_model.SGDRegressor(loss='squared_loss', penalty='l2', \
                                  shuffle=True, learning_rate='constant')
ridge_gridsearch = GridSearchCV(ridge, hp, cv=5, return_train_score=True)
```

```
ridge_gridsearch.fit(x_train,y_train)

plt.plot(hp['eta0'], hp['alpha'], ridge_gridsearch.
 ↪cv_results_['mean_train_score'], 'red', label='Training Set')
plt.plot( hp['eta0'], hp['alpha'], ridge_gridsearch.
 ↪cv_results_['mean_test_score'],'blue', label='Test Set')
plt.xlabel('Learning Rate')
plt.ylabel('mean score')
plt.legend()
plt.grid()
plt.show()
```



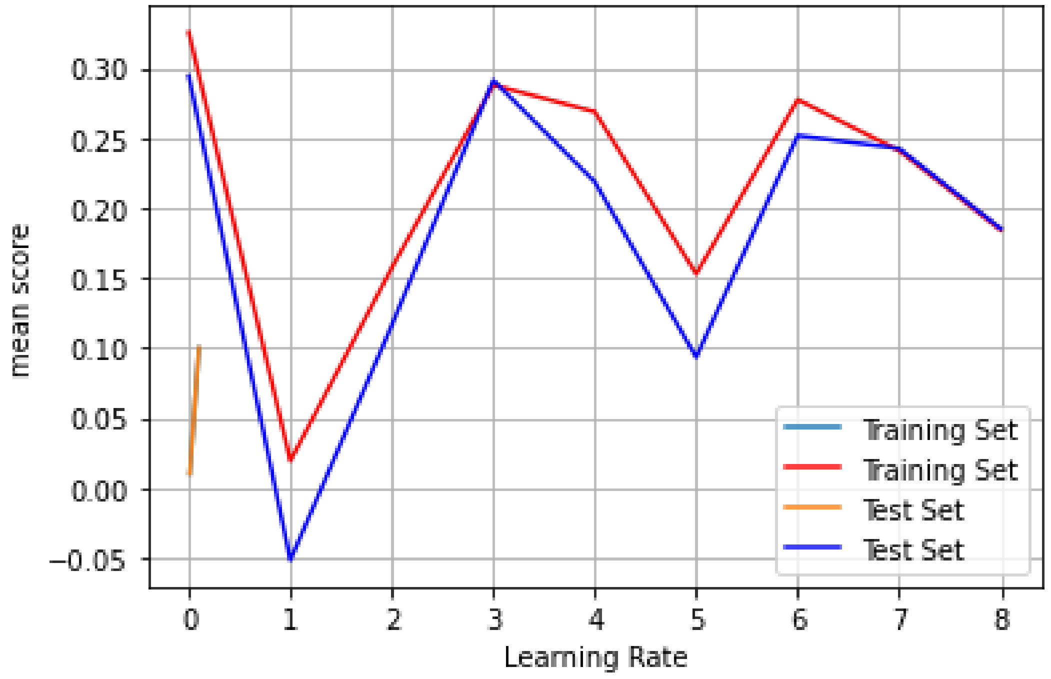
[23]:
```
## RIDGE REGRESSION
## Cross validation

ridge = linear_model.SGDRegressor(loss='squared_loss', penalty='l2',␣
 ↪alpha=ridge_gridsearch.best_params_['alpha'], \
                                  shuffle=True, learning_rate='constant',␣
 ↪eta0=ridge_gridsearch.best_params_['eta0'])
ridge.fit(x_train,y_train)
ridge_cv_train = cross_val_score(ridge,x_train,y_train,cv=4)
ridge_cv_test = cross_val_score(ridge,x_test,y_test,cv=4)
```

```
plt.title("Ridge Regression CV Results", fontsize = 15)
plt.plot(ridge_gridsearch.cv_results_["mean_test_score"])
plt.plot(ridge_gridsearch.cv_results_["mean_test_score"], "ro", label =␣
 ↪"separate models")
plt.plot(ridge_gridsearch.best_score_, "ro")
plt.xlabel('Models', fontsize = 15)
plt.ylabel('Error', fontsize = 15)
plt.legend()
plt.show()
```

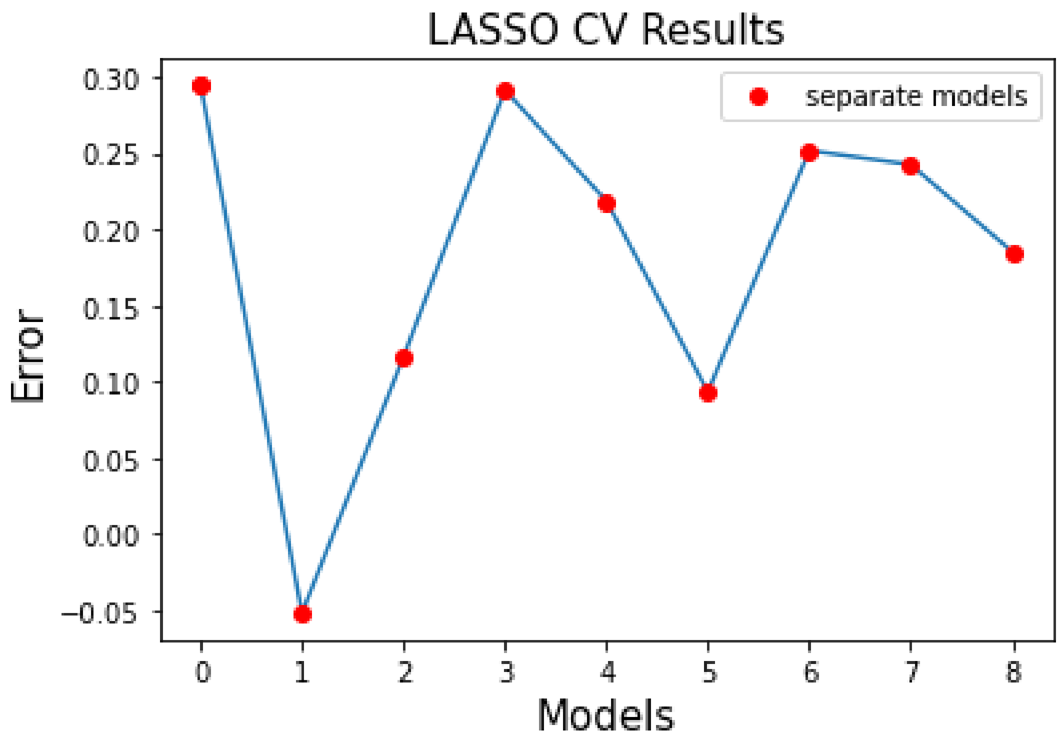


[27]:
```
## LASSO
## Tune the hyperparameters using scikit learn GridSearchCV
## Plot the results of cross validation for each model

hp = {'eta0':[0.01,0.05,0.1],'alpha':[0.01,0.05,0.1]}
lasso = linear_model.SGDRegressor(loss='squared_loss', penalty='l1', \
                                  shuffle=True, learning_rate='constant')
lasso_gridsearch = GridSearchCV(lasso, hp, cv=5, return_train_score=True)
lasso_gridsearch.fit(x_train,y_train)

plt.plot(hp['eta0'], hp['alpha'], lasso_gridsearch.
 ↪cv_results_['mean_train_score'], 'red', label='Training Set')
```

```
plt.plot( hp['eta0'], hp['alpha'], lasso_gridsearch.
 ↪cv_results_['mean_test_score'],'blue', label='Test Set')
plt.xlabel('Learning Rate')
plt.ylabel('mean score')
plt.legend()
plt.grid()
plt.show()
```



[28]:
```
## LASSO
## Cross validation

lasso = linear_model.SGDRegressor(loss='squared_loss', penalty='l1',␣
 ↪alpha=lasso_gridsearch.best_params_['alpha'], \
                                    shuffle=True, learning_rate='constant',␣
 ↪eta0=lasso_gridsearch.best_params_['eta0'])
lasso.fit(x_train,y_train)
lasso_cv_train = cross_val_score(lasso,x_train,y_train,cv=4)
lasso_cv_test = cross_val_score(lasso,x_test,y_test,cv=4)

plt.title("LASSO CV Results", fontsize = 15)
plt.plot(lasso_gridsearch.cv_results_["mean_test_score"])
plt.plot(lasso_gridsearch.cv_results_["mean_test_score"], "ro", label =␣
 ↪"separate models")
plt.plot(lasso_gridsearch.best_score_, "ro")
plt.xlabel('Models', fontsize = 15)
plt.ylabel('Error', fontsize = 15)
plt.legend()
plt.show()
```



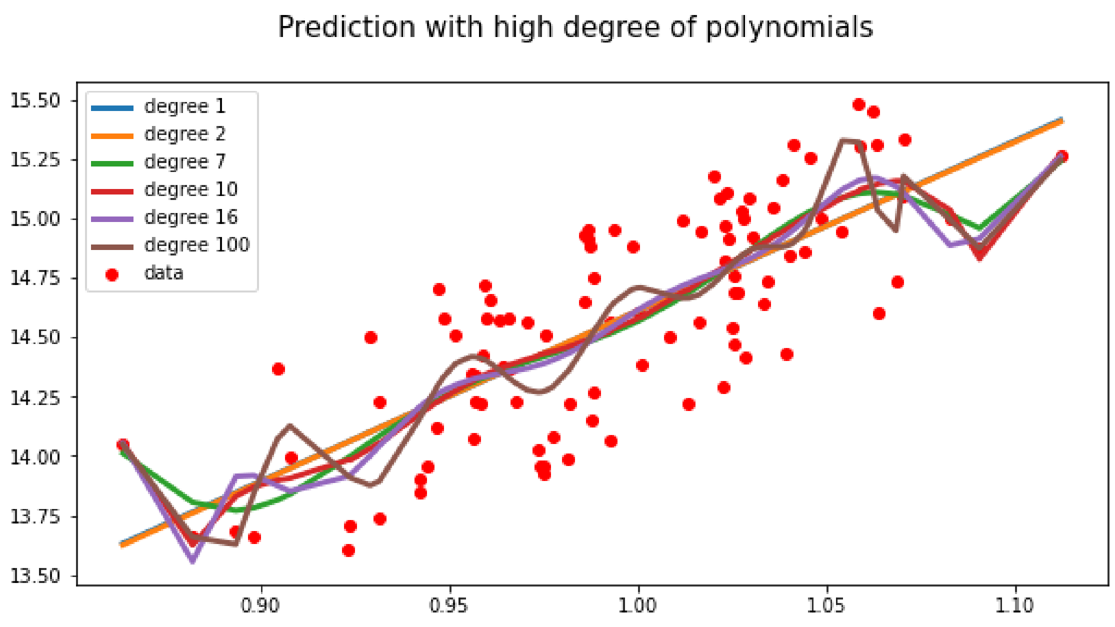
[30]:
```
## Using the optimal hyperparameter you have to evaluate each model on the Test␣
 ↪Set
##Report the results in a meaningful manner

print(f"OLS best score is {model_gridsearch.best_score_}")
print(f"Ridge Regression best score is {ridge_gridsearch.best_score_}")
print(f"LASSO best score is {lasso_gridsearch.best_score_}")

## Based on these we see that LASSO is the best model based on scores generated␣
 ↪by grid search cv
```

```
OLS best score is 0.29090210832334124
Ridge Regression best score is 0.2840577158597672
LASSO best score is 0.294460661890987
```

3 EXERCISE 2

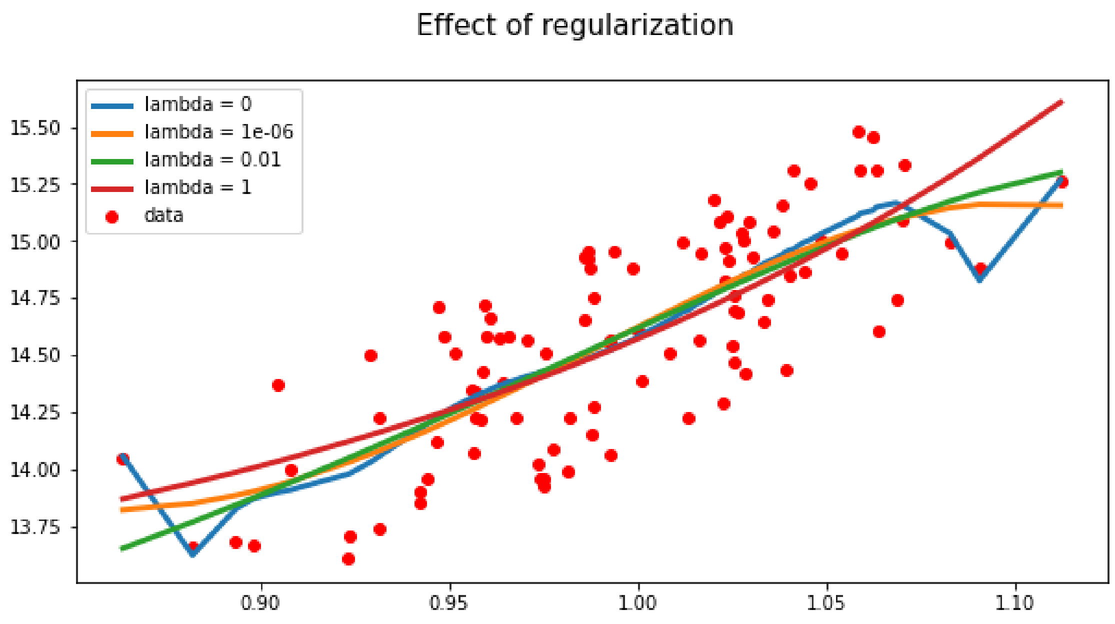
3.1 Higher Order Polynomial Regression

[68]:
```
## Use higher degrees of polynomial feature for your data i.e. degrees 1, 2, 7,␣
↪10, 16 and 100
## TASK A
## For each newly created dataset learn LinearRegression
## Plot the predicted curves for each dataset

## Plot for original data points
fig, axs = plt.subplots(1,1,figsize=(10,5))
axs.scatter(D1_x, D1_y, color='red', label="data")
axs.legend()

degrees = [1, 2, 7, 10, 16,100]

## Plot for models with different polynomial features
for i in degrees:
    poly = PolynomialFeatures(degree=i)
    poly_x = poly.fit_transform(D1_x)
    linreg = LinearRegression().fit(poly_x, D1_y)
    y_pred = linreg.predict(poly_x)
    concat = zip(*sorted(zip(D1_x,y_pred)))
    x,y = concat
    axs.plot(x, y, label=(f"degree {i}"), linewidth=3)
axs.legend()
plt.suptitle('Prediction with high degree of polynomials',fontsize=15)
plt.show()

## In this plot we see that the higher the polynomial features, the more the␣
↪line fits to the data points
## This is why models with high degrees of freedom tend to overfit the data
## Lines with too little degrees tend to underfit the data due to␣
↪oversimplification of the relationship
## Therefore we need to find a good balance between the two 09tyty
```



[69]:
```
## TASK B
## Fix the degree of polynomial to 10
## Pick 4 values of   (regularization constant) and learn Ridge Regression
## Plot the predicted curves for each dataset
## Explain the phenomena you observed for different prediction curves

## Plot for original data points
fig, axs = plt.subplots(1,1,figsize=(10,5))
axs.scatter(D1_x, D1_y, color='red', label="data")
axs.legend()

## Lambda values given in problem
lambdas = [0, (10**-6), (10**-2), 1]

for i in lambdas:
    poly = PolynomialFeatures(degree=10)
    poly_x = poly.fit_transform(D1_x)
    ridge = Ridge(alpha=i).fit(poly_x, D1_y)
    y_pred = ridge.predict(poly_x)
    concat = zip(*sorted(zip(D1_x,y_pred)))
    x,y = concat
    axs.plot(x, y, label=(f"lambda = {i}"), linewidth=3)
axs.legend()
plt.suptitle('Effect of regularization',fontsize=15)
plt.show()
```

```
## Using regularization we see that there is much less likelihood of␣
↪overfitting in this case
## When lambda = 0 this represents no regularization and we see that it is more␣
↪prone to being skewed by noise in data
```



4 EXERCISE 3

4.1 Implementing Coordinate Descent

[ ]:
```
## Implement the Coordinate Descent algorithm
## Maintain a history of your   values
## After training plot them against iterations in a single plot

## Reference: https://xavierbourretsicotte.github.io/lasso_implementation.
↪html#Implementing-coordinate-descent-for-lasso-regression-in-Python
def coordinate_descent(x,y,iters):
    m,n = x.shape
    theta = np.zeros((n,1))
    theta_list = []
    for i in range(iter):
        for j in range(len(theta)):
            xtheta = np.dot(x,theta)
            update = ((y-xtheta).T).dot(x[:,j]) / (x[:,j].T).dot(x[:,j])
            theta_list.append(update)
        theta[i+1] = theta[j]
    return theta, theta_list
```

```
[77]: ##Implement CD with L1 regularization
      ## Maintain a history of your  values
      ## After training plot them against iterations in a single plot

      ## Soft threshold constitutes part of the closed-form LASSO solution using L1␣
       ↪regularization
      def soft_threshold(rho,lamda):
          if rho < - lamda:
              return (rho + lamda)
          elif rho >  lamda:
              return (rho - lamda)
          else:
              return 0

      def coordinate_descent_lasso(x,y,lamda = .01, num_iters=100):
          m, n = x.shape
          theta = np.ones((n,1))
          theta_list = []
          for i in range(num_iters):
              for j in range(n):
                  x_j = x[:,j].reshape(-1,1)
                  y_pred = x @ theta
                  rho = x_j.T @ (y - y_pred  + theta[j]*x_j)
                  theta[j] =  soft_threshold(rho, lamda)
          return theta.flatten()

      m,n = x_train.shape
      initial_theta = np.ones((n,1))
      theta_list = list()

      theta = coordinate_descent_lasso(x_train,y_train, num_iters=100)
      theta_list.append(theta)

      #Stack into numpy array
      theta_lasso = np.stack(theta_list).T

      #Plot results
      n,_ = theta_lasso.shape
      plt.figure(figsize = (12,8))

      for i in range(n):
          plt.plot(lamda, theta_lasso[i])

      plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('Lasso Paths - Numpy implementation')
plt.legend()
plt.axis('tight')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-77-f0dc1e433d80> in <module>
     28 theta_list = list()
     29
---> 30 theta = coordinate_descent_lasso(x_train,y_train, num_iters=100)
     31 theta_list.append(theta)
     32

<ipython-input-77-f0dc1e433d80> in coordinate_descent_lasso(x, y, lamda,␣
 ↪num_iters)
     21             y_pred = x @ theta
     22             rho = x_j.T @ (y - y_pred  + theta[j]*x_j)
---> 23             theta[j] =  soft_threshold(rho, lamda)
     24     return theta.flatten()
     25

ValueError: could not broadcast input array from shape (1279,) into shape (1,)
```

[ ]:
```
## Compare the plots of the unregularized and regularized CD
## Highlight the difference
## What information can be inferred from these values?
```