MLChem Lab1 20210929

1. Basics on Python

Useful packages:

- **numpy**
- scipy
- **scikit-learn**
- pandas
- **matplotlib**/seaborn/lets_plot
- **jupyter notebook/lab**

For Deep Learning:

- **pytorch**
- tensorflow
- jax with haiku

For Reinforcement Learning:

- **gym**

1.1 Hello, world!

```
print("Hello, world!")
```

```
Hello, world!
```

1.2 Test package availability

```
import numpy as np
import scipy
import sklearn
import matplotlib.pyplot as plt
import pandas as pd
import torch
```

1.3 Useful websites

- Stack overflow https://stackoverflow.com/ for troubleshooting
- Documents (You can find them with Google/Bing easily.)
    - numpy: https://numpy.org/doc/stable/
    - pandas: https://pandas.pydata.org/docs/
    - scipy: https://docs.scipy.org/doc/scipy/reference/index.html
    - scikit-learn: https://scikit-learn.org/stable/modules/classes.html
    - matplotlib: https://matplotlib.org/stable/contents.html
- Google CoLab: https://colab.research.google.com/
- Google machine learning crash course: https://developers.google.com/machine-learning/crash-course/ml-intro
- 机器学习白板推导系列：https://www.bilibili.com/video/BV1aE411o7qd by shuhuai008

1.4 About Assignments

Your code for every Lab assignment should be encapsulated in a Python class with required methods and necessary annotation.

```
#!/usr/bin/env python
#coding=utf-8

"""
Author:        Fanchong Jian, Pengbo Song
Created Date:  2021/09/28
Last Modified: 2021/09/29
"""

from warnings import warn
from sklearn import linear_model

class MLChemLab1(object):
    """Template class for Lab1 -*- Linear Regression -*-

    Properties:
        model: Linear model to fit.
        featurization_mode: Keyword to choose feature methods.
    """

    def __init__(self):
        """Initialize class with empty model and default featurization mode to
identical"""
        self.model = None
        self.featurization_mode = "identical"

    def fit(self, x, y, featurization_mode : str):
        """Feature input X using given mode and fit model with featurized X and
y

        Args:
            x: Input X data.
            y: Input y data.
            featurization_mode: Keyword to choose feature methods.

        Returns:
            Trained model using given X and y, or None if no model is specified
        """

        # Catch empty model, a model should be added earlier
        if (self.model is None):
            warn("No model to fit. Nothing Returned.")
            return None
        self.featurization_mode = featurization_mode
        featurized_x = self.featurization(x)
        self.model.fit(featurized_x, y)
        return self.model

    def add_model(self, model : str, **kwargs):
        """Add model before fitting and prediction"""
        if model == "ridge":
            # Put your Ridge Regression model HERE
```

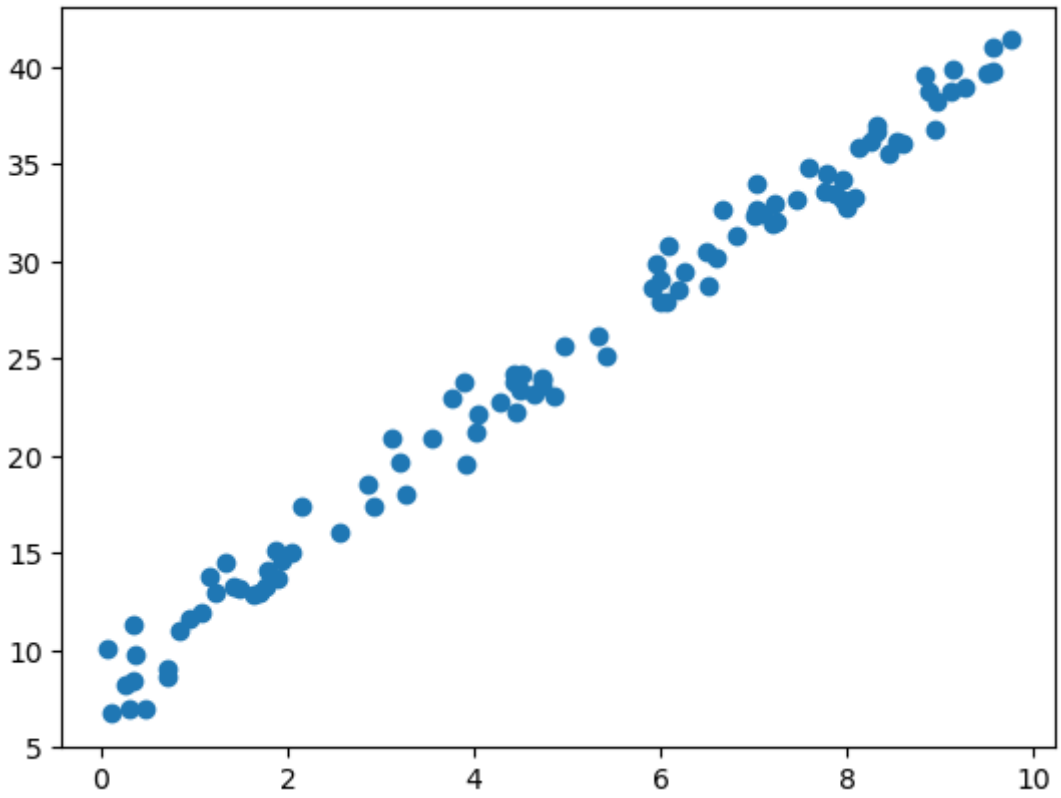
```
            self.model = ...
        elif model == "lasso":
            # Put your Lasso Regression model HERE
            self.model = ...
        elif model == "naive":
            # Put your naive model HERE
            self.model = ...
        else:
            # Catch incorrect keywords
            raise NotImplementedError("Got incorrect model keyword " + model)

    def featurization(self, x):
        """Feature input X data using preset mode"""
        if self.featurization_mode == "poly":
            # Put your polynomial featurization code HERE
            return ...
        elif self.featurization_mode == "poly-cos":
            # Put your polynomial cosine featurization code HERE
            return ...
        elif self.featurization_mode == "identical":
            # Do nothing, returns raw X data
            return x

    def predict(self, x):
        """Predict based on fitted model and given X data"""
        x = self.featurization(x)
        y_predict = self.model.predict(x)
        return y_predict

    def evaluation(self, y_predict, y_label, metric : str):
        """Eavluate training results based on predicted y and true y"""
        if metric == "RMS":
            # Returns RMS value
            return ...
        else:
            # Catch incorrect keywords
            raise NotImplementedError("Got incorrect metric keyword " + model)
```

2. Linear Regression

2.1 Simple least square

```
x = np.random.random(100) * 10.0 # generate 100 random numbers in [0,10]
noise = np.random.normal(0, 1, 100) # normally distributed noise, with mean=0,
sd=1, 100 dimensions
# or: noise = np.random.randn(100)
y = 3.4 * x + 8.0 + noise # target Y

print(f"X shape = {x.shape}")
print(f"y shape = {y.shape}")

plt.scatter(x, y)
plt.show()
```

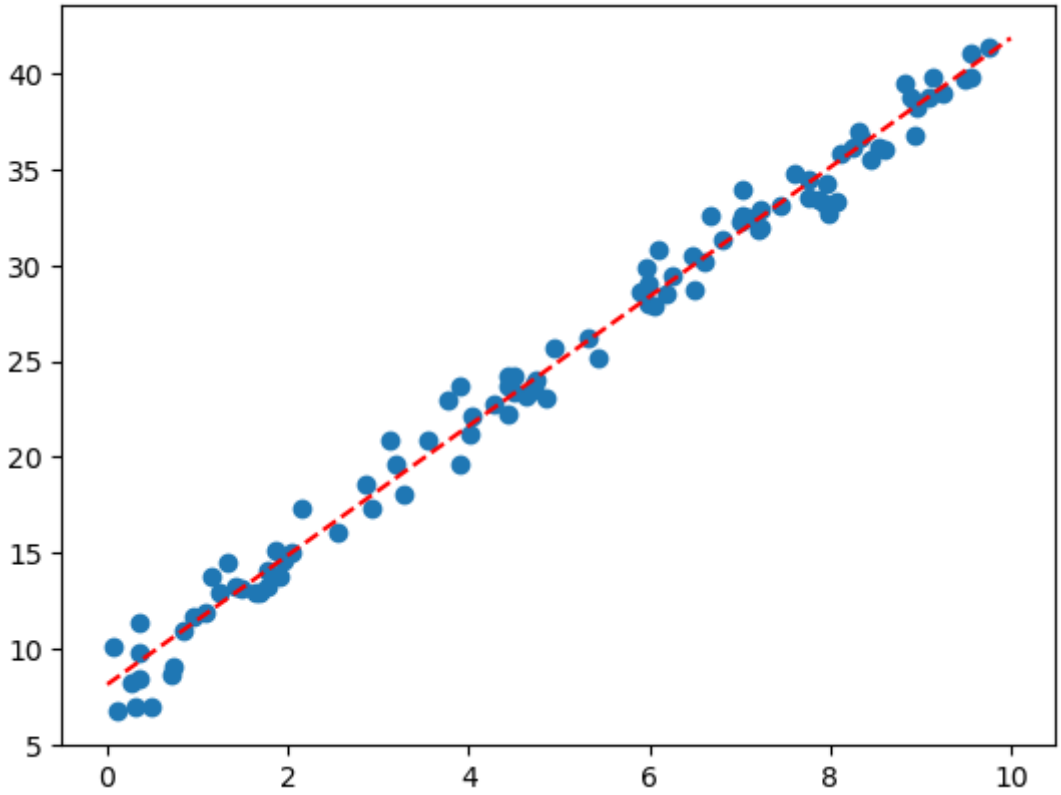
```
X shape = (100,)
y shape = (100,)
```



We can build a linear model with scikit-learn.

```
from sklearn import linear_model

x = x.reshape(len(x), 1) # we need [n, 1] matrix for input, instead of [n] array
print(f"X shape = {x.shape}")
print(f"y shape = {y.shape}")

reg = linear_model.LinearRegression() # model initiation
reg.fit(x.reshape(len(x),1),y) # model training

print(f"Linear Regression coefficients = {reg.coef_}")
print(f"Linear Regression intercept = {reg.intercept_:>.5f}")

k, b = (reg.coef_[0], reg.intercept_) # y=kx+b
x_ = np.linspace(0, 10, 100) # evenly spaced X, for plotting
y_predict = k*x_ + b # make prediction
plt.scatter(x, y)
plt.plot(x_, y_predict, "r--")
plt.show()
```

```
X shape = (100, 1)
y shape = (100,)
Linear Regression coefficients = [3.37037534]
Linear Regression intercept = 8.14620
```



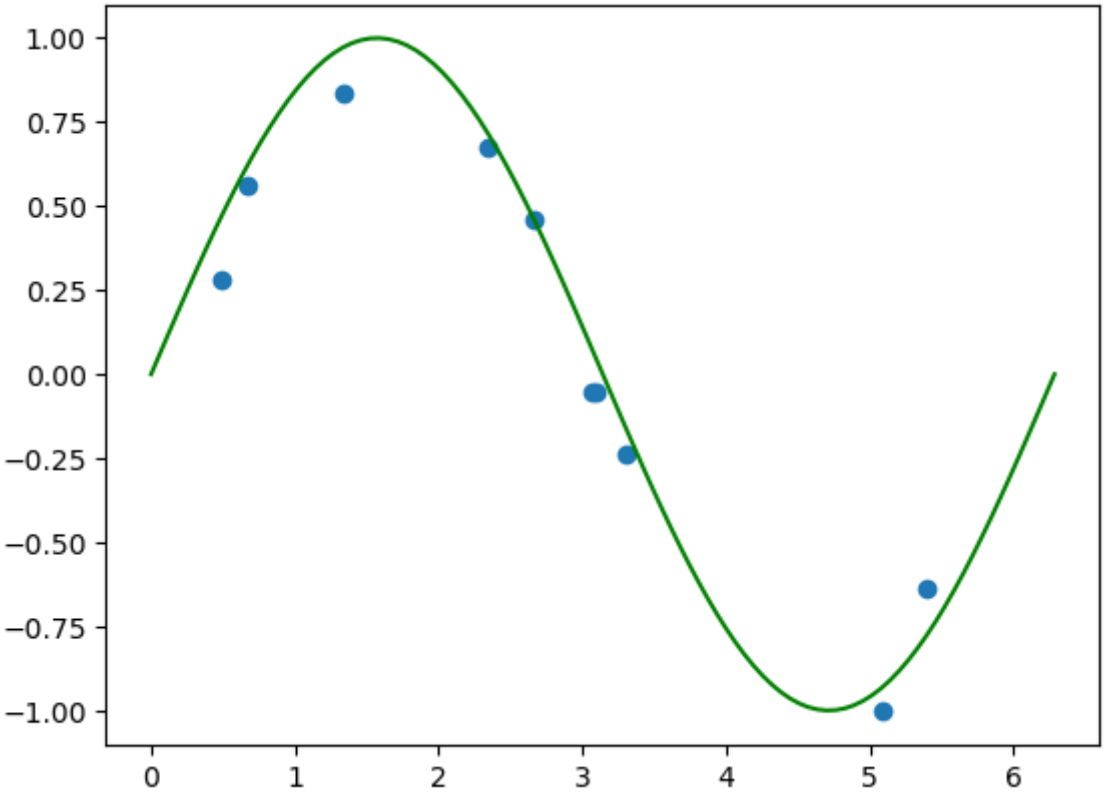
2.2 Ridge regression

Optimizing target is:

$$\min_{w} ||Xw - y||_2^2 + \alpha||w||_2^2$$

Let's try fitting a sine function with a polynomial model using this L2 regularization!

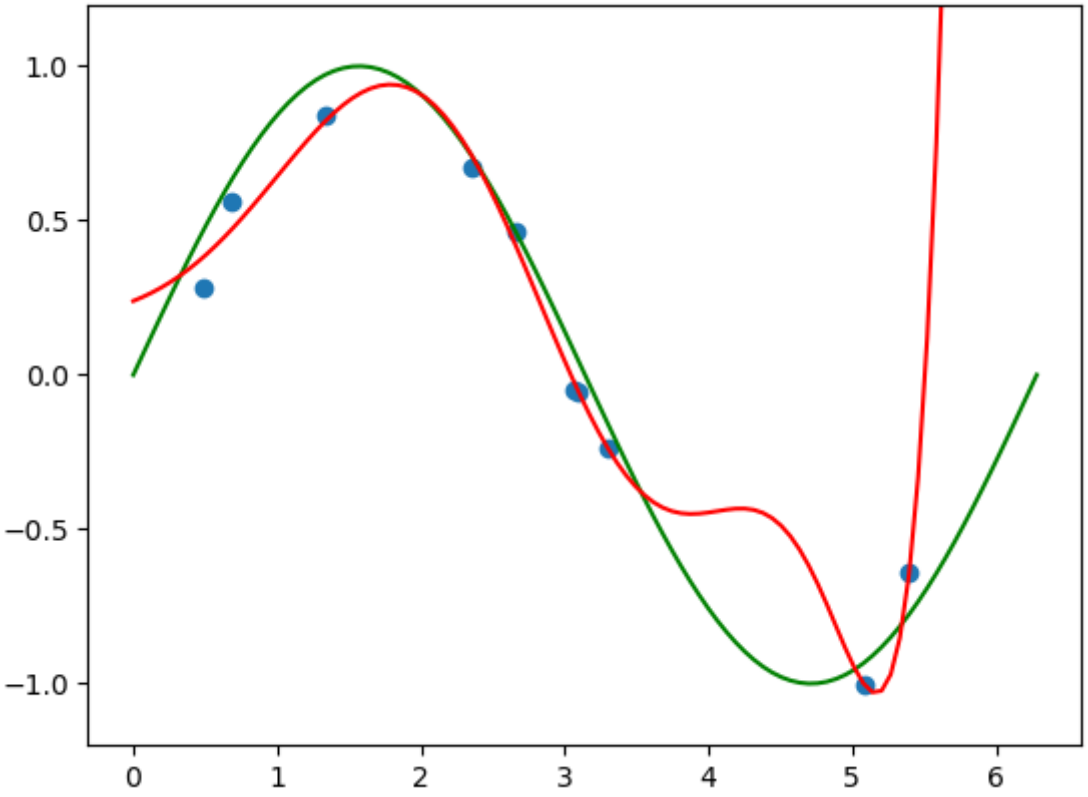
```
# data generation
N_samples = 10

x = np.random.random(N_samples)*2*np.pi # random x

noise = np.random.randn(N_samples) / 10 # normally distributed random noise
y = np.sin(x)+noise # target Y

# ground truth curve plotting
x_ = np.linspace(0, 2*np.pi, 100)
y_ = np.sin(x_)
plt.scatter(x, y)
plt.plot(x_, y_, color = "green")
plt.show()
```




```
deg = 9 # 9th-degree polynomial

x_multi = np.power(x.reshape(N_samples,1), np.linspace(1,deg,deg)) # get
polynomial features
### alternatively: ###
# from sklearn import PolynomialFeatures
# poly_features = PolynomialFeatures(degree=deg, include_bias=False)
# x_multi = poly_features.fit_transform(x.reshape(-1, 1))
"""
With deg=9 and include_bias=False, x_multi will be transformed to 9-feature data
series (without constant term)
Got train_X like this:
[[X0, X0^2, X0^3, ..., X0^9],
 [X1, X1^2, X1^3, ..., X1^9],
 ...
 [Xn-1, Xn-1^2, Xn-1^3, ..., Xn-1^9]]
"""

print(f"X shape = {x.shape}")
print(f"Featurized X shape = {x_multi.shape}")
print(f"y shape = {y.shape}")

print('='*30)
print(f"First 3 values of X: ", x[0:3,])
print(f"First 3 rows of featurized X: ", x_multi[0:3,])
print('='*30)

reg = linear_model.Ridge(alpha=0.1) # model initiation
reg.fit(x_multi, y) # training
```

```python
print(f"Ridge Regression coefficients = {reg.coef_}")
print(f"Ridge Regression intercept = {reg.intercept_:>.5f}")

# polynomial features for evenly spaced X
x_multi_ = np.power(x_.reshape(100,1), np.linspace(1,deg,deg))
# make prediction
y_predict = reg.predict(x_multi_)
plt.scatter(x, y)
plt.plot(x_, y_, color = "green")
plt.plot(x_, y_predict, color = "red")
plt.ylim(-1.2, 1.2)
plt.show()
```

```
X shape = (10,)
Featurized X shape = (10, 9)
y shape = (10,)
================================
First 3 values of X:  [2.66376221 0.67693395 5.08493877]
First 3 rows of featurized X:  [[2.66376221e+00 7.09562909e+00 1.89010686e+01
5.03479522e+01
  1.34114972e+02 3.57250394e+02 9.51630097e+02 2.53491629e+03
  6.75241420e+03]
 [6.76933946e-01 4.58239567e-01 3.10197918e-01 2.09983501e-01
  1.42144960e-01 9.62227486e-02 6.51364449e-02 4.40930707e-02
  2.98480963e-02]
 [5.08493877e+00 2.58566023e+01 1.31479240e+02 6.68563884e+02
  3.39960642e+03 1.72867905e+04 8.79022712e+04 4.46977667e+05
  2.27285407e+06]]
================================
Ridge Regression coefficients = [ 0.18541059  0.18114404  0.09302828 -0.00852257
-0.05215546 -0.00563928
  0.01319642 -0.00315807  0.0002257 ]
Ridge Regression intercept = 0.23929
```



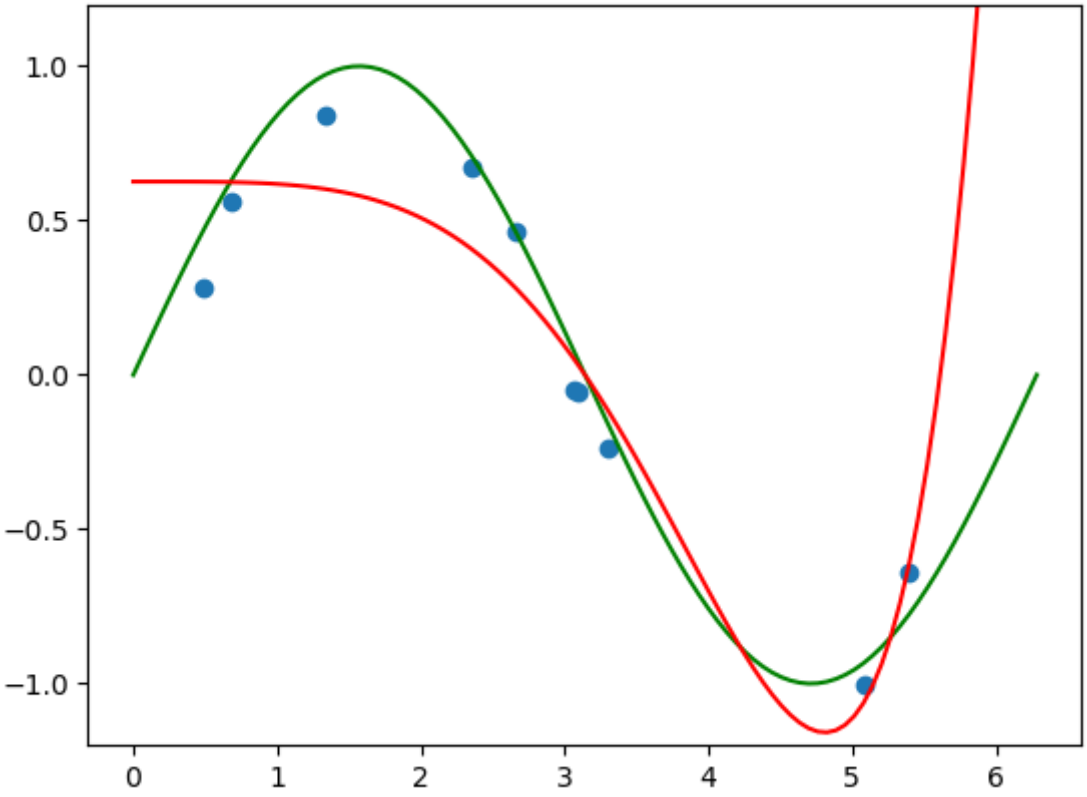
2.3 Lasso Regression

Similarly, we can build a model with L1 regularization for the same problem.
In `scikit-learn`, the optimizing target for `sklearn.linear_model.Lasso` is:

$$\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha||w||_1$$

```
reg = linear_model.Lasso(alpha=0.1) # model initiation
reg.fit(x_multi, y) # training with the same dataset
print(f"Lasso Regression coefficients = {reg.coef_}")
print(f"Lasso Regression intercept = {reg.intercept_:>.5f}")

# polynomial features for evenly spaced X
x_multi_ = np.power(x_.reshape(100,1), np.linspace(1,deg,deg))
# make prediction
y_predict = reg.predict(x_multi_)
plt.scatter(x, y)
plt.plot(x_, y_, color = "green")
plt.plot(x_, y_predict, color = "red")
plt.ylim(-1.2, 1.2)
plt.show()
```

```
Lasso Regression coefficients = [ 0.00000000e+00  0.00000000e+00  0.00000000e+00
-7.76020699e-03
 -2.77162094e-06  7.59929618e-05  1.23006882e-05  1.58414993e-06
  1.80813863e-07]
Lasso Regression intercept = 0.62604
```


1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6