L07_regulizers

Regulizers

Qa The Penalty Factor

Now, lets examine what $||\mathbf{w}||_2^2$ effectivly mean? It is composed of our well-known $\mathcal{L}_2^2$ norm and can also be expressed as

$$||\mathbf{w}||_2^2 = \mathbf{w}^\top \mathbf{w}$$

Construct a penaltiy function that implements $\mathbf{w}^\top \mathbf{w}$, re-using any functions from `numpy` (implementation could be a tiny *one-liner*).

Take w_0 into account, this weight factor should NOT be included in the norm. Also checkup on `numpy`s `dot` implementation, if you have not done so and are using it: it is a typical pythonic *combo* function, doing both dot op's (inner product) and matrix multiplication (outer product) dependent on the shape of the input parameters.

Then run it on the three test vectors below, and explain when the penalty factor is low and when it is high.

Qa Implementation

In [1]:
```
# Assignment Qa
import warnings
warnings.filterwarnings('ignore')

import numpy as np
from libitmal import utils as itmalutils

def Omega(w):
    return np.dot(w[1:].T, w[1:]) # w[1:] t exclude the w0 bias

# weight vector format: [w_0 w_1 .. w_d], ie. elem. 0 is the 'bias'
w_a = np.array([1, 2, -3]) #
w_b = np.array([1E10, -3E10])
w_c = np.array([0.1, 0.2, -0.3, 0])

p_a = Omega(w_a)
p_b = Omega(w_b)
p_c = Omega(w_c)

print(f"P(w0)={p_a}")
print(f"P(w1)={p_b}")
print(f"P(w2)={p_c}")

# TEST VECTORS
e0 = 2*2+(-3)*(-3)
e1 = 9e+20
e2 = 0.13
itmalutils.AssertInRange(1.0*p_a,1.0*e0)
itmalutils.AssertInRange(1.0*p_b,1.0*e1,eps=1E5)
itmalutils.AssertInRange(1.0*p_c,1.0*e2)
```

```
P(w0)=13
P(w1)=9e+20
P(w2)=0.13
```

Qa Results

The penalty factor is high when the regularization of the of a linear regression model. Thus if the penallty factor increases, it improves the conditioning and redcues the variance of the output.
For a Ridge regularization, it would mean that a high penalty factor would lead to fitting the data while keeping the model weights as low as possible.

Adding Regularization for Linear Regression Models

Adding the penalty $\alpha||\mathbf{w}||_2^2$ actually corresponds to the Scikit-learn model `sklearn.linear_model.Ridge` and there are, as usual, a bewildering array of regulized models to choose from in Scikit-learn with exotic names like `Lasso` and `Lars`

> https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)

Lets just examine `Ridge`, `Lasso` and `ElasticNet` here.

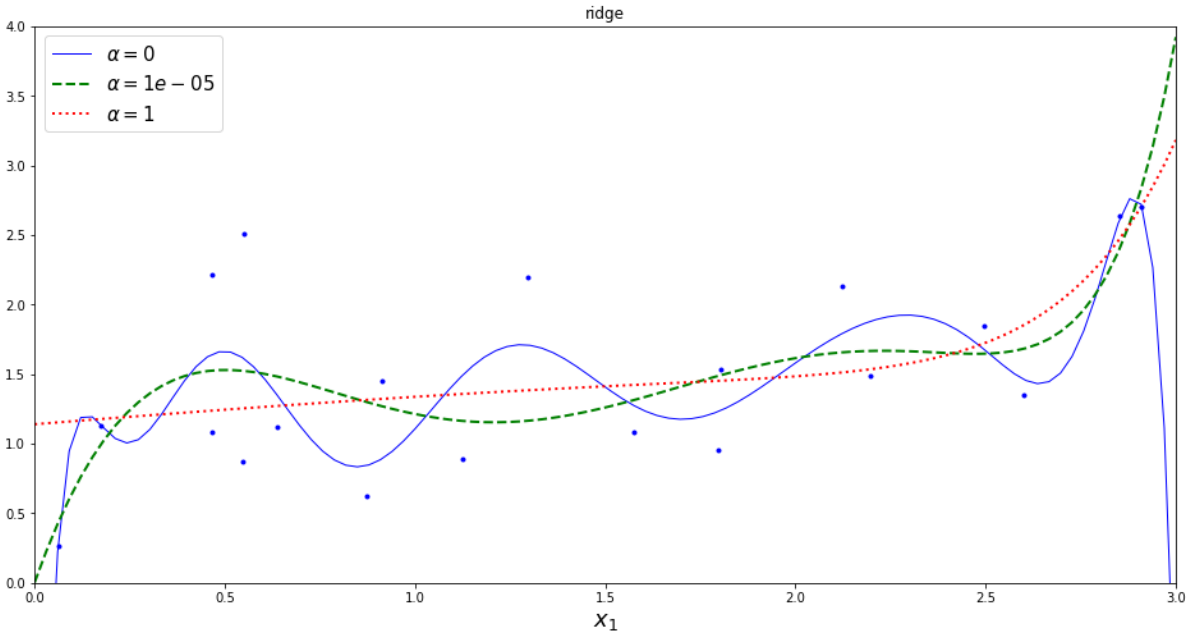
Qb Explain the Ridge Plot

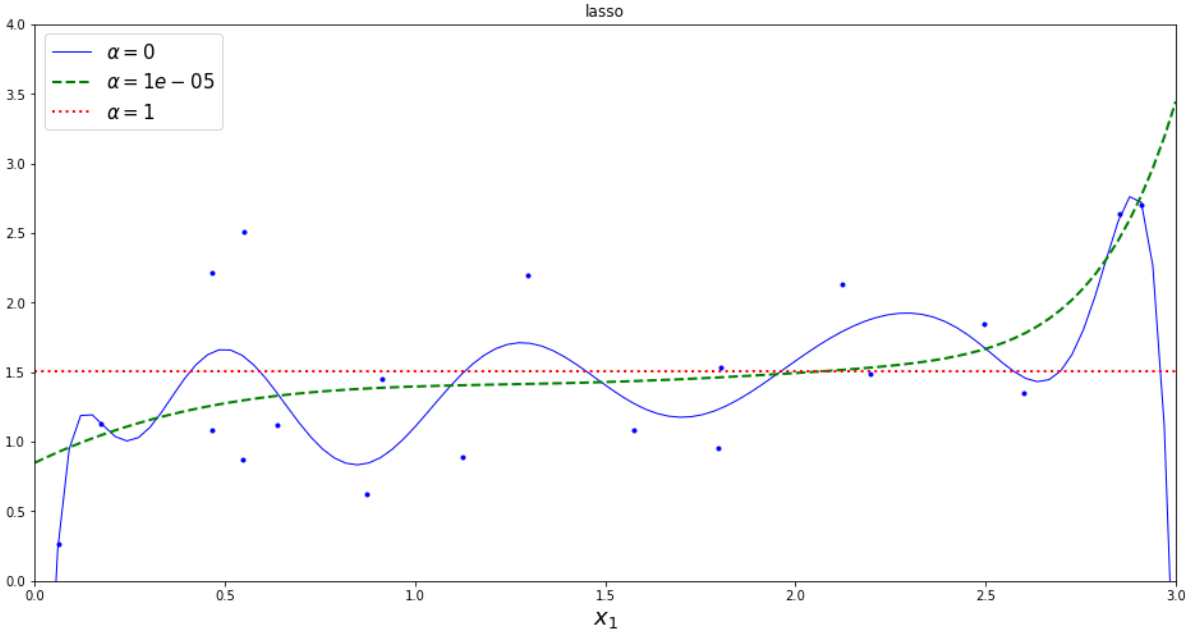
First take a peek into the plots (and code) below, that fits the `Ridge`, `Lasso` and `ElasticNet` to a polynomial model. The plots show three fits with different α values (0, 10^{-5}, and 1).

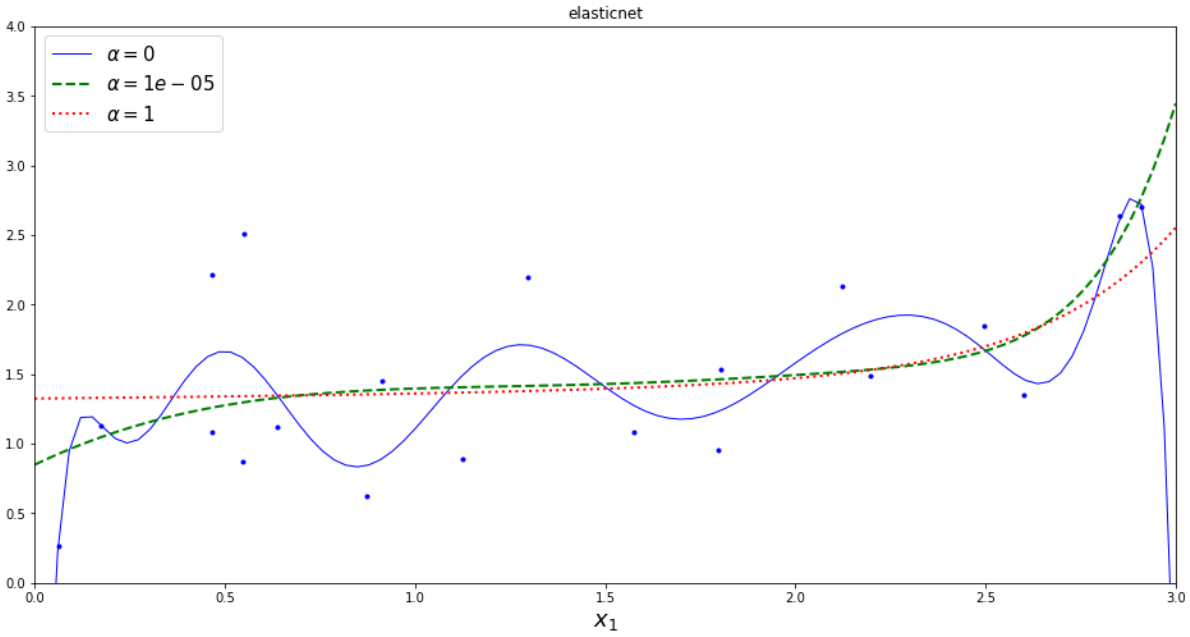
First, explain what the different α does to the actual fitting for the `Ridge` model in the plot.

In [2]:

```
# Assignment Qb

%matplotlib inline

from sklearn.linear_model import LinearRegression, SGDRegressor, Ridge, Elasti
cNet, Lasso

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

def FitAndPlotModel(name, model_class, X, X_new, y, **model_kargs):
    plt.figure(figsize=(16,8))

    alphas=(0, 10**-5, 1)
    random_state=42

    for alpha, style in zip(alphas, ("b-", "g--", "r:")):
        #print(model_kargs)
        model = model_class(alpha, **model_kargs) if alpha > 0 else LinearRegr
ession()
        model_pipe = Pipeline([
                ("poly_features", PolynomialFeatures(degree=12, include_bias=F
alse)),
                ("std_scaler", StandardScaler()),
                ("regul_reg", model),
            ])

        model_pipe.fit(X, y)
        y_new_regul = model_pipe.predict(X_new)

        lw = 2 if alpha > 0 else 1
        plt.plot(X_new, y_new_regul, style, linewidth=lw, label=r"$\alpha = {}
$".format(alpha))

    plt.plot(X, y, "b.", linewidth=3)
    plt.legend(loc="upper left", fontsize=15)
    plt.xlabel("$x_1$", fontsize=18)
    plt.title(name)
    plt.axis([0, 3, 0, 4])

def GenerateData():
    np.random.seed(42)
    m = 20
    X = 3 * np.random.rand(m, 1)
    y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5
    X_new = np.linspace(0, 3, 100).reshape(100, 1)
    return X, X_new, y

X, X_new, y = GenerateData()

FitAndPlotModel('ridge',      Ridge,        X, X_new, y)
FitAndPlotModel('lasso',      Lasso,        X, X_new, y)
FitAndPlotModel('elasticnet', ElasticNet,   X, X_new, y, l1_ratio=0.1)
```


ridge
$\alpha = 0$
$\alpha = 1e-05$
$\alpha = 1$
x_1


lasso
$\alpha = 0$
$\alpha = 1e-05$
$\alpha = 1$
x_1



Qb Results

α is the regularization strength which improves the conditioning of the problem and reduces the variance of estimates. Larger value in α means stronger regularization. Which means a large α makes the weights end up very close to zero and result is a flat line going through the data's mean.
We can observe on the ridge plot that the increasement of α "simplifies" the plot and reduces the weights of the model if compared with $\alpha = 0$ which is a linear regression. However, the ridge model tries to keep the weights in the beginning and end of the plot compared to the other models.

Qc Explain the Ridge, Lasso and ElasticNet Regulized Methods

Then explain the different regularization methods used for the `Ridge`, `Lasso` and `ElasticNet` models, by looking at the math formulas for the methods in the Scikit-learn documentation and/or using [HOML].

Qc Results

Ridge:
Most important element of the ridge method is that it uses the $L2$ penalization. This means if we were to say what a ridge regression does, it would be to fit the data while keeping the model weights as small as possible.

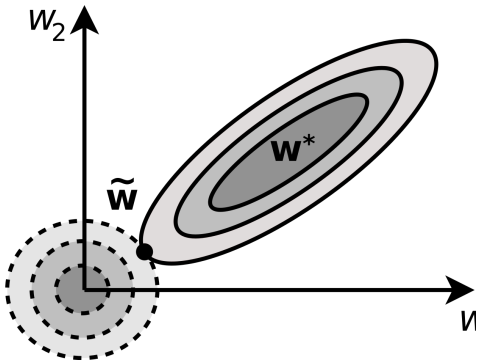
Lasso:
Lasso method replaces the $L2$ norm with $L1$ norm. The Lasso regression tends to completely elimnate the weights of the least important features.
This is very true if we were to look at the Lasso plot from Qa and compare it with the Ridge plot. It can be seen that the Lasso plot when $\alpha = 1$ is a straight line with no weights.

ElasticNet:
This regression is a middle ground between Ridge- and Lasso regression. It has a "mix ratio" defined as r and when $r = 0$ it is equivalent to the Ridge regression, and if $r = 1$ it is equivalent to the Lasso regression. Thus it is able to control the ratio between the two regression by using ElasticNet method.
In assignment Qa the ElasticNet plot was defined to be with $r = 0.1$ and thus it should resemble the Ridge regression with a small mix of the Lasso regression.

Qd Regularization and Overfitting

Finally, comment on how regularization may be used to reduce a potential tendency to overfit the data

Describe the situation with the tug-of-war between the MSE and regulizer terms in $\tilde{J}$ and the potential problem of $\mathbf{w}^*$ being far, far away from the origin (with $\alpha = 1$ in regulizer term).



Would data preprocessing in the form of scaling, standardization or normalization be of any help to that particular situation? If so, describe.

Qd Results

In linear regression regularization is about constraining the weights and disallowing them to grow widely. This leads to reduced overfitting where the model won't be able to learn from the background noise in the data.

The tug-of-war situation describes the sitution that $\tilde{J}$ is dealing with J as an addition of the multiplication of α and Ω since these two factors decide how much power the regulizers get in the tug-of-war. Furthermore, when $\mathbf{w}^*$ is far away from the origin it can potentially challenge the regulizer, creating difficulty in terms of getting good results. However with data-preprocessing in form of scaling, we are able to bring $\mathbf{w}^*$ closer to origin and thus yielding more power to the penalty.

Conclusion

Penalty factor Ω and regularization strength α has been covered while analyzing three different linear regression methods; Ridge, Lasso and ElasticNet.
The Ridge regression uses $L2$ regression, Lasso regression replaces the $L2$ with $L1$ and ElasticNet is the middle-ground between Ridge- and Lasso regression.
We discovered that regularization leads to reduced overfitting by constraining the weights and disallowing them to grow widely. Furthermore, the tug-of-war situation deals with J being an addition to the $\alpha\Omega$ term. And when $\mathbf{w}^*$ is far away from the origin, it can create difficulty with getting good results.