Regularization

Regularization involves adding a penalty term to our loss function. It turns out that this penalty term can help combat overfitting by making the model more biased but with less variance.

What we will accomplish

In this notebook we will:

- Introduce the general idea behind regularization,
- Discuss ridge and lasso regression as particular regularization algorithms
- Discuss how ridge can combat multicollinearity.
- Show how lasso is nice for feature selection.

Quick Note

This notebook is a little math heavy, I will do my best to provide both mathematical insight for those that want it and give a broad overview for those that do not want to delve too much into the math specifics.

In [2]:
```python
# import the packages we'll use
## For data handling
import pandas as pd
import numpy as np
from numpy import meshgrid

## For plotting
import matplotlib.pyplot as plt
import seaborn as sns

## This sets the plot style
## to have a grid on a white background
sns.set_style("whitegrid")
```

The idea behind regularization

Recall our supervised learning framework:

A **model** or **hypothesis class** is a collection of functions $\mathcal{H} = \{f_\theta : \mathbb{R}^p \to \mathbb{R}^m : \theta \in \Theta \subset \mathbb{R}^q\}$.

We need a way to evaluate the performance of each f_θ at reproducing the input/output pairs we actually observed in the data. This performance evaluation metric is called a **loss function**. It will a single real number score $\ell(\theta)$ to each parameter vector θ.

We *fit* the model by finding the parameters which minimize $\ell(\theta)$.

To summarize to do supervised learning we need

- Data which comes in pairs of input measurements and output measurements $(\vec{x}_i, y_i)$ for $i = 1, 2, 3, \dots, n$.
- We select a *model* f_θ which is the collection of functions we will consider as candidates and is parameterized by some list of numbers θ.
- We select a *loss function* ℓ which allows us to judge how any particular f_θ performs on our data.
- We minimize the loss function to obtain the *fitted model* $\hat{f}$ which has parameters $\hat{\theta}$.
    - Note: Often for "classic" machine learning algorithms the minimum is unique and can be found to high precision. This is not the case for all models: we will see that Neural Networks often have extremely complex loss landscapes with many local minima and saddles.

Penalizing large parameters

We can modify our loss function to penalize "large" parameters.

$$\ell(\theta) + \alpha \, \mathrm{Size}(\theta)$$

Here α is an adjustable constant which we will call a *hyperparameter*: a parameter that is not learned during the training of an estimator. Different ways of measuring the size of a vector lead to different regularization methods.

For $\alpha = 0$ we recover the unregularized estimate for θ, for $\alpha = \infty$ we get $\theta = 0$, values of α between those two extremes will give different parameter estimates. The value of α that gives the best model for your data depends on the problem and can be found through *nested* cross-validation (which we will discuss in a `3_hyperparameter_tuning.ipynb`).

Specific regularization models

Ridge regression and *lasso* are two forms of regularization where we make specific choices of how to measure the "size" of the parameters.

Ridge regression

In ridge regression we use the the size of θ as the square of the Euclidean length (or "ℓ_2-norm") of θ:

$$\text{Size}_{\text{Ridge}}(\theta) = ||\theta||_2^2 = \theta_1^2 + \theta_2^2 + \cdots + \theta_p^2.$$

This is the length you get using the Pythogorean Theorem!

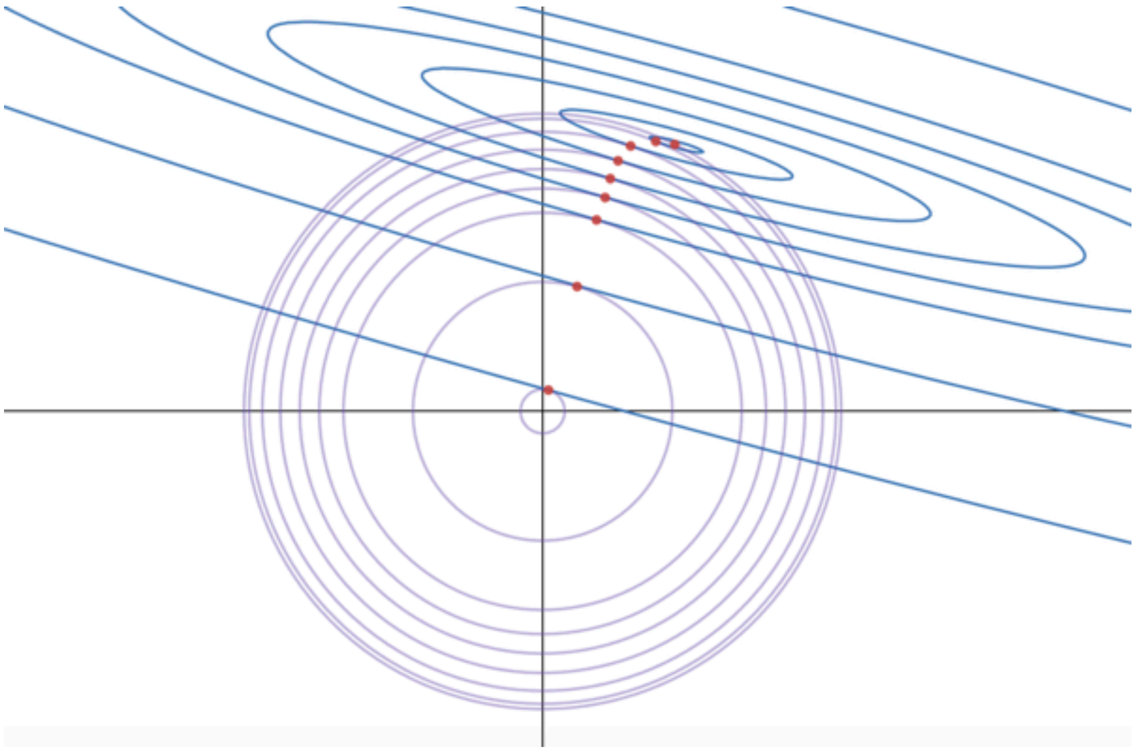
Lasso regression

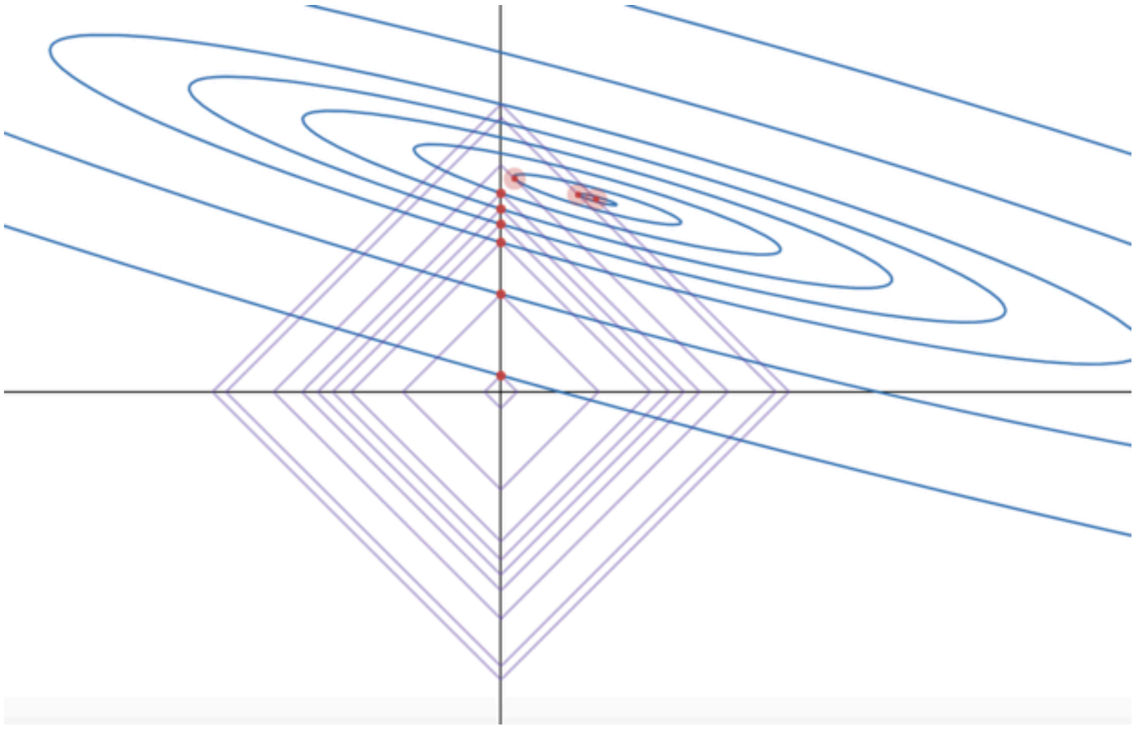
In lasso regression we take $\text{Size}(\theta)$ to be the ℓ_1-norm:

$$\text{Size}_{\text{Lasso}}(\theta) = ||\theta||_1 = |\theta_1| + |\theta_2| + \cdots + |\theta_p|.$$

Some geometric intuition

In the particular case of linear regression, the loss function is quadratic.





On the left we see a picture of contour lines (in blue) for the MSE of a particular linear regression problem with two features. The purple circles are the contours of the ℓ_2 norm. The OLS estimate, which minimizes the MSE alone, is at the center of all of the ellipses. The origin minimizes the ℓ_2 norm. Ridge regression gives us a "tug of war" between these two quantities. Note that a ridge regression solution $\hat{\theta}_\alpha$ must be at a point of tangency: if it were at a point of transverse intersection between contours you could move along one contour while decreasing the other. We can see some different ridge estimates here: the small α are close to the OLS estimates, while large α is close to the origin.

We get a similar picture for Lasso regression, but the contours of the ℓ_1 norm are squares instead of circles! The same argument about tangency applies *until* we intersect a coordinate axis. Notice that it is clear, from this picture, that the $\hat{\theta}_\alpha$ must follow a **piecewise linear** path as we vary α (a fact which has some cool applications)!

We can also see that while Ridge regression will never zero out a parameter, Lasso will! In this way, Lasso can be used for "automatic feature selection".

Let's see this play out by fitting a degree 10 polynomial our data using both Ridge and Lasso regression and seeing how the coefficients change as we adjust α.

Implementing in `sklearn`

We can implement both of these linear models in `sklearn` with `Ridge` https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html for ridge regression and `Lasso` https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html for lasso regression.

Note: ridge and lasso regression are examples of algorithms/models where scaling the data is a step that should be taken prior to fitting the model. This is because vastly different scales can impact the scales of the components of θ. This can make it so that there is not enough room in the θ-budget to afford the actual values of the individual coefficients.

In [18]:
```
## Import the models here
## Ridge and Lasso regression are stored in linear_model

from sklearn.linear_model import Ridge, Lasso
```

In [19]:
```
## This code will allow us to demonstrate the effect of
## increasing alpha

## set values for alpha
alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]

## The degree of the polynomial we will fit
n=10

#$ These will hold our coefficient estimates
ridge_coefs = np.empty((len(alpha),n))
lasso_coefs = np.empty((len(alpha),n))

## for each alpha value
for i in range(len(alpha)):
    ## set up the ridge pipeline
    ## first scale
    ## then make polynomial features
    ## then fit the ridge regression model
    ridge_pipe = Pipeline([('scale',StandardScaler()),
                           ('poly',PolynomialFeatures(n, interactio
                           ('ridge', Ridge(alpha=alpha[i], max_iter
                           ])

    ## set up the lasso pipeline
    ## same steps as with ridge
    lasso_pipe = Pipeline([('scale',StandardScaler()),
                           ('poly',PolynomialFeatures(n, interactio
                           ('lasso', Lasso(alpha=alpha[i], max_iter
                       ])

    ## fit the ridge
    ridge_pipe.fit(x.reshape(-1,1), y)
    ## fit the lasso
    lasso_pipe.fit(x.reshape(-1,1), y)


    # record the coefficients
    ridge_coefs[i,:] = ridge_pipe['ridge'].coef_
    lasso_coefs[i,:] = lasso_pipe['lasso'].coef_
```

In [20]:
```
print("Ridge Coefficients")

pd.DataFrame(np.round(ridge_coefs,8),
             columns = ["x^" + str(i) for i in range(1,n+1)],
             index = ["alpha=" + str(a) for a in alpha])
```

```
Ridge Coefficients
```

Out[20]:

	x^1	x^2	x^3	x^4	x^5	x^
alpha=1e-05	-0.532096	-3.291722	-4.387906	11.909853	5.025352	-9.18368
alpha=0.0001	-0.534826	-3.258441	-4.375907	11.821701	5.010790	-9.10058
alpha=0.001	-0.561268	-2.953679	-4.259673	11.014577	4.869705	-8.33981
alpha=0.01	-0.760828	-1.358686	-3.380449	6.796907	3.801530	-4.36916
alpha=0.1	-1.228197	0.381332	-1.262057	2.294823	1.199363	-0.20802
alpha=1	-1.273637	0.897989	-0.530516	1.197639	0.093808	0.47231
alpha=10	-0.887189	0.696417	-0.477340	0.713626	-0.171016	0.41101
alpha=100	-0.319023	0.221442	-0.257106	0.286443	-0.223319	0.30650
alpha=1000	-0.070440	0.057039	-0.077307	0.093393	-0.091999	0.13164

In [21]:
```
print("Lasso Coefficients")

pd.DataFrame(np.round(lasso_coefs,8),
             columns = ["x^" + str(i) for i in range(1,n+1)],
             index = ["alpha=" + str(a) for a in alpha])
```

```
Lasso Coefficients
```

Out[21]:

	x^1	x^2	x^3	x^4	x^5	x^
alpha=1e-05	-0.538353	-3.250257	-4.359495	11.800042	4.990309	-9.0800
alpha=0.0001	-0.597412	-2.843806	-4.091758	10.723675	4.660312	-8.0640
alpha=0.001	-1.201256	-0.000000	-1.353878	3.251137	1.285138	-1.0597
alpha=0.01	-1.434115	0.570207	-0.264264	1.808744	-0.000000	-0.0000
alpha=0.1	-1.277036	0.000000	-0.081929	1.501704	-0.137108	0.0000
alpha=1	-0.000000	0.000000	-0.000000	0.000000	-0.167810	0.4751
alpha=10	-0.000000	0.000000	-0.000000	0.000000	-0.000000	0.0000
alpha=100	-0.000000	0.000000	-0.000000	0.000000	-0.000000	0.0000
alpha=1000	-0.000000	0.000000	-0.000000	0.000000	-0.000000	0.0000

An important note about scaling.

OLS linear regression is scale invariant. That means that if you scale your features and run OLS linear regression you will get exactly the same predictions as if you had not scaled them. Another way of thinking about this is that OLS will give you the same results no matter what units you use for your features: for example if x_1 is height in meters and x_1' is height in centimeters, then the OLS estimates $\hat{\theta}_1$ and $\hat{\theta}_1'$ would be related by $\hat{\theta}_1' = 100\hat{\theta}_1$. Changing the unit from m to cm just changes the unit of θ from $\frac{\text{units}}{m}$ to $\frac{\text{units}}{cm}$.

Both Ridge and Lasso regression **are not** scale invariant. It is easy to see why: if we change from m to cm the ``size'' the OLS parameters will change by a factor of $\frac{1}{100}$. As a consequence, both Ridge and Lasso will prioritize keeping these predictors in the model, since the coefficient is not very "expensive" in terms of parameter size but does a lot to decrease the MSE.

To avoid this it is highly advisable to *scale and center your data* before Ridge or Lasso regression.

Ridge and Lasso Regularization in other sklearn models

Models with Explicit L2 (Ridge-style) Regularization

- **Ridge** — parameter: `alpha`
- **LogisticRegression** (`penalty="l2"`) — parameter: `C` (inverse strength)
- **LinearSVC**, **LinearSVR** (`penalty="l2"`) — parameter: `C`
- **MLPClassifier / MLPRegressor** — parameter: `alpha`
- **ElasticNet** (`l1_ratio < 1`) — parameters: `alpha`, `l1_ratio`
- **SGDClassifier / SGDRegressor** (`penalty="l2"`) — parameter: `alpha`

Models with Explicit L1 (Lasso-style) Regularization

- **Lasso** — parameter: `alpha`
- **LogisticRegression** (`penalty="l1"` , solver= `'saga'` or `'liblinear'`) — parameter: `C`
- **LinearSVC** (`penalty="l1"` , solver= `'liblinear'`) — parameter: `C`
- **ElasticNet** (`l1_ratio > 0`) — parameters: `alpha`, `l1_ratio`
- **SGDClassifier / SGDRegressor** (`penalty="l1"`) — parameter: `alpha`

Models with ElasticNet (L1 + L2) Regularization

- **ElasticNet** (`sklearn.linear_model.ElasticNet`) — parameters: `alpha`, `l1_ratio`
- **LogisticRegression** (`penalty="elasticnet"` , solver= `'saga'`) — parameters: `C`, `l1_ratio`
- **SGDClassifier / SGDRegressor** (`penalty="elasticnet"`) — parameters: `alpha`, `l1_ratio`

Which one to use?

Which type of regularization is the better choice? Well that depends on the problem. Both are good at addressing overfitting concerns, but each has a couple unique pros and cons.

Lasso

Pros

- Works well when you have a large number of features that do not have any effect on the target
- Feature selection is a plus, this can allow for a sparser model which is good for computational reasons.

- Feature selection can also produce a more interpretable model.

Cons

- Can have trouble with highly correlated features (colinearity), it typically chooses one variable among those that are correlated, which may be random.

Ridge

Pros

- Works well when the target depends on all or most of the features and
- Can handle colinearity better than lasso.

Cons

- Because ridge typically keeps most of the predictors in the model, this can be a computationally costly model type for data sets with a large number of predictors.
- Keeping all features also makes interpretation of the model difficult.

Elastic Net

Sometimes the best model will have both types of regularization penalties.
