Ridge Regression

Understand the need for regularization in linear regression.

Chapter Goals:

- Learn about regularization in linear regression
- Learn about hyperparameter tuning using cross-validation
- Implement a cross-validated ridge regression model in scikit-learn

While ordinary least squares regression is a good way to fit a linear model onto a dataset, it relies on the fact that the dataset's features are each independent, i.e. uncorrelated. When many of the dataset features are linearly correlated, e.g. if a dataset has multiple features depicting the same price in different currencies, it makes the least squares regression model highly sensitive to noise in the data.

Because real life data tends to have noise, and will often have some linearly correlated features in the dataset, we combat this by performing *regularization*. For ordinary least squares regression, the goal is to find the weights (coefficients) for the linear model that minimize the sum of squared residuals:

$$\sum_{i=1}^{n}(\mathbf{x}_i \cdot w - y_i)^2$$

where each x_i represents a data observation and y_i is the corresponding label.

For regularization, the goal is to not only minimize the sum of squared residuals, but to do this with coefficients as small as possible. The smaller the coefficients, the less susceptible they are to random noise in the data. The most commonly used form of regularization is ridge regularization.

With ridge regularization, the goal is now to find the weights that minimize the following quantity:

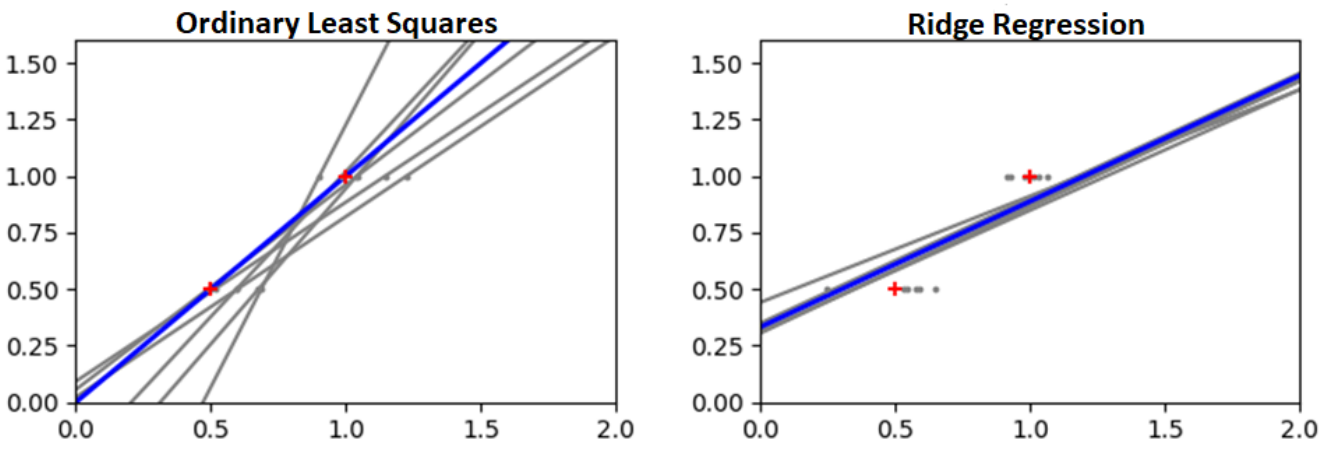
the following quantity:

$$\alpha||w||_2^2 + \sum_{i=1}^{n}(\mathbf{x}_i \cdot w - y_i)^2$$

where α is a non-negative real number hyperparameter and $||w||_2$ represents the L2 norm of the weights. The additional

$$\alpha||w||_2^2$$

is referred to as the *penalty term*, since it penalizes larger weight values. Larger quantities of α will put greater emphasis on the penalty term, forcing the model to have even smaller weight values.



The plot above shows an example of ordinary least squares regression models vs. ridge regression models. The two red crosses mark the points (0.5, 0.5) and (1, 1), and the blue lines are the regression lines for those two points. Each of the grey lines are the regression lines for the original points with added noise (which is signified by the grey points).

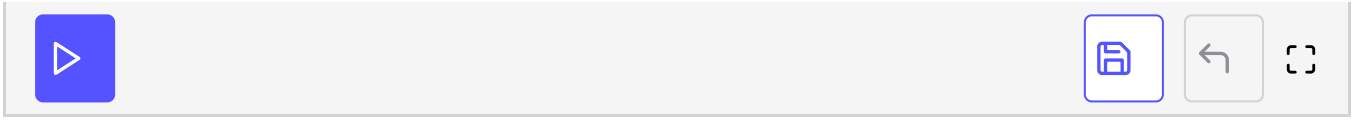
The ordinary least squares regression is much more susceptible to being influenced by the added noise, as there is a much larger degree of variance in the grey regression lines compared to the ridge regression.

B. Choosing the best alpha

In scikit-learn, we implement ridge regression in essentially the same way we implement ordinary least squares regression. We use the `Ridge` object (part of the `linear_model` module) to implement ridge regression.

The code below fits a `Ridge` object on the pizza dataset from the previous chapter.

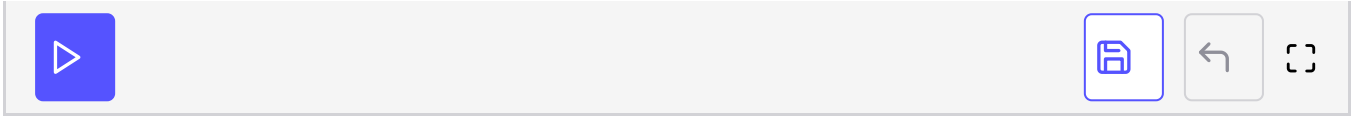
```
from sklearn import linear_model
reg = linear_model.Ridge(alpha=0.1)
reg.fit(pizza_data, pizza_prices)
print('Coefficients: {}\n'.format(repr(reg.coef_)))
print('Intercept: {}\n'.format(reg.intercept_))
r2 = reg.score(pizza_data, pizza_prices)
print('R2: {}\n'.format(r2))
```



We can specify the value of the α hyperparameter when initializing the `Ridge` object (the default is 1.0). However, rather than manually choosing a value, we can use *cross-validation* to help us choose the optimal α from a list of values.

We'll discuss the specifics of cross-validation in the chapters Cross-Validation, Applying CV to Decision Trees, and Exhaustive Tuning, but for now just know that we can implement a cross-validated ridge regression using the `RidgeCV` object.

```
from sklearn import linear_model
alphas = [0.1, 0.2, 0.3]
reg = linear_model.RidgeCV(alphas=alphas)
reg.fit(pizza_data, pizza_prices)
print('Coefficients: {}\n'.format(repr(reg.coef_)))
print('Intercept: {}\n'.format(reg.intercept_))
print('Chosen alpha: {}\n'.format(reg.alpha_))
```



Time to Code!

The coding exercise in this chapter uses the `RidgeCV` object of the `linear_model` module (imported in backend) to complete the `cv_ridge_reg` function.

The function will fit a ridge regression model to the input data and labels. The model is cross-validated to choose the best α value from the input list `alphas`.

Set `reg` equal to `linear_model.RidgeCV` initialized with the input list, `alphas`, for the `alphas` keyword argument.

Call `reg.fit` with `data` and `labels` as the two input arguments. Then return `reg`.

```
def cv_ridge_reg(data, labels, alphas):
  # CODE HERE
  pass
```