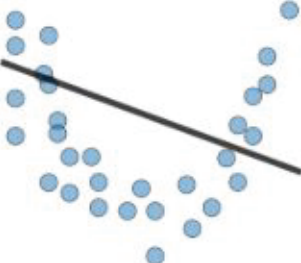
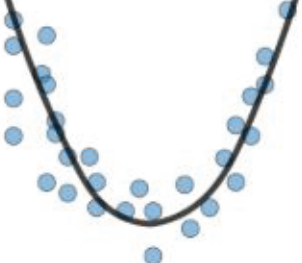
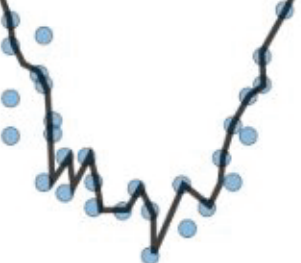
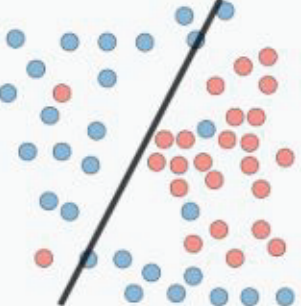
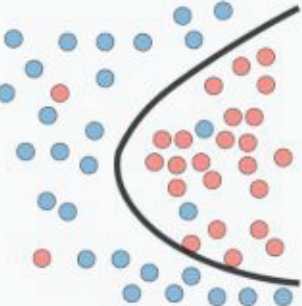
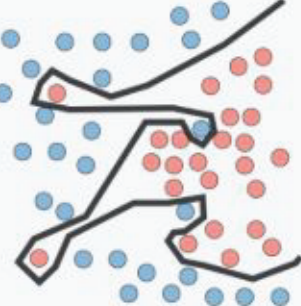

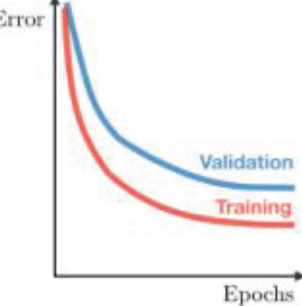
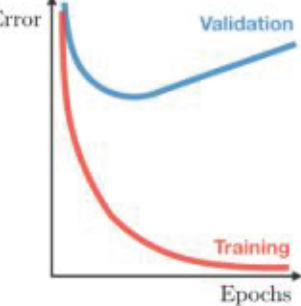


## 7.1. BIAS–VARIANCE TRADE–OFF

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> <li>• High training error</li> <li>• Training error close to test error</li> <li>• High bias</li> </ul>	<ul style="list-style-type: none"> <li>• Training error slightly lower than test error</li> </ul>	<ul style="list-style-type: none"> <li>• Very low training error</li> <li>• Training error much lower than test error</li> <li>• High variance</li> </ul>
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> <li>• Complexify model</li> <li>• Add more features</li> <li>• Train longer</li> </ul>		<ul style="list-style-type: none"> <li>• Perform regularization</li> <li>• Get more data</li> </ul>

<https://www.kaggle.com/getting-started/166897>

We consider a regression problem with predictor  $X$  and a quantitative output  $y$ . Assume that there is some general relationship

$$y = f(X) + \epsilon$$

where  $f$  is a fixed (but unknown) function and  $\epsilon$  is a random error term which is independent of  $X$  with mean 0 and standard deviation  $\sigma$ . Assume that our machine learning model gives an estimate  $\hat{f}$  of  $f$  and we have predicted output

$$\hat{y} = \hat{f}(X).$$

For any learning model, we define the bias and the variance by

$$\text{bias}(\hat{f}) := E[\hat{f} - f] \qquad \text{variance}(\hat{f}) := \text{Var}(\hat{f}).$$

The bias and variance are not directly related and there are models which have high or low bias and have either high or low variance. Note that the bias depends on the unknown function  $f$ .

For a fixed estimate  $\hat{f}$  and a fixed input  $X$ , we have

$$\begin{aligned} E[(y - \hat{y})^2] &= E\left[\left(f(X) - \hat{f}(X) + \epsilon\right)^2\right] \\ &= E\left[f(X) - \hat{f}(X) + \epsilon\right]^2 + \text{Var}\left[f(X) - \hat{f}(X) + \epsilon\right] && \text{using } E[A^2] = E[A]^2 + \text{Var}[A] \\ &= \left(E\left[f(X) - \hat{f}(X)\right] + E[\epsilon]\right)^2 + \text{Var}[\epsilon] && \text{since } f(X) - \hat{f}(X) \text{ is constant} \\ &= \left[f(X) - \hat{f}(X)\right]^2 + \text{Var}[\epsilon] && \text{since } \epsilon \text{ has mean 0.} \end{aligned}$$

The left hand-side of the above expression measures the accuracy of  $\hat{y}$  as a prediction of  $y$  and is the expected value (or average) of the squared distance between  $y$  and  $\hat{y}$ . The right-hand side is the sum of the *reducible* and *irreducible* errors. In general,  $\hat{f}$  will not be a perfect estimate of  $f$  and this inaccuracy is the reducible error because we can potentially find a better machine learning algorithm to estimate  $f$ . But, even if we found a perfect estimate of  $f$  and  $\hat{f} = f$ , our prediction would still have some error since  $y$  also depends on  $\epsilon$  which cannot be predicted using  $X$ . Therefore, variability in  $\epsilon$  affects the accuracy of our predictions and this is defined as the irreducible error. The reasons for this are, for example, that  $\epsilon$  may contain unmeasured variables that are useful in predicting  $y$  as well as that  $\epsilon$  also may contain some unmeasured variation. Our goal, as data scientists, is to try to minimize the reducible error.

Next, for an unseen test data instance  $(x_*, y_*)$ , note that

$$\begin{aligned}
\text{MSE}(\hat{f}(x_*)) &= E \left[ (y_* - \hat{y}_*)^2 \right] \\
&= E \left[ \left( \epsilon + f(x_*) - \hat{f}(x_*) \right)^2 \right] \\
&= E \left[ \epsilon^2 + 2 \cdot \epsilon \cdot (f(x_*) - \hat{f}(x_*)) + \left( f(x_*) - \hat{f}(x_*) \right)^2 \right] \\
&= E[\epsilon^2] + 2 \cdot E[\epsilon] \cdot E \left[ f(x_*) - \hat{f}(x_*) \right] + E \left[ \left( f(x_*) - \hat{f}(x_*) \right)^2 \right] \\
&= E[\epsilon^2] + E \left[ \left( f(x_*) - \hat{f}(x_*) \right)^2 \right] && \text{since } E[\epsilon] = 0 \\
&= E[\epsilon]^2 + \text{Var}[\epsilon] + E \left[ f(x_*) - \hat{f}(x_*) \right]^2 + \text{Var}[f(x_*) - \hat{f}(x_*)] \\
&= \text{Var}[\epsilon] + E \left[ f(x_*) - \hat{f}(x_*) \right]^2 + \text{Var}[\hat{f}(x_*)] && \text{since } f(x_*) \text{ is constant}
\end{aligned}$$

where in the 4th line we use that for independent random variables  $A$  and  $B$  we have  $E[A \cdot B] = E[A] \cdot E[B]$  and in the 6th line we use  $E[A^2] = (E[A])^2 + \text{Var}[A]$ .

Therefore, we have

$$\boxed{\text{MSE}(\hat{f}(x_*)) = \text{irreducible error} + (\text{bias})^2 + \text{variance}}$$

The left hand-side side is the expected test MSE at  $x_*$  and refers to the average test MSE that we would obtain if we repeatedly estimated  $f$  using a large number of training sets and tested each at  $x_*$ . This means that in order to minimize the expected test error we need to select a machine learning model that simultaneously achieves *low bias* and *low variance*. This relationship between MSE, bias, and variance is called the bias-variance trade-off. Note that expected MSE can never be lower than the irreducible error.

Variance refers to the amount by which estimate  $\hat{f}$  would change if we estimated it using a different training data set. If a model has high variance it means that a small change in the training data set results in a large change in  $\hat{f}$ .

Bias refers to the error that is introduced by approximating a real-life problem, which might be very complicated, by a very simple model. For example, if the data is not linearly related, linear regression model will have a high bias.

In general, as we use more flexible models, the variance will increase and the bias will decrease.

*Remark:* Such a clean decomposition into irreducible error, bias, and the variance terms exists only for the squared error loss. Proposals have been made for more general loss functions, but none are widely accepted.

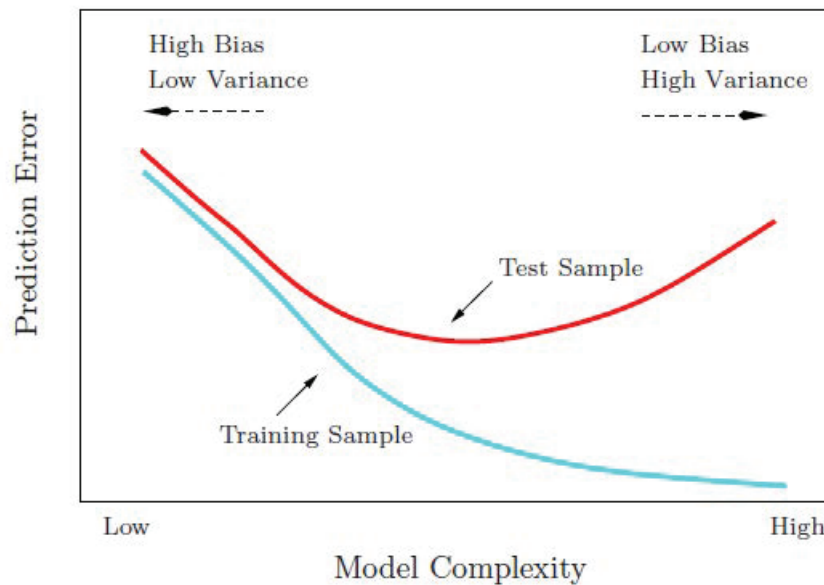
Bias-variance decomposition relates to underfitting and overfitting.

- Underfitting refers to having a high bias model. To fight underfitting, we need to reduce bias in the model by using a more flexible model, using more features, or decreasing regularization.
- Overfitting refers to having a high variance in the model. To fight overfitting, we need to reduce variance in the model by using a less flexible model, decreasing the number of features, increasing regularization, or obtaining larger data set.

To improve the generalization error (i.e., the test error) we need to determine which component (bias or variance) in the decomposition has the highest contribution and go after that component. Some suggestions are as follows.

- Training error can be treated as the amount of the bias in the model. If the model is unable to fit the training data, then it is likely that the model has high bias and this indicates underfitting.
- The difference between the training error and the cross-validation error can be treated as the amount of the variance in the model. If the training error is low and the cross-validation error is high, this indicates high variance, and is the overfitting case.

We should always analyze the model performance by looking at the training error and the cross-validation error simultaneously as this is the only way to obtain an estimate of the bias and variance components. Often strategies taking to fight one (either high bias or high variance) can end up worsening the other.



<https://online.stat.psu.edu/stat508/book/export/html/788>

### References and Reading Material:

- [1] Andrew Ng's notes on Bias Variance Analysis
- [2] *Introduction to Statistical Learning*, James et al., Sections 2.1.1, 2.2.2

## 7.2. CROSS-VALIDATION

- **Parameters and Hyperparameters**

A model parameter is a variable that is internal to the model and whose value can be estimated from data automatically. In other words, parameters are the part of the model that is learned from historical training data. Whether a model has a fixed or variable number of parameters determines whether it may be referred to as parametric or nonparametric. Some examples of model parameters are:

- the coefficients in linear regression

A model hyperparameter is a variable that is external to the model and whose value cannot be estimated from data and it is set manually. The model hyperparameters are

- often specified by the practitioner
- often set using heuristics
- often tuned for a given predictive modeling problem

We cannot know the best value for a model hyperparameter on a given problem in advance. We may use rules of thumb, copy values used on other problems, or search for the best value by trial and error. When a machine learning algorithm is tuned for a specific problem, such as when we employ a grid search or a random search, we are tuning the hyperparameters in order to discover the hyperparameters of that result in the most skillful predictions. Some examples of hyperparameters include:

- the learning rate for training a logistic regression model
- the number of hidden layers in a neural network

- **The Validation Set Approach**

Three phases in the machine learning process are:

1. training: training the model
2. validation and selection: evaluating and comparing different models which may represent the same broad type of an algorithm (different number of hidden layers in ANNs) or a completely different algorithm (Logistic Regression and ANN)
3. testing and assessment: final, last-chance estimate of how the machine learning model will perform in the real world

The datasets used for these phases are:

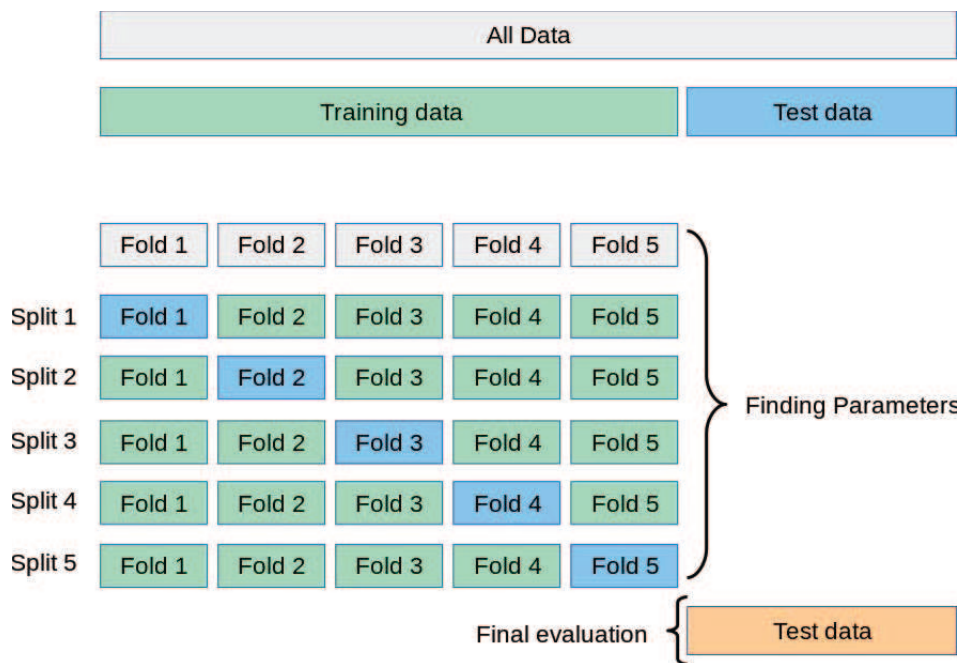
1. training set: used to fit the parameters of the machine learning model
2. validation set: used to get an evaluation of a learning model by giving an estimate of the test error ("model assessment") and to help us pick optimal hyperparameters or choose between different models ("model selection")
3. test set: used to make sure that the entire process of building one or more machine learning models, optimizing them, evaluating them, and picking among them is evaluated fairly. The test data we have never used before in any training or validation is necessary to protect us from the indirect peeking ("data leakage") and to give us a fair evaluation of how our final system will perform on the novel data.



[https://commons.wikimedia.org/wiki/File:ML\\_dataset\\_training\\_validation\\_test\\_sets.png](https://commons.wikimedia.org/wiki/File:ML_dataset_training_validation_test_sets.png)

- **$k$ -Fold Cross-Validation**

A single train-validation-test split is a simple method, however, we might get unlucky and underestimate the real-world performance. A better way is to use cross-validation and generate multiple datasets. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing cross-validation.



[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

In a  $k$ -fold cross-validation, the data is broken into  $k$ -folds of almost equal sizes. In each step,  $k - 1$  folds are used for training and the remaining fold is used for testing. We record the performance of our methods in each of these  $k$  steps and we can graph as well as summarize these values. Graphing them can help us understand how variable our performance is with respect to different training and testing datasets. If we see a very wide spread in our performance measures, we would be justified in being skeptical of any single performance score for our system (if the spread in performance is about 10%, that is considered large enough to warrant our attention). On the other hand, if the scores are all similar, we have some certainty that, regardless of the specific split, our method's performances will be similar.

## How do we pick $k$ ?

A bigger  $k$  means more folds which in turn means more training and testing phases. The value of  $k = 2$  makes two disjoint folds of data and two estimates of our test error. The largest value  $k$  can take is the number of data points we have,  $k = n$ , which results in each example being in its own fold and  $n$  total models and estimates. A suggested set of values for  $k$  are between 5 and 10.

## Variations on $k$ -Fold Cross-Validation:

- *Leave-One-Out Cross-Validation (LOOCV)*

This is when  $k = n$  and it means that a single observation is used for the validation set and the remaining  $n - 1$  make up the training set. Since there is no randomness in the splits, performing LOOCV multiple times will always give the same results. LOOCV can be expensive to implement since the model needs to be fit  $n$  times, which can be time consuming if  $n$  is large and if each individual model is slow to fit. LOOCV will give almost unbiased estimate of the test error since it is using almost all of data from the training set. However, LOOCV may have very high variance since the folds are highly correlated with each other while in  $k$ -CV with  $k < n$ , the folds have less overlap.

- *Stratified*

The specific splitting of data which ensures that each fold has the same proportion of observations with a given categorical value such as the class outcome value.

- *Repeated*

This is where the  $k$ -fold cross-validation is repeated several times and the data is shuffled prior to each repetition which results in a different split each time. Repeated  $k$ -fold cross-validation usually improves the estimate of the mean model performance at the cost of fitting and evaluating many more models. Common numbers of repeats include 3, 5, and 10. For example, if 3 repeats of 10-fold cross-validation are used to estimate the model performance, this means that 30 different models would need to be fit and evaluated. This approach is suited for small- to modestly-sized datasets and/or models that are not too computationally costly to fit and evaluate.

- *Nested or double-cross*

This is where  $k$ -fold cross-validation is performed within each fold of cross-validation, often to perform hyperparameter tuning during model evaluation.

## References and Reading Material:

- [1] *Introduction to Statistical Learning*, James et al., pages 175-186

## 7.3. VARIABLE SELECTION AND SHRINKAGE (REGULARIZATION) METHODS

- Consider a linear regression model

$$\hat{y} = \theta_0 + \theta_1 x_1 + \dots \theta_d x_d.$$

There are two main issues.

- Prediction accuracy: If the underlying relationship between predictors and the target variable is linear, the linear regression model based on least squares (i.e., based on minimizing MSE) will have low bias. In addition, if we have lots of data available for training ( $n \gg d$ ), this model will also have low variance. However, if  $n$  is not much larger than  $d$ , the linear regression model will have high variance. If  $n < d$ , then there will be no solution given by the least squares model.
- Interpretability: Including irrelevant predictors leads to unnecessary complexity of the model and its poor interpretability.

Three main ways on improving linear models are the following.

### – Variable selection

The goal is to select a subset of the original  $d$  predictors that we believe are related to the target variable.

- \* Best Subset Selection: We consider all possible subsets of  $k \in \{0, \dots, d\}$  among all  $d$  predictors and for each  $k$  we choose the best model (for example, we choose the model with the lowest MSE). The case  $k = 0$  means predicting the sample mean. Next, we choose the best model among these  $d + 1$  models. Note that as the number of predictors increases, the MSE will decrease monotonically, which means that the training error will get lower. However, our goal is to select a model with the lowest test error, and so we use cross-validation on these  $d + 1$  models to select the best model.

This approach is simple, but since there are  $2^d$  subsets of the set of  $d$  predictors, this approach suffers from computational cost and even modern computers will not be able to perform it if  $d$  is around 40.

- \* Forward Stepwise Selection: We start with the model with no predictors and then at each step  $k \in \{0, \dots, d - 1\}$  add predictors one at a time until all predictors are in the model. At each step  $k$ , we pick the model with lowest MSE and we choose the best model among all these models using cross-validation. The total number of models is

$$1 + d + (d - 1) + (d - 2) + \dots + 2 + 1 = 1 + \frac{d(d + 1)}{2}$$

Forward stepwise selection can be applied even when  $n < d$ ; however, it is not guaranteed that it will find the best possible model out of  $2^d$  models.

- \* Backward Stepwise Selection: We start with the full model containing all  $d$  predictors and then we remove the least useful predictor, iteratively one at a time. It requires that  $n$  is larger than  $d$  so that the full model can be fit.



- **Shrinkage (or Regularization) methods**

This approach involves fitting the model with all  $d$  predictors included, but the coefficients are shranken to 0 or they might be exactly 0. Therefore, this model can also perform variable selection.

- **Dimensionality reduction**

This approach involves computing  $m < d$  different linear combinations (or projections) of predictors. These  $m$  projections are then used as predictors to fit a model.

- **Shrinkage (or Regularization) methods**

We fit the model using all  $d$  predictors using a technique that constraints or regularizes the coefficients by shrinking them to 0. Shrinking coefficients reduces the variance of the model.

- Ridge Regression (or L2 Regularization)

In this case we minimize the function

$$\begin{aligned} L_{\text{ridge}}(\theta_0, \theta_1, \dots, \theta_d) &= \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 + \lambda \sum_{i=1}^d \theta_i^2 \\ &= \text{MSE} + \lambda (\|\theta\|_2)^2 \end{aligned}$$

where  $\theta = (\theta_1, \dots, \theta_d)$  and  $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_d x_d$ .

Here  $\lambda \geq 0$  is a hyperparameter that can be determined using cross-validation. The second term is the *shrinkage penalty* and it has the effect of shrinking  $\theta$  towards 0.

- \* When  $\lambda = 0$ , the penalty has no effect and the ridge regression is the same as the least square regression. If  $\lambda \rightarrow \infty$ , the impact of shrinkage grows and the coefficients become closer to 0.
- \* The second term is the  $L^2$  norm of vector  $\theta$ .
- \* Note that the penalty is applied to  $\theta_1, \dots, \theta_d$  and not to the intercept  $\theta_0$ .
- \* Ridge regression works best when the least squares estimates have high variance. As  $\lambda$  increases, the flexibility of ridge regression decreases leading to the model with reduced variance, but increased bias.
- \* Unlike best subset, ridge regression includes all  $d$  predictors. The penalty will shrink all of the coefficients to 0, but it will not set any of them exactly to 0. Hence, when  $d$  is large, ridge regression model may have challenges in interpretability.
- \* Note that the regularization term is added to MSE only during training. Once the model is trained, we use the unregularized performance measure to evaluate how the model performed.

This is very common in ML – for example, in classification, we often minimize the cross entropy loss, but we evaluate models based on their accuracy, precision, recall, etc. One of the reasons is that the function we want to optimize should have optimization friendly derivatives, but the evaluation measures should be as close as possible to the final objective.

- \* The above minimization problems has the analytical solution

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

- \* Ridge regression is affected by the predictors' scales and it is best to apply it after standardizing the predictors using the `StandardScaler` so that all standardized predictors have a standard deviation of 1.

- \* There are two ways to implement ridge regression in sklearn:
  - `from sklearn.linear_model import Ridge`  
which uses the analytical solution and matrix factorization by Cholesky
  - `from sklearn.linear_model import SGDRegressor(penalty="l2")`  
which uses stochastic gradient descent

– Lasso Regression (or L1 Regularization)

In this case, the coefficients minimize the function

$$\begin{aligned} L_{\text{lasso}}(\theta_0, \theta_1, \dots, \theta_d) &= \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 + \lambda \sum_{i=1}^d |\theta_i| \\ &= \text{MSE} + \lambda \|\theta\|_1 \end{aligned}$$

where  $\theta = (\theta_1, \dots, \theta_d)$ ,  $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_d x_d$ , and  $\lambda$  is a hyperparameter to be determined using cross-validation. The penalty forces some of the coefficients to be exactly 0 when  $\lambda$  is large.

- \* Note that lasso (Least Absolute Shrinkage and Selection Operator) regression performs variable selection since it eliminates weights of the least important predictors. Therefore, lasso models are generally much easier to interpret than those produced by ridge regression.
- \* There is no analytical solution to lasso regression since  $L_{\text{lasso}}$  is not differentiable. However, gradient descent works fine by using a subgradient

$$\frac{d}{dz}|z| = \text{sign}(z) = \begin{cases} 1, & z > 0 \\ 0, & z = 0 \\ -1, & z < 0 \end{cases}$$

- \* We can implement lasso regression in sklearn using
  - `from sklearn.linear_model import Lasso`
  - `from sklearn.linear_model import SGDRegressor(penalty="l1")`

– Elastic Net

The regularization term is a mix of ridge and lasso regularization terms.

The cost function is defined by

$$L_{\text{elastic net}}(\theta_0, \theta_1, \dots, \theta_d) = \text{MSE} + r \lambda \sum_{i=1}^d |\theta_i| + \frac{1-r}{2} \lambda \sum_{i=1}^d \theta_i^2$$

When  $r = 0$ , elastic net is the same as ridge regression, and when  $r = 1$ , it becomes lasso regression. To implement elastic net using sklearn, use

`from sklearn.linear_model import ElasticNet`

– **Remarks:**

- \* Lasso regression performs better than ridge regression when a smaller number of predictors have substantial coefficients. Ridge regression performs better when the output is a function of many predictors all with coefficients of roughly same size. Cross-validation can be used to determine which approach is better for a given dataset.
- \* It can be shown that ridge and lasso regression coefficients solve the minimization problems

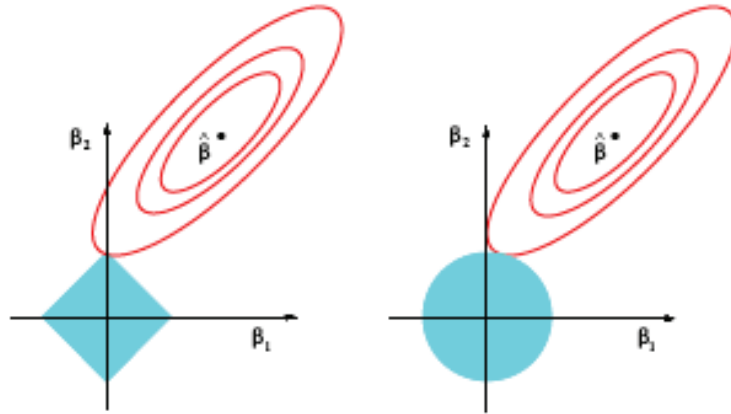
$$\min_{\theta_1, \dots, \theta_d} \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 \quad \text{subject to} \quad \sum_{i=1}^d \theta_i^2 \leq s$$

and

$$\min_{\theta_1, \dots, \theta_d} \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 \quad \text{subject to} \quad \sum_{i=1}^d |\theta_i| \leq s$$

for some value of  $s$  that exists for every value of  $\lambda$ .

(In the following diagram, replace  $\beta$  by  $\theta$ .)



**Figure 3.12:** *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions  $|\beta_1| + |\beta_2| \leq t$  and  $\beta_1^2 + \beta_2^2 \leq t^2$ , respectively, while the red ellipses are the contours of the least squares error function.*

<https://online.stat.psu.edu/stat857/node/158/>

**References and Reading Material:**

- [1] *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, by Geron, pages 134-142
- [2] *An Introduction to Statistical Learning*, by James et al., pages 203-224