# Lecture 14: Model tuning with Cross Validation

So far we have seen that SL models can perform wildly differently on training & testing data sets. This gets even worse if training set is small or if training & test sets have wildly different distributions.

$y \in \{-1, +1\}$

$\text{sign}(\hat{f}(x)) \approx y$

$\{x, y\}$

$\hat{f}(x) = \sum_{j=0}^{J-1} \beta_j \psi_j(x)$,

minimize $\|\hat{f}(X) - Y\|^2 + \lambda \|\beta\|^2$ over $\beta$

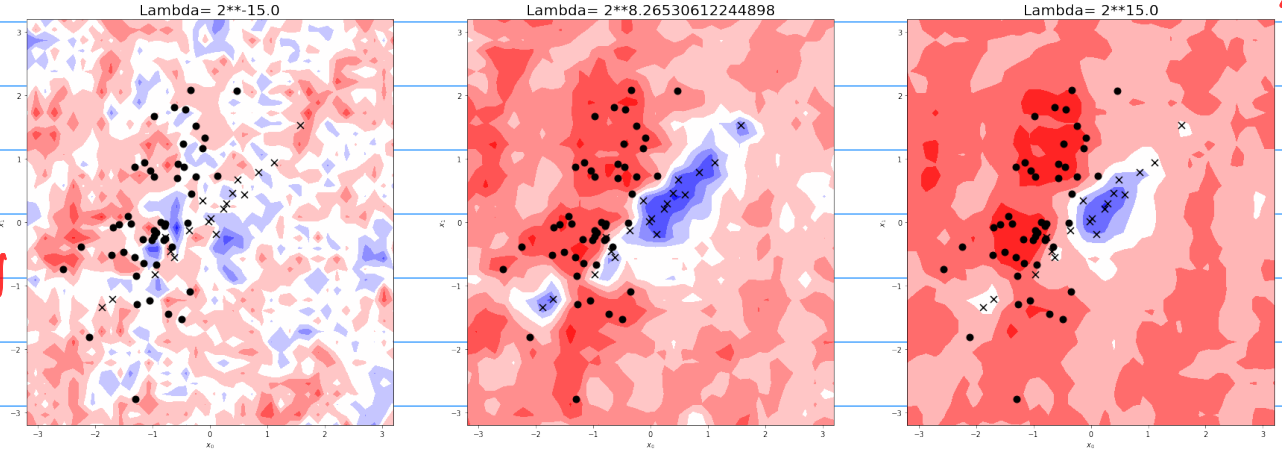
Additionally, SL models are very sensitive to the choice of parameters such as $\lambda$, the penalty parameter in Ridge regression. This sensitivity is very high if our model is complicated & has many features.

ex Consider our data set from prev. two lectures.

Good

too simple

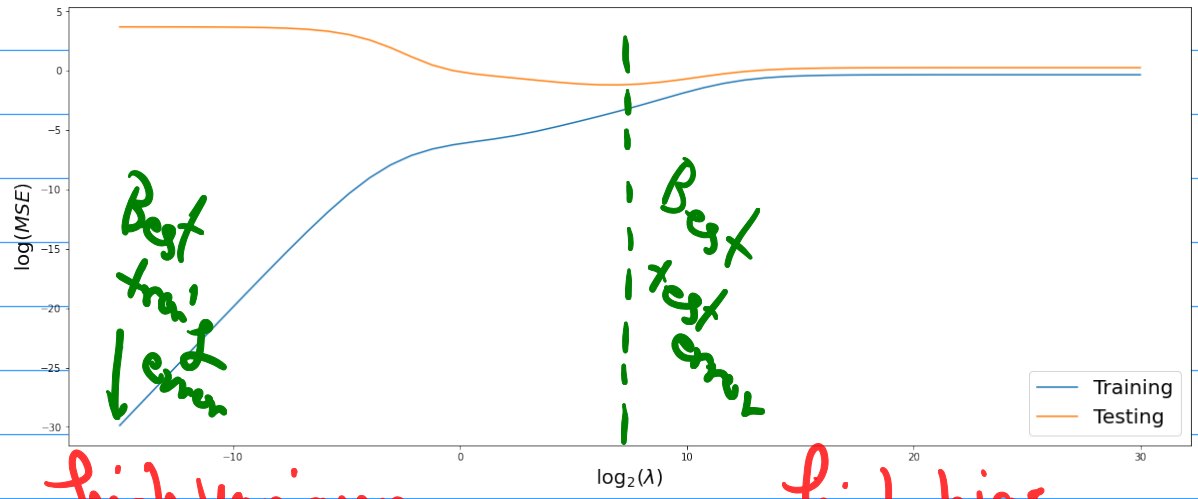
Too complicated




$$\hat{f}(x) = \hat{\beta}_0 + \sum_{j=1}^{200} \hat{\beta}_j \psi_j(x), \quad \psi_j(x) = \cos(w_1^T x) \sin(w_2^T x)$$

$w_1, w_2$ - random normal vectors.

# "Bias-Variance trade-off"




high variance ← Complicated model Overfitted

high bias → simple model Underfitted

In real world we cannot see the test error so we essentially have access only to the train error & the train data set.

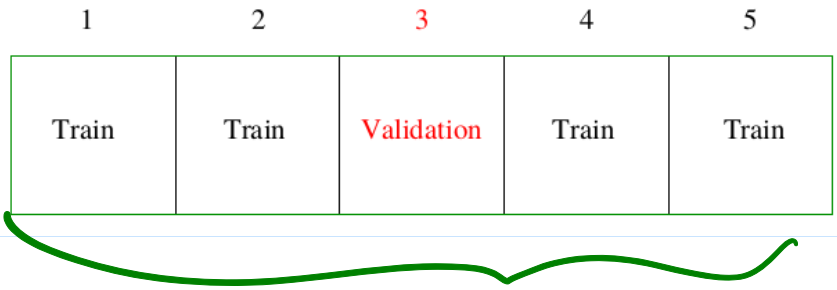
so the question is, what is a good strategy for choosing a good value of $\lambda$?

(See Ch7 of Hastie & Tibshirani for an in depth discussion).

## 14.1 Cross-Validation

In the ideal setting we would have a large train set & a large test set & use both to tune $\lambda$ & train the SL model. But if we don't have high quality test sets, the most intuitive & simple thing we can do is to split the train set into pieces.

Some pieces will be used for train & others will be used for testy. This is precisely the idea of K-fold Cross Validation (CV).




Here is an example of the CV recipe (others exist too).

- take train data

$$X = \{\underline{x}_0, \underline{x}_1, \cdots, \underline{x}_N\}$$
$$Y = \{y_0, \cdots, y_N\}$$

- Randomly permute the $(\underline{x}_j, y_j)$ pairs.

$$\tilde{X} = \{\tilde{\underline{x}}_0, \tilde{\underline{x}}_1, \cdots, \tilde{\underline{x}}_{N-1}\} = \{\underline{x}_{10}, \underline{x}_{01}, \cdots, \underline{x}_{13}\} \quad \text{ex}$$
$$\tilde{Y} = \{\tilde{y}_0, \cdots, \tilde{y}_{N-1}\} = \{y_{10}, y_{01}, \cdots, y_{13}\}$$

- split $\tilde{X}$ & $\tilde{Y}$ into $K$-subsets, $\alpha = \frac{N}{K}$

$$\tilde{X}_k = \{\tilde{\underline{x}}_{\alpha k+j}, \cdots, \tilde{\underline{x}}_{\alpha(k+1)-1}\}, \; j = 0, \cdots, \alpha-1,$$

$$\tilde{Y}_k = \{\tilde{y}_{\alpha k+j}, \cdots \tilde{y}_{\alpha(k+1)-1}\}, \; k = 0, \cdots, K-1$$

- Iterate over $k = 0, \dots, K-1$ & fit the model (solve SL problem) to the traing data with the k-th fold $(\tilde{X}_k, \tilde{Y}_k)$ removed.

Notation $(\tilde{X}_{-k}, \tilde{Y}_{-k}) = (\tilde{X}, \tilde{Y}) \setminus (\tilde{X}_k, \tilde{Y}_k)$

This will result in K models which we denote as $\hat{f}_{-k}$, $k = 0, \dots, K-1$.

- Finally, define the CV prediction error (CV cost) of $\hat{f}$ as

$$CV(\hat{f}) := \frac{1}{N} \sum_{k=0}^{K-1} \| \hat{f}_{-k}(\tilde{X}_k) - \tilde{Y}_k \|^2$$

* Note the ambiguity in the language & notation used above. By $\hat{f}$ we really mean a family of models, such as ridge regression parameterized by respective feature maps & choices of penalty parameters $\lambda$.

In the particular case of the penalty param. we write

$$CV(\hat{f}, \lambda) = \frac{1}{N} \sum_{k=0}^{K-1} \| \hat{f}_{-k}(\tilde{X}_k, \lambda) - \tilde{Y}_k \|^2$$

to emphasize the $\hat{f}_{-k}$ are trained using the same parameter $\lambda$.

We are now in position to find an optimal choice of $\lambda$ by minimizing the CV loss,

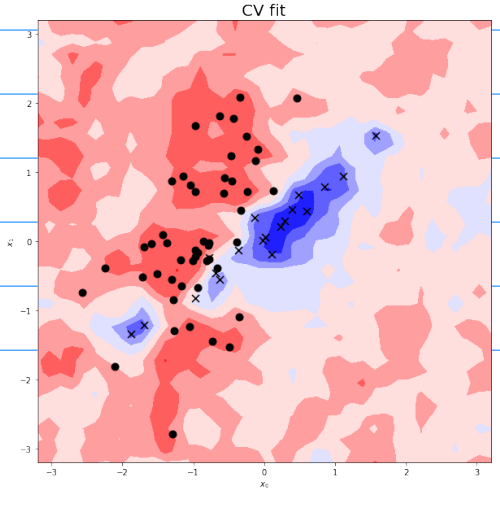
$$\lambda^* = \underset{\lambda \in (0,+\infty)}{\operatorname{argmin}} \; CV(\hat{f}, \lambda)$$

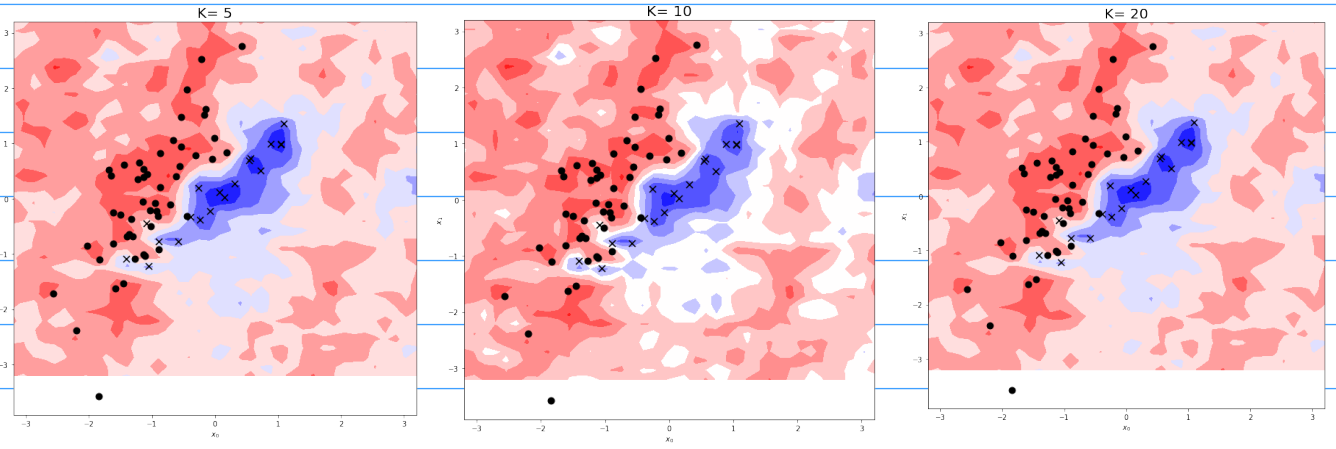
Observe that CV is not just defined for ridge regression & param $\lambda$. It is much more general & can be used to tune all sorts of things in your model. Such as number of features, shape of features, etc.

Lucky for you most ML packages come with CV train built in or at least have CV functions for tuning model parameters. Scikitlearn already has CV implemented for Ridge

$\lambda^* \approx 2^{8.8}$

Using K=1, ie, leave-one out CV.






# A few things to keep in mind:

- The term "hyper-parameter" is often used in ML literature to refer to parameters in a model that are not trained/fit directly to the data. This is not a well-defined concept. For example $\lambda$ is a hyper parameter, the functions $\psi_j$ chosen in our regression model are also hyperparameters.
- In this light, CV is a "hyper-param. learning/tuning algorithm". In principle one can use CV to tune large families of hyper-param. But most packages such as scikitlearn accomodate restricted families.
- CV is not the only method for this task & many other techniques exist. This is very much a hot research topic.