# Machine Learning - Michaelmas Term 2017
# Lecture 6 : Regularization, Validation, Model Selection

Lecturers: Christoph Haase & Varun Kanade

## 1 Regularization

Let us now consider a few approaches to reducing overfitting. Of course, one approach is to reduce the number of features used in the model, which in turn reduces the number of model parameters and hence overfitting. However, this is somewhat less satisfying as this might leave our models unable to capture interesting relationships in the data. For example, it is not possible to know *a priori* which higher-degree monomials may be important when using polynomial basis expansion. Thus, we add all terms up to a certain degree.

However, as discussed in the previous lecture, having a large number of (possibly irrelevant) features makes the learning algorithms attempt to fit noise. What we would like is a penalty to be imposed for putting weights on features that contribute little to predicting the signal. However, we don't know which features are irrelevant, and so we add a weight penalty for every feature. The optimization objective is now a combination of the loss and penalty term; the optimization procedure has to balance the tradeoff between minimizing the loss and the penalty.

### 1.1 Ridge Regression

In ridge regression, we use the least squares objective and add a penalty on the sum of the squares of the weight parameters. For $\lambda > 0$,

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \sum_{i=1}^{D} w_i^2 \tag{1}$$

Before proceeding to find the $\mathbf{w}$ that minimises $\mathcal{L}_{\text{ridge}}(\mathbf{w})$, a few words are in order. First, notice that we've left the $w_0$ term (for the constant 1 feature) out of the penalty. We think of the magnitudes of the weights $w_i$ as a measure of the complexity of the model. However, a translation of the output does not correspond to any additional model complexity. As a more concrete example, if we think of predicting the temperature using measurements of pressure, moisture, *etc.*, we may choose to output the answer in $^\circ C$ (celsius) or $K$ (kelvin). The fact that we need to add 273 to every output to get the value in $K$ does not make the model any more complex!

#### Standardizing Inputs

Likewise, let's consider the inputs $\mathbf{x}$. Let's consider a very simple model, $\widehat{y} = w_0 + w_1 x$, where $x$ is the temperature measured in $^\circ C$ (celsius). Now, if instead we use $x'$ which is the temperature in $^\circ F$ (fahrenheit), the model becomes $\widehat{y} = \left(w_0 - \frac{160}{9} w_1\right) + \frac{5}{9} w_1 x'$. Thus, in one case, we would get the term $w_1^2$, in the other $25 w_1^2/81$, which is less than one third of $w_1^2$.

To avoid issues of scaling and translation, it is good practice to standardise all the input variables, make them have mean 0 and variance 1 before fitting a model to the data. Don't forget to apply the same transformation to the test data!

If in addition we also centre the output variables, $y_i$s, then in the case of ridge regression we will always get $w_0 = 0$ (Problem Sheet 2). Thus, we can succinctly re-write the objective $\mathcal{L}_{\text{ridge}}$ in vector form as shown below.

**Remark 1.** *Don't forget to standardize the inputs and center the outputs before applying the estimates derived in this section!*

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w} \tag{2}$$

Suppose the data is $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^{N}$ with inputs standardised and output centered. Let us now take the gradient of the above expression to find the optimal $\mathbf{w}$.

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w} - 2\mathbf{y}^{\mathsf{T}}\mathbf{X}\mathbf{w} + \mathbf{y}^{\mathsf{T}}\mathbf{y} + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w}$$

Taking the gradient with respect to $\mathbf{w}$

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L}_{\text{ridge}} &= 2(\mathbf{X}^{\mathsf{T}}\mathbf{X})\mathbf{w} - 2\mathbf{X}^{\mathsf{T}}\mathbf{y} + 2\lambda\mathbf{w} \\ &= 2\left(\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}_D\right)\mathbf{w} - \mathbf{X}^{\mathsf{T}}\mathbf{y}\right)\end{aligned}$$

Above $\mathbf{I}_D$ is the $D \times D$ identity matrix. We set the gradient to $\mathbf{0}$ and solve for $\mathbf{w}$

$$\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}_D\right)\mathbf{w} = \mathbf{X}^{\mathsf{T}}\mathbf{y}$$

to obtain the solution

$$\mathbf{w}_{\text{ridge}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}_D\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y} \tag{3}$$

Unlike in the case of the least-squares estimate, we do not need to be concerned about whether or not the matrix $(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}_D)$ is invertible. For $\lambda > 0$, this is always invertible (Exercise: Show that this is the case.). The quantity $\lambda$ controls the tradeoff between minimising the prediction error and reducing the model complexity. As $\lambda \to 0$, we recover the least-squares estimate, where we are only concerned with minimising the sum of the squares of the residuals. On the other hand as $\lambda \to \infty$, we will get $\mathbf{w} = \mathbf{0}$ as the solution. Clearly, this is not desirable and the goal is to pick a $\lambda$ that balances the two parts of the objective more evenly.

We'll return to the question of choosing $\lambda$ shortly, but let's look at an alternative formulation of ridge regression.

$$\begin{aligned}&\text{minimise} && (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &\text{subject to:} && \mathbf{w}^{\mathsf{T}}\mathbf{w} \le R\end{aligned}$$

It is not that hard to show that these two formulations are equivalent (the form in Eq. (2) is called the Lagrangean form), however, we'll leave that as an exercise for the interested student. When $R \to \infty$, there is essentially no constraint and the solution is that given by the least squares estimate, let's call it $\mathbf{w}_{\text{LS}}$ (in fact, this is the case for any $R \ge \mathbf{w}_{\text{LS}}^{\mathsf{T}}\mathbf{w}_{\text{LS}}$). For smaller $R$, we'll get a solution $\mathbf{w}$ such that $\mathbf{w}^{\mathsf{T}}\mathbf{w} = R$, and the contour curves of $\mathbf{w}^{\mathsf{T}}\mathbf{w}$ and $(\mathbf{X}\mathbf{w}-\mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w}-\mathbf{y})$ are tangent at the solution. Figure 1(a) shows the solution to the objective function as a function of $R$ (equivalently $\lambda$). Figure 1(b) shows how the weights on features vary as a function of $\lambda$ for ridge regression performed on the diabetes dataset (available in scikit-learn).

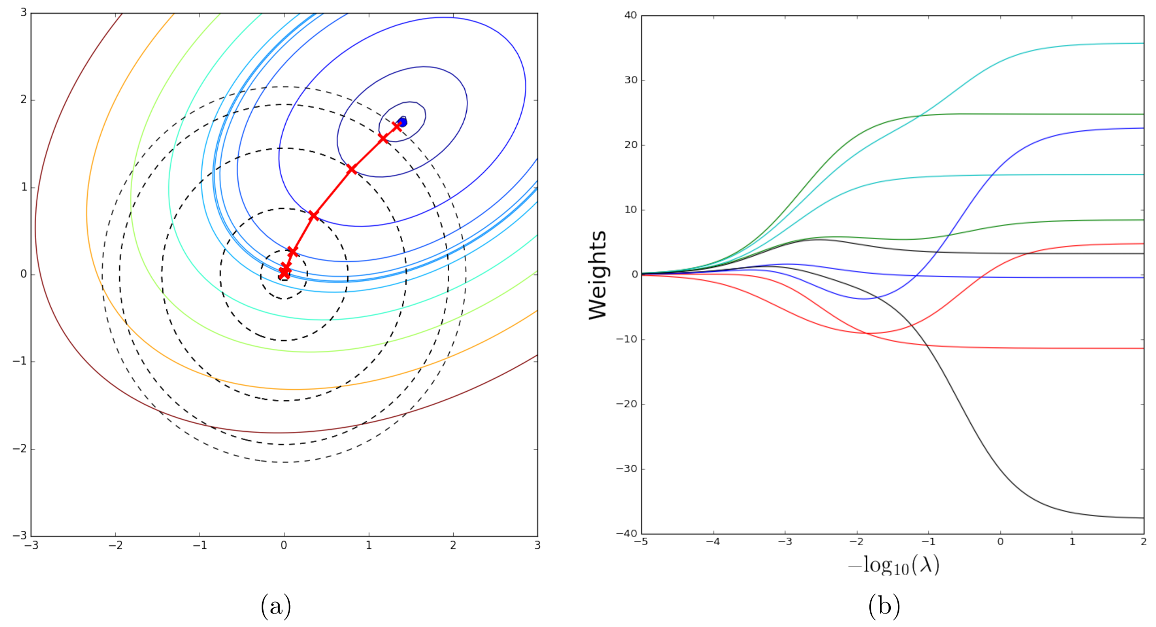


Figure 1: (a) Solution to ridge regression as a function of $R$ (or $\lambda$) (b) Plot showing the weights of each feature obtained in ridge regression as a function of $-\log(\lambda)$.

## 1.2 The Lasso

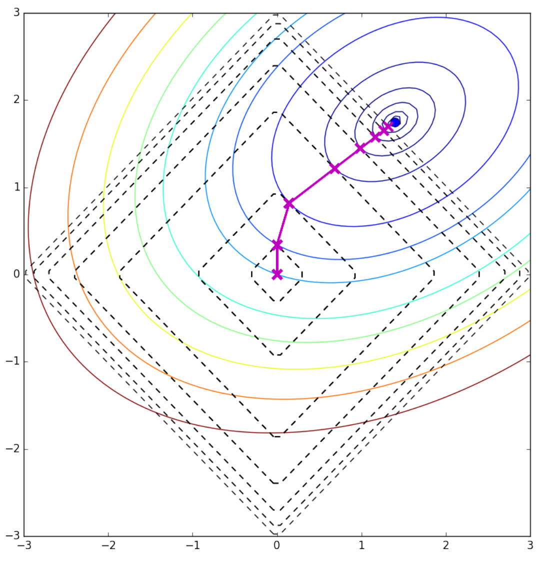
The Lasso (which stands for least absolute shrinkage and selection operator) is an alternative way to penalise model complexity. In the Lasso objective, instead of adding the sum of the squares of the weights as a penalty term, we add the sum of the absolute values of the weights, *i.e.*, $\sum_{i=1}^{D} |w_i|$, as a penalty term to the objective function. As in the case of Ridge regression, it is a good idea to standardize the input variables. For $\lambda > 0$, the lasso objective is expressed as:

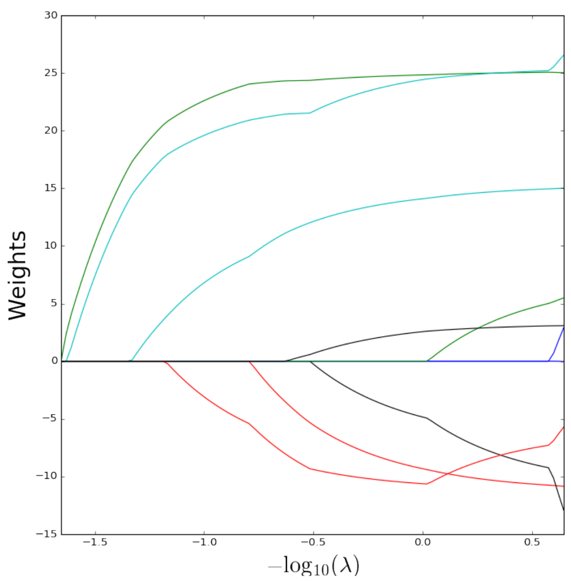
$$\mathcal{L}_{\text{lasso}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \sum_{i=1}^{D} |w_i| \tag{4}$$

Unlike Ridge Regression, there is no closed form solution for $\mathbf{w}$ that minimises the Lasso objective; we have to resort to general optimisation methods. Clearly, as in the case of Ridge Regression, when $\lambda = 0$, we recover the least squares solution, whereas when $\lambda \to \infty$, we get the solution $\mathbf{w} = \mathbf{0}$. The equivalent constrained optimisation form for the Lasso objective is the following:

$$\begin{aligned} \text{minimise} \qquad & (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ \text{subject to:} \qquad & \sum_{i=1}^{D} |w_i| \leq R \end{aligned}$$

As $R \to \infty$ (which is equivalent to $\lambda \to 0$), we recover the least squares solution; when $R = 0$ (which is equivalent to $\lambda \to \infty$) we get $\mathbf{w} = 0$ as the solution. For intermediate values of $R$ we get a solution such that $\sum_{i=1}^{D} |w_i| = R$ and the contour curves of $\sum_{i=1}^{D} |w_i|$ and $(\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}} (\mathbf{X}\mathbf{w} - \mathbf{y})$ are tangent at the solution. However, the contour curves of $\sum_{i=1}^{D} |w_i|$ have corners, where the tangent is not well defined. This happens when some of the $w_i$ are exactly zero! As a result, the model fit by Lasso may have several zero weights, and hence it can be seen as a form of *variable selection* (hence the name Lasso). Figure 2(a) shows the solution to the Lasso objective as a function of $R$ (equivalently $\lambda$). Figure 2(b) shows how the


(a)



(b)

Figure 2: (a) Solution to the Lasso as a function of $R$ (or $\lambda$) (b) Plot showing the weights of each feature obtained using Lasso as a function of $-\log(\lambda)$.

weights of features vary as a function of $\lambda$ for Lasso performed on the diabetes dataset (available in scikit-learn). We can see that for $\lambda$ up to a certain level, several of the weights are exactly 0; compare this to Ridge Regression in Fig. 1(b).

### 1.3 Discussion

Let us return to the toy problem introduced in Section 2.2 in the previous lecture in which the learning algorithm used irrelevant features in order to fit noise. Figure 3 shows the training and test error for least squares, Ridge Regression and Lasso on the problem. We see that both least squares and Ridge Regression do quite badly as we allow the model to use more and more irrelevant features. Recall that the first feature is the only relevant feature; but as we introduce more irrelevant features these models start fitting noise. Lasso actually does very well even when all 100 features (99 of which are irrelevant) are used. The reason why Ridge Regression performs poorly, while Lasso does very well, in this case is probably due to the fact that it is important to have most features other than the first one have zero weight. As discussed above this is more likely to be the case with Lasso than with Ridge Regression because of the corners in the Lasso penalty term. This can be seen in Figures 3(b), (c), (d) where the actual weights on the features are shown as a function of the number of features used in the model. As more and more irrelevant features are allowed, both the least squares and Ridge Regression model put weight on irrelevant features; this happens to a much lesser extent in the case of Lasso.

In the next lecture, we will see a good theoretical justification for Ridge Regression through the lense of the Bayesian Approach to Machine Learning.

## 2 Model Selection

Let us now return to the question alluded to several times so far about how to select various hyperparmeters such as $\lambda$ (in Ridge and Lasso), the degree (in polynomial basis expansion), and the width parameter $\gamma$ (in kernel regression). In general, as we start using more and more complex models to fit the data, we might have more hyperparameters that need to be selected.

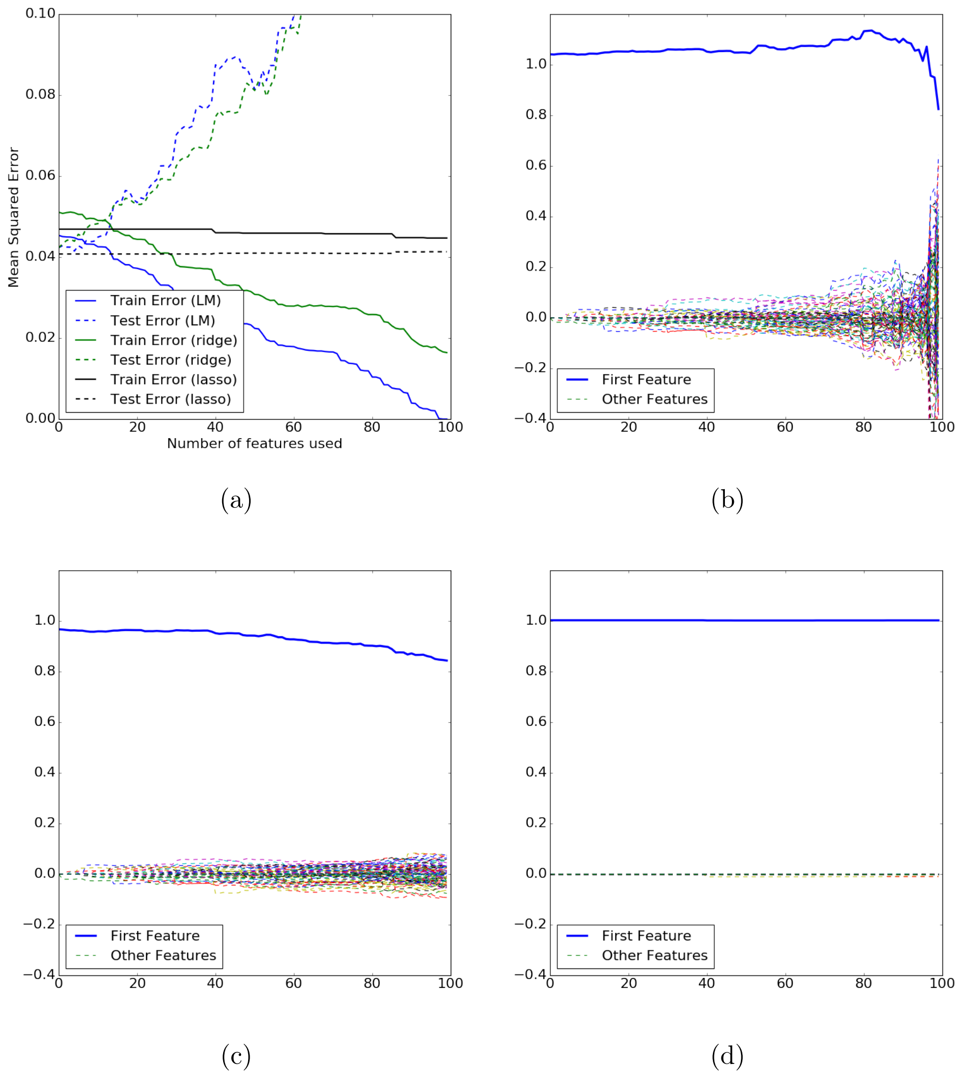


Figure 3: All plots concern the toy problem introduced in Lecture 5 (a) Training and test error for least squares, Ridge Regression and Lasso as a function of the number of features used. (b) Weights as a function of the number of features used in the model for least squares (c) Weights as a function of the number of features used for Ridge Regression (d) Weights as a function of the number of features used for Lasso

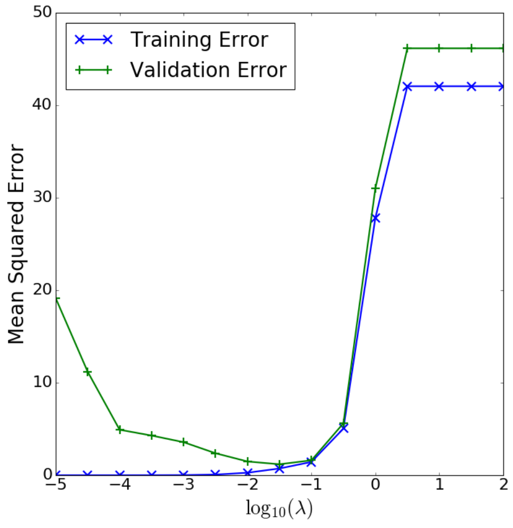
## 2.1 Validation

Let us start with the setting where the data is relatively plenty. In this case, we divide the data into a *training* set and a *validation* set. The training set will be used to actually train the model parameters (such as $\mathbf{w}$, not the hyperparameters!) and the validation set will be used to pick suitable values for the hyperparameters. Of course, in reality we care about testing our model on completely unseen data. When applying machine learning in the real-world, tests will present themselves! However, in academic settings, we can keep aside yet another part of the data called the *test* set, where we'll evaluate the performance of our models after performing training and validation. In academic settings, the test set should not be touched at all except for reporting the performance of the models!

Now, let's say we have only one hyperparameter to choose, say $\lambda$. We start with a possible set of values that we may want to assign to $\lambda$, *e.g.,* $\lambda \in \{0.01, 0.1, 1, 10, 100\}$. In general, the range of hyperparameters should be chosen depending on the sensitivity of the trained model to

| $\lambda$ | training error(%) | validation error(%) |
|---|---|---|
| 0.01 | 0 | 89 |
| 0.1 | 0 | 43 |
| 1 | 2 | 12 |
| 10 | 10 | 8 |
| 100 | 25 | 27 |

(a)



(b)

Figure 4: (a) (Made-up) Errors on training and validation sets. (b) Curves showing error on training and validation sets as a function of $\lambda$ for Lasso.

the hyperparameters; thus they may be on a log scale, linear scale, etc.[1] Fig 4(a) summarises (made-up values of) the error on the training set and the validation set for some model. Since, we've not used the validation set as part of the training set, we'll trust the performance on the validation set as being more representative than that on the training set. On the training set, we may have overfit depending on the complexity of the model. Thus, in this case we'll pick $\lambda = 10$ as the value for the hyperparameter. Once the value of the hyperparameter is fixed, it is often a good idea to train the model using all available data (including the one previously used for validation), since the more data we use in the training the more accurate our model is likely to be. If we plot the curves for the training and validation error as a function of the hyperparameter, the validation error curve typically has a $U$-shape, where the validation error is high on one-side because of overfitting (where the training error is typically low) and on the other side because of underfitting (where the training error is also high). The optimal hyperparameter to be chosen is at the bottom of the $U$ shape (see Fig. 4(b)). When there are multiple hyperparameters to be chosen, we can make a "grid" of all possible combinations of values for the hyperparameters and pick the combination that is most suitable; this is called *grid search*. Grid search may be very costly even when the number of hyperparameters is relatively modest because of the exponential size of the search space. Techniques such as random search, or Bayesian black-box optimisation can be applied; we will not cover them in this course.

When data is scarce, keeping aside a validation set is not a good idea. In this case, it is more common to divide the data into $K$ parts (called folds) and then use $K - 1$ parts as the training set and use the performance on the $K$th part for validation (see Fig. 5). This is then repeated across all the folds and the average error on the fold used as the validation set (over the $K$ different choices) is used as a proxy for the validation error. This has the effect of reducing the variance in the validation error and hence is usually more suitable than using a small validation set. It is common to use $K = 5$ or $K = 10$. One extreme case is to use $K = N$, where $N$ is the number of datapoints. In this case, in each instance we are training on $N - 1$ datapoints and testing the performance on the $N^{th}$ one, averaging over all possible choices. This method is called leave-one-out-cross-validation or LOOCV for short. However, this means that

[1]While looking at existing literature will provide clues as to what scales should be chosen for various hyperparmeters, there will be times when this will only be clear after one round of validation.

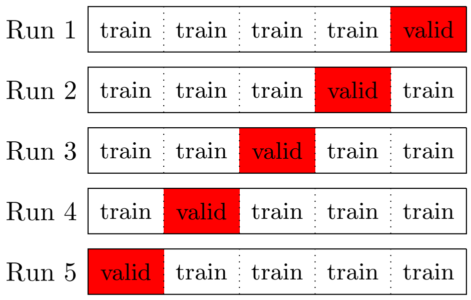


Figure 5: 5-fold cross validation.

we are running the training algorithm $N$ times which can be computationally expensive. Some methods (such as least squares) have the property that the influence of one datapoint from the trained model can be quickly removed without explicitly re-training, however in general this is unlikely to be the case.

## 2.2 Feature Selection

As discussed earlier, having a small training set with a large number of features may result in overfitting due to a learning algorithm trying to fit noise. There are situations were it is conceivable due to prior knowledge of the application domain that only a small number of features are actually relevant for the learning task. Thus, if we knew which features were irrelevant, we could remove them from our training set and hence prevent overfitting from happening. If we have $n$ features, there are $2^n$ different subsets of features that we could remove and check in each case whether the generalization error decreases. It is obvious that this approach is infeasible already for relatively modest $n$. In this section, we describe two computationally feasible heuristics that are often applied in practice for discovering relevant subsets of features.

The first is *forward search*, a greedy heuristic that is described in Algorithm 1, and commonly used in practice in order to circumvent searching through the exponential set of possible feature subsets. It wraps around an existing learning algorithm that is repeatedly executed in Line 5 of the algorithm for different subsets of features. Even though forward search avoids exploring $2^n$

**Algorithm 1** Forward search

```
1: $F \leftarrow \emptyset$
2: **do**
3:     **for all** $i \in \{1, \dots, n\} \setminus F$ **do**
4:         $F_i := F \cup \{i\}$
5:         $E_{F_i}$ := generalization error when trained using only features from $F_i$
6:     **end for**
7:     $F := F_i$ for the $i$ for which $E_{F_i}$ is minimal
8: **while** $F \neq \{1, \dots, n\}$          ▷ alternatively while $|F|$ is less than some fixed threshold
9: **return** the $F$ with the smallest generalization error encountered
```

subsets, in the version above it will still make $O(n^2)$ calls to the underlying learning algorithm, which can be too expensive.

An alternative, computationally less heavy, way of selecting potentially relevant features is *filter feature selection*. It is, informally speaking, based on the idea that some features $x_i$ provide more information about the output $y$ than others. One way to measure the contribution of a feature $x_i$ is to compute the *mutual information* between $x_i$ and the output $y$ (assuming for simplicity that $x_i$ and $y$ are discrete):

$$I(x_i, y) = \sum_{x_i \in X} \sum_{y \in Y} p(x_i, y) \cdot \log \frac{p(x_i, y)}{p(x_i) \cdot p(y)}.$$

The above probabilities can be obtained from their empirical distribution on the training set. Mutual information is a concept coming from information theory and provides a measure of how much bits of information one can obtain about a random variable through another.[2] If $x_i$ and $y$ are uncorrelated then $I(x_i, y) = 0$, whereas a strong correlation between $x_i$ and $y$ would result in a high value of $I(x_i, y)$ and would thus mean that $x_i$ is relevant for $y$. It should now be obvious how filter feature selection works: compute $I(x_i, y)$ for all $x_i$ and keep the $k$ features with the top $k$ values of $I(x_i, y)$. Of course, other scoring functions than $I(x_i, y)$ can be used.

### Discussion

The question of model selection is an important one. We've only seen the very basic approaches employed in practice. We'll return to these questions a few times later in the course. In particular, for classification problems, it is often necessary to treat errors of different kinds differently. Predicting that a tumor is malignant when it is not vs predicting a tumor is benign when it is not are not equally problematic errors! Thus, when performing model selection, it is important to assign different costs to different types of errors. In unsupervised learning, where we have no access to the ground truth the problem of model selection requires using different criteria.

The machine learning pipeline we've see so far is: get data, choose a model, train a model and select hyperparameters, and test the model. What happens when the test turns out to be bad? We have no option but to start from scratch; however, in general we may not always get new data. We may simply be left with the option of choosing a different model or hyperparameters and use the existing data. However, notice that we've already "seen" the test set, although only implicitly, when testing our model. Thus, through this process we've *leaked* some information from the test data to the machine learning pipeline and hence can no longer assume that our model is completely blind to the test set. When using the test set sparingly, this does not usually pose a huge problem; however, if the test set is not utilised carefully, this can lead to serious overfitting! For example, in Kaggle competitions it often happens that some top teams on the public leaderboard are not close to the top on the private leaderboard! This may happen because these teams are submitting too many entries and implicitly overfitting on the data used for the public leaderboard.[3]

## References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[2]Mutual information can also be expressed in terms of the Kullback-Leibler (KL) divergence: $I(x_i, y) = \mathrm{KL}(p(x_i, y) || p(x_i) \cdot p(y))$. The Kullback-Leibler convergence measures the difference between two probability distributions. The interested reader is referred to (Goodfellow et al., 2016, Chap 3.13) for a more formal treatment.

[3]An interesting blog post about topping the public leaderboard in Kaggle without reading the data is available here: `http://blog.mrtz.org/2015/03/09/competition.html`.