Machine Learning Review

Zhongjian Lin
University of Georgia

October 7, 2024

Machine Learning

Machine learning is about extracting knowledge from data. There are input and output data. There are two types of learning: supervised learning and unsupervised learning.

- Machine learning algorithms that learn from input-output pairs are called supervised learning.
- In unsupervised learning, only the input data is known and there is no known output data given to the algorithm.

In supervised learning, we have two types of models: classification and regression.

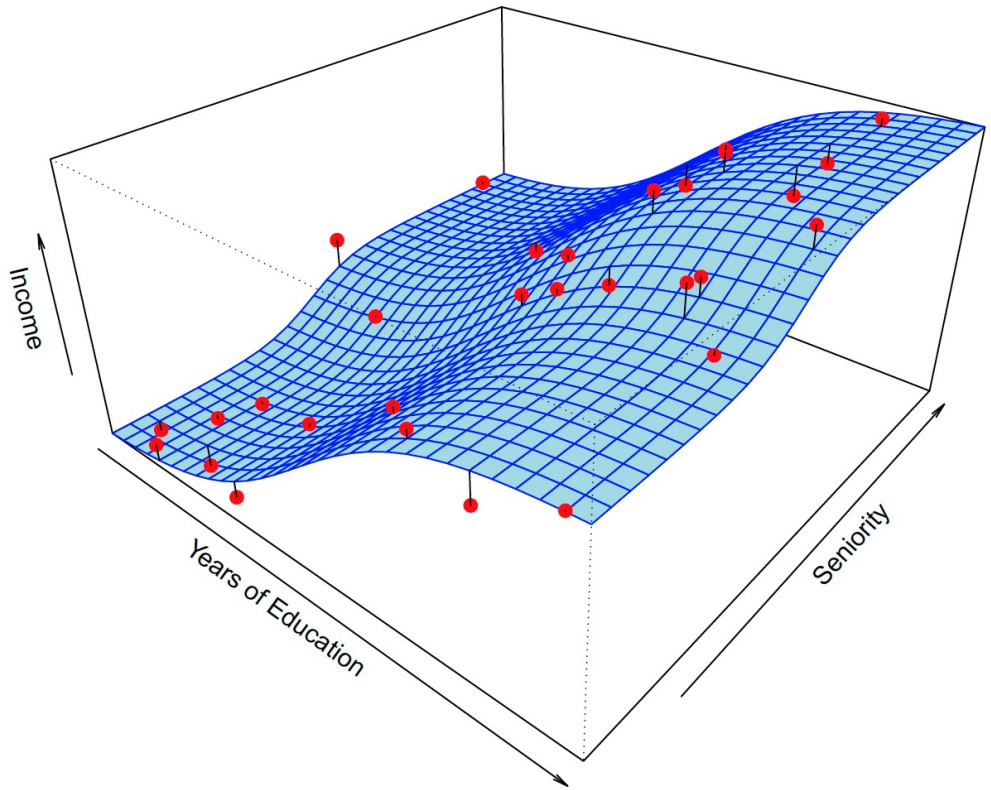
- Classification model is used to analyse data when output is discrete/categorical.
- Regression model is for continuous output.

Measuring Success: Training and testing data

- For classification, we use accuracy of prediction as the measurement of model performance.
- For regression, we use R^2/MSE as the measurement of model performance.

Model

f is some fixed but unknown function of X, and ϵ is a random error term, which is independent of X and has mean zero. In this formula, f represents the systematic information that X provides about Y. In general, the function f may involve more than one input variable. For example, we can plot *income* as a function of *years of education* and *seniority*. Here f is a two-dimensional surface that must be estimated based on the observed data.



Prediction and Inference

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. We can predict Y using

$$\widehat{Y} = \hat{f}(X)$$

In this setting, $\hat{f}$ is often treated as a black box.

Inference

We are often interested in understanding the association between Y and $X_1, \cdots, X_p$.

- Which predictors are associated with the response?
- What is the relationship between the response and each predictor?
- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

Accuracy of Prediction

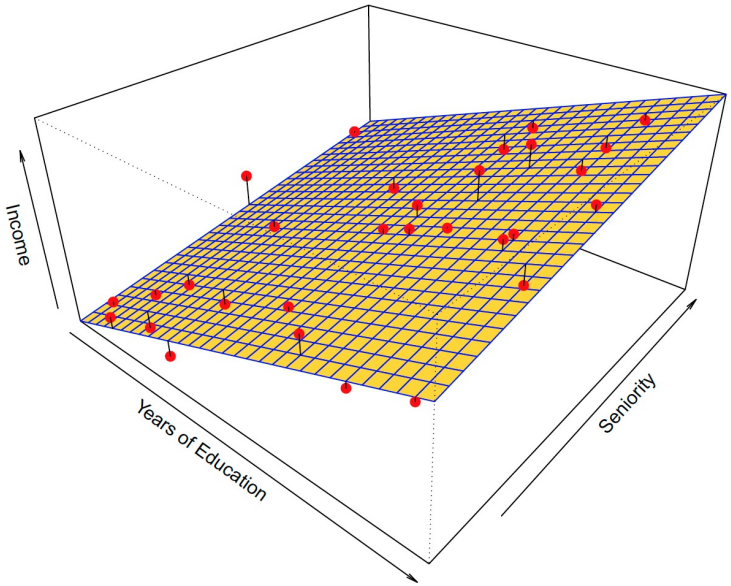
$$E[(Y-\widehat{Y})^2|X] = E\Big[(f(X)-\hat{f}(X)+\epsilon)^2|X\Big] = [f(X)-\hat{f}(X)]^2 + Var(\epsilon)$$

- Reducible error: $[f(X)-\hat{f}(X)]^2$
- Irreducible error: $Var(\epsilon)$

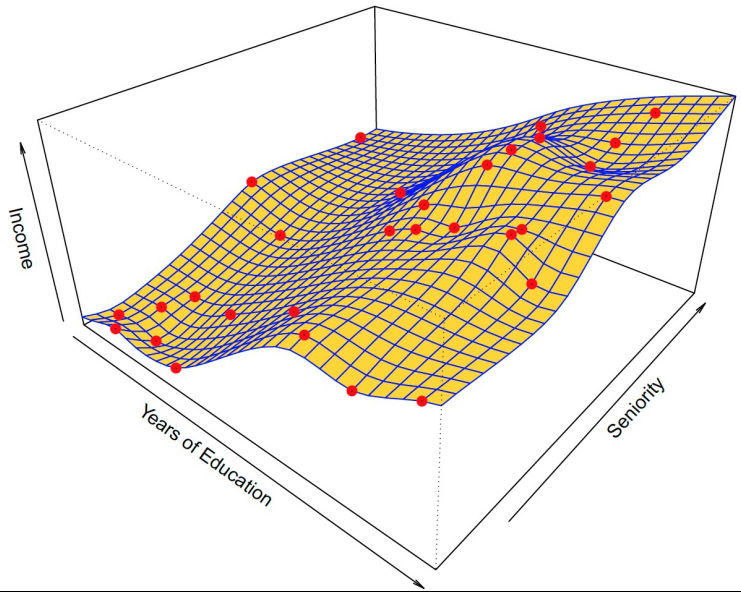
The focus of this course is on techniques for estimating f with the aim of minimizing the reducible error. It is important to keep in mind that the irreducible error will always provide an upper bound on the accuracy of our prediction for Y. This bound is almost always unknown in practice.

Parametric vs Non-parametric Estimation

Income–Years of Education and Seniority



[Parametric]



[Non-parametric]

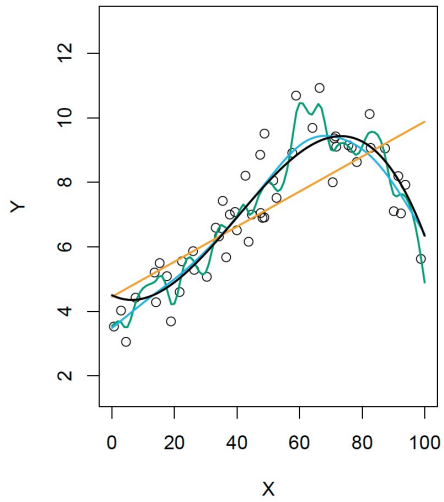
Accuracy and Interpretability

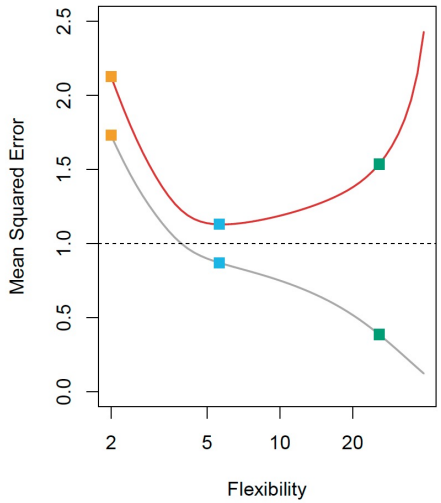
Of the many methods, some are less flexible, or more restrictive, in the sense that they can produce just a relatively small range of shapes to estimate f. One might reasonably ask the following question: *why would we ever choose to use a more restrictive method instead of a very flexible approach?*

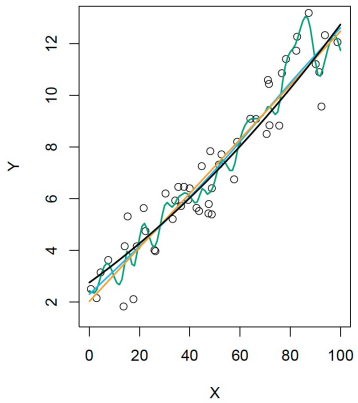
- For inference restrictive models are much more interpretable, e.g., linear regression.
- In some settings, however, we are only interested in prediction, and the interpretability of the predictive model is simply not of interest.
- When interpretability is not a concern, we might expect that it will be best to use the most flexible model available. Surprisingly, this is not always the case!

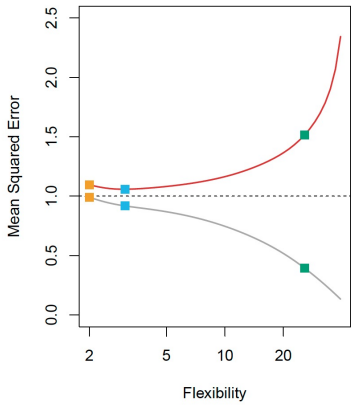
Overfitting

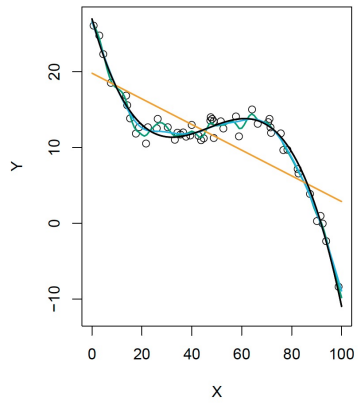
As model flexibility increases, the training MSE will decrease, but the test MSE may not. When a given method yields a small training MSE but a large test MSE, we are said to be *overfitting* the data.

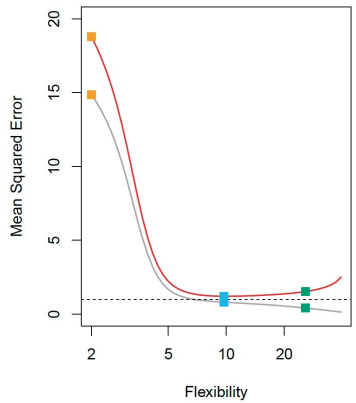












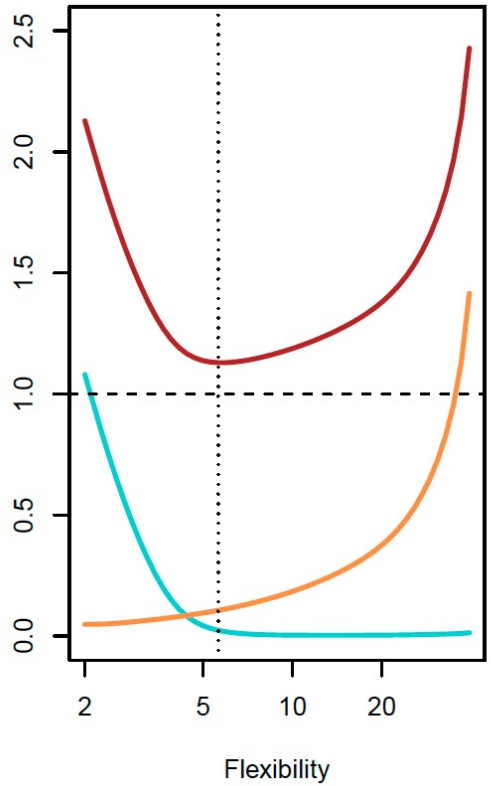
The Bias-Variance Trade-Off

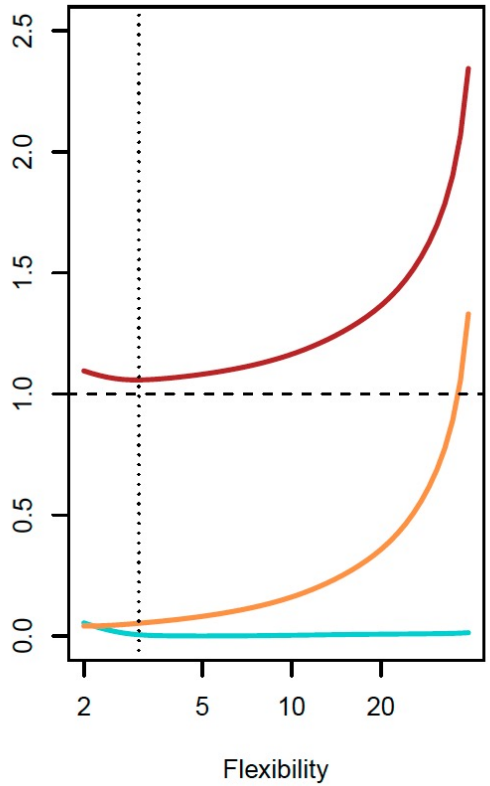
The U-shape observed in the test MSE curves turns out to be the result of two competing properties of machine learning methods. Suppose we train the data and get $\hat{f}$ using $(y_i, x_i)_{i=1}^n$. For a given value x_0, the expected test MSE, can always be decomposed into the sum of three fundamental quantities: the variance of $\hat{f}(x_0)$, the squared bias of $\hat{f}(x_0)$ and the variance of the error variance terms ϵ.

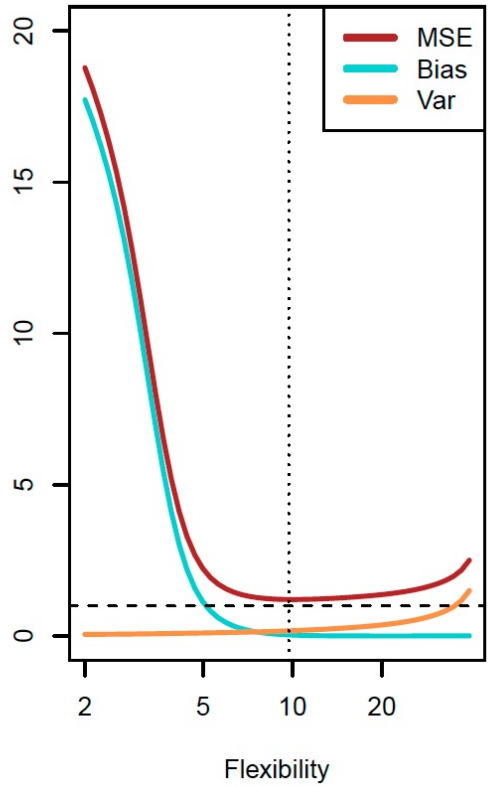
$$E[(y_0 - \hat{f}(x_0)]^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + var(\epsilon).$$

- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. In general, more flexible statistical methods have higher variance.
- *Bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

The Bias-Variance Trade-Off



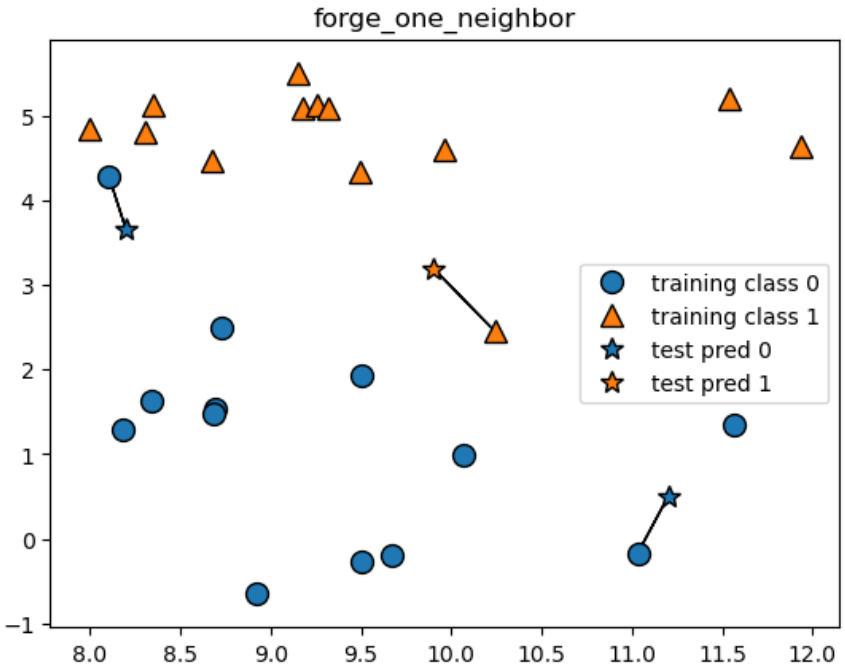




k-Nearest Neighbor

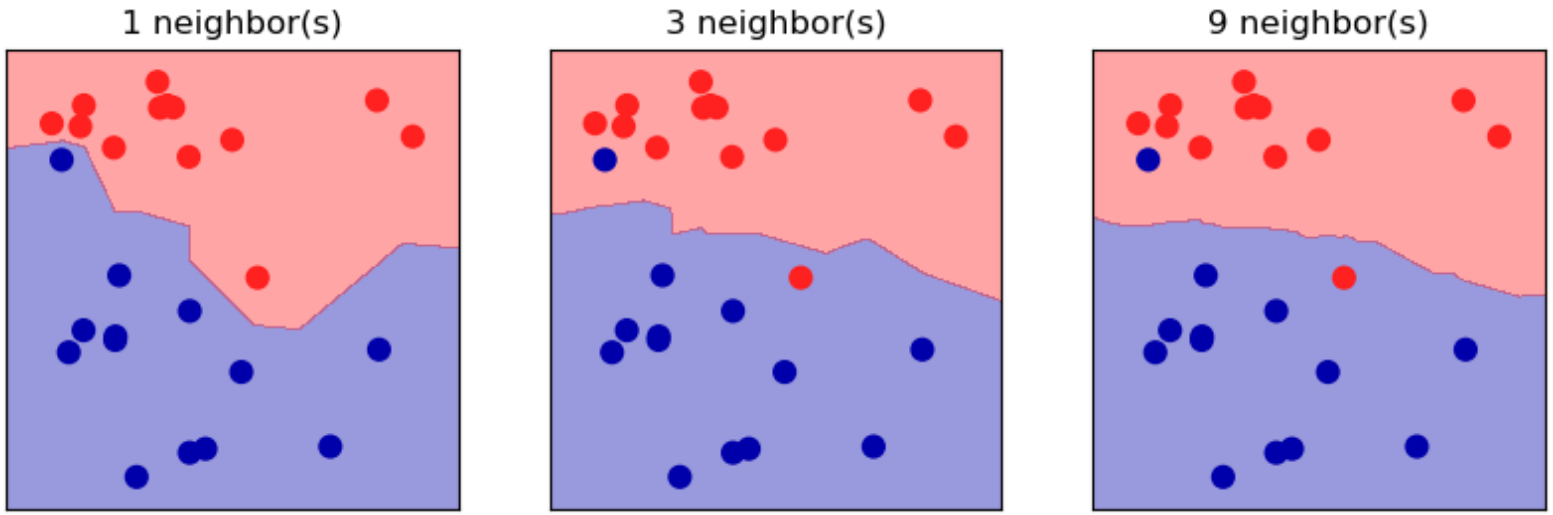
The k-Nearest Neighbors (kNN) algorithm is arguably the simplest machine learning algorithm. The algorithm finds the closest data points in the training dataset, it "nearest neighbors".
In its simplest version, the algorithm only considers exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for.

```
mglearn.plots.plot_knn_classification(n_neighbors=1)
plt.title("forge_one_neighbor");
```



Analyzing KNeighborsClassiier

For two-dimensional datasets, we can also illustrate the prediction for all possible test point in the xy-plane. We color the plane red in regions where points would be assigned the red class, and blue otherwise. This lets us view the decision boundary, which is the divide between where the algorithm assigns class red versus where it assigns class blue.



Ordinary Least Squares

For datasets with many features, linear models can be very powerful. There are many different linear models for regression. The difference between these models is how the model parameters β are learned from the training data, and how model complexity can be controlled. Linear regression or Ordinary Least Squares (OLS) is the simplest and most classic linear method for regression. Linear regression finds the parameters β that minimize the mean squared error between predictions and the true regression targets y on the training set.

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{1}$$

Ridge Regression

Ridge regression is also a linear model for regression, so the formula it uses to make predictions is still Formula (1), as for ordinary least squares. In Ridge regression,the coefficients β are chosen not only so that they predict well on the training data, but there is an additional constraint. We also want the magnitude of coefficients to be as small as possible; in other words, all entries of β should be close to 0.

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda(\beta^T \beta) \tag{2}$$

Intuitively, this means each feature should have as little effect on the outcome as possible (which translates to having a small slope), while still predicting well. This constraint is an example of what is called regularization (L2). Regularization means explicitly restricting a model to avoid overfitting.

LASSO

An alternative to Ridge for regularizing linear regression is the Lasso (least absolute shrinkage and selection operator). The lasso also restricts coefficients to be close to zero, similarly to Ridge regression, but in a slightly different way, called "L1" regularization.

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{k=0}^{p} |\beta_k| \tag{3}$$

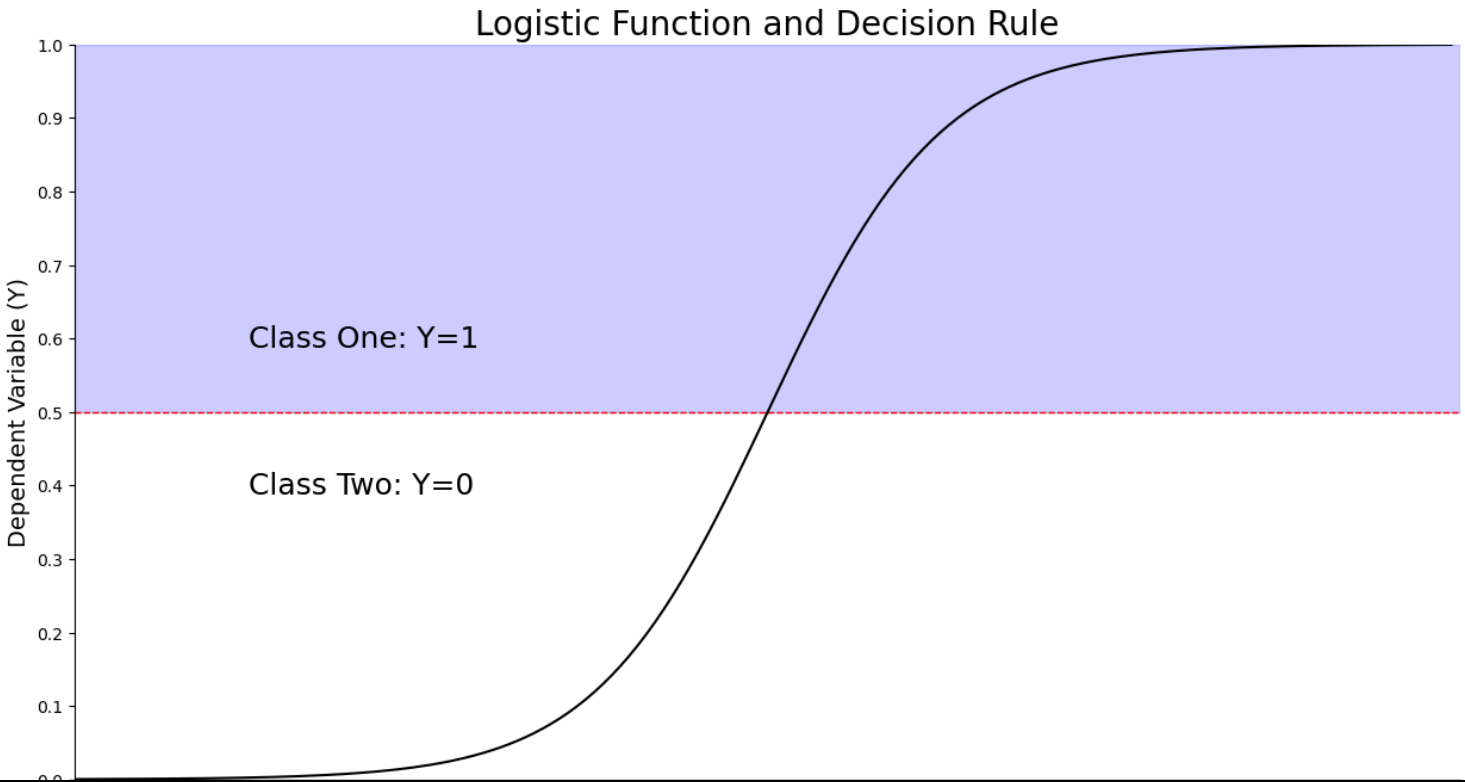
The consequence of l1 regularization is that when using the Lasso, some coefficients are exactly zero. This means some features are entirely ignored by the model. This can be seen as a form of automatic feature selection. Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of your model.

$$y = 1\{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon > 0\}. \tag{4}$$

When impose parametric distribution assumption on ε, i.e., $\varepsilon \sim Logistic(0, 1)$, we have

$$P(y = 1|X) = \frac{exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)}{1 + exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)}$$

The classifier gives us a set of outputs or classes based on probability. Here, we predict the outcome variable is true whenever $P(y = 1|X) \geq 0.5$.



Optimization (MLE)

When we fit the model to the training data, the algorithm chooses β such that:

$$\hat{\beta} = \arg\max_{\beta} L(\beta)$$

where

$$L(\beta) \equiv \sum_{i=1}^{n} y_i \cdot \log[P(y_i = 1|X_i)] + (1 - y_i) \cdot \Big(\log[1 - P(y_i = 1|X_i)] \Big)$$

is the log-likelihood function.

Logistic Regression with L_2 regularization

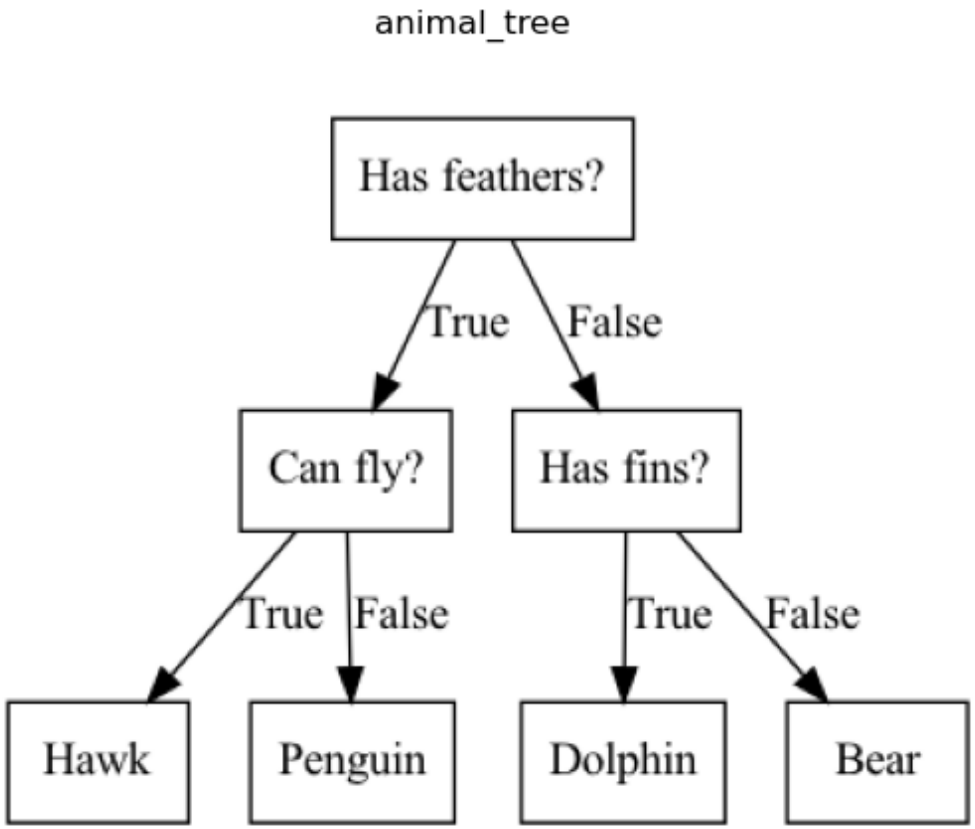
$$\hat{\beta} = \arg\max_{\beta} L(\beta) + C\beta'\beta$$

Logistic Regression with L_1 regularization

$$\hat{\beta} = \arg\max_{\beta} L(\beta) + C|\beta|$$

Decision Trees

Decision trees are a widely used models for classification and regression tasks. Essentially, they learn a hierarchy of "if-else" questions, leading to a decision.



Instead of building these models by hand, we can learn them from data using supervised learning.

Recursive Binary Splitting

The recursive binary splitting approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

1. We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
2. In greater detail, for any j and s, we define the pair of half-planes

$$R_1(j,s) = \{X|X_j < s\}, \quad R_2(j,s) = \{X|X_j \geq s\}.$$

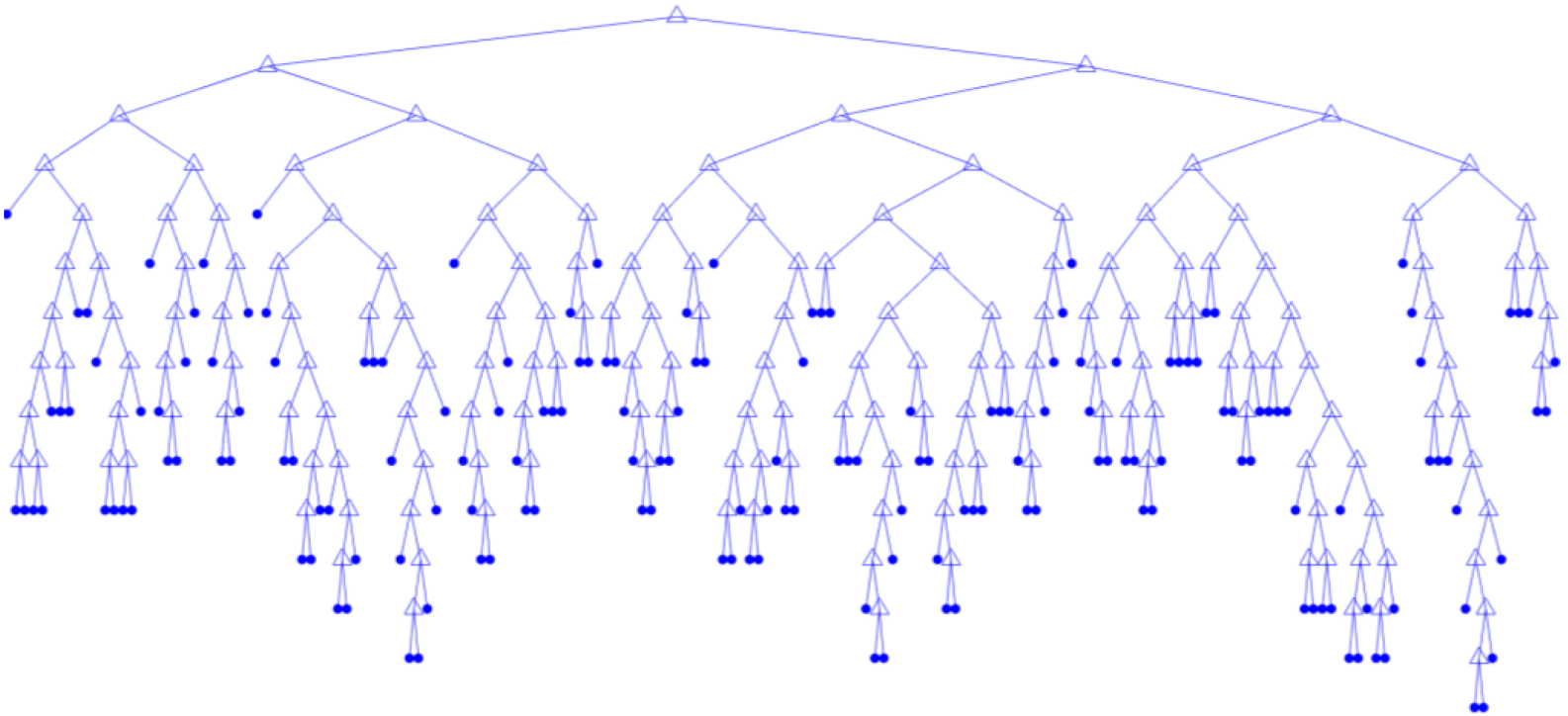
   We seek the value of j and s that minimize the equation

$$\sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

3. Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
4. Once the regions $R_1, \cdots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Overfitting

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because the resulting tree might be too complex.



Tree Pruning

A smaller tree with fewer splits (that is, fewer regions $R_1, \cdots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

- One possible alternative to the process described above is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split.
- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. Intuitively, our goal is to select a subtree that subtree leads to the lowest test error rate. Instead, we need a way to select a small set of subtrees for consideration.

Cost complexity pruning

We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{5}$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response associated with R_m. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0, because then Equation (5) just measures the training error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity in Equation (5) will tend to be minimized for a smaller subtree.

Algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \cdots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.

- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$- that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.
- Doing this allows us to build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. This reduction in overfitting, while retaining the predictive power of the trees, can be shown using rigorous mathematics.

Random Forests

Random forests is not even allowed to consider a majority of the available predictors. This may sound crazy, but it has a clever rationale.

- Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. This causes problem of "not diversifying well".
- Random forests inject randomness to avoid such strong deterministic trend and to ensure each tree is different: by selecting the data points used to build a tree and by selecting the features in each split test.
- The idea of random forests is that each tree might do a relatively good job of predicting and should also be different from the other trees. Then they will likely overfit on part of the data.

Randomness from Bootstrap and Features Choice

To build a random forest model, you need to decide on the number of trees to build ("n_estimators").

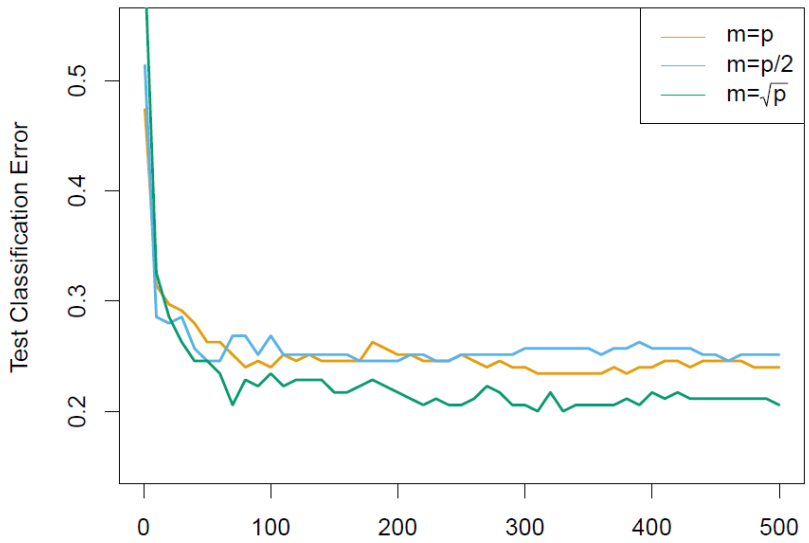
- The first way to inject randomness is by "Bootstrap": From our n_samples data points, we repeatedly draw an example randomly with replacement (i.e. the same sample can be picked multiple times), n_samples times. This will create a dataset that is as big as the original dataset, but some data points will be missing from it, and some will be repeated.
- In each node the algorithm randomly selects a subset of the features, and looks for the best possible test involving one of these features. The amount of features that is selected is controlled by the max_features parameter.

The bootstrap sampling leads to each decision tree in the random forest being built on a slightly different dataset. Because of the selection of features in each node, each split in each tree operates on a different subset of features. Together these two mechanisms ensure that all the trees in the random forests are different.

Model Complexity

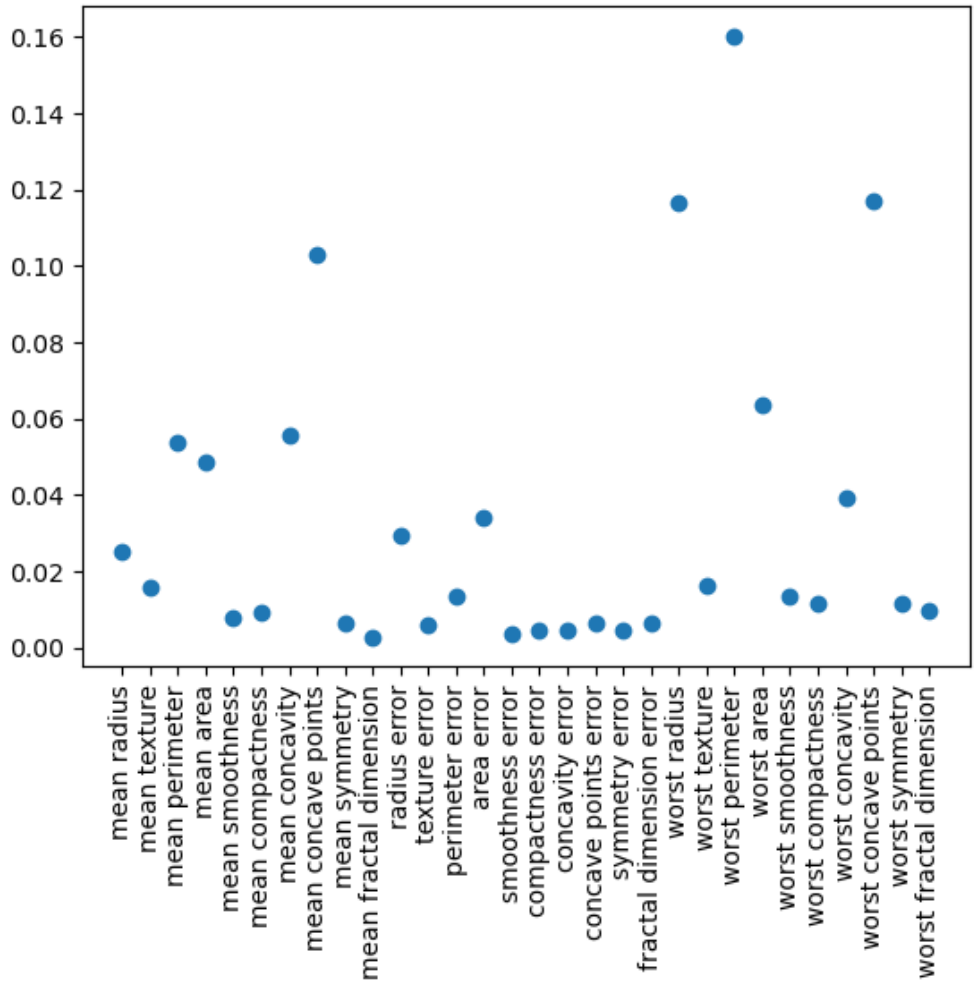
A critical parameter in this process is max_features (m).

- If we set max_features to n_features, that means that each split can look at all features in the dataset, and no randomness will be injected (*bagging*). If we set max_features to one, that means that the splits have no choice at all on which feature to test, and can only search over different thresholds for the feature that was selected randomly.
- Therefore, a high max_features means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features. A low max_features means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.



Features Importance

Similarly to the decision tree, the random forest provides feature importances, which are computed by aggregating the feature importances over the trees in the forest. Typically the feature importances provided by the random forest are more reliable than the ones provided by a single tree.



Gradient Boosted Regression Trees

- Gradient boosted regression trees is another ensemble method that combines multiple decision trees to a more powerful model. In contrast to random forests, gradient boosting works by building trees in a serial manner, where each tree tries to correct the mistakes of the previous one. Gradient boosted trees often use very shallow trees, of depth one to five, often making the model smaller in terms of memory, and making predictions faster.
- The main idea behind gradient boosting is to combine many simple models (in this context known as weak learners), like shallow trees. Each tree can only provide good predictions on part of the data, and so more and more trees are added to iteratively improve performance.
- They are generally a bit more sensitive to parameter settings than random forests, but can provide better accuracy if the parameter are set correctly.

Boosting Algorithm

Boosting for Regression Trees

1. Set $\hat{f}(x) = 0$ and residual $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \cdots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Tuning Parameters in Boosting

Boosting has three tuning parameters:

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Support Vector Classifiers

The hyperplane is chosen to correctlyseparate most of the training observations into the two classes, but may misclassify a few observations. It is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\cdots,\beta_p,\epsilon_1,\cdots,\epsilon_n,M} M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) > M(1-\epsilon_i),\ \forall i = 1,\cdots,n,$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

where C is a nonnegative tuning parameter. $\epsilon_1,\cdots,\epsilon_n$ are *slack variables* that allow individual observations to be on slack the wrong side of the margin or the hyperplane. If $\epsilon_i = 0$ then the ith observation is on the correct side of the margin. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, and we say that the ith observation has violated the margin. If $\epsilon_i > 1$ then it is on the wrong side of the hyperplane.

Furthermore, it turns out that α_i is nonzero only for the support vectors in the solution-that is, if a training observation is not a support vector, then its α_i equals zero. Let $\mathcal{S}$ be the set of all support vectors, we then have

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}}^{n} \alpha_i <x, x_i>.$$

To summarize, in representing the linear classifier $f(x)$, and in computing its coefficients, all we need are inner products. Now suppose that we replace the inner product with a generalization of the inner product of the form

$$K(x_i, x_{i'}),$$

where K is some function that we will refer to as a kernel. A kernel is a function that quantifies the similarity of two observations. We now arrive at the *Kernelized SVMs*.

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}}^{n} \alpha_i K(x, x_i).$$

Kernels

There are many possible Kernel functions

- Linear Kernel:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

- Polynomial kernel of degree d:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

  Using such a kernel with $d > 1$, instead of the standard linear kernel, in the support vector classifier algorithm leads to a much more flexible decision boundary.

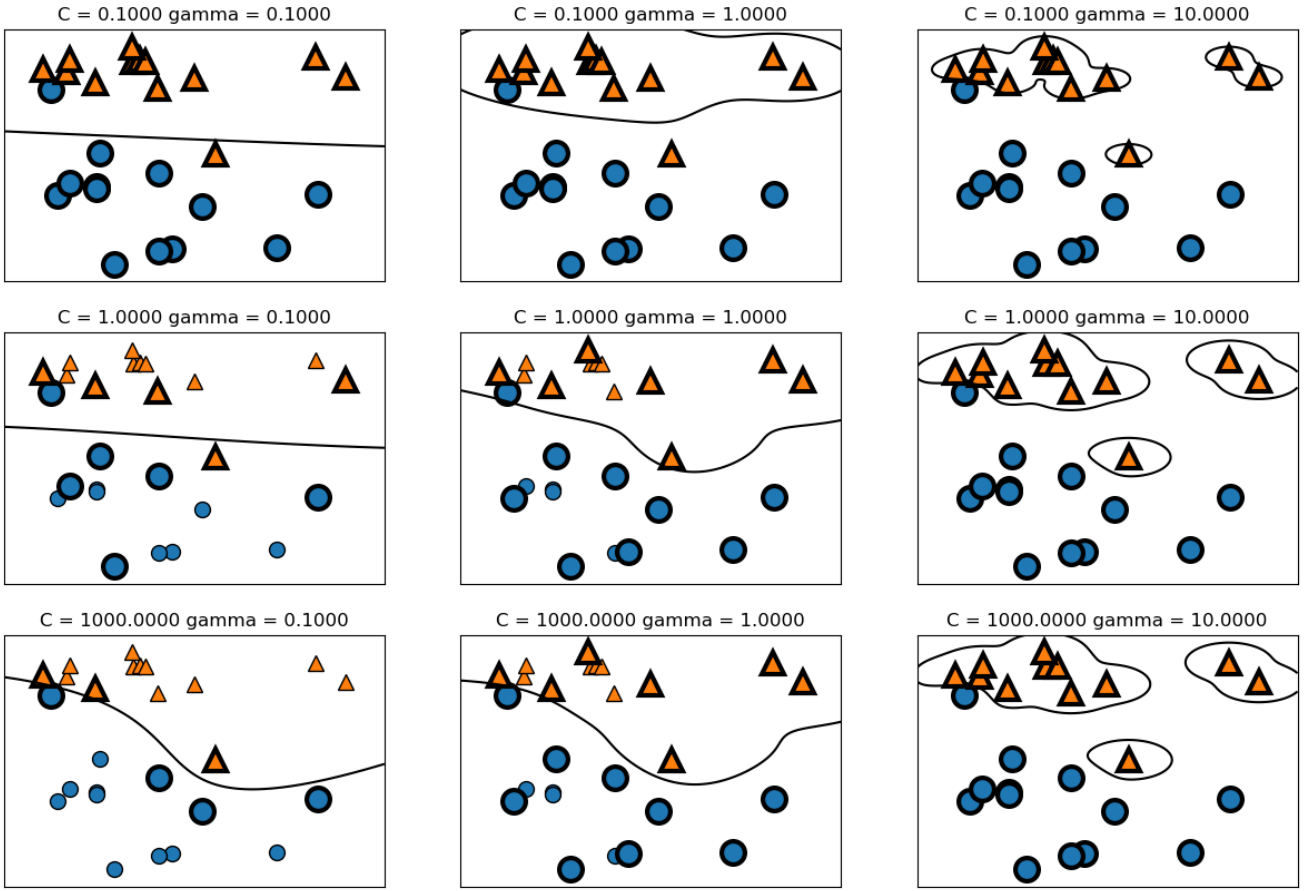
- The radial basis function (rbf) kernel, also known as Gaussian kernel.

$$k_{\text{rbf}}(x_1, x_2) = \exp(-\gamma ||x_1 - x_2||^2)$$

  The radial basis function kernel has very local behavior, in the sense that only nearby training observations have an effect on the class label of a test observation.

Parameter Choices

A small C means a very restricted model, where each data point can only have very limited influence. Increasing C, as shown on the bottom right, allows these points to have a stronger influence on the model, and makes the decision boundary bend to correctly classify them. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.



Models

- Nearest neighbors: for small datasets, good as a baseline, easy to explain.
- Linear models: Go-to as a first algorithm to try, good for very large datasets, good for very high-dimensional data.
- Decision trees: Very fast, don't need scaling of the data, can be visualized and easily explained.
- Random forests: Nearly always perform better than a single decision tree, very robust and powerful. Don't need scaling of data. Not good for very highdimensional sparse data.
- Gradient Boosted Decision Trees: Often slightly more accurate than random forest. Slower to train but faster to predict than random forest, and smaller in memory. Need more parameter tuning than random forest.
- Support Vector Machines: Powerful for medium-sized datasets of features with similar meaning. Needs scaling of data, sensitive to parameters.

k-fold cross-validation

1. When performing five-fold crossvalidation, the data is first partitioned into five parts of (approximately) equal size, called folds.
2. Next, a sequence of models is trained. The first model is trained using the first fold as the test set, and the remaining folds 2-5 as the training set.
3. Then another model is build, this time using fold 2 as the test set, and the data in folds 1, 3, 4 and 5 as the training set.
4. This process is repeated using the folds 3, 4 and 5 as test sets. For each of these five splits of the data into training and test set, we computed the accuracy. In the end, we have collected five accuracy values.