Nonlinear Regression Models

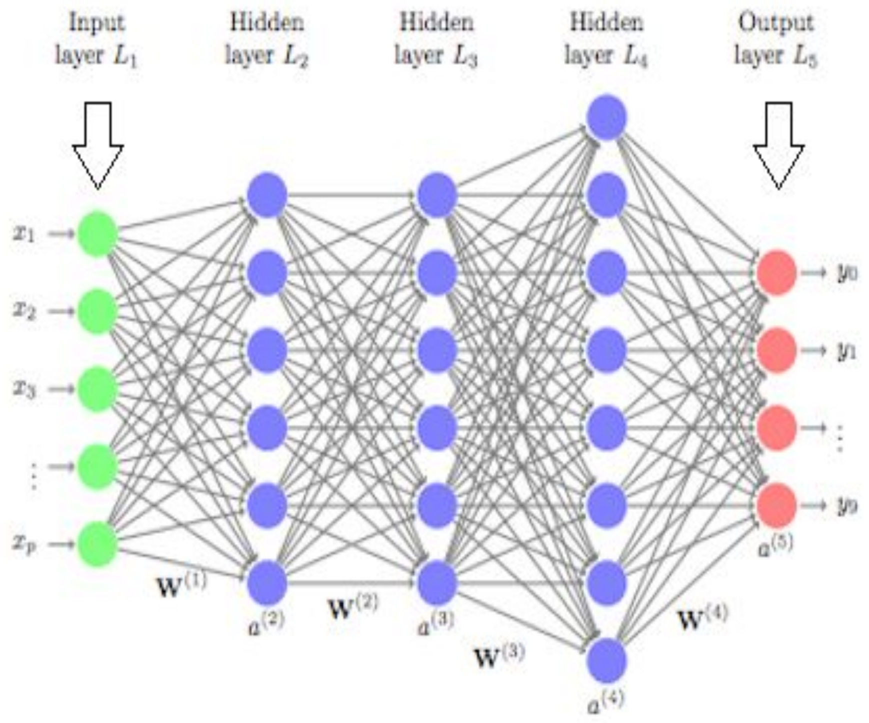
Chapter 7

Dominika Markowska-Desvallons, John Mazon, Orli Khaimova

Contents

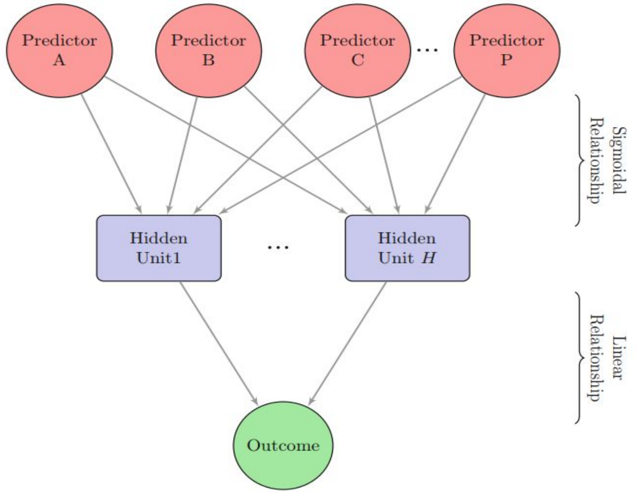
Neural Networks

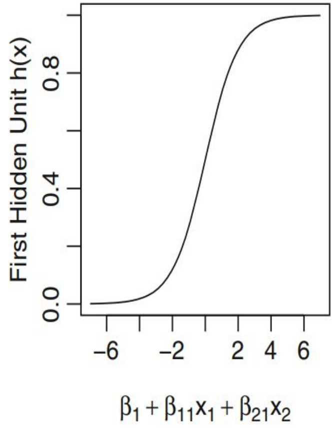
- powerful nonlinear regression method based by theories about how the brain works
- series of algorithms that recognize relationships between vast amounts of data
- the outcome is modeled by an intermediary set of hidden variables or hidden units .
- usually involves multiple hidden units to model the outcome.

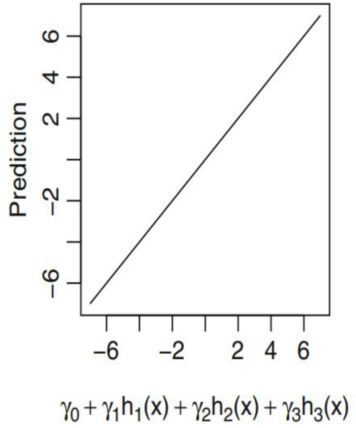


NN: description

- ➢ hidden units are linear combinations of the original predictors without any predefined order
- ➢ transformed by a nonlinear function, such as logistic (ex. sigmoidal function)
- ➢ no constraints that help define these linear combinations
- ➢ because of that, there is little likelihood that the coefficients in each unit represent some coherent piece of information







NN: Processing

- define the number of hidden units, each unit must be related to the outcome (linear combination)
- ex. for the solubility data (textbook) there are 228 predictors
    - *H(P+1)+H+1 parameters*
        - *228 predictors and 3 Hidden units would be:*
            - *3(228+1)+3+1=691 parameters*
- parameters are initialized to random values
- specialized algorithms like the back-propagation (derivatives) algorithm is used to solve the equations
- not a ‘global solution’- no guarantee that the resulting set of parameters are uniformly better than any other set

NN: Common Problems

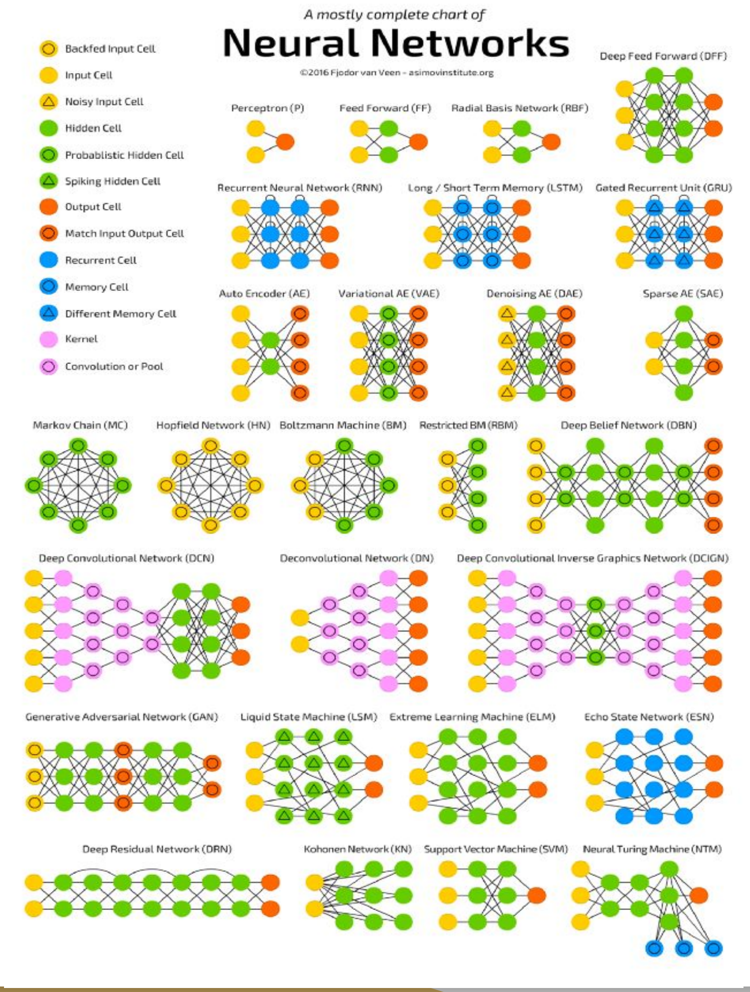
- neural networks have a tendency to over-fit the relationship between the predictors and the response due to the large number of regression coefficients
  - Solution: ***early stopping approach*** - stop the optimization procedure when estimate of the error rate starts to increase. It can be problematic since it is hard to estimate the model error (already has some amount of uncertainty associated)
  - Solution: **weight decay approach**, a penalization method to regularize the model, adding a penalty for large regression coefficients so that any large value must have a significant effect on the model errors to be tolerated, the value of this parameter must be specified (between 0 and 0.1)
  - Solution: **model averaging,** as an alternative, several models can be created using different starting values and averaging the results of these model to produce a more stable prediction adversely affected by high correlation among the predictor variables,
    - Two approaches -pre-filter the predictors to remove the predictors that are associated with high correlations - extraction technique, such as principal component analysis,can be used prior to modeling to eliminate correlations.

Chart of Neural Networks

The model described in textbook is the simplest neural network: a single-layer feed-forward network.

References:

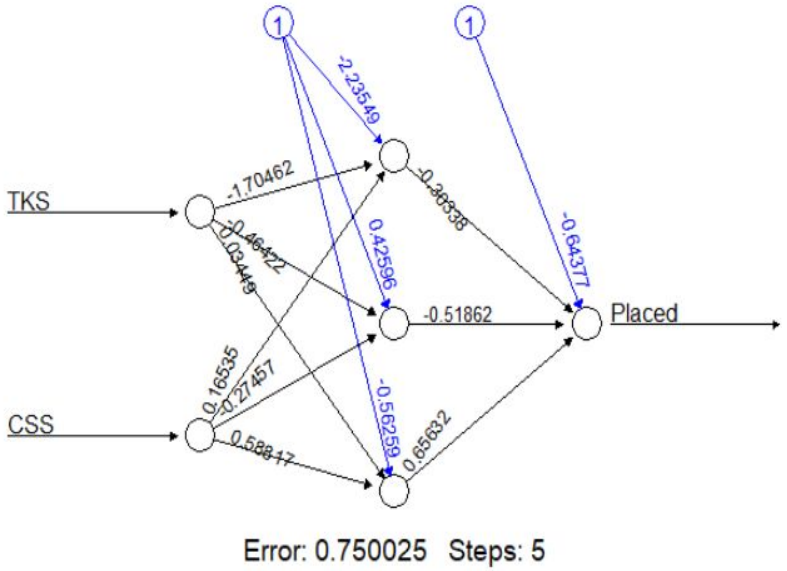
https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

NN: R

```
library(neuralnet)
nn=neuralnet(Placed~TKS+CSS,data=df, hidden=3,act.fct = "logistic",
                linear.output = FALSE)
plot(nn)
```
```

- ❏ hidden - number of nodes/neuron in a layer
- ❏ by adding lifesign = 'full' , 'minimal' or 'none' would give us different number of the calculation of the neural network
- ❏ rep = number of repetition times to run model
- ❏ when using rep can use plot(n, num) to show plot for rep


```

SVM (Supervised Machine Learning Algorithm)

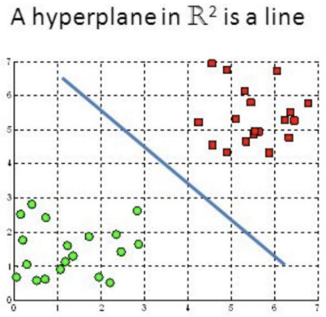
Key points(support vector machine) :

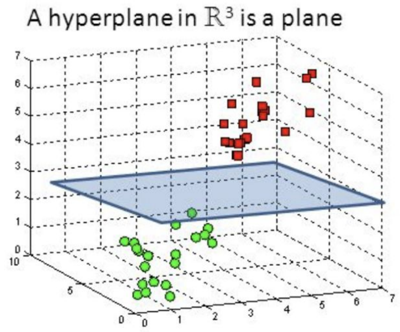
*SVMs are a class of powerful, highly flexible modeling techniques

1. We *motivate this technique* in the framework of robust regression where we seek to minimize the effect of outliers on the regression equations.
2. *It's applicable(solves) to both supervised **regression** and **classification** problems*

Involves optimally separating (maximal margin) hyperplanes

3. For example, in two dimensions, a hyperplane is a flat one-dimensional subspace (a line). In three dimensions, a hyperplane is a flat two-dimensional subspace – a plane.

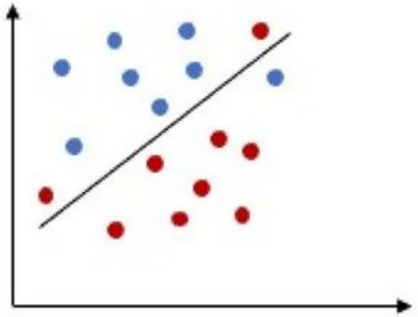


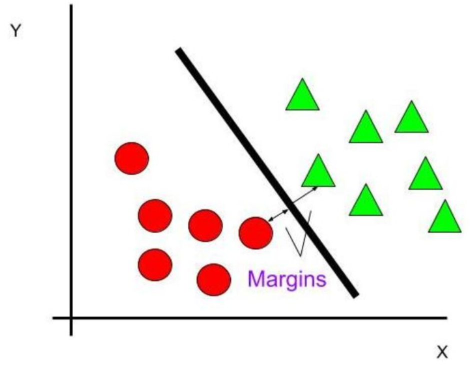


- *For non-separable cases, a **non-linear mapping** transforms the data into a kernel-induced feature space F, and then a linear machine is used to classify them in the feature space*

Linear SVM

Two classes such as dot and triangle, we can draw a straight line to seperator (100%)
Here we will build our initial concept of SVM by classifying perfectly separated dataset (linear classification). This is also called “Linear SVM – Hard Margin Classifier”. We define the objective function.

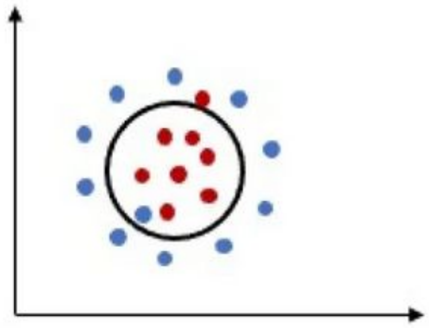


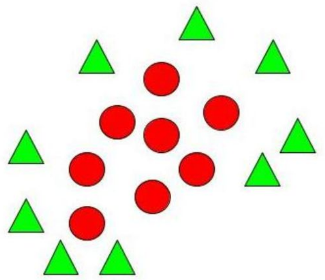


Linear Separable Data

Non – Linear SVM

-Nonlinear classification: SVM can be extended to solve nonlinear classification tasks when the set of samples cannot be separated linearly. By applying kernel functions, the samples are mapped onto a high-dimensional feature space, in which the linear classification is possible.
-After we turn it into a hyperdimensional space we are able to separate it. Originally we cannot draw a straight line to seperate





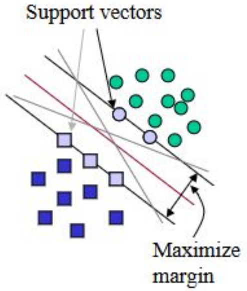
Non-Linear Separable Data

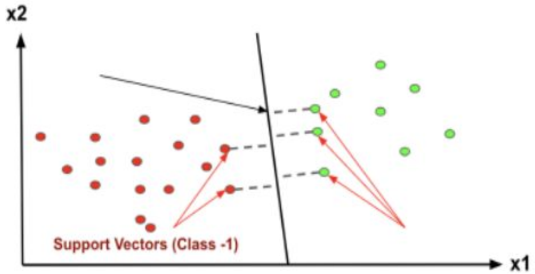
Classification using hyperplane

Generally we desire to construct a hyperplane that separates the training observations perfectly according to their class labels

Suppose we have a *n* x *p* data matrix *X* that consists of *n* training observations in *p*-dimensional space, which fall into two classes (-1, 1). The goal is to develop a classifier based on the training data that will correctly classify the test observations using the available features

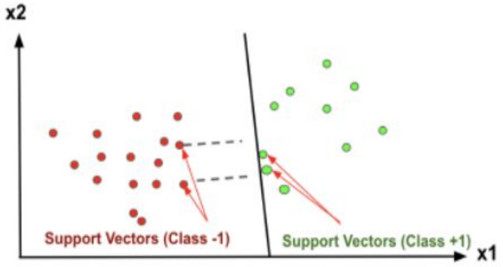
.





Good Margin

- all sector vectors have the same distance with the maximum margin hyperplane



Bad Margin

- very close to either class -1 support vectors o class +1 support vectors

Support vectors

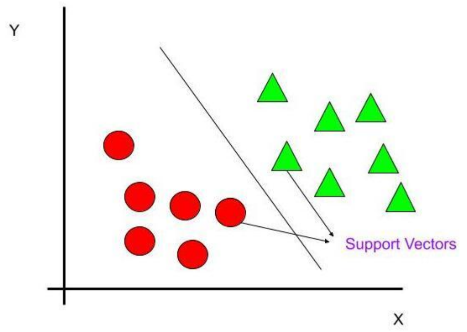
• Support vectors are the data points that lie closest to the decision surface (or hyperplane). These are the critical elements. Since removing them may alter the position of the dividing hyperplane.

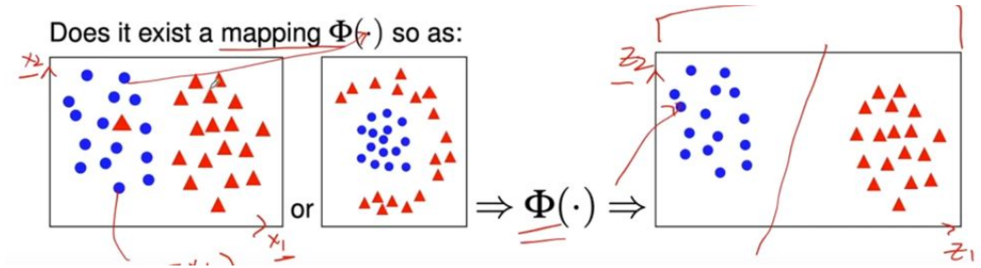
•• Support vectors are the elements of the training set that would change the position of the dividing hyperplane if remove.They are the data points most difficult to classify

-We use the mapping function, we send points from our dataset in the function which will map points as so. It takes its form in hyper dimensional space which then we can apply the linear SVM theory to design a hyperplane in the hyperdimensional space. We substitute every single coordinate into the function

-We are going to map X into Z and then design the hyperplane in Z space.

-Original feature space is mapped to a higher-dimensional feature space through a feature mapping function. In the higher dimensional feature space we will be able to come up with a "linear seperable dataset" to separate the two classes. We want a classifier (linear separator) with as big a margin as possible.




Does it exist a mapping $\Phi(\cdot)$ so as:
x_2
x_1
or
$\Rightarrow \Phi(\cdot) \Rightarrow$
z_2
z_1

Soft margin formulation

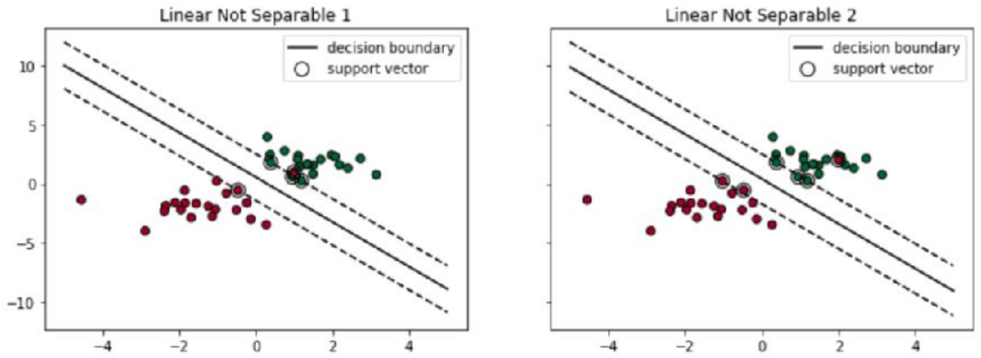
Aka tries to find a line to separate, but tolerate one or few misclassified dots.

Why use this ?

1. it tolerates a few dots to get misclassified
2. it tries to balance the trade-off between finding a line that maximizes the margin and minimizes the misclassification.

Two types of misclassifications can happen:

1. The dot is on the wrong side of the decision boundary but on the correct side/ on the margin
2. The dot is on the wrong side of the decision boundary and on the wrong side of the margin



Kernel function

-when using SVMs you can apply an almost miraculous mathematical technique called the kernel trick . It makes it possible to get the same result as if you added many polynomial features, even with very high-degree polynomials, without actually having to add them.

- implementing support vector classifiers requires specifying a kernel function

-There can be many transformations that allow the data to be linearly separated in higher dimensions, but not all of these functions are actually kernels. The kernel function has a special property that makes it particularly useful in training support vector models, and the use of this property in optimizing non-linear support vector classifiers is often called the kernel trick.

-The radial basis function has been shown to be very effective. However, lwhen the regression line is truly linear, the linear kernel function will be a better choice

** radial basis - RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other.

-Since the predictors enter into the model as the sum of cross products, differences in the predictor scales can affect the model. Therefore, we recom-mend centering and scaling the predictors prior to building an SVM model.

$$\underline{K(\mathbf{x}_i,\mathbf{x})} = \overline{\underline{\Phi(\mathbf{x}_i)}^T \underline{\Phi(\mathbf{x})}}$$

types of kernel functions that can be used to general-ize the regression model and encompass nonlinear functions of the predictors:

$$\begin{aligned} \text{polynomial} &= (\phi\,(\mathbf{x}'\mathbf{u}) + 1)^{degree} \\ \text{radial basis function} &= \exp(-\sigma\|\mathbf{x}-\mathbf{u}\|^2) \\ \text{hyperbolic tangent} &= \tanh\,(\phi\,(\mathbf{x}'\mathbf{u}) + 1)\,, \end{aligned}$$

$$L_i = \begin{cases} \left(z^{(i)}\right)^2 & \text{for } \left|z^{(i)}\right| \leq \delta \\ 2\delta\left|z^{(i)}\right| - \delta^2 & \text{for } \left|z^{(i)}\right| > \delta \end{cases}$$

*What Kernel Trick does is it utilizes existing features, applies some transformations, and create new features. Those new features are the key for SVM to find the nonlinear decision boundary.

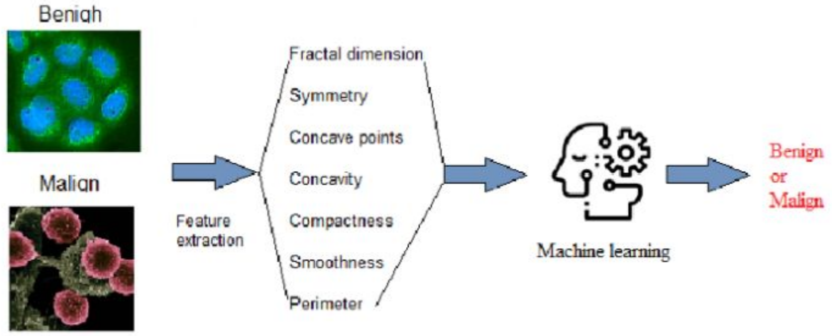
Real World Use

Where do we see this used ?

The aim of using SVM is to correctly classify unseen data. SVMs have a number of applications in several fields.

- Bioinformatics – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.

- Face detection – SVMc classify parts of the image as a face and non-face and create a square boundary around the face.



SVM Conclusion

PROS

-effective in high dimensional spaces (dim > N)

-works well with even unstructured & semi-structured data (text, images)

-does not suffer from local optimal and multicollinearity

CONS

-difficult to interpret the final model, variable weights, and meld with business logic

forcing separation of the data can easily lead to overfitting, particularly when noise is present in the data

-choosing a “good” kernel function is not easy

-long training time for large datasets

Multivariate Adaptive Regression Splines (MARS)

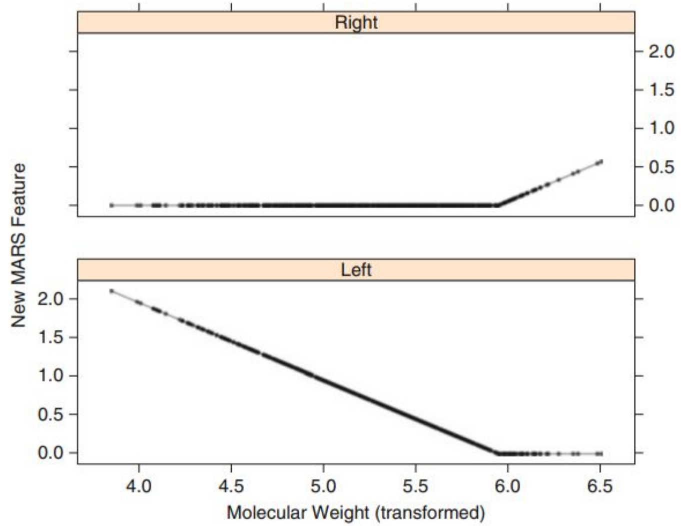
- Uses surrogate features instead of the original predictors
  - Creates 2 contrasted versions of a predictor
  - Breaks the predictor into 2 groups
    - models a linear relationship between the predictor and response in each group
  - Given a cut point, the 2 new features are hinge of the original
- Piecewise linear model
  - Each new feature models an isolate portion of the original data
- Determining the cut point
  - Each data point is a potential candidate
  - Linear model is created with candidate features and model error is calculated
  - Cut point with the smallest error is chosen for the model
  - A simple linear regression with no hinge/cut point is also evaluated
- Then, given the first set of features, the model looks for the next set of features
  - Process continues until the stopping point (can be defined by user)

$$h(x) = \begin{cases} x & x > a \\ 0 & x \leq a \end{cases}$$

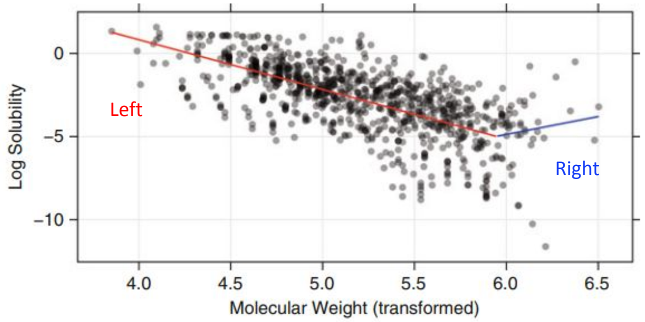
Hinge functions:

$$h(x - a)$$
$$h(a - x)$$

MARS: Example



- Cut point = 5.9
- Right: all values less than cut point = 0
- Left: all values greater than cut point = 0
- These two predictors are used for linear regression model

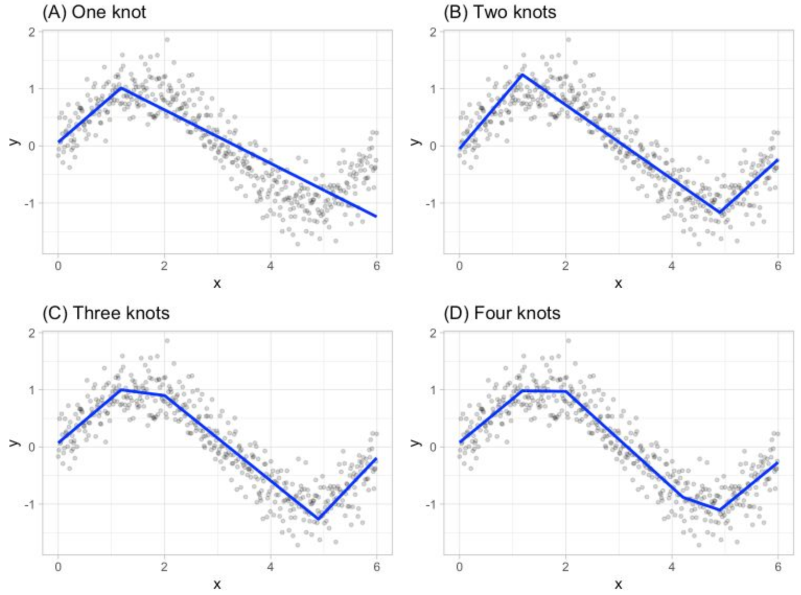


$$-5 + \underbrace{2.1 \times h(MolWeight - 5.94516)}_{\text{right}} + \underbrace{3 \times h(5.94516 - MolWeight)}_{\text{left}}.$$

MARS : Multiple Cut Points

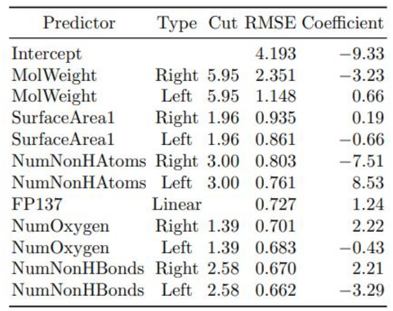
- Sometimes, there can be more than 1 cut point.
- Although, it may fit the training data well, it may not do well on other data.

$$y = \begin{cases} \beta_0 + \beta_1(1.183606 - x) & x < 1.183606, \\ \beta_0 + \beta_1(x - 1.183606) & x > 1.183606 \quad \& \quad x < 4.898114, \\ \beta_0 + \beta_1(4.898114 - x) & x > 4.898114 \end{cases}$$

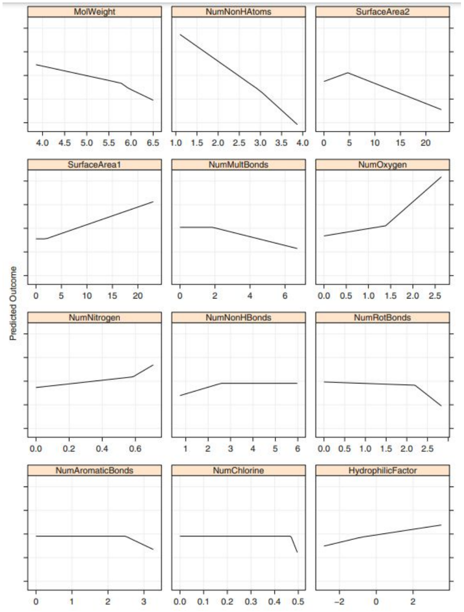


MARS: Pruning

- When there is a full set of features, MARS removes features that are not significant
- It assesses each predictor and estimates how much the error has decreased with the predictor included
- The Generalized Cross Validation (GCV) statistic is used
    - Has better estimates than the actual error rate
- The amount of terms to be removed can be set by the user or a tuning parameter can be used along with resampling
- Can also quantify importance of each predictor by tracking the reduction in the RMSE

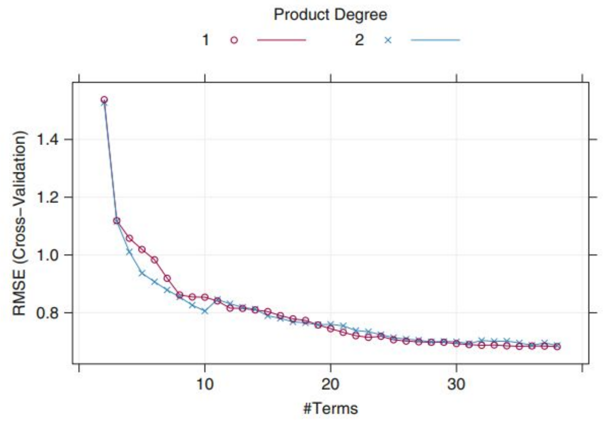


Predictor	Type	Cut	RMSE	Coefficient
Intercept			4.193	−9.33
MolWeight	Right	5.95	2.351	−3.23
MolWeight	Left	5.95	1.148	0.66
SurfaceArea1	Right	1.96	0.935	0.19
SurfaceArea1	Left	1.96	0.861	−0.66
NumNonHAtoms	Right	3.00	0.803	−7.51
NumNonHAtoms	Left	3.00	0.761	8.53
FP137	Linear		0.727	1.24
NumOxygen	Right	1.39	0.701	2.22
NumOxygen	Left	1.39	0.683	−0.43
NumNonHBonds	Right	2.58	0.670	2.21
NumNonHBonds	Left	2.58	0.662	−3.29



MARS: Tuning

- Parameters
  - Degree of features added to the model
    - First degree is additive model
      - No interaction, one predictor at a time
    - Second degree is where the features can include multiple predictors simultaneously
      - Instability is more likely to occur
  - Number of retained terms
    - Can be manually set, GCV, or a resampling method

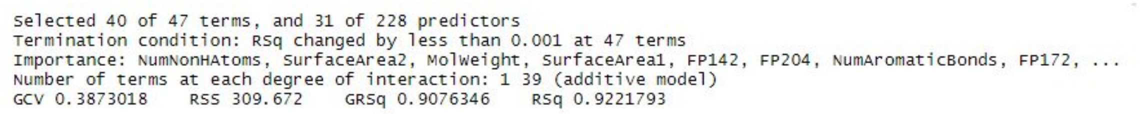


MARS: Advantages

- The model conducts feature selection automatically
  - MARS will select the predictors while building the model
- Interpretability
  - Each hinge feature models a specific region since it is a piecewise linear model
  - In an additive model, predictors can be isolated to see how they relate to the response
  - In a nonadditive model, interpretability still remains
- Requires very little pre-processing
  - No need to transform data or filter
  - Zero variance predictor will not be chosen as it provides no information
  - Correlated predictors only affect interpretability

MARS: R

- `earth` package
    - `marsFit <- earth(X_train, Y_train)`

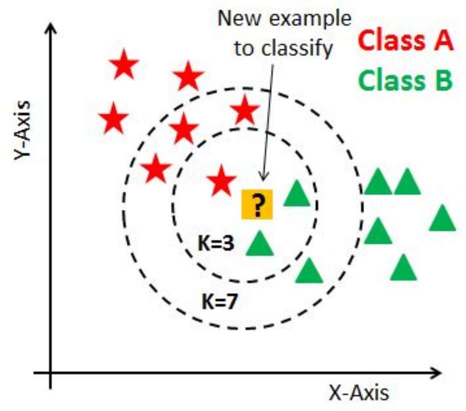


```
Selected 40 of 47 terms, and 31 of 228 predictors
Termination condition: RSq changed by less than 0.001 at 47 terms
Importance: NumNonHAtoms, SurfaceArea2, MolWeight, SurfaceArea1, FP142, FP204, NumAromaticBonds, FP172, ...
Number of terms at each degree of interaction: 1 39 (additive model)
GCV 0.3873018    RSS 309.672    GRSq 0.9076346    RSq 0.9221793
```

```
# create a tuning grid
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)
# tune
marsTune <- train(X_train, Y_train, method = "earth",
                  tuneGrid = marsGrid,
                  trControl = trainControl(method="cv", number = 10))
mars_predict <- predict(marsTune, X_test)
```

K-Nearest Neighbors (KNN)

- Predicts a new sample using *K* closest samples in the training data
- Can't be summarized by a model
- Based only on the individual samples
- The predicted response will take the average of the *K* nearest neighbors
  - Can use median
- The optimal *K* can be determined by resampling



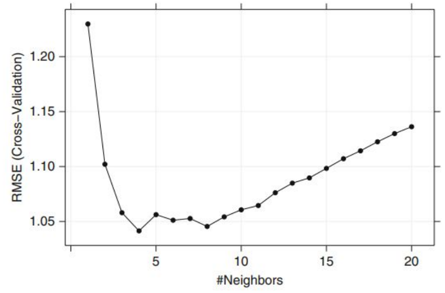
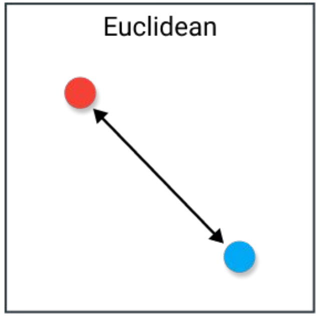
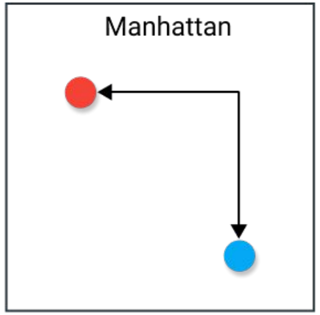


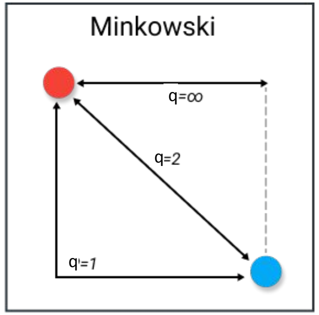
Fig. 7.10: The RMSE cross-validation profile for a *K*NN model applied to the solubility data. The optimal number of neighbors is 4

KNN: Distance

- KNN depends on how distance is defined
- The most common are:







$$\left(\sum_{j=1}^{P}(x_{aj}-x_{bj})^2\right)^{\frac{1}{2}}$$

$$\left(\sum_{j=1}^{P}|x_{aj}-x_{bj}|^q\right)^{\frac{1}{q}}$$

q=2 Euclidean
q=1 Manhattan

KNN: Pre-Processing

- Centering and Scaling
    - Scale has a big impact on the distances between samples
    - Predictors with larger scales will contribute more to the distance
    - Best for each predictor to contribute equally
- Missing Values
    - Can exclude if the data is sparse
    - Impute the data
        - Mean, nearest neighbors, etc.

KNN: Common Problems

- Computation time
    - Computation increases as *n* increases
    - Solution:
        - Can replace data with a representation that takes less memory
            - Ex. *k*-dimensional tree
- Disconnection between local structure and the predictive ability
    - Will perform poorly if the predictor structure is not relevant
    - Noisy and irrelevant predictors will cause large variability in the distance
    - Solution:
        - Remove irrelevant, noisy predictors
        - Weight the nearest neighbors
            - Samples that are closer contribute more and further samples contribute less

KNN: R

- caret package

  -
    ```
    # fitting model with no tuning
    knnFit <- knnreg(X_transformed_train, Y_transformed_train, k = 3)

    # create a tuning grid
    knnGrid <- expand.grid(.k = 1:20)
    # tune
    knnTune <- train(X_transformed_train, Y_train, method = "knn",
                     tuneGrid = knnGrid,
                     trControl = trainControl(method="cv"),
                     preProc = c("center", "scale"))
    knn_predict <- predict(knnTune, X_test)
    ```

Sources

- Boehmke, B., & Greenwell, B. (2020). *Hands-on machine learning with R*. CRC Press.
- Grootendorst, M. (2021). *Distance Measures* [Online image]. Towards Data Science. https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.
- Kuhn, M., & Johnson, K. (2019). *Applied predictive modeling*. Springer.