INT104 knowledge domain

(For final exam)

1 Data pre-procession

1.1 Data type

• Highly organized

• Usually with a label

1.2 Data Storage and Presentation

CSV, TSV, XML, JSON

1.3 Pearson's r correlation

$$r = \frac{N\sum xy - \sum x \sum y}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}}$$

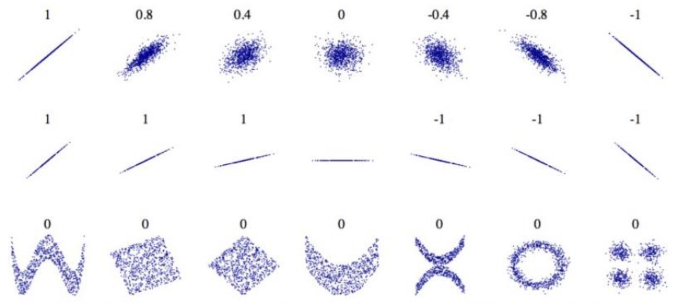


Figure 2-14. Standard correlation coefficient of various datasets (source: Wikipedia; public domain image)

1.4 Data Integration

Combine, Resolve conflicts, Remove redundant

1.5 Data Transformation

• Min–max normalization

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

• Z-score normalization

(Normalizing every value in a dataset such that the mean of all of the values is 0 and the standard deviation is 1)

$$x_{scaled} = \frac{x - mean}{sd}$$

• Normalization by decimal scaling

$$x_{scaled} = \frac{x}{10^j}$$

1.6 Feature Selection

• Filter methods – features are selected and ranked according to their relationships with the target

• Wrapper methods – it's a search for well-performing combinations of features

• Embedded methods – perform feature selection as part of the model training process

2 Classification & Model selection

2.1 Binary Classifier

• Classification algorithms find the mapping function to map the "x" input to "y" discrete output.

- Binary Classification refers to those classification tasks that have two class labels.

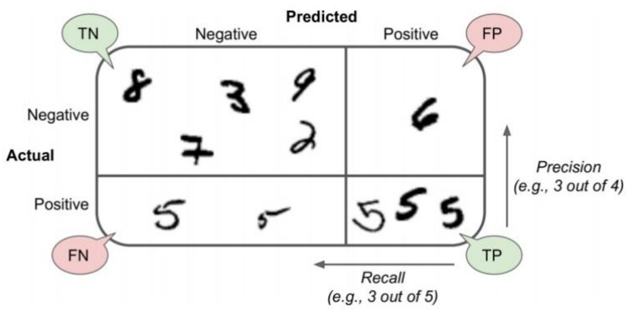
2.2 Performance Measures

Metrics to evaluate Classification models:

- Accuracy

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- Confusion Matrix (not a metric but fundamental to others)



  - True Positive (TP): Predict an observation belongs to a class and it actually does belong to that class.
  - True Negative (TN): Predict an observation does not belong to a class and it actually does not belong to that class.
  - False Positives (FP): Predict an observation belongs to a class and it actually does not belong to that class.
  - False Negatives (FN): Predict an observation does not belong to a class and it actually does belong to that class.

- Precision and Recall

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

- F1-score

$$F_1\ score = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- F_β-score: a more flexible F score that combines precision and recall

$$F_\beta\ score = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

  - $\beta < 1$ focuses more on precision
  - $\beta > 1$ focuses more on recall

- Specificity (Selectivity)

$$Specificity = \frac{TN}{TN + FP}$$

- Fall-out

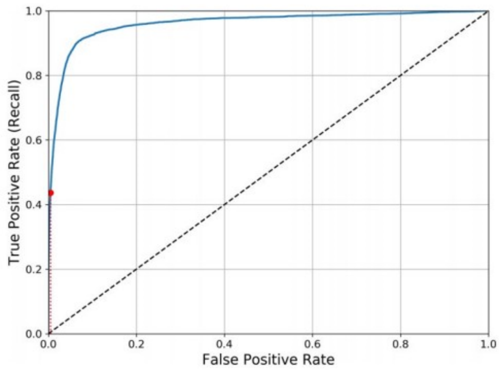
$$Fall - out = \frac{FP}{TN + FP}$$

- Miss

$$Miss = \frac{FN}{TP + FN}$$

• AUC&ROC

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. In another word, it presents Recall (True Positive Rate) VS Fall-out (False Positive Rate)

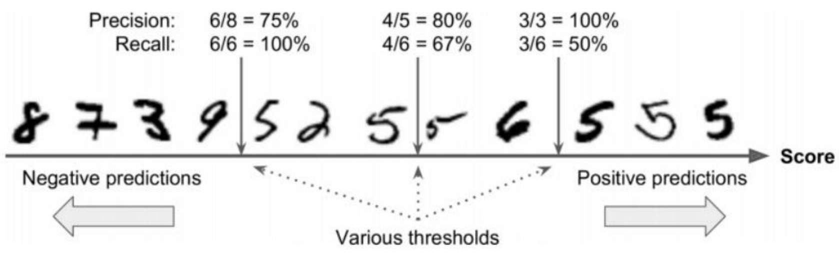


2.3 Precision/Recall Trade-off

The threshold of a decision function:

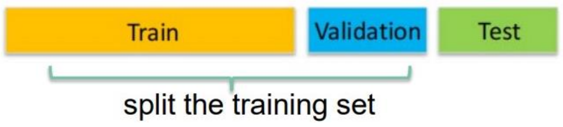
• If the threshold is small, FP is large → high recall, low precision (Strict, most real objects are detected)

• If the threshold is large, FN is large → low recall, high precision (Generous, most of the prediction are correct)


Precision: 6/8 = 75% 4/5 = 80% 3/3 = 100%
Recall: 6/6 = 100% 4/6 = 67% 3/6 = 50%
Score
Negative predictions
Positive predictions
Various thresholds

2.4 Cross Validation

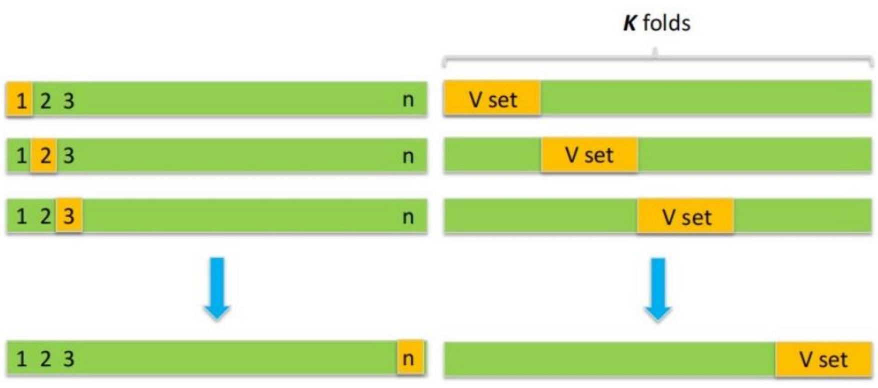
To avoid selecting the parameters that perform best on the test data but maybe not the parameters that generalize best, we can further split the training set into training fold and validation fold:



• Training fold: used to fit the model

• Validation fold: used to estimate prediction error for model selection

• Test set: used for assessment of the prediction error of the final chosen model



Leave-one-out Cross Validation (LOOCV):

1. Split the training data set of size n into:
    • Training fold (n-1 elements)
    • Validation fold (1 element)

2. Fit the model using the training data set.
3. Evaluate the model using validation set and compute the corresponding mean squared error (MSE).
4. Repeat this process n times, producing n squared errors. The average of these n squared errors estimates the test MSE.

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MSE}_i$$

Merits:

• Less bias: it trains on almost all of the data means that the model is trained on as much data as possible

Demerits:

• Sensitive to outliers

• Due to n times model fitting, the computation of LOOCV is intensive

K-fold Cross Validation

1. Randomly divide the data set into K folds (they are mutually exclusive)
2. Treat one fold as a validation set (normally from the 1st to the Kth), and fit the model on the remaining K-1 folds. Then, computing the MSE on the observations in the validation set. (This operation is repeated K times)
3. Compute the mean MSE (or other performance measure) as following

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i$$

2.5 Multiclass Classification

Multiclass classification refers to classification tasks that can distinguish between more than two classes:

• One-versus-the-rest(OvR) strategy: train multiple binary classifiers for each class, select the class whose classifier outputs the highest score.

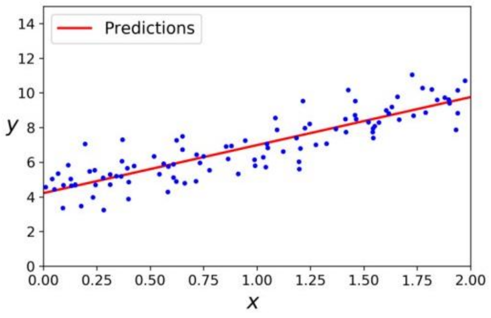
  • train N times

• One-versus-one (OvO) strategy: train a binary classifier for every pair of classes

  • train N(N-1)/2 times

3 Training models

3.1 Simple Linear Regression



• Y = kx + b

• Y: predicted value (output)

• x: feature value (input)

• k, b: model parameters (b: bias term, k: weight)

Usually the predicted value (fitted value) y is not perfect. The difference between the fitted value and real value is known as residuals $\hat{e}$

$$\hat{y}^{(i)} = kx^{(i)} + b$$

$$\hat{e}_i = y^{(i)} - \left(kx^{(i)} + b\right) = y^{(i)} - \hat{y}^{(i)}$$

The regressed value usually pursues a minimum of residual sum of square (RSS)

$$RSS = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the bias term (also called the intercept term):

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i^{th} feature value.
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Linear Regression model prediction (vectorized form):

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- $\boldsymbol{\theta}$ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.
- $h_{\boldsymbol{\theta}}$ is the hypothesis function, using the model parameters $\boldsymbol{\theta}$.

Evaluation:

• Cost function: Mean Squared error (MSE) for a Linear Regression model

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

Training the model is the process to find the value of $\theta\theta$ that minimizes the cost function.

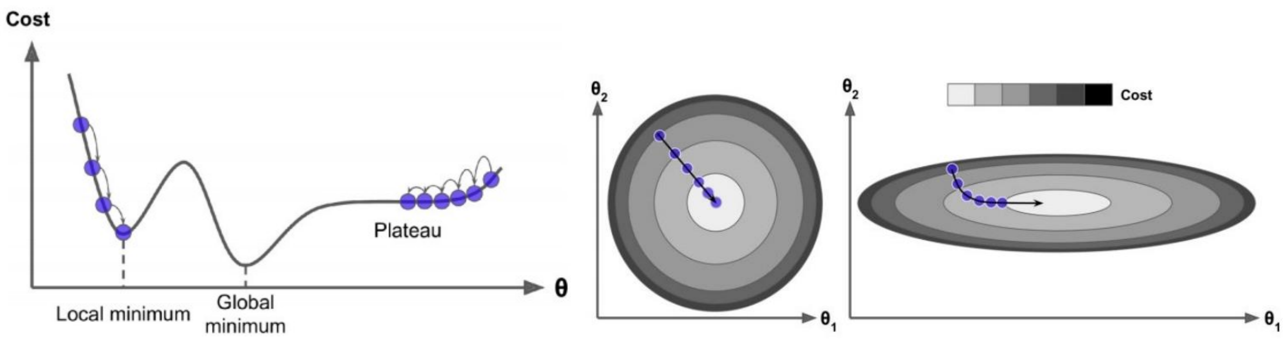
• Normal Equation:

$$\widehat{\boldsymbol{\theta}} = (X^{\intercal}X)^{-1}X^{\intercal}\mathrm{y}$$

- $\hat{\theta}$ is the value of θ that minimizes the cost function
- y is the vector of targeted values containing $y^{(1)}$ to $y^{(m)}$

3.2 Gradient Descent

The MSE cost function for a Linear Regression model is continuous and convex function.
Gradient Descent is guaranteed to approach arbitrarily close the global minimum.
Features with very different scales have different cost time.



3.2.1 Batch Gradient Descent

Use the whole training set to compute the gradients at every step

$$\nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \operatorname{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \operatorname{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \operatorname{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\intercal (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta})$$

$$(\boldsymbol{\theta} = [\theta_0, \theta_1 \dots . \theta_n]^\intercal)$$

3.2.2 Stochastic Gradient Descent

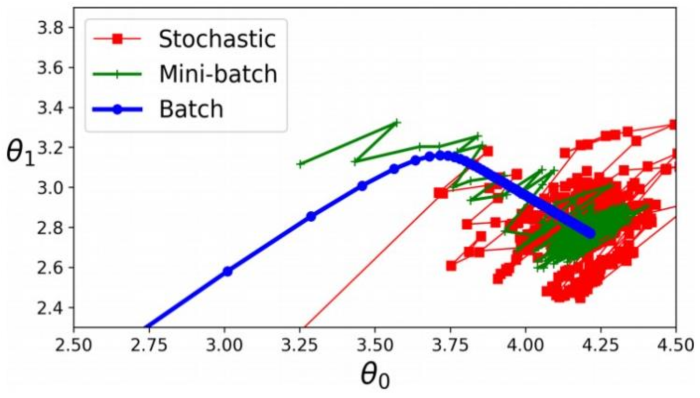
Picks a random instance in the training set at every step and compute based only on that single instance

$$\boldsymbol{\theta}^{(next\ step)} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} MES\left(\boldsymbol{\theta}; x^{(i)}, y^{(i)}\right)$$

3.2.3 Mini-batch Gradient Descent

Computes the gradients on small random sets of instances called mini-batches

Compare these three methods



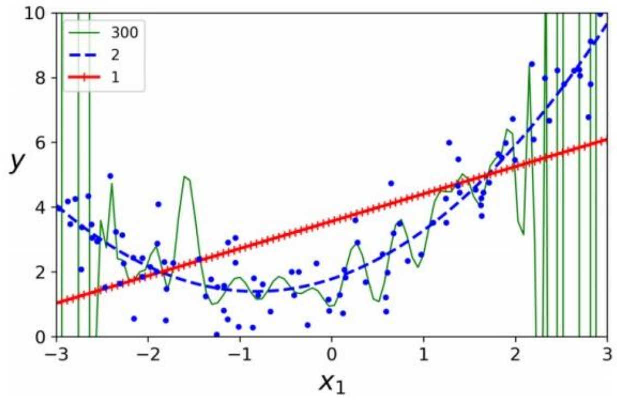
Method	Pros	Cons
Batch Gradient Descent	Guaranteed convergence to the global minimum.	Computationally expensive for large datasets. Requires the entire dataset to be loaded into memory.
Stochastic Gradient Descent	Computationally efficient for large datasets.	Convergence to the global minimum is not guaranteed. The noise in the updates can cause the loss function to oscillate.
Mini-Batch Gradient Descent	Better convergence than stochastic gradient descent. Computationally efficient for large datasets.	Convergence to the global minimum is not guaranteed.

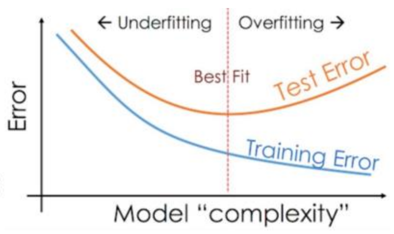
3.3 Polynomial Regression

Better but problems:

• Bias: refers to the error from erroneous assumptions in the learning algorithm (inability to capture the underlying patterns in the data). [Generalization]

• Variance: refers the error from sensitivity to small fluctuations in the training data (difference in fits between data sets). [Robustness]





- Underfitting (1d): Poor performance on training data and poor performance on validation data
- Overfitting(300d): Good performance on training data but poor performance on validation data

3.4 Regularized Linear Models

- Ridge Regression(L2) cost function:

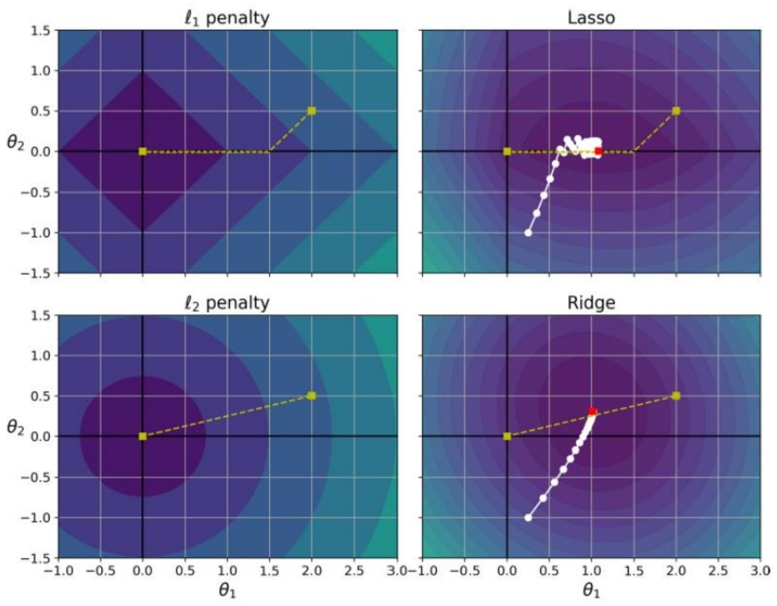
$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} {\theta_i}^2$$

<Forces the learning algorithm to not only fit the data but also keep the model weights as small as possible>

- Lasso Regression(L1) cost function:

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$$

<Lasso Regression automatically performs feature selection and outputs a sparse model>

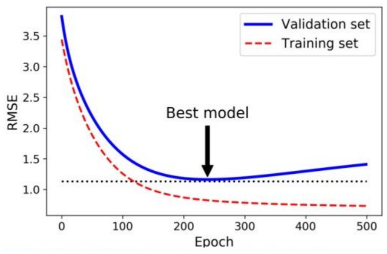


- Elastic Net cost function:

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} {\theta_i}^2$$

<It is a middle ground between Ridge Regression and Lasso Regression>

- Early stopping



<To stop training as soon as the validation error reaches a minimum>

3.5 Logistic Regression

Commonly used to estimate the probability that an instance belongs to a particular class

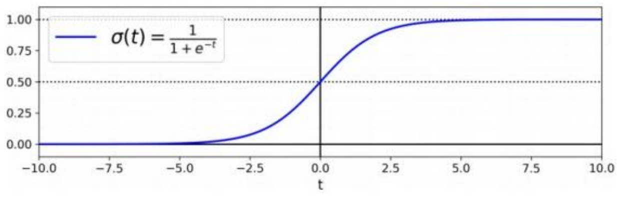
Logistic Regression model:

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^{\mathsf{T}}\boldsymbol{\theta}\right)$$

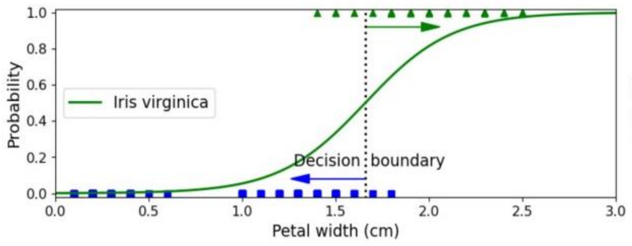
Sigmoid function:

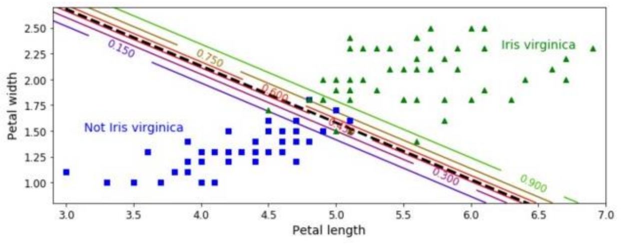
$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

$$\sigma(t) = \frac{1}{1+\exp(-t)}$$



Decision Boundaries:





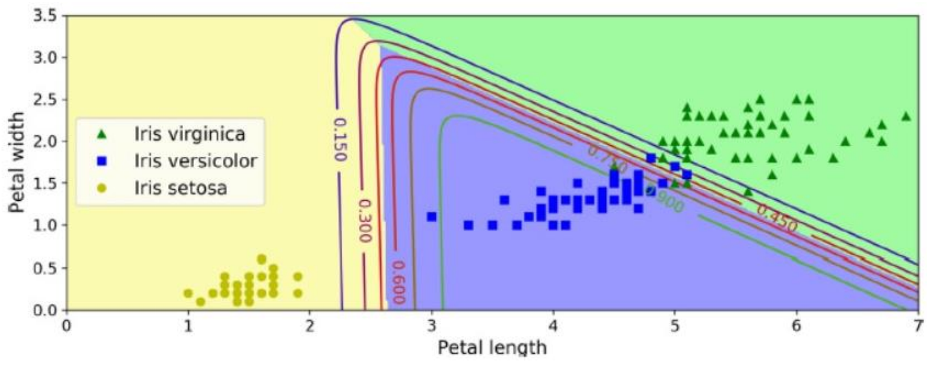
Softmax Regression: for Multinomial Logistic Regression

Softmax score for class k:

$$s_k(\mathbf{x}) = \mathbf{x}^{\mathsf{T}}\boldsymbol{\theta}^{(k)}$$

Softmax function:

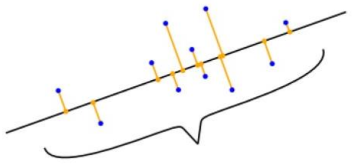
$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$$



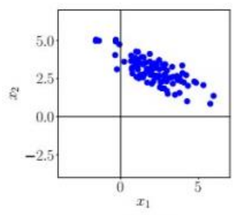
4 Dimensionality reduction

4.1 Principal Component Analysis (PCA)

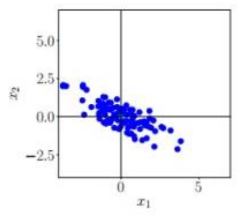
PCA identifies the axis that accounts for the largest amount of variance in the training set. Focuses on the mutual independence of the components. Vectors are orthogonal.



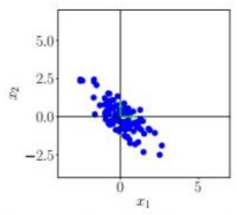
Steps:



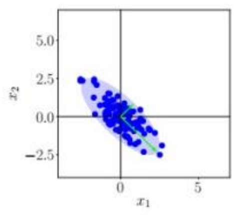
(a) Original dataset.



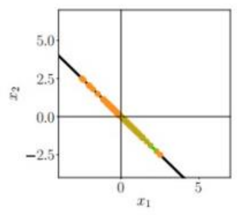
(b) Step 1: Centering by subtracting the mean from each data point.



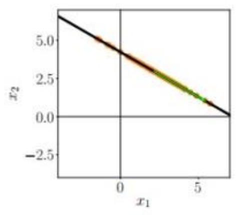
(c) Step 2: Dividing by the standard deviation to make the data unit free. Data has variance 1 along each axis.



(d) Step 3: Compute eigenvalues and eigenvectors (arrows) of the data covariance matrix (ellipse).



(e) Step 4: Project data onto the principal subspace.



(f) Undo the standardization and move projected data back into the original data space from (a).

Variance on C1:

$$V_1 = \frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{c}_1^{\top}\boldsymbol{x}^{(i)})^2 = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{c}_1^{\top}\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\top}\boldsymbol{c}_1 = \boldsymbol{c}_1^{\top}\left(\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\top}\right)\boldsymbol{c}_1$$
$$= \boldsymbol{c}_1^{\top}S\boldsymbol{c}_1$$

Data covariance matrix:

$$S = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\top}$$

(S is an N*N matrix, N is the number of features, M is the total number of data points)

Practice: Given a dataset that consists of the following points below:

A=(2, 3), B=(5, 5), C=(6, 6), D=(8,9)

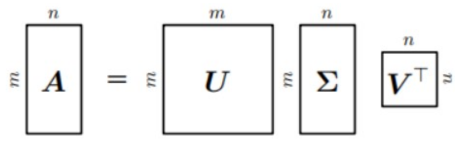
1. Calculate the covariance matrix for the dataset.
2. Calculate the eigenvalues and eigenvectors of the covariance matrix.

```R
# Define the dataset
data <- matrix(c(2, 3, 5, 5, 6, 6, 8, 9), ncol = 2, byrow = TRUE)
colnames(data) <- c("x", "y")

# Calculate the covariance matrix
cov_matrix <- cov(data)

# Calculate the eigenvalues and eigenvectors
eigen_result <- eigen(cov_matrix)
eigenvalues <- eigen_result$values
eigenvectors <- eigen_result$vectors

# Print the results
cat("Covariance matrix:\n")
print(cov_matrix)
cat("\nEigenvalues:\n")
print(eigenvalues)
cat("\nEigenvectors:\n")
print(eigenvectors)
```

Singular Value Decomposition (SVD)

Theorem:

Let $A \in R^{m*n}$ be a rectangular matrix of rank r $\in$ [0, min(m, n)]. The SVD of A is a decomposition of the form:



Formula:

$$A = [x_1 \quad \cdots \quad x_m]_{n*m} = U\Sigma V^{\mathsf{T}} = [u_1 \quad \cdots \quad u_n]_{n*n} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_m \\ 0 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 0 \end{bmatrix}_{n*m} [v_1 \quad \cdots \quad v_m]^{\mathsf{T}}_{m*m}$$

Principal components matrix:

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

($\boldsymbol{c}_1$, $\boldsymbol{c}_2$ … $\boldsymbol{c}_n$ are orthogonal)

Projecting Down to d Dimension:

$$X_{d-proj} = XW_d$$

(W_d is the first d eigen vectors of data covariance matrix)

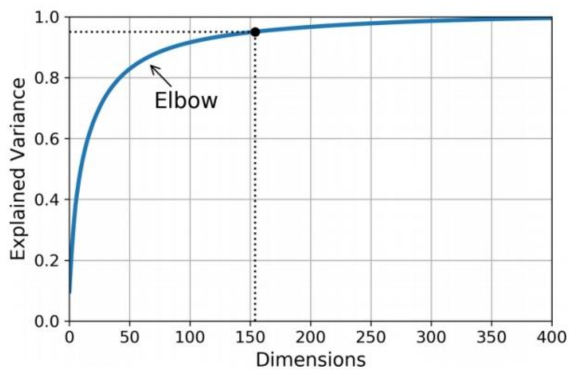
Explained Variance Ratio:

$$\frac{\lambda_1}{\lambda_1+\lambda_2 \ldots +\lambda_n} \text{ (eigenvalue/ total eigenvalue)}$$

Choosing the Right Number of Dimensions:

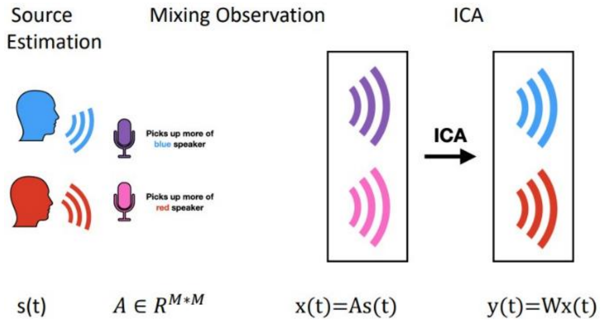
• Choose the number of dimensions that add up to sufficiently large portion of the variance (e.g., 95%)

• Variance Ratio > 95%

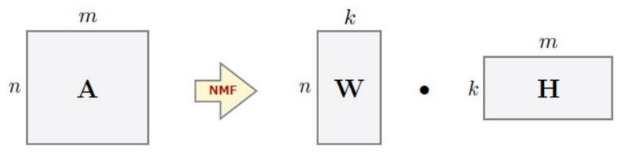


4.2 Independent Component Analysis (ICA)

ICA attempts to decompose (分解) a multivariate signal into independent non-Gaussian signals. Focuses on the mutual independence of the components. Vectors are not orthogonal.



4.3 Non-negative Matrix Factorization (NMF)



$$A \approx WH$$

Optimization: minimize $D(A\|WH)$ $\quad s.t.\ W, H \geq 0$

A matrix factorization where everything is non-negative

- $A \in R^{n*m}$ is original non-negative data
- $W \in R^{n*k}$ is matrix of basis vectors, dictionary elements
- $H \in R^{k*m}$ is matrix of activations, weights or gains
- NMF is not unique

4.4 Other Techniques

- Manifold Learning

  Data lies on d-dimensional manifold is a part of an n-dimensional space (where d < n)
- Locally Linear Embedding (LLE)

  LLE is a powerful nonlinear dimensionality reduction (NLDR) technique. It is a Manifold Learning technique that does not rely on projections.
- Multidimensional Scaling (MDS)

  Trying to preserve the distances between the instances.
- Isomap

  Trying to preserve the geodesic distances between the instances.
- t-Distributed Stochastic Neighbour Embedding (t-SNE)

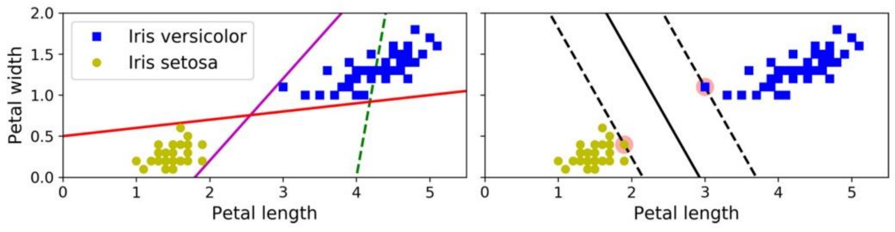
  Trying to keep similar instances close and dissimilar instances apart.

5 Support Vector Machine

5.1 Linear SVM Classification

5.1.1 Large Margin Classification

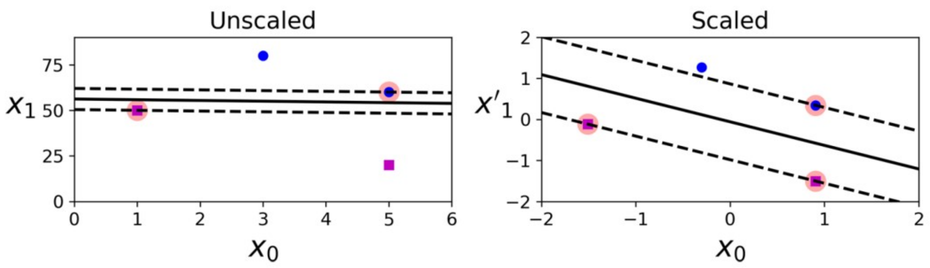
- If classes are linearly separable, two classes can be separated with a straight line
- SVM pursues the widest possible street between classes



- Decision boundary is not affected by more training instances
- It is determined by support vectors (instances located on the edge of street)

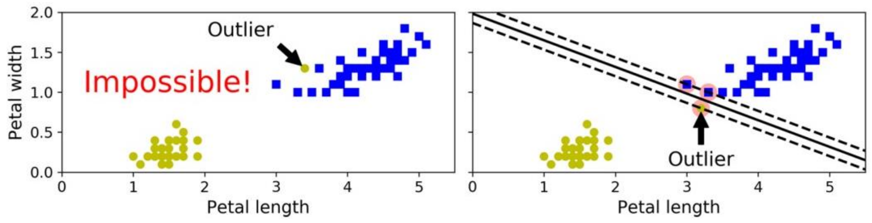
5.1.2 Feature Scales

The decision boundary could be much better if the feature is scaled:



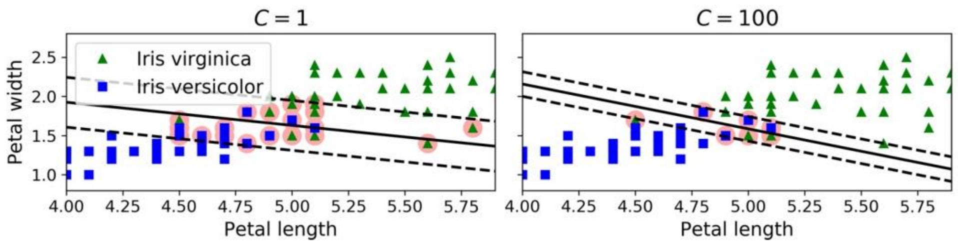
5.1.3 Hard Margin Classification

- All instances being off the street and on the right side is named "hard margin classification"
- The main limitation of hard margin classification is
    - The data must be linearly separable
    - There are little outliers



5.1.4 Soft Margin Classification

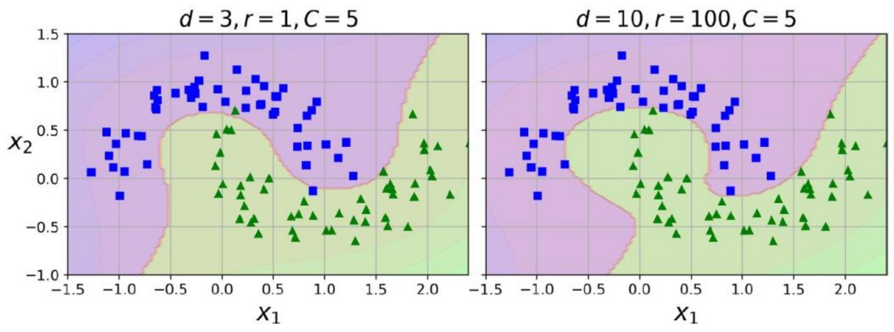
- Soft Margin Classification provides more flexibility
- The algorithm balances
    - The width of street
    - The amount of margin violations
- A hyper-parameter C is defined
    - A low value of C leads to more margin violation
    - A high value of C limits the flexibility



5.2 Nonlinear SVM Classification

5.2.1 Polynomial Kernel

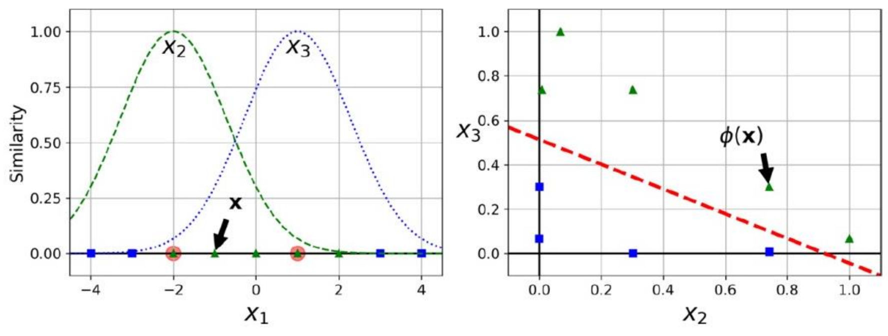
A kernel can be used instead of adding the polynomial features



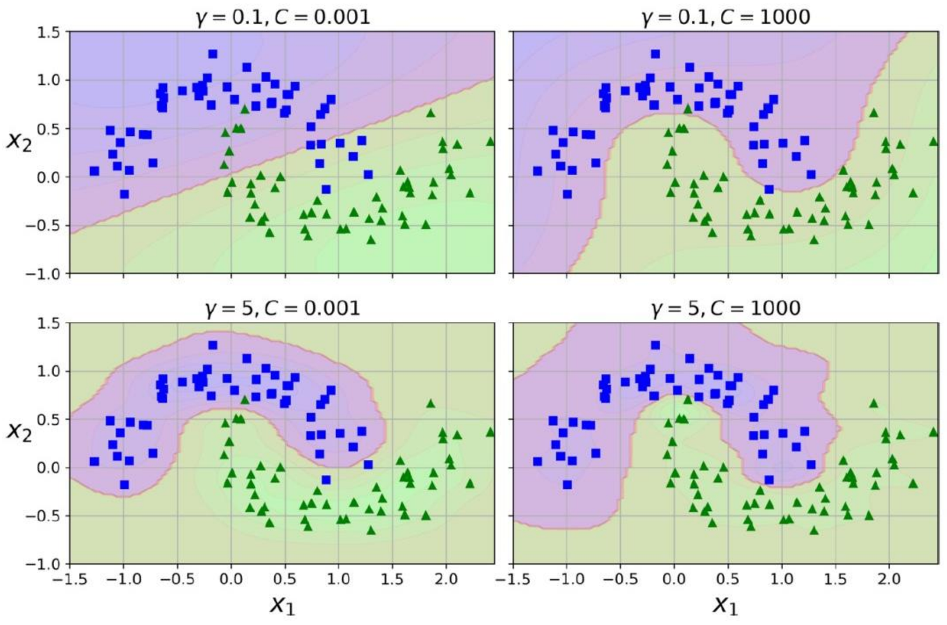
5.2.2 Similarity Feature

- Samples could be clustered with a landmark (reference point) l
- The similarity function could be defined as a Gaussian Radial Basis Function:

$$\Phi_\gamma(\mathbf{x}, l) = e^{-\gamma||\mathbf{x}-l||^2}$$

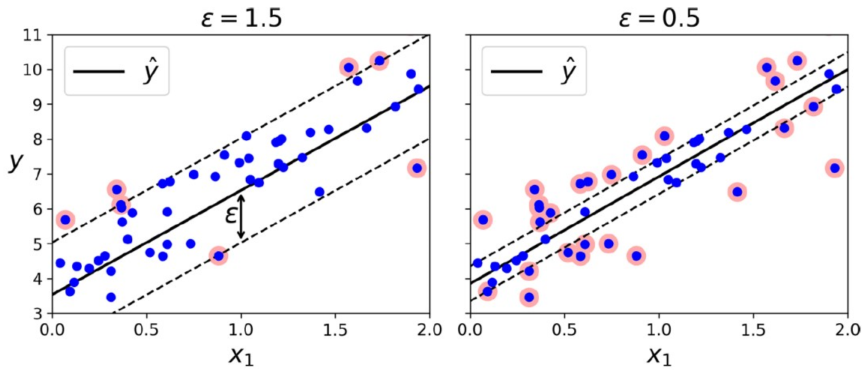


5.2.3 Gaussian RBF Kernel

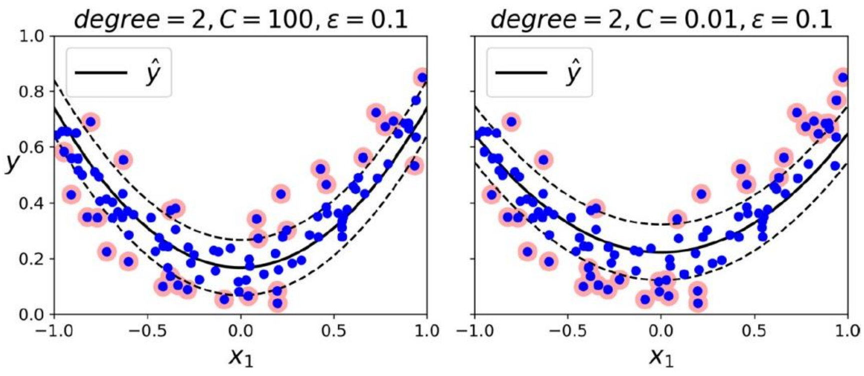


5.3 SVM Regression

- Fit as many instances as possible on the street
- Limit margin violations
- Width of street is controlled by a hyper-parameter ϵ



Use Kernel tricks:



6 Naïve Bayes

6.1 Bayes' Rule

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}$$

(Where c is considered as a class, **x** is considered as a set of samples)

- P(c) is named as prior probability
- P(**x**|c) is named as class-conditional probability (CCP, also known as "likelihood")

- P(c|**x**) is named as posterior probability
- P(**x**) is considered as evidence factor (observation)

6.2 Bayes' Rule for Classification

MAP estimation: maximise the posterior probability of observations

Presume that x ∊ Dc means that a sample x belongs to class c (Dataset)

Recall Bayes' Rule:

$$PosteriorProbability = \frac{CCP \times PriorProbability}{Observation}$$

As the observation is same (the same training dataset), we have:

$$PosteriorProbability \propto CCP \times PriorProbability$$

According to Law of Large Numbers, the prior probability can be taken as the probability resulted from the frequency of observations:

$$p(D|\Theta) = p(c)\prod_{c}\prod_{x_c \in D_c} p(x_c|\Theta_c)$$

6.3 Naïve Bayes Classifier

A way to simplify the process is to assume the conditions / features of Θ_c are independent to each other

So, assume that $\Theta_c = (\theta_{c1}, \theta_{c2}, \ldots, \theta_{cl})$, we have:

$$p(x_c|\Theta_c) = \prod_{i=1}^{l} p(c|\theta_i)$$

Example:

<Dataframe>

Outlook	Temprature	Humidity	Windy	Play
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No
Sunny	Hot	High	True	No
Sunny	Hot	High	False	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Q1: Will you play on the day of Mild?

Temp.	Yes	No	p
Hot	2	2	0.28
Mild	4	2	0.43
Cool	3	1	0.28
p	0.64	0.36	

1. By this table we have p(Mild|Yes) = 4/9 = 0.44 and p(Mild|No) = 2/5 = 0.4
2. Posterior p(Yes|Mild) = (p(Mild|Yes) x p(Yes)) / p(Mild) = (0.44 × 0.64) / 0.43 = 0.65
3. Posterior p(No|Mild) = (p(Mild| No) x p(No)) / p(Mild) = (0.4 × 0.36) / 0.43 = 0.33
4. As p(Yes|Mild) > p(No|Mild), it is likely to play.

Q2: Check INT104 NOTE (zhtc.one) Chapter-3 to see multiply conditions Naïve Bayes Classifier by using R

7 Non-parametric Classification

Parametric Methods limitations:

• We seldom obtain a "true" model due to the lack of prior knowledge

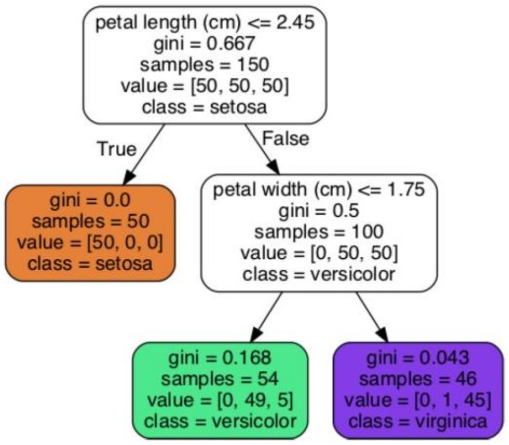
• For the same reason, we never know which model should be used

Non-parametric Methods merit:

• Can be used with arbitrary distributions and without the assumption that the forms of the underlying densities are known

• Can be used with multimodal distributions which are much more common in practice than unimodal distributions

7.1 Decision Tree



Content:

• Root
• Leaf
• Node
• Depth

7.1.1 Impurity

Gini is used to measure the impurity of node:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

($p_{i,k}$ is the ratio of class k instances among the training instances in the i-th node)

Entropy is also one of the possible ways to evaluate impurity:

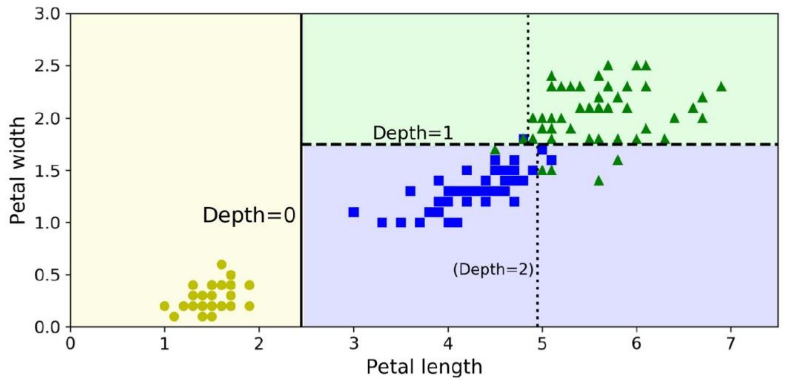
$$H_i = -\sum_{k=1}^{n} p_{i,k} \log_2(p_{i,k})$$

($p_{i,k}$ is the ratio of class k instances among the training instances in the i-th node ($p_{i,k}$ cannot be 0))

Question: A node applies to 0 Iris setosa, 49 Iris versicolor and 5 Iris virginica

```
Gᵢ = 1 - (49/54)² = 0.1766118
Hᵢ = -(49/54) * log₂(49/54) = 0.1271982
```

7.1.2 Decision Boundaries



<White box method>

7.1.3 Instability

Decision tree is sensitive to both training set rotation and the samples in training set:

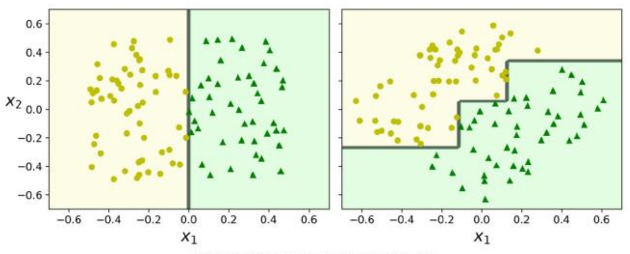


Figure 6-7. Sensitivity to training set rotation

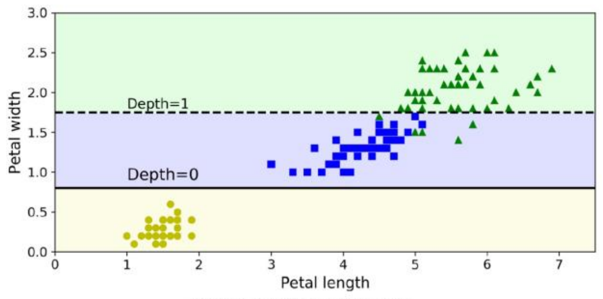


Figure 6-8. Sensitivity to training set details

7.1.4 Growing a tree

Classification and Regression Tree (CART)

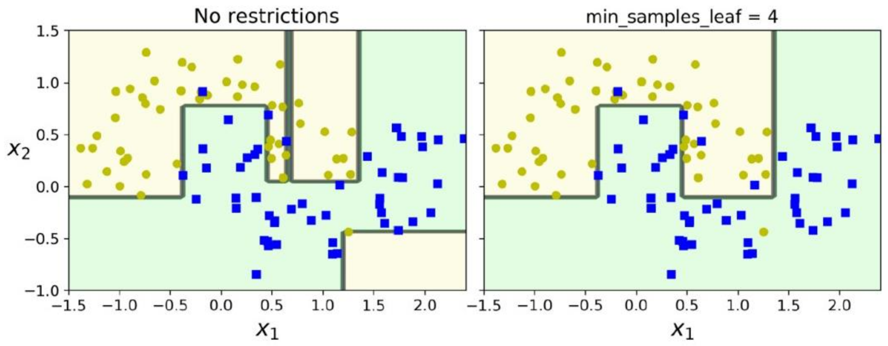
- Splitting a node into two subsets via single feature k and a threshold t_k

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

  - $G_{\text{left/right}}$ measures the impurity of the left/right subset
  - $m_{\text{left/right}}$ is the number of instances in the left/right subset

Question:

- When the tree should stop grow?
- What problem is introduced if the tree grows until perfect impurity?
- If we allow the tree to grow too much, it may become too complex and memorize the training data instead of generalizing to new data. To avoid this, we can use a stopping criterion that tells the tree when to stop growing. Some commonly used to stop criteria include: maximum depth, minimum number of samples per leaf node, minimum impurity decrease, and maximum number of leaf nodes.
- If we let the tree grow until it perfectly separates the training data into groups with the same class, it will likely overfit the training data and perform poorly on new data. This is because the tree is too specific to the training data and cannot generalize well to new data. Therefore, it is important to use an appropriate stopping criterion to balance model complexity and generalization performance.



7.2 k-Nearest Neighbour (KNN)

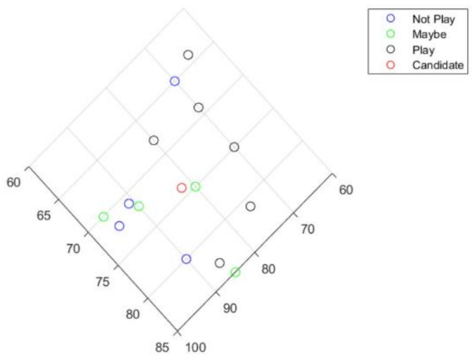
1. Load the data: First, we need to load the training dataset into memory.
2. Define the value of K: We need to choose a value of K, which represents the number of nearest neighbors to consider when making a prediction.
3. Calculate distance: For each new instance to be classified, we calculate its distance to all instances in the training dataset using a distance metric such as Euclidean distance or Manhattan distance.
4. Select the K-nearest neighbors: We then select the K instances in the training dataset that are closest to the new instance based on the calculated distances.
5. Assign the label: We assign the most common class label among the K-nearest neighbors to the new instance.
6. Repeat the process: We repeat this process for each new instance we want to classify.

7. Evaluate the model: Finally, we evaluate the performance of the KNN model by testing it on a separate testing dataset and calculating metrics such as accuracy, precision, recall, and F1 score.

Example:

outlook	temperature	humidity	windy	play
sunny	85	85	1.0	maybe
sunny	80	90	3.0	no
overcast	83	86	0.8	yes
rainy	70	96	0.2	maybe
rainy	68	80	0.1	yes
rainy	65	70	2.8	no
overcast	64	65	2.6	yes
sunny	72	95	0.3	no
sunny	69	70	0.5	yes
rainy	75	80	0.4	maybe
sunny	75	70	2.2	yes
overcast	72	90	2.4	maybe
overcast	81	75	0.0	yes
rainy	71	91	2.9	no

Q1: Will we play golf in the case of 74 (temperature), 74 (humidity)? What value of k should be selected?

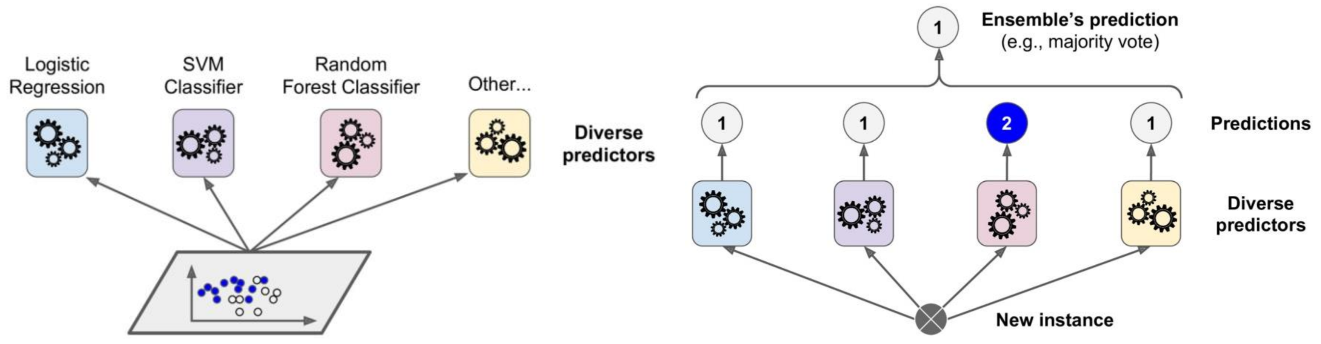


K=3

Q2: Check INT104 NOTE (zhtc.one) Chapter-1 to see the example of KNN by using R

8 Ensemble Learning & Random Forests

8.1 Voting Classifiers



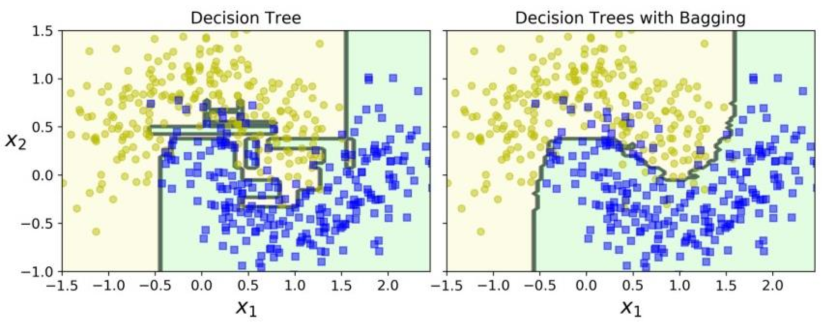
8.1.1 Ways to merge multiple classifiers:

Majority Voting, Weighted Voting, Bagging (Bootstrap Aggregating), Boosting, Stacking

8.1.2 Bagging and Pasting

Training with different subsets of data:

- Bagging: Sampling with Replacement. Since bootstrap sampling allows multiple instances to be selected multiple times in each subset, some instances may be included in multiple subsets, while others may be left out.
- Pasting: Sampling without Replacement. Each classifier is trained on a different subset, and their predictions are aggregated using averaging or voting.



8.2 Random Forest

8.2.1 Concepts

- Multiple decision trees with different samples and features can be grown
- Final decision could be made by voting

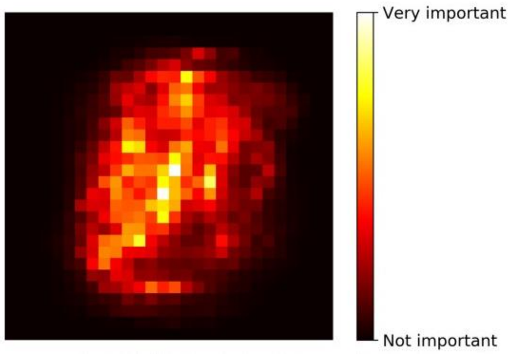
Random Forest achieves better performance than a single decision tree because it reduces overfitting, handles complex nonlinear relationships, handles high-dimensional data, and is robust to noisy data. It does this by building multiple trees on different subsets of the data and features and combining their predictions.

8.2.2 Random Patches & Random Subspaces

- Random Subspace: randomly select features for training
- Random Patches: randomly select features and randomly select samples

8.2.3 Feature Importance

Some common methods for measuring feature importance:

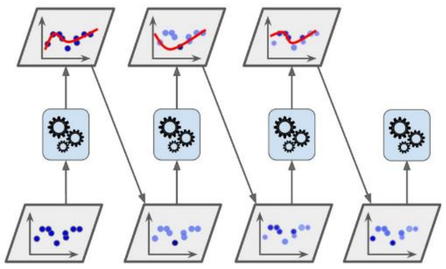


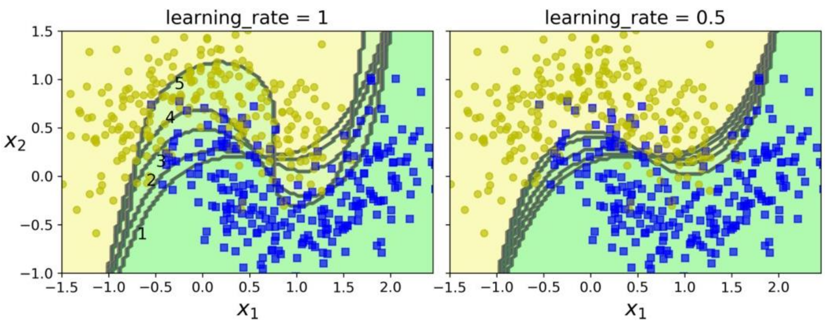
- Gini importance: For decision trees and random forests, the Gini importance is a measure of how often a feature is used to split the data across all trees in the ensemble. Features that are used more frequently to split the data have a higher importance.
- Permutation importance: This method measures the decrease in the model's performance when a feature's values are randomly permuted. Features that lead to a larger decrease in performance when permuted have a higher importance.
- Coefficient magnitude: For linear models, the magnitude of the coefficients reflects the contribution of each feature to the predicted outcome. Larger coefficients indicate higher feature importance.
- Recursive feature elimination: This method involves recursively removing features from the dataset and fitting a model on the remaining features until the optimal subset of features is found. The importance of a feature can be inferred from the order in which it is eliminated from the dataset.
- LASSO regularization: LASSO regularization can shrink some coefficients to zero, effectively removing some features from the model. The magnitude of the nonzero coefficients reflects the importance of each feature.

8.3 Boosting

8.3.1 AdaBoost (迭代推进算法)

- Weight samples in an iterative manner
- Train the classifier accordingly





8.3.2 Gradient Boosting (梯度推进算法)

- Residue is learned after each iteration

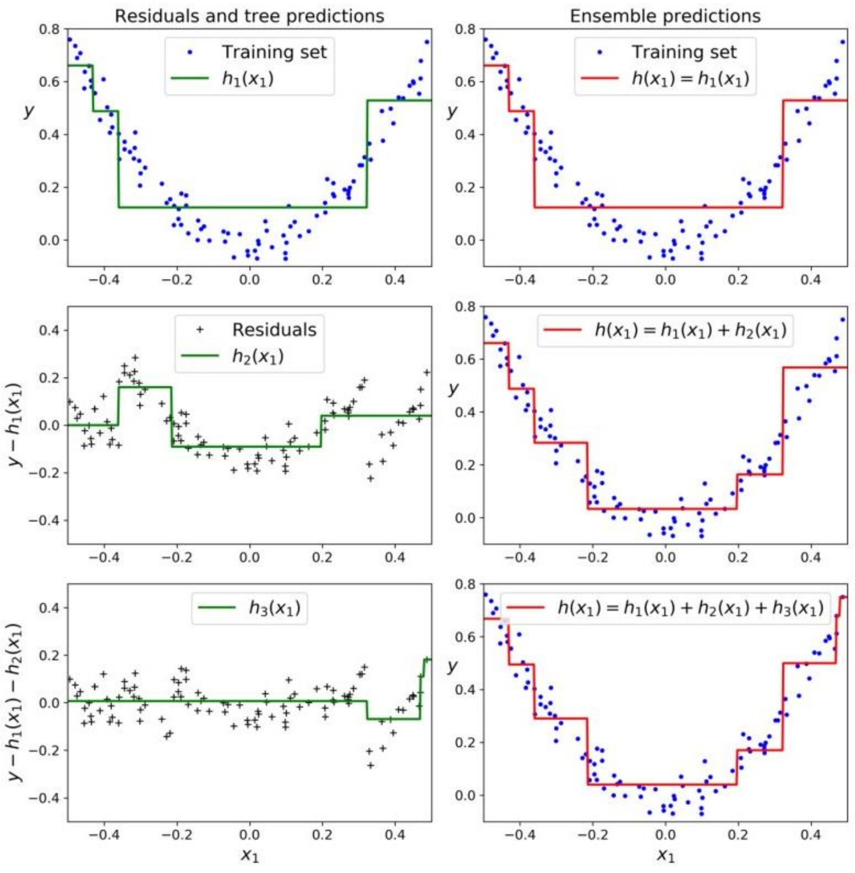


Figure 7-9. In this depiction of Gradient Boosting, the first predictor (top left) is trained normally, then each consecutive predictor (middle left and lower left) is trained on the previous predictor's residuals; the right column shows the resulting ensemble's predictions

8.3.3 Prevent Overfitting

By using early stop:

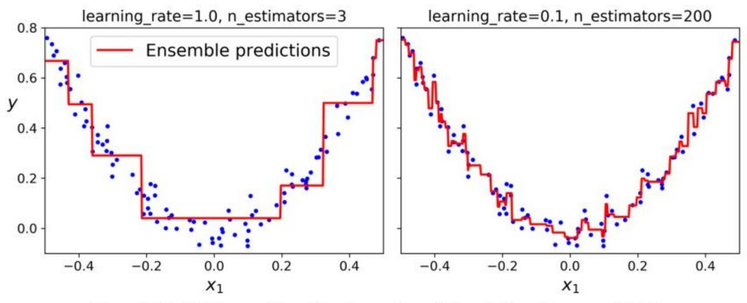


Figure 7-10. GBRT ensembles with not enough predictors (left) and too many (right)

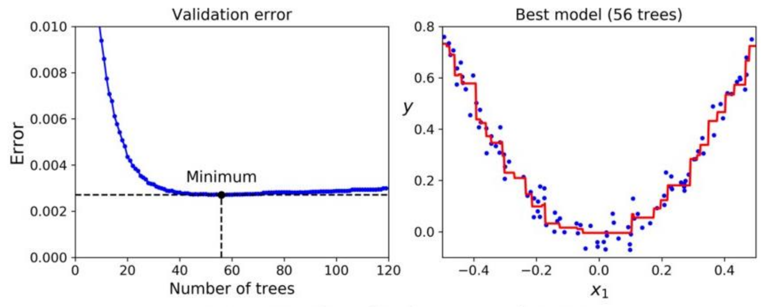
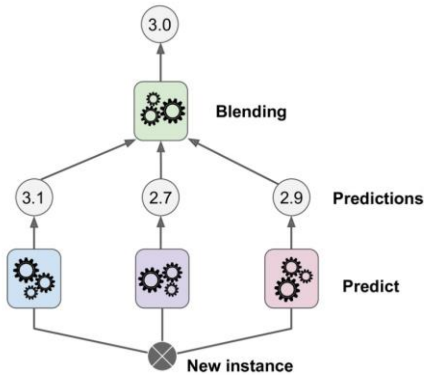


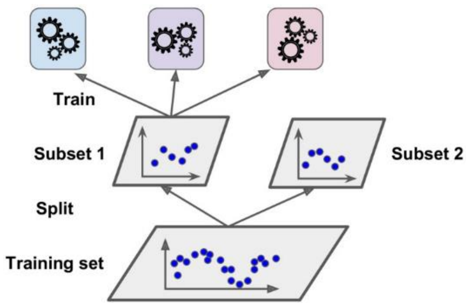
Figure 7-11. Tuning the number of trees using early stopping

8.3.4 **Stacking**

Combine the predictions of multiple base models by training a meta-model on the outputs of the base models:



1. Training the First Layer



[1] Split the data into training and testing sets.

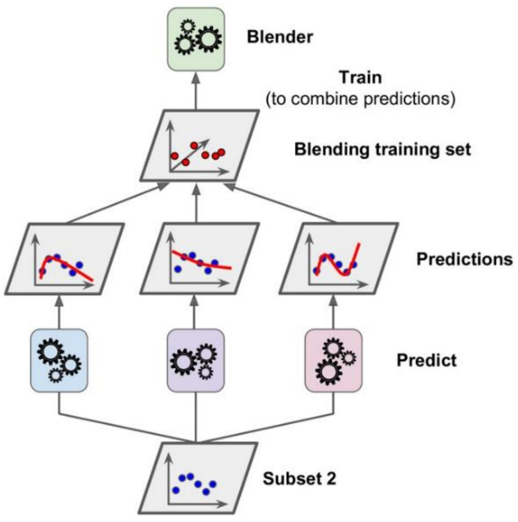
[2] Define the base models to use. These can be any machine learning models, such as decision trees, SVMs, neural networks, etc.

[3] Train each of the base models on the training data and record their predictions on the testing data.

[4] Use the predicted values as features for the meta-model.

[5] Train the meta-model on the predicted values and the actual target values to predict the final output.

2. Training the Blender



[1] Split the original training data into two parts: training and hold-out set.

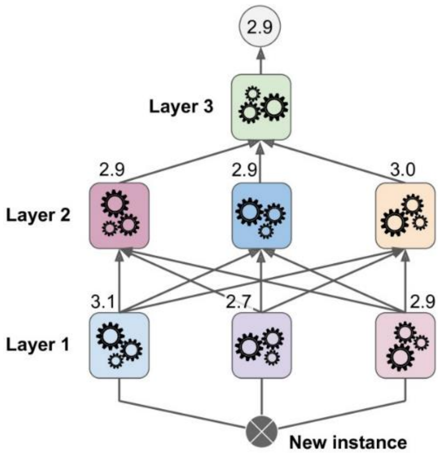
[2] Define the base models and train them on the training set.

[3] Use the trained base models to make predictions on the hold-out set.

[4] Concatenate the predictions made by the base models to create a new dataset.

[5] Train the blender on the new dataset and the target variable (or labels) of the hold-out set.

3. Predictions in a Multilayer Stacking Ensemble



[1] Split the original training data into three parts: training, validation, and hold-out set.

[2] Define the base models and train them on the training set.

[3] Use the trained base models to make predictions on the validation set.

[4] Concatenate the predictions made by the base models to create a new dataset, which becomes the input to the next layer.

[5] Define the second layer of models and train them on the predictions made by the base models in the first layer using the validation set.

[6] Use the trained second layer models to make predictions on the hold-out set.

[7] Concatenate the predictions made by the second layer models with the predictions made by the base models to create a new dataset, which becomes the input to the blender.

[8] Train the blender on new dataset and the target variable (or labels) of the hold-out set.

[9] Use the trained blender to make predictions on new data by passing the data through the base models to create the predicted features, and then passing those features through the second layer models to create a new set of features, and then finally passing those features through the blender to make the final prediction.