MATH5836: Data and Machine Learning

Week 2: Classification and Regularisation

Sarat Moka *UNSW, Sydney*

Key Topics

Books:

(A) *Pattern Recognition and Machine Learning* by Christopher Bishop (2006).

(B) *Mathematical Engineering of Deep Learning* book by Liquet, Moka, and Nazarathy (2024).

2.1 Logistic regression

Recall that in classification tasks, the goal is to predict categorical outcomes. In particular, for each sample i, we have a true label y_i and a vector of prediction probabilities $(\hat{p}_{i,1}, \ldots, \hat{p}_{i,K})$ with K denoting that number of classes and $\hat{p}_{i,k} = \mathbb{P}(y_i = k)$. For binary classification, we take the probability vector to be $(\hat{p}_i, 1 - \hat{p}_i)$ with $\hat{p}_i = \mathbb{P}(y_i = 1 \mid x_i)$, where x_i is the i-th row of the data matrix X given by

$$X = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,d} \\ 1 & x_{2,1} & \cdots & x_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,d} \end{bmatrix}.$$

Logistic regression is a classic statistical method used for binary classification tasks. It models the probability of a binary outcome (such as success/failure or 0/1) given one or more independent variables. In this section, we present the basic formulation and interpretation of logistic regression.

Formulation

Let $x_i = (1, x_{i,1}, \ldots, x_{i,d})$ be a vector of independent variables in the i-th sample and y_i be the corresponding binary dependent variable. The logistic regression model assumes a linear relationship between the independent variables and the log odds of the binary outcome:

$$\log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_p x_{i,d} = x_i^\top \beta, \quad i = 1, \ldots, n.$$

where p_i is the probability of the positive outcome, and $\beta = (\beta_0, \beta_1, \ldots, \beta_d)$ is the vector of the coefficients to be estimated.

In other words, for each vector of independent variable $x_i = (x_{1,1}, \ldots, x_{i,d})$, the corresponding dependent variable y_i takes values 0 or 1 with

$$\mathbb{P}(y_i = 1 \mid x_i) = p_i, \quad \text{and} \quad \mathbb{P}(y_i = 0 \mid x_i) = (1 - p_i).$$

Logistic Function: The *logistic function* $\sigma(z) = 1/(1+e^{-z})$—also known as the *sigmoid function*—is used to transform the linear combination of independent variables into a probability between 0 and 1:

$$p_i \ = \ \mathbb{P}(y_i = 1 \mid x_i) \ = \ \sigma(\beta^\top x_i) = \frac{1}{1 + \exp(-x_i^\top \beta)},$$

where exp is the exponential function.

Remark 2.1.1

Interpretation: The coefficients $\beta_1, \beta_2, \ldots, \beta_p$ represent the change in the log odds of the binary outcome for a one-unit change in the corresponding independent variable, holding all other variables constant.

Model Training: Maximum Likelihood Estimation

To estimate the coefficients of the logistic regression model, various optimization techniques such as maximum likelihood estimation or gradient descent are used. The goal is to find the coefficients that maximize the likelihood of the observed data given the model.

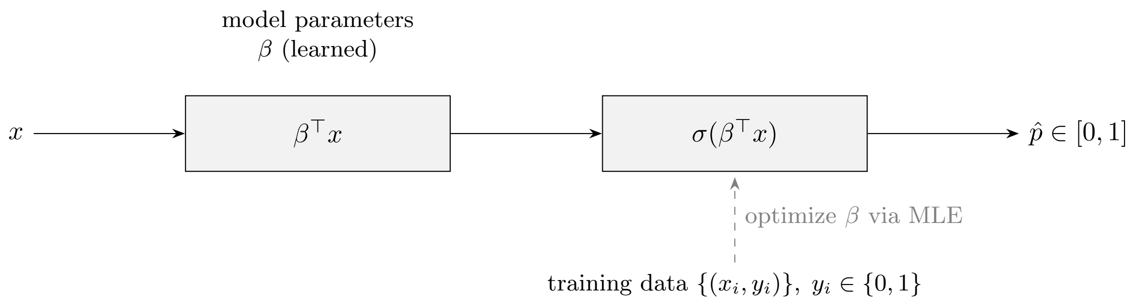


Figure 2.1: Logistic regression model. The linear combination $\beta^T x$ is passed through a sigmoid function $\sigma(\cdot)$ to produce a probability $\hat{p} = P(y = 1 \mid x)$. Parameters β are optimized via maximum likelihood estimation (MLE).

Specifically, we seek the parameter vector $\beta = (\beta_0, \beta_1, \ldots, \beta_d)$ that makes the observed data most probable under the model. Recall that

$$p_i \;=\; \mathbb{P}(y_i = 1 \mid x_i) \;=\; \sigma(\beta^\top x_i)\,,$$

where the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$.

Likelihood: Assuming independent samples,

$$L(\beta) \;=\; \prod_{i=1}^{n} p_i^{y_i} \left(1 - p_i\right)^{1-y_i}.$$

Log-Likelihood: Working with the log-likelihood simplifies products into sums as

$$\ell(\beta) \;=\; \log L(\beta) \;=\; \sum_{i=1}^{n} \big[y_i \log p_i + (1 - y_i) \log(1 - p_i)\big].$$

Gradient: Differentiating $\ell(\beta)$ w.r.t. β gives

$$\nabla_\beta \, \ell(\beta) = \sum_{i=1}^{n} \big(y_i - p_i\big) \, x_i.$$

Exercise 2.1.1

Use the following to train the logistic model:

- Show that maximizing the likelihood $L(\beta)$ is identical to minimizing $-\ell(\beta)$, the negative of log-likelihood.
- Show that $-\ell(\beta)$ is convex by proving that its Hessian is positive semi-definite.
- Thus, conclude that we can obtain a maximum likelihood estimate β^* by using the gradient descent on $C(\beta) = -\ell(\beta)$.

Model Evaluation

Once the logistic regression model is trained, it is evaluated using metrics such as accuracy, precision, recall, F1 score, and the receiver operating characteristic (ROC) curve, which we study soon. These metrics assess the performance of the model in predicting the binary outcome.

Extensions

Logistic regression can be extended to handle multi-class classification tasks using techniques such as one-vs-rest or multinomial logistic regression. Additionally, regularization techniques such as L1 and L2 regularization can be applied to prevent overfitting; more details later.

Remark 2.1.2

For complete details on the logistic regression, refer to Section 3.1 in Chapter 3 of the book *Mathematical Engineering of Deep Learning.*

Logistic Regression as a Shallow Neural Network

Let us first represent the logistic regression model as

$$\hat{y} = p = \underbrace{\sigma\left(\overbrace{b + w^\top x}^{z}\right)}_{a}, \tag{2.1}$$

with $b = \beta_0$, $w = (w_1, \ldots, w_d) = (\beta_1, \ldots, \beta_d)$, and σ denoting the sigmoid function, which we refer to as a scalar *activation function*. In this form, logistic regression model is a neural network; see Figure 2.2.

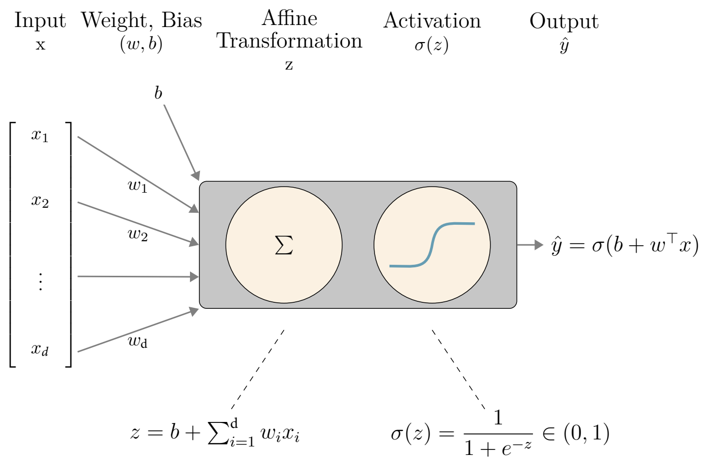


Figure 2.2: Logistic regression represented with neural network terminology as a shallow neural network. The gray box represents an artificial neuron composed of an affine transformation to create z and an activation $\sigma(z)$.

Remark 2.1.3

A trivial alternative activation function to the logistic function is the *identity activation function* $\sigma(z) = z$. With this identity activation function, the model in (2.1) is clearly just the linear model $\hat{y} = b + w^\top x$, which we studied in linear regression.

2.2 Softmax Regression

Softmax regression generalizes logistic regression to multi-class problems by modeling class probabilities using a normalized exponential function. This section presents the formulation and interpretation of softmax regression.

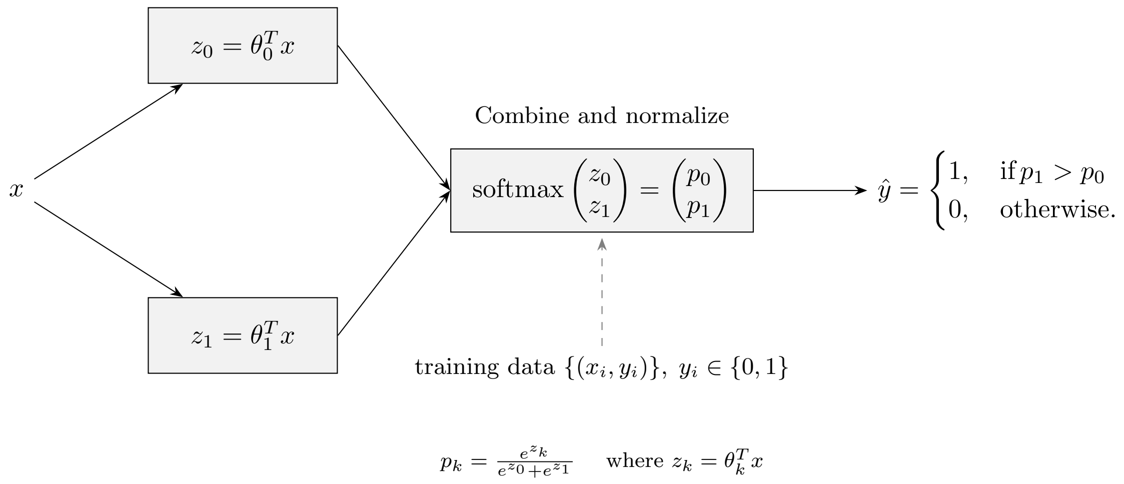
Let $x_i = (1, x_{i,1}, \ldots, x_{i,d})^\top$ be the feature vector for the i-th sample and $y_i \in \{1, \ldots, K\}$ its class label. The model assumes K linear relationships between features and logits:

$$z_{i,k} = x_i^\top \theta_k \quad \text{for } k = 1, \ldots, K \tag{2.2}$$

where $\theta_k = (\theta_{k,0}, \theta_{k,1}, \ldots, \theta_{k,d})^\top$ are the coefficients for class k (with $\theta_{k,0}$ being the intercept).

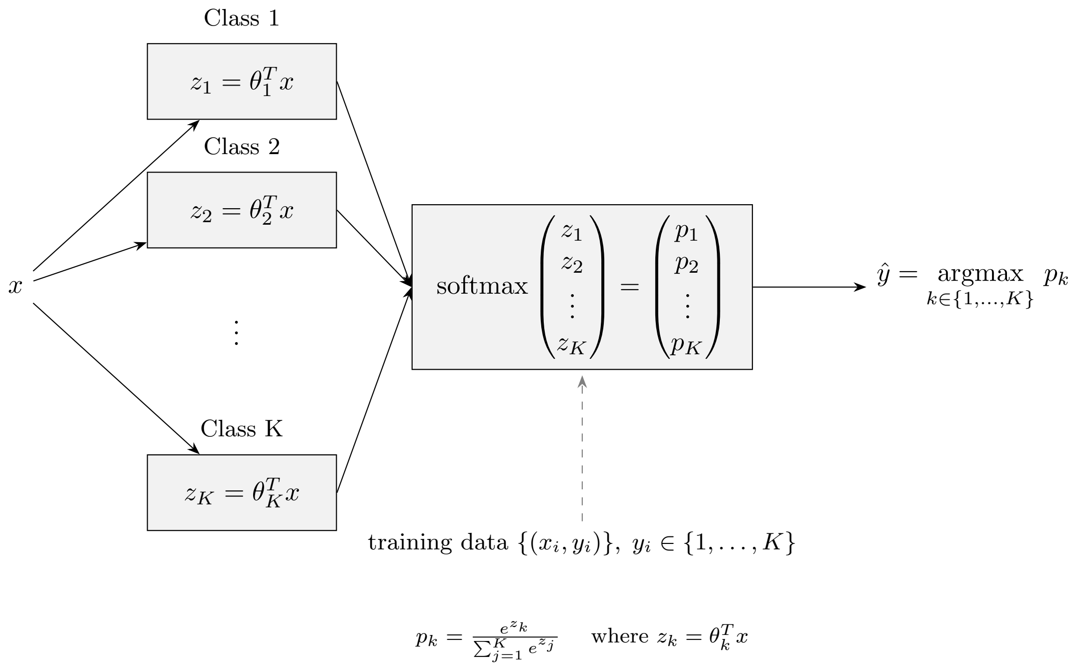
The softmax function converts these logits to class probabilities:

$$\mathbb{P}(y_i = k \mid x_i) = \frac{e^{z_{i,k}}}{\sum_{j=1}^{K} e^{z_{i,j}}} = \frac{e^{x_i^\top \theta_k}}{\sum_{j=1}^{K} e^{x_i^\top \theta_j}} \tag{2.3}$$



$z_0 = \theta_0^T x$
$z_1 = \theta_1^T x$
x
Combine and normalize
$\text{softmax}\begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$
$\hat{y} = \begin{cases} 1, & \text{if } p_1 > p_0 \\ 0, & \text{otherwise.} \end{cases}$
training data $\{(x_i, y_i)\},\ y_i \in \{0,1\}$
$p_k = \frac{e^{z_k}}{e^{z_0}+e^{z_1}}$ where $z_k = \theta_k^T x$

Figure 2.3: Softmax regression for binary classification. Input x is transformed by two parallel linear layers ($\theta_0^T x$ and $\theta_1^T x$), then combined via softmax to produce class probabilities. The predicted class $\hat{y}$ selects the highest probability.



Class 1
$z_1 = \theta_1^T x$
Class 2
$z_2 = \theta_2^T x$
x
$\vdots$
Class K
$z_K = \theta_K^T x$
$\text{softmax}\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_K \end{pmatrix}$
$\hat{y} = \underset{k\in\{1,\dots,K\}}{\text{argmax}}\ p_k$
training data $\{(x_i, y_i)\},\ y_i \in \{1,\dots,K\}$
$p_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}$ where $z_k = \theta_k^T x$

Figure 2.4: Softmax regression for multi-class classification. Input x feeds K parallel linear transformations (starting at class 1), with outputs normalized via softmax to produce class probabilities. The predicted class $\hat{y}$ selects the highest probability.

Decision Rule: The predicted class follows:

$$\hat{y}_i = \underset{k\in\{1,\dots,K\}}{\text{argmax}}\ \mathbb{P}(y_i = k \mid x_i)$$

Binary Special Case: When $K = 2$, this reduces to logistic regression:

$$\mathbb{P}(y_i = 1 \mid x_i) = \sigma(x_i^\top(\theta_1 - \theta_2))$$

where recall that $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function; see Figure 2.3.

2.3 Bias and variance

Bias and variance are two fundamental concepts in machine learning and statistics that describe the behavior of models in terms of their prediction errors. In this section, we discuss bias, variance, and the bias-variance tradeoff.

Bias: Bias refers to the error introduced by approximating a real-world problem with a simplified model. A model with high bias makes strong assumptions about the underlying data and may fail to capture complex patterns. Common examples of high bias models include linear regression with insufficient features and decision trees with shallow depths.

Mathematically, bias is defined as the difference between the expected prediction of the model $\hat{y} = \hat{f}(\boldsymbol{x})$ and the true value $y = f(\boldsymbol{x})$ for an unseen data point $(\boldsymbol{x}, y)$:

$$\text{Bias}(\hat{y}) = \mathbb{E}[\hat{y}] - y$$

where $\mathbb{E}[\hat{y}]$ is the expected value of the predicted value.

Variance: Variance measures the variability of the model's predictions for a given data point. A model with high variance is sensitive to small fluctuations in the training data and may overfit by capturing noise instead of true patterns. Examples of high variance models include decision trees with deep depths and complex neural networks.

Mathematically, variance is defined as the expected squared difference between the predicted value of the model and the expected prediction:

$$\text{Var}(\hat{y}) = \mathbb{E}[(\hat{y} - \mathbb{E}[\hat{y}])^2] = \mathbb{E}[\hat{y}^2] - (\mathbb{E}[\hat{y}])^2$$

Bias-Variance Tradeoff: The bias-variance tradeoff refers to the balance between bias and variance when designing machine learning models. In general, increasing the complexity of a model reduces bias but increases variance, and vice versa. The goal is to find the optimal balance that minimizes the overall prediction error.

- **High Bias, Low Variance:** Simple models with high bias and low variance tend to underfit the data by oversimplifying the underlying patterns.
- **Low Bias, High Variance:** Complex models with low bias and high variance tend to overfit the data by capturing noise instead of true patterns.

- **Optimal Tradeoff:** The optimal tradeoff between bias and variance depends on the specific problem and the amount of available data. Techniques such as regularization and cross-validation can help find the optimal balance.

Understanding bias and variance is essential for designing effective machine learning models that generalize well to unseen data.

Generally, as model complexity increases, expected training performance improves (decreases) since complex structured models can explain the training data better. At high extremes this is overfitting. Similarly an opposite phenomenon is that models with low complexity are not able to describe the data well. The tradeoff between these two regimes is obtained at the minimum of the expected generalized performance on an unseen data, marked by the vertical dashed line in Figure 2.5.

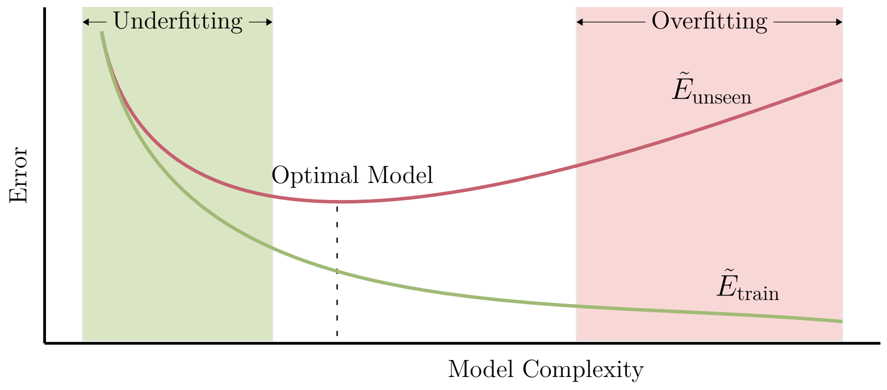


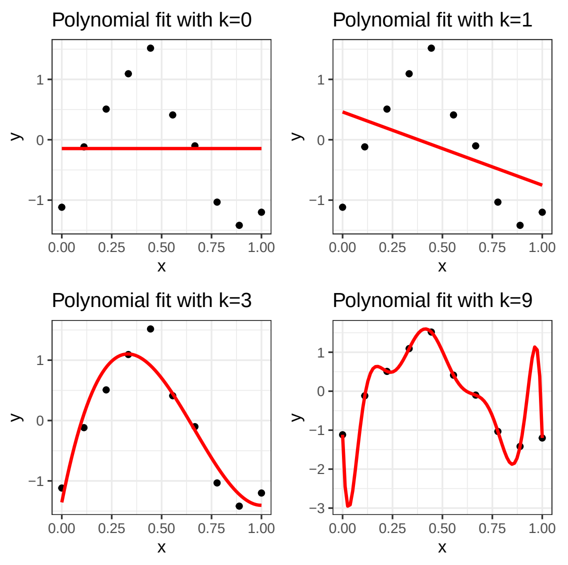
Figure 2.5: Typical behaviour of expected generalization performance on unseen data (red curve) and expected training performance on training data (blue) as a function of model complexity

Example 2.3.1

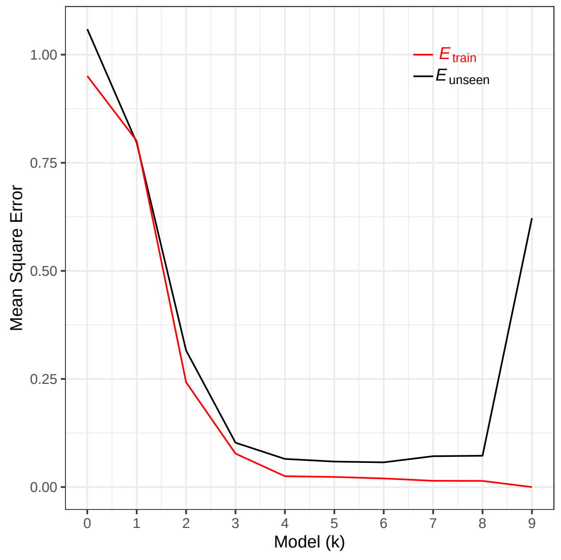
As one simple illustrative example capturing the tradeoffs of model complexity, let us consider linear models with polynomial features applied to synthetic univariate ($p = 1$) data. The model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_k x^k + \epsilon, \tag{2.4}$$

where k is the order of the polynomial. Hence, the model with $k = 0$ is the constant model, the model with $k = 1$ is the simple linear model, the model with $k = 2$ is the quadratic model, and so on. In this framework, model complexity corresponds to the degree of the polynomial model is illustrated in Figure 2.6.



(a)



(b)

Figure 2.6: Increasing model complexity illustrated via linear models with polynomial features where k, the order of the polynomial, captures the complexity. (a) Fitting several models to a single realization with $n = 10$ data-points. (b) The training performance in red and simulation estimates of the generalization performance in black.

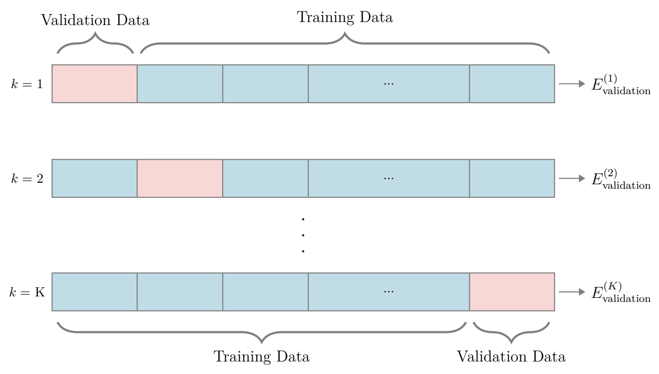
2.4 Cross validation

Cross-validation is a widely used technique in machine learning for assessing the generalized performance of a predictive model. It helps to estimate how well the model will generalize to unseen data. In this section, we discuss the concept of cross-validation and various commonly used cross-validation techniques.

The main idea behind cross-validation is to partition the available data into multiple subsets, called folds. The model is trained on a subset of the data and evaluated on the remaining data. This process is repeated multiple times, with different subsets used for training and evaluation each time. The performance metrics obtained from each iteration are then averaged to obtain a more reliable estimate of the model's performance.

K-Fold Cross-Validation

K-fold cross-validation is one of the most commonly used cross-validation techniques. In K-fold cross-validation, the given data is divided into K subsets of approximately equal size. The model is trained K times, each time using K-1 folds for training and the remaining fold for evaluation. The performance metrics obtained from each iteration are then averaged to obtain the final performance estimate which provides an estimation of the expected generalized performance; see Figure 2.7 for an illustration.



Validation Data
Training Data
$k = 1$
$E^{(1)}_{\text{validation}}$
$k = 2$
$E^{(2)}_{\text{validation}}$
$k = \text{K}$
$E^{(K)}_{\text{validation}}$
Training Data
Validation Data

Figure 2.7: K-fold cross validation. For each $k = 1, \ldots, K$ the data is split into training data and validation data differently. This yields K estimates for performance and these estimates can be averaged.

Leave-One-Out Cross-Validation (LOOCV)

Leave-One-Out Cross-Validation (LOOCV) is a special case of K-fold cross-validation where K equals the number of data points in the dataset. In LOOCV, the model is trained K times, each time using all but one data point for training and the remaining data point for evaluation. This process is repeated for each data point in the dataset. LOOCV provides an unbiased estimate of the model's performance but can be computationally expensive, especially for large datasets.

Stratified Cross-Validation

Stratified cross-validation is used when the dataset is imbalanced, i.e., the distribution of classes is uneven. In stratified cross-validation, the data is divided into folds such that each fold contains approximately the same proportion of samples from each class as the original dataset. This ensures that the model is trained and evaluated on representative samples from each class.

Nested Cross-Validation

Nested cross-validation is used for model selection and hyperparameter tuning. In nested cross-validation, the data is divided into an outer loop and an inner loop. The outer loop is used for model evaluation, while the inner loop is used for hyperparameter tuning. Nested cross-validation provides an unbiased estimate of the model's performance and helps to prevent overfitting during model selection.

Remark 2.4.1

Cross-validation is a powerful technique for estimating the performance of predictive models and selecting the best model for a given task. By using cross-validation, researchers and practitioners can ensure that their models generalize well to unseen data and make reliable predictions in real-world scenarios.

2.5 Confusion matrix

A confusion matrix is a performance measurement tool used in classification tasks to evaluate the performance of a predictive model. It provides a summary of the predicted and actual class labels for a given dataset. In this section, we discuss the components of a confusion matrix and how it is used to assess the performance of a classification model.

Components of a Confusion Matrix

A confusion matrix is typically represented as a square matrix with rows and columns corresponding to the actual and predicted class labels, respectively. To define the components of this matrix, suppose for a classifier is constructed via a decision rule based on a threshold τ, with the predicted output being,

$$\hat{y}_i = \begin{cases} 1, & \text{if } \hat{p}_i \geq \tau, \\ 0, & \text{if } \hat{p}_i < \tau, \end{cases} \tag{2.5}$$

Then the main components of a confusion matrix are as follows:

- **True Positives (TP):** The number of instances that belong to the positive class (actual positive) and are correctly predicted as positive. That is, the number of instances i where $\hat{Y}_i = 1$ and $y_i = 1$.
- **False Positives (FP):** The number of instances that belong to the negative class (actual negative) but are incorrectly predicted as positive. That is, the number of instances i where $\hat{Y}_i = 1$ for $y_i = 0$.
- **True Negatives (TN):** The number of instances that belong to the negative class (actual negative) and are correctly predicted as negative. That is, the number of instances i where $\hat{Y}_i = 0$ and $y_i = 0$.
- **False Negatives (FN):** The number of instances that belong to the positive class (actual positive) but are incorrectly predicted as negative. That is, the number of instances i where $\hat{Y}_i = 0$ for $y_i = 1$.

Interpretation

The confusion matrix provides a detailed breakdown of the model's performance in terms of true positives, false positives, true negatives, and false negatives. It allows us to calculate various performance metrics, such as accuracy, precision, recall, and F1 score, which provide insights into different aspects of the model's performance.

Example 2.5.1

Consider a binary classification task where we are predicting whether an email is spam (positive class) or not spam (negative class). A confusion matrix for this task might look like the following:

$$\begin{bmatrix} \text{TN} & \text{FP} \\ \text{FN} & \text{TP} \end{bmatrix} = \begin{bmatrix} 950 & 20 \\ 30 & 1000 \end{bmatrix}$$

In this example, there are 950 true negatives (non-spam emails correctly classified), 20 false positives (non-spam emails incorrectly classified as spam), 30 false negatives (spam emails incorrectly classified as non-spam), and 1000 true positives (spam emails correctly classified).

Remark 2.5.1

The confusion matrix is a valuable tool for evaluating the performance of classification models and gaining insights into their strengths and weaknesses. By analyzing the components of the confusion matrix, we can identify areas for improvement and make informed decisions about model selection and optimization.

2.6 Receiver Operating Characteristics (ROC) curve

The Receiver Operating Characteristics (ROC) curve is a graphical tool widely used in binary classification to assess the performance of a classifier across different decision thresholds. Let's explore the mathematical formulation and interpretation of the ROC curve in detail.

Mathematical Concept

The ROC curve is a plot of the true positive rate (TPR) (sensitivity) against the false positive rate (FPR) (1 - specificity) for various decision thresholds τ. To define formally, recall the binary classifier given in (2.5) and define,

$$\text{TPR}(\tau) = \frac{TP}{TP + FN}, \quad \text{and} \quad \text{FPR}(\tau) = \frac{FP}{TN + FP}.$$

Then, ROC is the curve of TPR(τ) vs FPR(τ)).

Interpretation

The ROC curve provides a visual representation of a classifier's trade-off between sensitivity and specificity at different decision thresholds. A curve that hugs the upper left corner indicates superior performance, as it achieves high sensitivity with low false positive rate. Conversely, a curve closer to the diagonal line suggests poorer discrimination ability.

Example 2.6.1

Consider a logistic regression set-up with logistic function being

$$p_i = \frac{1}{1+\exp(-(\beta_0+\beta_1 x_1+\beta_2 x_1^2+\cdots+\beta_k x_i^k))},$$

or, equivalently,

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0+\beta_1 x_1+\beta_2 x_1^2+\cdots+\beta_k x_i^k$$

is a polynomial of degree k. Compare this with (2.4).
Figure 2.8 illustrates ROC for an example dataset. For more details on ROC and the dataset used in the figure, refer to Section 2.2 in Chapter 2 of the book *Mathematical Engineering of Deep Learning.*

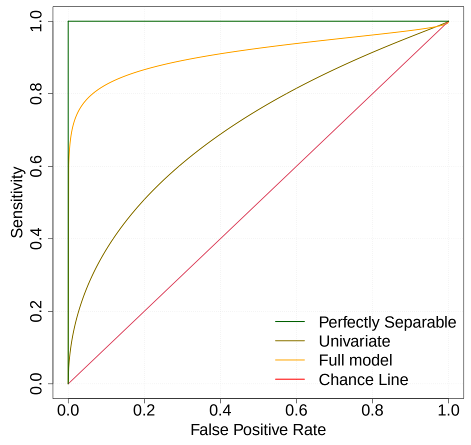


Figure 2.8: Receiver operating characteristic (ROC) curves for the breast cancer data. One model is a univariate model ($k = 1$), and the other is with $k = 30$. A chance line (guessing model) and a perfectly separable line (ideal model) are also plotted. For each model, the ROC captures the tradeoff between the sensitivity and the false positive rate (one minus specificity).

Advantages

The ROC curve is advantageous as it provides a comprehensive visualization of a classifier's performance across different decision thresholds. It allows for easy comparison between different classifiers and facilitates the selection of an appropriate operating point based on the specific application requirements.

Area Under The Curve (AUC)

ROC curves allow us to asses the quality of models taking all possible threshold parameters into account. A related measure that tries to quantify the quality of a curve into a single number is the *area under the curve*(AUC) measure. For a classifier with an ROC curves that achieves perfect sensitivity under any level of specificity this measure is at 1 and corresponds to the perfectly separable green curve in Figure 2.8. However for classifiers that just choose a random class, this measure is at 0.5 corresponding to the chance line, red line in Figure 2.8. In the case of the breast cancer data we see that on the test set the AUC for the $k = 1$ model is 0.70 and for the $k = 30$ model it is at 0.92. This may give an indication that the additional features in the richer model help obtain a better predictor.

2.7 Regularisation

Regularization is a technique used in machine learning and statistics to prevent overfitting by adding a penalty term to the loss function. It discourages overly complex models by penalizing large parameter values. In this section, we discuss three commonly used regularization techniques: Lasso, Ridge, and ElasticNet.

A common way to keep model parameters at bay is to augment the optimization objective $\min_\theta C(\theta)$ with an additional *regularization term* $R_\lambda(\theta)$. The revised objective is then,

$$\min_{\theta} \; C(\theta) + R_\lambda(\theta). \tag{2.6}$$

The regularization term $R_\lambda(\theta)$ (also called penalty term) depends on a *regularization parameter* λ, which is often a scalar in the range $[0, \infty)$ but also sometimes a vector. This hyper-parameter allows us to optimize the bias and variance tradeoff.

Lasso (L1 Regularization)

Lasso, short for Least Absolute Shrinkage and Selection Operator, adds a penalty term proportional to the absolute value of the coefficients to the loss function. Mathematically, the Lasso regularization term is defined as:

$$R_\lambda(\theta) = \lambda \sum_{j=1}^{p} |\theta_j|.$$

where λ is the regularization parameter and β_j are the model coefficients. Lasso encourages sparsity in the model, leading to feature selection by shrinking some coefficients to zero.

Ridge (L2 Regularization)

Ridge regularization adds a penalty term proportional to the square of the coefficients to the loss function. Mathematically, the Ridge regularization term is defined as:

$$R_\lambda(\theta) = \lambda \sum_{j=1}^{p} \theta_j^2.$$

Ridge regularization penalizes large coefficients while maintaining all features in the model. It tends to shrink the coefficients towards zero without completely eliminating them.

ElasticNet Regularization

ElasticNet regularization combines Lasso and Ridge penalties by adding both L1 and L2 regularization terms to the loss function. The ElasticNet penalty is defined as a weighted combination of the Lasso and Ridge penalties:

$$R_\lambda(\theta) = \lambda_1 \sum_{j=1}^{p} |\theta_j| + \lambda_2 \sum_{j=1}^{p} \theta_j^2.$$

where λ_1 and λ_2 are the regularization parameters controlling the strength of L1 and L2 regularization, respectively.

Choice of Regularization Parameter

The choice of the regularization parameter (λ for Lasso and Ridge, λ_1 and λ_2 for ElasticNet) is crucial and is typically determined using techniques such as cross-validation or grid search.

Remark 2.7.1

In summary, regularization techniques such as Lasso, Ridge, and ElasticNet are powerful tools for preventing overfitting and improving the generalization of machine learning models. By adding penalty terms to the loss function, these techniques encourage simpler models and feature selection, leading to better performance on unseen data.

Example: Regularized Linear Regression Optimization

Recall that in the linear regression, model parameters are $\beta = (\beta_0, \beta_1, \dots, \beta_d)^\top$ and the loss function is the mean squared error given by

$$C(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2.$$

We now explicitly state the optimization problems for all three regularization cases:

1. Lasso Regression (ℓ_1 regularization):

$$\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2 + \lambda \sum_{j=1}^{d} |\beta_j| \right\}.$$

2. Ridge Regression (ℓ_2 regularization):

$$\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \beta_j^2 \right\}.$$

3. Elastic Net (Combined $\ell_1 + \ell_2$):

$$\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2 + \lambda_1 \sum_{j=1}^{d} |\beta_j| + \lambda_2 \sum_{j=1}^{d} \beta_j^2 \right\}.$$