Data, Features, and Evaluation*

February 20, 2025

1 Introduction

In previous lecture notes, we delved into various classification techniques and classifiers. While understanding these methods is crucial, it's equally important to comprehend the fundamental aspects of the data we work with, the features we extract, and the evaluation metrics we employ to assess the performance of our models. Understanding data, feature engineering, and choosing appropriate evaluation metrics are fundamental pillars of successful machine learning practice.

2 Data

Processing data is a crucial step in the machine learning. It prepares the data for use in building and training machine learning models. The overall goal is to clean, transform, and prepare the data in a format suitable for modelling.

Figure 1 illustrates the data processing workflow. (1) Collecting data. It involves pulling data from numerous sources. An example is to collect point cloud data using LiDAR scanning devices; (2) Preparing data. It includes cleaning and structuring data to remove errors, anomalies, redundancy, etc. It also includes data annotation, and splitting the data into training/validation/testing sets; (3) Transport the data to a machine learning algorithm; (4) Analyze the data using the selected algorithm to derive insights or knowledge; (5) Output the analysis and interpret the results; (6) Store the data where the information is preserved for future use.

When it comes to up-to-date deep learning, more complex data processing techniques can be applied. One important technique is batch processing, which tackles massive data by grouping them into small batches. It allows to process the grouped batches in parallel, maximizing the efficiency and encouraging faster convergence. Another noteworthy technique is data augmentation. It generates high-quality artificial data by creating modified copies of existing data. This helps to train a deep learning network and significantly reduces overfitting.

*References

- Christopher Bishop. Pattern Recognition and Machine Learning. 2006
- Sergios Theodoridis, Konstantinos Koutroumbas. Pattern Recognition. 2009
- Eugenio Zuccarelli. Performance Metrics in Machine Learning. 2020

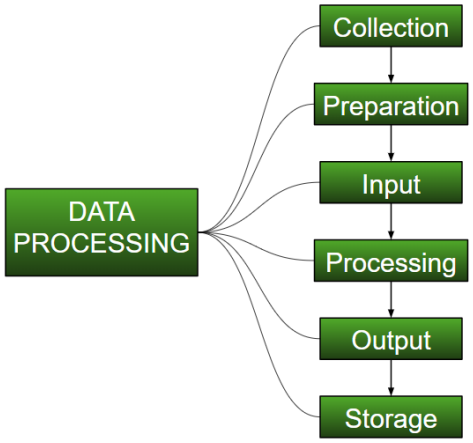

Figure 1: Data processing workflow in machine learning. Image sourced from online.

2.1 Data representation

Data in machine learning is typically represented in tabular form, where rows correspond to instances or samples, and columns represent attributes or features. It's imperative to comprehend the structure and nature of the data we're working with.

For most machine learning algorithms, it's a common practice to structure the input data as a matrix $\mathbf{X} \in R^{n \times m}$, e.g.,

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1m} \\ x_{21} & x_{22} & x_{23} & ... & x_{2m} \\ x_{31} & x_{32} & x_{33} & ... & x_{3m} \\ ... & ... & ... & ... & ... \\ x_{n1} & x_{n2} & x_{n3} & ... & x_{nm} \end{bmatrix}, \tag{1}$$

where each column represents a feature, an individual measurable property of a phenomenon observed. Each row represents a recorded data sample $\mathbf{x}_i$, e.g.,

$$\mathbf{x}_i = [x_{i1}, x_{i2}, x_{i3}, ..., x_{im}]^T \tag{2}$$

2.2 Data types

Data can be categorical, numerical, ordinal, or textual. Each type requires distinct preprocessing techniques before feeding into machine learning algorithms.

- **Categorical data** derives from observations made of qualitative data that are summarised as counts, or from observations of quantitative data grouped within given intervals. The probability distribution associated with a random categorical variable is called a categorical distribution.
- **Numerical data**, also referred to quantitative data, is in the form of numbers. Unlike categorical data which uses description or vocabulary, numerical data only consists of numbers that can be measured and counted. Numerical data include discrete data and continuous data.
- **Ordinal data** is a specific categorical data type where the variables have natural, ordered categories and the distances between the categories are not known.

- **Textual data** is information stored and written in a text format. It's any data that has been expressed in words.

In our course, both numerical data and categorical data are frequently used. For example, in A2, point cloud features are extracted and stored in the numerical format. A supervised machine learning classifier takes numerical features as input, and then predicts a probability distribution over semantic classes which can be viewed as the categorical data type. Processing textual data typically requires another group of techniques (i.e., Natural Language Processing techniques) and therefore is out of the scope of our course.

2.3 Data preprocessing

This involves handling missing values, handling outliers, scaling features, encoding categorical variables, and splitting data into training and testing sets.

- **Data cleaning**. Raw data possibly consists of noises, outliers, and missing values under certain fields. Therefore, they need to be cleaned before feeding into a machine learning algorithm. Missing values can occur due to failure to record data or data corruption. Often, we handle them by either deleting the data entry or substituting reasonable estimates or guesses for missing values; On the other hand, outliers, deviating significantly from the norm, can greatly distort measures of data tendency. They can be removed by visualization tools or statistical analyses.

- **Data scaling and normalization**. The purpose is to ensure all features have the same scale to prevent dominance by certain features. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. Also, in deep neural networks, gradient descent converges much faster with feature scaling than without it.

- **Label encoding**. This is used to convert categorical labels into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project. It assigns a unique integer to each category in the data, making it suitable for machine learning models that work with numerical targets.

- **Data spliting**. Given data is divided into two or more subsets so that a model can get trained, tested and evaluated. It is always recommended to split the data into three sets: training set, testing set and optionally validation set. The model is trained on the training set and its performance is evaluated in another independent test set. Sometimes, an extra validation set is used to understand the performance of the model in comparison to different models and hyperparameter choices.

3 Feature engineering

Feature engineering involves creating new features or modifying existing ones to enhance the performance of machine learning models. Well-engineered features can significantly impact the performance of machine learning models, sometimes even more so than the choice of algorithm.

3.1 Feature selection

A major problem associated with machine learning is the so-called *curse of dimensionality*[1]. In practice, it is very common to design a classification system with dozens, or even hundreds of features. There is more than one reason to reduce the number of features to a sufficient minimum. Computational complexity is the obvious one. A related reason is that, although two features may carry good classification information when treated separately, there is little gain if they are combined into a feature vector because of a high mutual correlation. Thus, complexity increases without much gain. Another major reason is imposed by the required generalization properties of the classifier. The higher the ratio of the number of training patterns N to the number of free classifier parameters, the better the generalization properties of the resulting classifier.

A large number of features are directly translated into a large number of classifier parameters (e.g., weights in a linear classifier, synaptic weights in a neural network). Thus, for a finite and usually limited number of training patterns, keeping the number of features as small as possible is in line with our desire to design classifiers with good generalization capabilities. To this end, different strategies will be adopted. One is to select features with sufficient discriminatory capability. An alternative is to use feature reduction techniques such as Principle Component Analysis (see Section 3.2). In this section, we focus on the feature selection techniques.

We consider selecting d features from p features in the original feature set. The optimal way of doing it is to brute-force search for all possible combinations of features, evaluate their accuracy over the input dataset, and pick the feature combination that gives the highest performance. However, such a method is usually too computationally expensive. When the feature set and the dataset become large in practice, it's almost impossible to implement such a searching method in a reasonable time. Therefore, we consider the sub-optimal techniques for feature selection.

Instead of the actual model performance (i.e., accuracy), several metrics are introduced to quantitatively measure if a feature is good or not:

- **Within-class scatter matrix**

$$S_W = \sum_{k=1}^{K} \frac{N_k}{N} \Sigma_k, \tag{3}$$

  where K is the total number of classes. N_k is the number of data samples of the k^{th} class. N is the total number of data samples. Σ_k is the covariance matrix[2] of the data samples of the k^{th} class using a given feature set.

- **Between-class scatter matrix**

$$S_B = \sum_{k=1}^{K} \frac{N_k}{N} (\mu_{\mathbf{k}} - \mu)(\mu_{\mathbf{k}} - \mu)^T, \tag{4}$$

  where $\mu_{\mathbf{k}}$ is the mean of per-class samples. μ is the mean of all data samples.

Obviously, given the feature set, trace(S_W) measures the average, overall classes, the variance of the features. trace(S_B) measures the average (overall classes) distance of the

[1] Wikipedia: Curse of dimensionality.
[2] Wikipedia: Covariance matrix.

mean of each class from the respective global value. A good feature set should achieve a small variance within per-class (i.e., small trace(S_W)) and a large distance between classes (i.e., large trace(S_B)). Figure 2 further gives a visualization of S_W and S_B. A common criterion for selecting features is to compute

$$J = \frac{trace(S_B)}{trace(S_W)}.$$

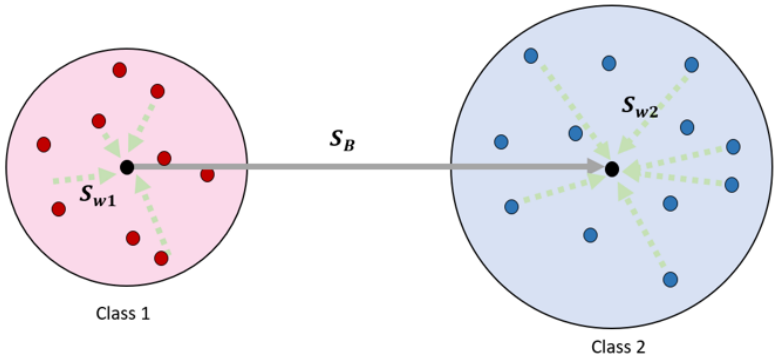

Figure 2: Visualization of scatter matrices.

Using the scatter matrix criterion, we aim to search for the optimal d features from p original features. Instead of brute-force search, several sub-optimal searching algorithms are proposed and used in practice:

1) Compute the criterion value for each of the features. Select the d optimal features with the top values.

2) *Forward search.* Starting from the empty feature set, add one feature each time to the current set which leads to the optimal value according to your criterion, until the d features have been chosen in the set.

3) *Backward search.* Starting from the original feature set, remove one feature each time such that the remaining features have the optimal value of a given criterion, until the $(p - d)$ features have been removed.

3.2 Feature reduction (optional)

Feature reduction, or dimensionality reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. One of most popular techniques to reduce feature dimensionality is Principle Component Analysis (PCA).

Key idea of PCA is to linearly transform the raw data onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified. It is shown that the principal components are eigenvectors of the data's covariance matrix, therefore are often computed by eigendecomposition of the data covariance matrix. Figure 3 illustrates the eigenvectors of the covariance matrix, with their magnitudes scaled by the corresponding eigenvalues.

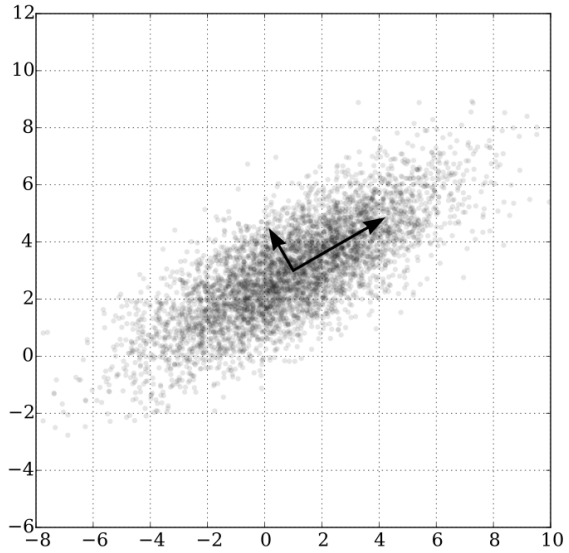

Figure 3: Visualization of PCA on a 2D Gaussian distributed dataset. Image sourced from online.

PCA is commonly performed under the following steps:

1) Normalize the input data such that the range of the continuous initial features does not affect the analysis.

2) Compute the covariance matrix and perform eigen-decomposition. Organize the eigenvectors in a descending order with respect to their corresponding eigenvalues. These eigenvectors are *Principal Components* since they describe directions of the axes where there is the most variance.

3) If we want to reduce the dimension to p dimension, we keep only p largest eigenvectors (components). These components form the columns of the resulting transformation matrix.

4) Recast the data along the principal components axes by multiplying the original data matrix with the transformation matrix. $FinalData = OriginalData * TransformationMatrx$

4 Classification evaluation

For classification, it is key to be able to correctly evaluate the model being produced to guarantee that the predictions are accurately describing the intended phenomenon (e.g., disease prediction, urban object classification). There are so many different performance metrics (accuracy, precision, recall, etc.), which is often overwhelming to choose which one to use. Selecting the right metric for a specific model, however, is key to be able to measure the performance of the model objectively and in the right setting. In this section, we will explore the different metrics, to be able to apply the most appropriate metrics for different tasks.

4.1 Basic concepts

Before delving into the performance metrics themselves, it is important to make sure some concepts are clearly understood as they are consistently used across most performance metrics.

4.1.1 True Value vs. Predicted Value

When evaluating the performance of a classification model, two concepts are key, the *real outcome* (usually called y) and the *predicted outcome* (usually called $\hat{y}$). For instance, a model can be trained to predict whether a person will develop a particular disease. In this case, it is trained with samples, e.g. a person's data, containing predictive information, such as age, gender, etc., and each person is labelled with a flag stating whether the disease will develop or not. In this case, the label can be whether the disease will happen ($y = 1$) or will not happen ($y = 0$). A machine learning model aims at making sure that every time a sample is presented to it, the predicted outcome corresponds to the true outcome. The more the model's predictions are the same as the true values the higher is the performance of the model. There are many different ways of evaluating a model's performance, but in general, models make mistakes, lowering performance.

True Positive, False Positive, True Negative, and False Negative. Each prediction from the model can be one of four types with regards to performance:

- **True Positive** (TP): A sample is predicted to be positive ($\hat{y} = 1$, i.e., the person is predicted to develop the disease) and its label is actually positive ($y = 1$, i.e., the person will actually develop the disease). In other words, a true positive occurs when the classifier correctly predicts a positive instance as positive.

- **False Positive** (FP): A sample is predicted to be positive ($\hat{y} = 1$, i.e., the person is predicted to develop the disease) and its label is actually negative ($y = 0$, i.e., the person will actually not develop the disease). In this case, the sample is "falsely" predicted as positive. In other words, a false positive occurs when the classifier incorrectly predicts a negative instance as positive.

- **True Negative** (TN): A sample is predicted to be negative ($\hat{y} = 0$, i.e., the person is predicted to not develop the disease) and its label is actually negative ($y = 0$, i.e., the person will actually not develop the disease). In other words, a true negative occurs when the classifier correctly predicts a negative instance as negative.

- **False Negative** (FN): A sample is predicted to be negative ($\hat{y} = 0$, i.e., the person is predicted to not develop the disease) and its label is actually positive ($y = 1$, i.e., the person will actually develop the disease). In this case, the sample is "falsely" predicted as negative. In other words, a false negative occurs when the classifier incorrectly predicts a positive instance as negative.

Note: Even though the classes are usually labelled 1 and 0, these values are arbitrary and they can often be found labelled as 1 and -1, which is the reason "Positive" and "Negative" are used. Remembering False Positive and False Negative meanings is usually relatively tricky and it is common for data scientists to have to stop for a second to think about the meaning of each to remember which one represents what. An easy trick to remember the difference is focus first on the second part of the name ("Positive" or "Negative"). This relates to the prediction, basically saying "The sample is predicted to

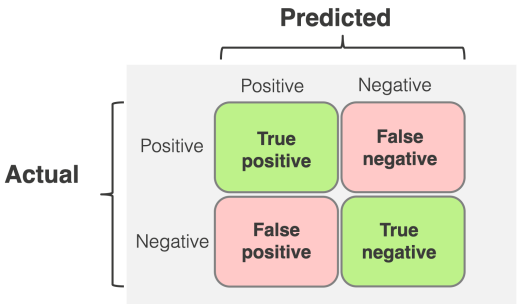

Figure 4: The structure of confusion matrix (for binary classification)

be Positive/Negative (belong to class 1/0)...". Then, we can look at the first part of the name to understand whether the prediction was correct or not ("True" or "False"). In this case, we are adding whether the prediction was correct or incorrect, and therefore if the sample was actually belonging to that class. For example, False Positive means that the sample is predicted to be Positive, but this is False/incorrect becasue the sample is actually Negative.

4.1.2 Confusion matrix

True Positive, False Positive, True Negative, and False Negative are usually presented in a tabular format in the so-called confusion matrix, which is simply a table organizing the four values. A confusion matrix has a specific table layout that allows the visualization of the performance of a classifier. Figure 4 shows such how the values are organized in a confusion matrix, and Figure 5 shows an example.

Confusion matrix can also be used in multi-class classification problems. In this case, the matrix will have more than two rows and columns. Their number depends on the number of labels the model is tasked to predict. The construction of a confusion matrix in multi-class classification follows the same logic as in binary classification. Figure 6 shows an example of confusion matrix for a 3-class classification problem.

The main limitation of using the confusion matrix in production model evaluation is that you must get the true labels on every model prediction. This might be possible, for example, when subject matter experts (e.g., payment disputes team) review the model predictions after some time. However, often you only get feedback on some of the predictions or receive only partial labels. Depending on your exact machine learning product, it might be more convenient to dynamically monitor specific metrics, such as precision. For example, in cases like payment fraud detection, you are more likely to send suspicious transactions for manual review and receive the true label quickly. This way, you can get the data for some of the confusion matrix's components faster than others.

4.2 Evaluation metrics

To evaluate the performance of a classifier over the test set, the following metrics are commonly used: confusion matrix, overall accuracy, mean per-class accuracy, precision, recall (sensitivity), F1_score, ROC curve and AUC.

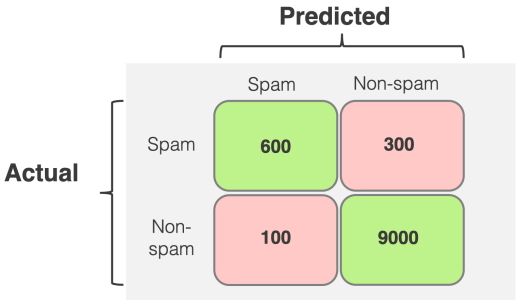

Figure 5: An example of confusion matrix. In this example, we have an email spam classification model, and it is a binary classification problem. The two possible classes are "spam" and "not spam." After training the model, we generated predictions for 10000 emails in the validation dataset. We already know the actual labels and can evaluate the quality of the model predictions. We can see that the number of true positives is 600 (top left, in green), which means 600 emails were predicted as spam, and they are indeed spam. Similarly, the number of true negatives is 9000 (bottom right, in green), which means 9000 emails were predicted as non-spam, and they are indeed non-spam; the number of false positives is 100 (bottom left, in pink), which means 100 emails were predicted as spam, but they are actually not; the number of false negatives is 300 (top right, in pink), which means 300 emails were predicted as non-spam, but they are actually spam.

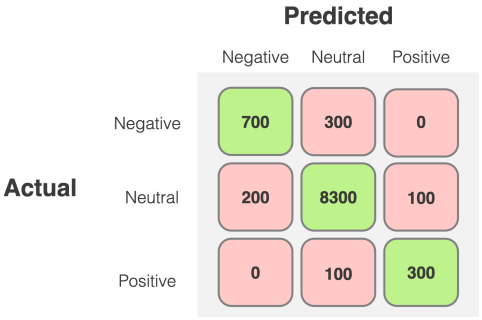

Figure 6: An example of confusion matrix in multi-class classification. In this example, we are classifying reviews that users leave on the website into three groups: "negative", "positive", and "neutral". So it is a 3-class classification problem. In this confusion matrix, each row represents the actual review label, while each column represents the predicted review label. What's convenient, the diagonal cells show correctly classified samples, so we can quickly grasp them together. The off-diagonal cells show model errors. In the top row, the model correctly labeled 700 (out of all 1000) negative reviews. In the second row, the model correctly labeled 8300 (out of 8600) neutral reviews but misclassified 200 as negative and 100 as positive. In the third row, the model correctly predicted 300 (out of 400) positive reviews but misclassified 100 as neutral.

4.2.1 Accuracy

Accuracy is the fraction of predictions our model got right out of all the predictions. This means that we sum the number of predictions correctly predicted as Positive (TP) or correctly predicted as Negative (TN) and divide it by all types of predictions, both correct (TP, TN) and incorrect (FP, FN).

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{5}$$

For multi-class classification, the **Overall Accuracy** measures the proportion of correctly classified instances, i.e.,

$$OA = \frac{1}{N}\sum_{i=1}^{C} n_i, \tag{6}$$

where C is the total number of classes. n_i is the number of objects correctly classified in class i. N is the number of objects in total. OA measures the overall performance of the classifier. In addition to Overall Accuracy, the performance of a classifier can also be measured by **Mean per-class Accuracy** that measures the average of per-category classification accuracy, i.e.,

$$mA = \frac{1}{C}\sum_{i=1}^{C} \frac{n_i}{N_i}, \tag{7}$$

where n_i is the number of objects correctly classified in class i, and N_i is the total number of objects in the i-th class. This metric can be used to **relieve the class imbalance issue** by averaging the accuracy of all classes.

Accuracy ranges between 0 and 1. These extreme cases correspond to completely missing the predictions or having always correct predictions. For instance, if our model is able to perfectly predict, the model will have no False Positives or False Negatives, making the numerator be equal to the denominator, bringing the accuracy to 1. Conversely, if our system is always off, incorrectly predicting each time, the number of True Positives and True Negatives will be zero, making the equation be zero divided by something positive, leading to an accuracy equal to 0.

Accuracy, however, is not a great metric, especially when the data is imbalanced. When there is a significant disparity between the number of positive and negative labels, Accuracy does not tell the full story. For instance, let's consider an example where we have 100 samples, 95 of which labelled as belonging to class 0, and 5 labelled as class 1. In this case, a poorly built "dummy" model which always predicts class 0, achieves a 95% accuracy, which indicates a very strong model. However, this model is not really predictive and accuracy is not the right performance metric to evaluate the power of this model. If we used only accuracy to evaluate this model, we would end up providing a model that is not performant or predictive.

4.2.2 Precision and Recall

To overcome the limitations of the Accuracy metric, we usually use Precision and Recall.

Precision tells what *proportion of positive predictions* was actually correct. It achieves this by counting the samples correctly predicted as positive (TP) and dividing it by the total positive predictions, correct or incorrect (TP, FP), i.e.,

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{8}$$

Recall (also called Sensitivity, True Positive Rate, or Hit Rate) aims at measuring what *proportion of actual positives* was identified correctly. It does so by dividing the correctly predicted positive samples (TP) by the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (TP, FN), i.e.,

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{9}$$

The generalization to multi-class problems is to sum over rows/columns of the confusion matrix. Given that the matrix is oriented as above (see Figure 6), i.e., that a given row of the matrix corresponds to specific value for the "truth", we have

$$\begin{aligned} \text{Precision}_i &= \frac{M_{ii}}{\sum_j M_{\boldsymbol{j}i}} \\ \text{Recall}_i &= \frac{M_{ii}}{\sum_j M_{i\boldsymbol{j}}} \end{aligned} . \tag{10}$$

That is, precision is the fraction of events where we correctly declared category i out of all instances where the algorithm declared category i. Similarly, recall is the fraction of events where we correctly declared category i out of all of the cases where the true of state of the world is category i.

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Considering the aforementioned example that showed the shortcomings of Accuracy, we get: Accuracy = 0.95 and Recall = 0. By using both Precision and Recall, we can better understand that a model predicting the majority class all the time is actually a low-performance model (Recall = 0) even though Accuracy is high (Accuracy = 0.95).

When is calculating precision and recall by class a good idea? When you have specific classes of interest, calculating the metrics by category is useful when you want to evaluate the performance of a particular class (or classes) and to know how well the classifier can distinguish this class from the others. It can also be helpful when you deal with imbalanced classes. Calculating precision and recall for the minority class (or classes) will clearly expose any issues. When you have a small number of classes, you can judge each metric individually and grasp them together, which is often the simplest solution. However, the downside of using precision and recall for each class is that it can result in a large number of performance metrics. The more classes, the more metrics you get. They can be challenging to interpret and grasp simultaneously. When you have a large number of classes, you might prefer to use macro or micro averages to provide a more concise summary of the performance. Specifically, the micro-average compute the precision and recall globally by counting the total true positives, false positives, and false negatives across all classes, while the macro-average compute the precision and recall for each class individually and then average them.

Micro-average:

$$\begin{aligned} \text{Micro-avg Precision} &= \frac{\sum_{i=1}^{n} \text{TP}_i}{\sum_{i=1}^{n} \text{TP}_i + \sum_{i=1}^{n} \text{FP}_i} \\ \text{Micro-avg Recall} &= \frac{\sum_{i=1}^{n} \text{TP}_i}{\sum_{i=1}^{n} \text{TP}_i + \sum_{i=1}^{n} \text{FN}_i} \end{aligned} \tag{11}$$

Macro-average:

$$\begin{aligned} \text{Macro-avg Precision} &= \frac{1}{n} \sum_{i=1}^{n} \text{Precision}_i \\ \text{Macro-avg Recall} &= \frac{1}{n} \sum_{i=1}^{n} \text{Recall}_i \end{aligned} \tag{12}$$

4.2.3 F1 Score

The F1 score is another performance metric indicating the harmonic mean of Precision and Recall. The highest value of an F1 Score is 1, indicating perfect Precision and Recall, and the lowest possible value is 0 if either the Precision or the Recall is zero.

$$\text{F1 Score} = \frac{2 \times TP}{(2 \times TP + FP + FN)} \tag{13}$$

In the case of multi-class classification, the F1 Score can be calculated using different approaches, including micro-average, macro-average, and weighted-average.

- **Micro-average F1 Score**. Micro-average F1 score is calculated by considering the total number of true positives, false positives, and false negatives across all classes.

$$\text{Micro-avg F1 Score } = \frac{2 \times \text{ Micro-avg Precision } \times \text{ Micro-avg Recall}}{\text{Micro-avg Precision } + \text{ Micro-avg Recall}} \tag{14}$$

- **Macro-average F1 Score**. Macro-average F1 score is calculated by computing the F1 score for each class individually and then averaging them, i.e.,

$$\text{Macro-avg F1 Score} = \frac{1}{n} \sum_{i=1}^{n} \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}, \tag{15}$$

  where n is the total number of classes.

- **Weighted-average F1 Score**. Weighted-average F1 score calculates the F1 score for each class individually and then averages them with weights proportional to the number of true instances for each class, i.e.,

$$\text{Weighted-avg F1 Score } = \frac{\sum_{i=1}^{n} (\text{F1 Score}_i \times \text{Weight}_i)}{\sum_{i=1}^{n} \text{Weight}_i}, \tag{16}$$

  where $\text{F1 Score}_i = \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$. Weight_i is the weight assigned to class i, usually proportional to the number of true instances for that class.

4.2.4 AUC (Area Under the ROC Curve)

As we've seen, one of the issues of Accuracy is that it can lead to overly inflated performance if the distribution of the classes is not very well balanced. AUC, which stands for "Area Under the ROC Curve" (see Section 4.3.1), measures the entire two-dimensional area underneath the entire ROC curve. It is an aggregate measure of performance across all possible classification thresholds[3]. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0; one whose predictions are 100% correct has an AUC of 1. Another way of interpreting AUC is as the probability that the model

[3]Let's pick an example of fraudulent transaction identification to understand how a threshold (or cutoff) works. The fraud detection model outputs the probability of a transaction being fraudulent, which intrinsically gives away the probability of a regular transaction. A transaction is said to be predicted as fraudulent if the output probability is greater than the chosen threshold, else it is declared as a regular transaction. When a threshold which is as stringent as 0.9 is applied, more transactions are marked as regular, whereas fewer transactions are marked as fraudulent. Thus, whether a transaction is predicted as fraudulent or regular depends largely on the threshold value.

ranks a random positive sample higher than a random negative sample. AUC is a great metric when dealing with imbalanced classes, and is one of the most frequently used performance measures in classification.

Some of the properties of the AUC metric, which may make AUC desirable for some cases and may limit the usefulness of AUC in certain use cases:

- Scale-invariance. AUC measures how well predictions are ranked, instead of their absolute values. However, scale invariance may not be always desirable. For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- Classification-threshold invariance. AUC measures the quality of the model's predictions regardless of what classification threshold is chosen. Similarly, classification-threshold invariance is not always desirable. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

4.3 Performance charts (optional)

Additionally, performance measures can be not only communicated as single numbers but also as charts. Some common charts showing a machine learning model's performance are the ROC Curve and the Precision/Recall Curve.

4.3.1 ROC Curve

An ROC curve (Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model at different classification thresholds (see Figure 7). The chart's y-axis is the True Positive Rate (TPR), while the x-axis is the False Positive Rate (FPR) and the plot consists of the TPR and FPR values varying the threshold. Lowering the classification threshold, the model classifies more items as positive, increasing both False Positives and True Positives. The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random); points below the line represent bad results (worse than random).

The best-case scenario consists of an angled line, going vertically first and horizontally after. The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a perfect classification.

A worst-case scenario, i.e., random guess, would give a point along a diagonal line (the so-called line of no-discrimination) from the bottom left to the top right corners (regardless of the positive and negative base rates). An intuitive example of random guessing is a decision by flipping coins. As the size of the sample increases, a random classifier's ROC point tends towards the diagonal line. In the case of a balanced coin, it will tend to the point (0.5, 0.5).

In a multi-class model, we can plot the N number of ROC curves for N number classes using the One vs ALL strategy. So for example, if you have three classes named X, Y, and Z, you will have one ROC for X classified against Y and Z, another ROC for Y classified against X and Z, and the third one of Z classified against Y and X.

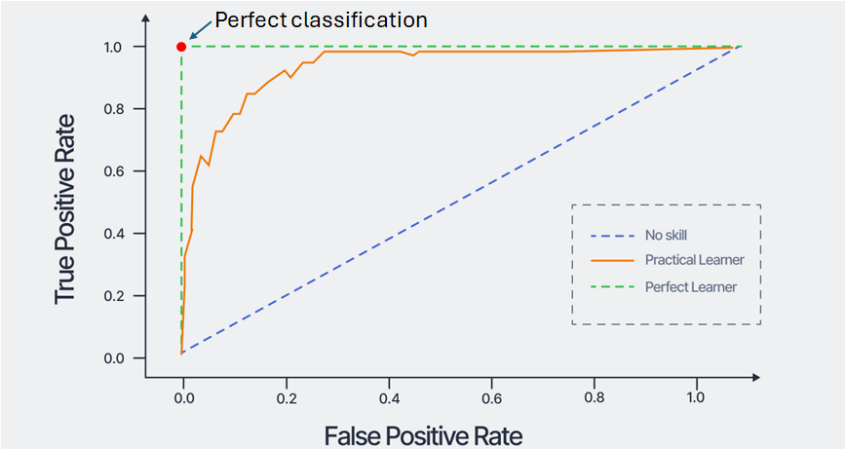

Figure 7: An example of an ROC curve for a binary classification problem.

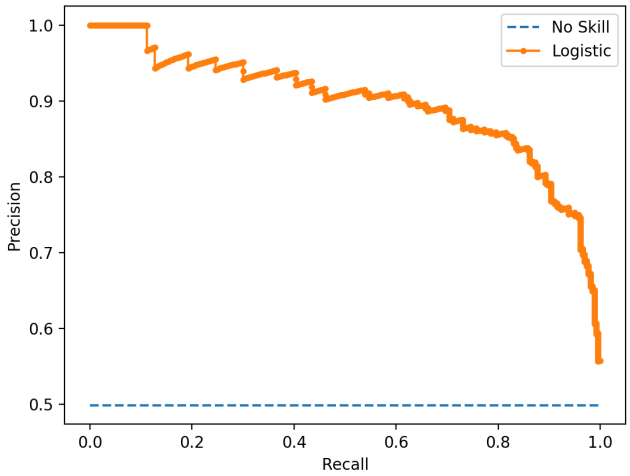

Figure 8: Precision/Recall curve

4.3.2 Precision/Recall Curve

Precision and recall are key evaluation metrics to measure the performance of machine learning classification models. However, the trade-off between the two depends on the business prerogative and is best resolved through the PR curve.

A Precision/Recall curve plots performance over a y-axis showing Precision and an x-axis which is Recall (see Figure 8). Each point is evaluated at different threshold values. To construct a precision-recall curve, the classifier's predictions are sorted based on their confidence scores or probability estimates. Then, different thresholds are applied to these scores to classify instances as positive or negative. For each threshold, precision and recall are calculated, and a point is plotted on the precision-recall plane. By varying the threshold, different points are plotted, resulting in a curve that shows how precision and recall change as the threshold changes.

Ideally, a classifier should achieve high precision and high recall simultaneously, leading to a curve that hugs the upper-right corner of the plot. In practice, the precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High

scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

The precision-recall curves are particularly useful when dealing with imbalanced datasets, as they provide insights into the classifier's ability to correctly identify positive instances while minimizing false positives. In practice, the precision-recall curve can help practitioners choose an appropriate threshold based on their specific requirements. For instance, if high precision is desired, a threshold can be chosen that maximizes precision even if it sacrifices some recall.

In multi-class classification, the precision-recall curve is for each class at different classification thresholds. The precision and recall values are calculated for each class individually.

When to Use ROC vs. Precision-Recall Curves? Generally, ROC curves should be used when there are roughly equal numbers of observations for each class, while precision-recall curves should be used when there is a moderate to large class imbalance[4]. The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance (because of the use of true negatives in the False Positive Rate in the ROC Curve and the careful avoidance of this rate in the Precision-Recall curve).

4.4 Impact of choosing the right performance metric

Choosing the right metric is key, especially in cases where False Positives and False Negatives do not have the same impact. Ideally, we would want to have a perfect prediction both in terms of False Positive and False Negative (both zero), but with machine learning models there is usually a tradeoff between detecting False Positives or False Negatives well. For instance, if our model predicts whether a person has got a deadly disease, like cancer, it could be said that False Positives are more important. We want to make sure that if that person has the disease, we correctly flag them. We are less concerned if we accidentally misclassify a person as having the disease even though they didn't have it. Conversely, if our model predicts whether a person is innocent or not, it might be argued that False Negatives are more important. We want to make sure that no innocent person is incorrectly jailed.

5 Cross validation

We have previouly introduced some commonly used metrics for measuring the performance of a classifier on a given dataset (e.g., overall accuracy, mean per-class accuracy, confusion matrix). However, it is not a good idea to measure the performance of a classifier on the training set, as it will generally yield an optimistic estimate of the classifier. Due to the problem of over-fitting (i.e., the classifier performs well on the training set while it gives bad performance when extended to unseen data samples), the performance on the training set is not a good indicator of predictive performance on unseen data.

If data is plentiful, then one approach is simply to use some of the available data (i.e., training set) to train a range of models or a given model with a range of values for its complexity parameters, and then to compare them on independent data (i.e., test set), and select the one having the best predictive performance. In some scenarios, if the model

[4]The Relationship Between Precision-Recall and ROC Curves. ICML 2006

design is too complex and involves too many hyper-parameters, it becomes necessary to keep aside a third validation set on which the effects of various hyper-parameters are evaluated. Then, the performance of the selected model is finally evaluated over the independent test set.

Given a certain amount of data available, how to determine the ratio between the training set and test set?

- Using the same data for training and testing may give a good classifier but will yield an optimistically biased performance estimate.
- Using a large portion of the data set for training will lead to a well-trained classifier. However, correspondingly, a small test set is likely to give an independent but unreliable (i.e., with large variance) estimate of the model performance.
- Using a large portion of the data for testing instead will give a reliable, unbiased estimate of a badly-trained classifier, due to the small amount of training set.
- In practice, the ratio of the train-test split is usually set to 7:3, 6:4, or 5:5, to achieve a relative balance between these two sets.

In many applications, the supply of data for training and testing will be limited, and to build good models, we wish to use as much of the available data as possible for training. However, we also don't want the test set to be small, as it will give a relatively noisy estimate of predictive performance. One solution to this dilemma is to use *cross-validation*.

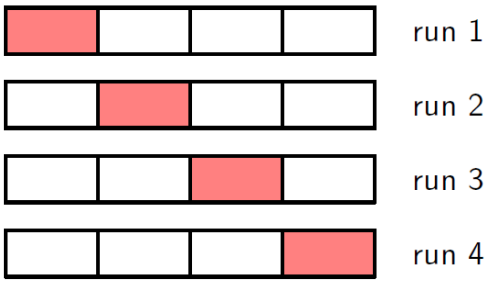

Figure 9: An illustration of K-fold cross validation. In this example $K = 4$. The red block is used for testing while the rest 3 blocks are used for training in each run. The performance score is then averaged over the 4 runs in total.

The technique of K-fold cross-validation, illustrated in Figure 9 for the case of $K = 4$, involves taking the available data and partitioning it into K groups. Then $(K-1)$ of the groups are used to train a classifier, which is then evaluated on the remaining group. This means a proportion $(K-1)/K$ of the available data is used for training while making use of all of the rest for testing. The procedure is repeated for all K possible choices for the held-out group, indicated here by the red blocks, and the performance scores from the K runs are then averaged. Sufficient data is crucial for most existing machine learning classifiers, therefore the goal of cross-validation is to allow all the data samples to be seen and trained, while still yielding a reasonable evaluation of the classifier.

6 Learning curve and feature curve

Learning curve shows the performance of a classifier on the training and testing set for varying numbers of training samples. It is a tool to analyze how much we benefit from adding more training data and how much the classifier suffers from overfitting. The plot's x-axis shows the number of training samples and the y-axis shows the classification error rate (see Figure 10). **Note**: the y-axis can also use other performance metrics (e.g., overall accuracy, precision, recall, F1-score, depending on the problem). So it is also common to see that the y-axis shows the classification accuracy (so the curves are flipped in the vertical direction).

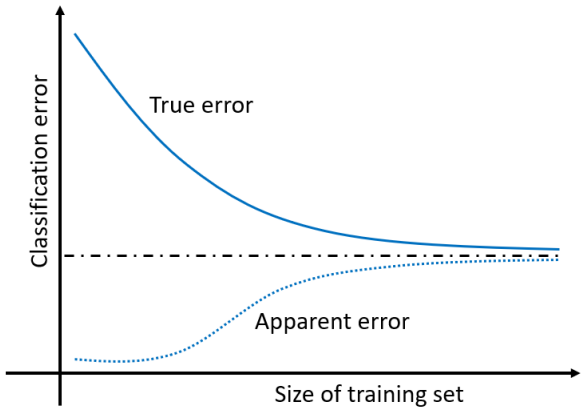

Figure 10: Learning curve

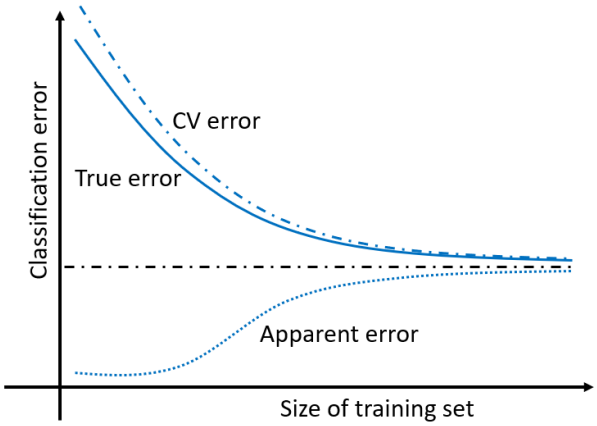

Figure 11: Cross validation error curve in comparison with the true error curve

The apparent error, also called training error, is the error rate obtained on the training set. The true error is the actual performance of the classifier, which cannot be obtained but can be approximated by evaluating on the testing set[5]. Obviously, the apparent error is always smaller than the true error, since evaluating the classifier's performance on its training set will give a way too optimistic estimate of the error. At certain range of

[5]**Note**: If you use the learning curve function provided in Scikit-Learn, you will not obtain the true error curve but the validation error curve, which is estimated by K-folds cross validation. By default, K is set to 5. As shown in Figure 11, the true error curve serves as a lower bound of the cross validation error curve. The reason is that to estimate the cross validation error, only $\frac{K-1}{K}$ of the training samples are used to train a classifier, resulting in a less sufficiently trained classifier with worse performance. The smaller the K, the larger the cross validation error.

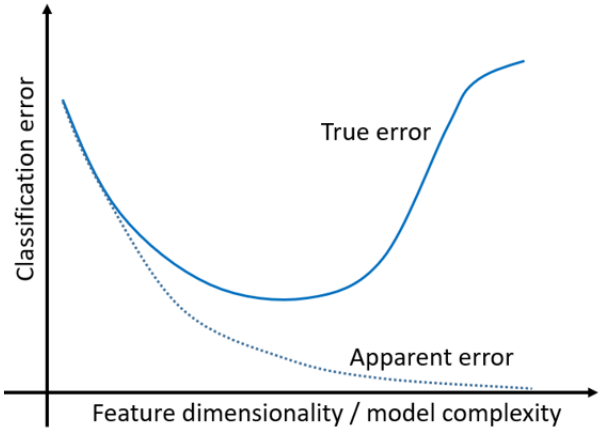

Figure 12: Feature curve

training set size you may observe a noticeable gap between the two curves, indicating the risk of model overfitting. The two curves will eventually converge as the training set size goes to infinity. The convergence value is the Bayes error, the minimum attainable error of this classification task.

Feature curve shows the performance of varying classifiers with increasing model complexity or feature dimensionality under a fixed amount of training samples. It is a tool to analyze how we should trade off to find the classifier with the most proper level of complexity. The plot's x-axis shows the model complexity or feature dimensionality, and the y-axis shows the classification error rate (see Figure 12). It indicates that given a limited amount of data, a too complex classifier or a too enriched feature combination will result in severe overfitting (i.e., the noticeable gap between the apparent error and the true error). This phenomenon is often refered to as *curse of dimensionality*[6].

[6]Wikipedia: Curse of dimensionality.