Algorithms of Information Security: Machine learning-based malware detection techniques II

Faculty of Information Technology
Czech Technical University in Prague

December 1, 2021

Naive Bayes

- Naive Bayes classifier for the binary (i.e. 2 classes) classification problem works as follows.
- Naive Bayes classifier is a probability algorithm based on Bayes' theorem, which predicts the class with the highest a posteriori probability.
- Bayes' theorem states that:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \quad (P(B) \neq 0).$$

- Let's have a set of two classes $\{\mathcal{C}, \mathcal{M}\}$, where $\mathcal{C}$ denotes the class of benign samples and $\mathcal{M}$ denotes the malware class.

Naive Bayes

- Let's have a training dataset and a test element (with unknown label) that is represented by a feature vector $x = (x_1, \ldots, x_n)$.
- Let $P(\mathcal{M}|x)$ denote the probability that the sample is malware given the feature vector x, which represents the sample. Similarly, let $P(\mathcal{C}|x)$ denote the probability that the sample is benign given the feature vector x. The Naive Bayes classification rule is defined by

$$\begin{aligned} \text{If } P(\mathcal{M}|x) &< P(\mathcal{C}|x), \ x \text{ is classified as benign} \\ \text{If } P(\mathcal{M}|x) &> P(\mathcal{C}|x), \ x \text{ is classified as malware} \end{aligned} \quad (1)$$

Naive Bayes

- a posteriori probabilities $P(C|x)$ can be expressed as a priori probabilities and $P(x|C)$ probabilities using the Bayesian theorem:

$$P(C|x) = \frac{P(x|C)\ P(C)}{P(x)} \tag{2}$$

- Assuming that the values of the features are conditionally independent of each other, the equation (2) can be expressed as

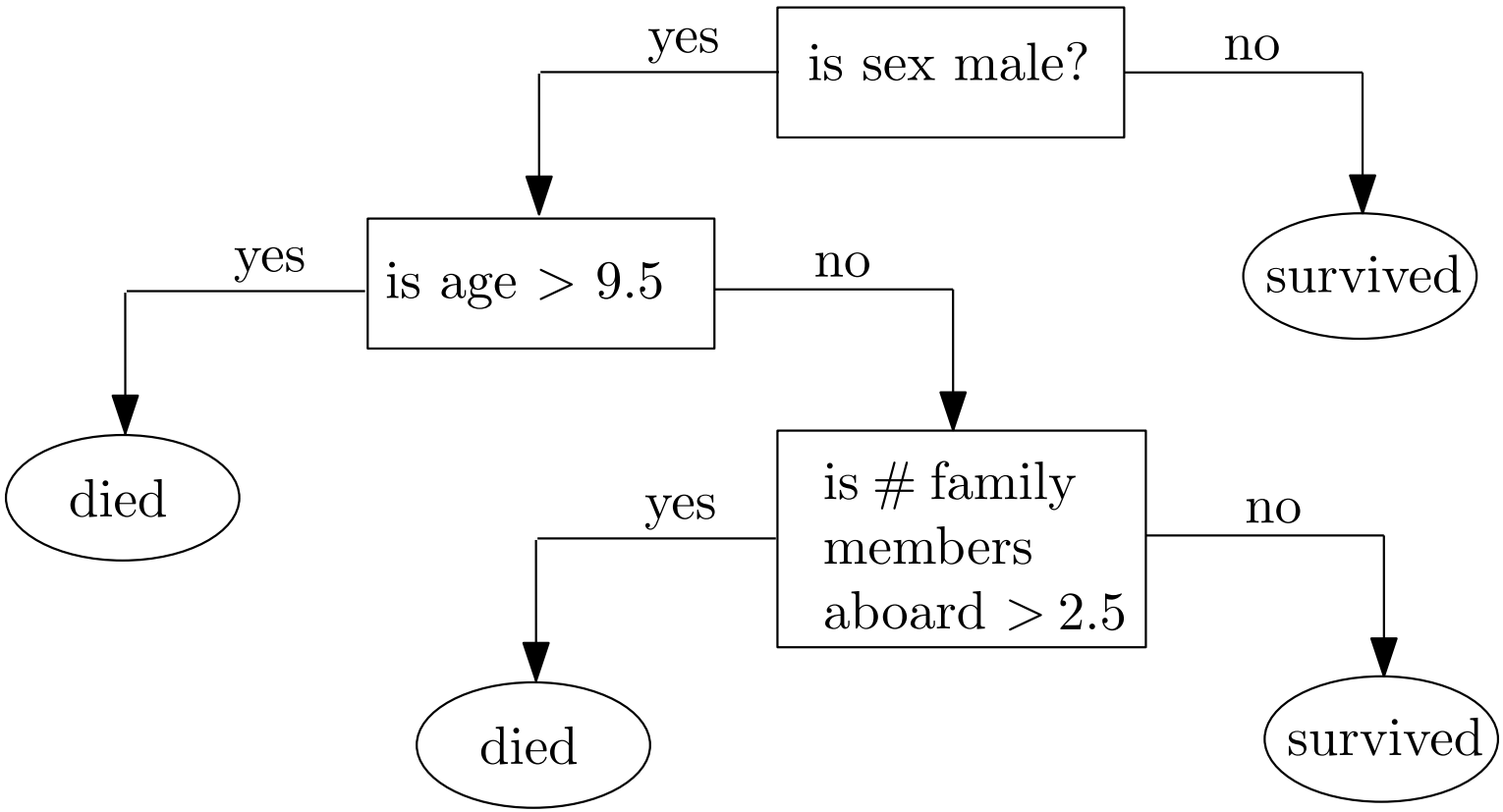
$$P(C|x) = \frac{\prod_{i=1}^{n} P(x_i|C)\ P(C)}{P(x)} \tag{3}$$

Naive Bayes

- The probabilities $P(x_i|C)$ can be estimated from the training set by calculating the feature values for each class.
- More specifically, the probability $P(x_i = h|C)$ is represented as the number of elements of class C in the training set with the value h for the feature x_i divided by the number of elements of class C in the training set.
- The output of the classifier is the class C' with the highest probability:

$$C' = \arg\max_{C} \left(P(C) \prod_{i=1}^{n} P(x_i|C) \right) \tag{4}$$

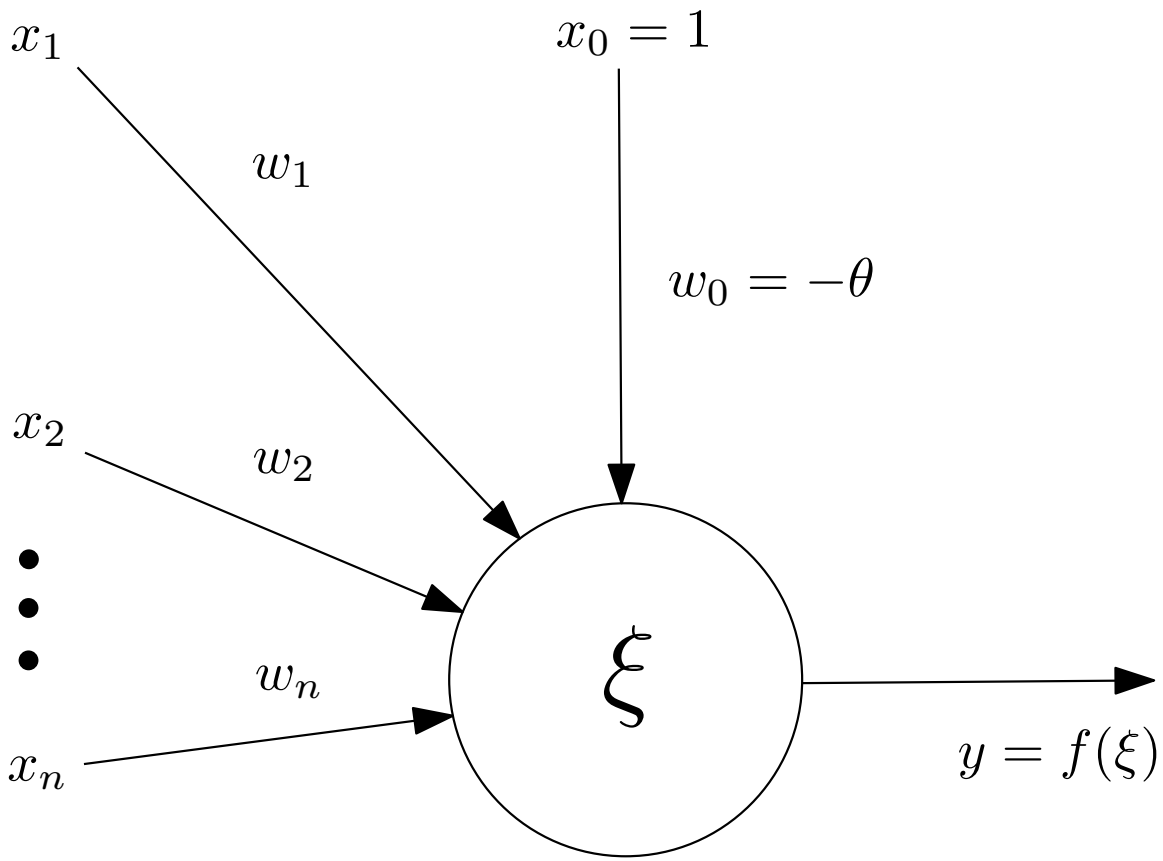
Decision Tree



Decision Tree - Basics

- goal: achieve perfect classification with minimal number of decisions
- each level splits the data according to different attributes
- 1. select attribute or value along a dimension that gives "the best" split
  2. create child nodes based on the split
  3. recurse on each child using child data until a stopping criterion is reached
- finding optimal tree for arbitrary data is NP-hard

Single Neuron (Perceptron)

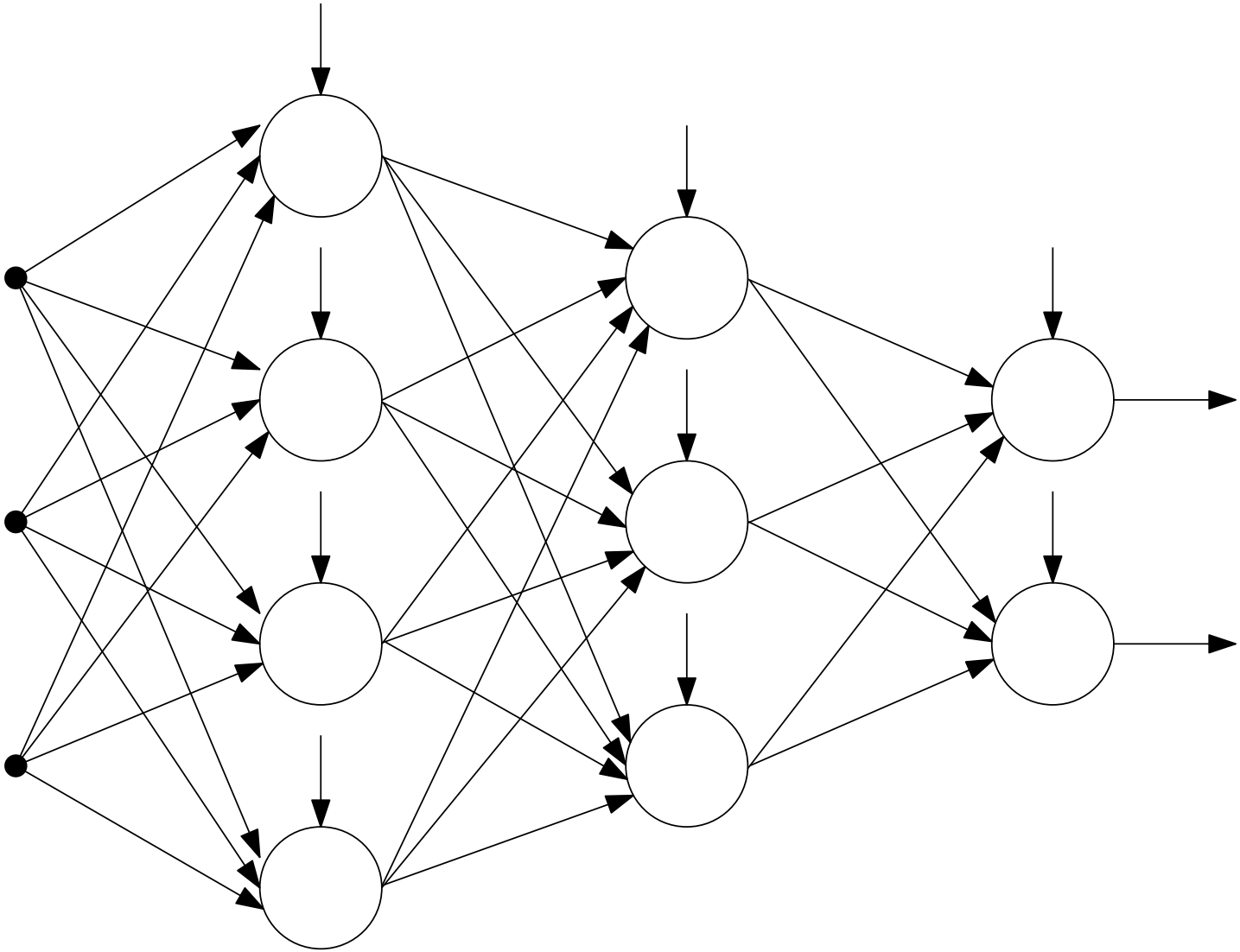


Single Neuron - Details

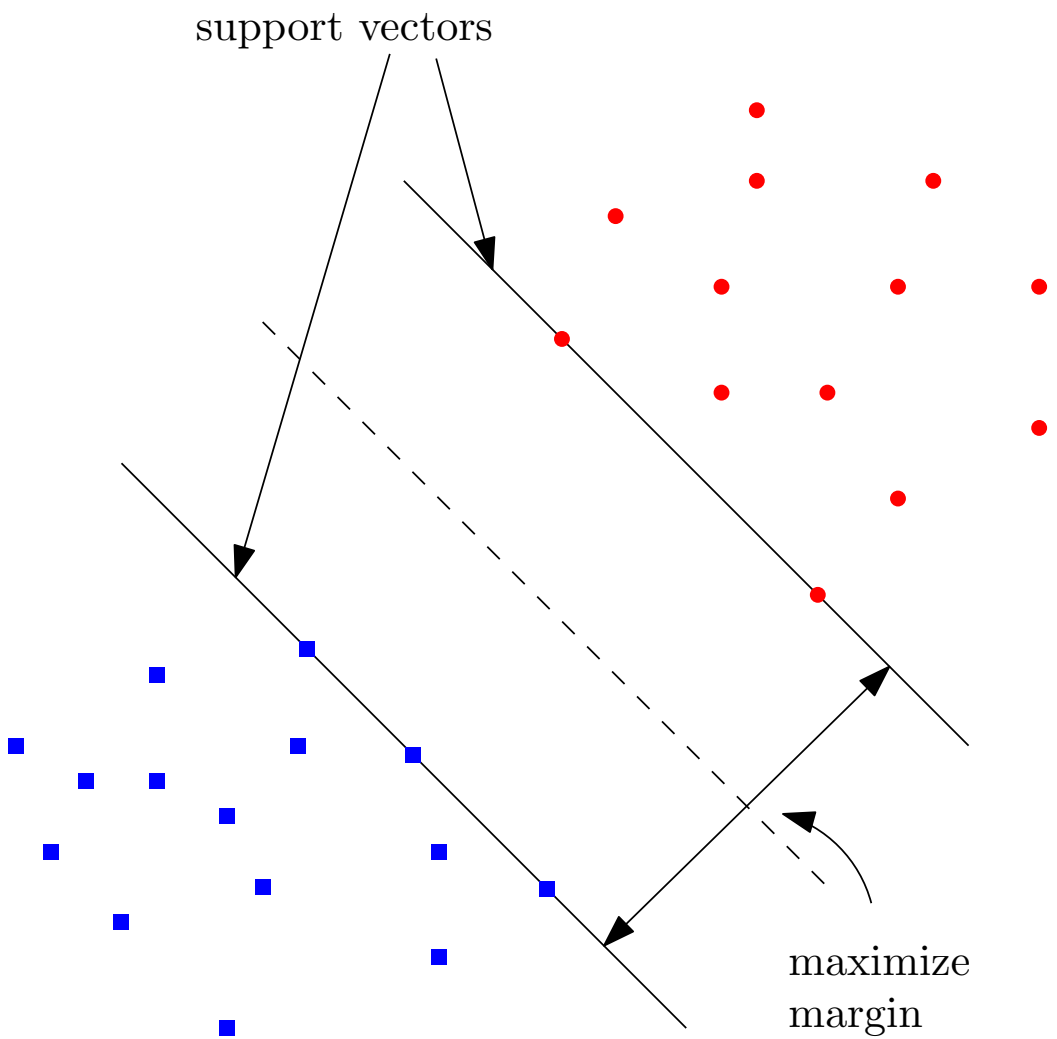
$$\xi = \sum_{i=1}^{n} w_i x_i - \theta = \sum_{i=0}^{n} w_i x_i \qquad \text{action potential}$$

$$f(\xi) = \frac{1}{1 + e^{-\lambda \xi}} \qquad \text{activation function}$$

Neural Network



Support Vector Machines



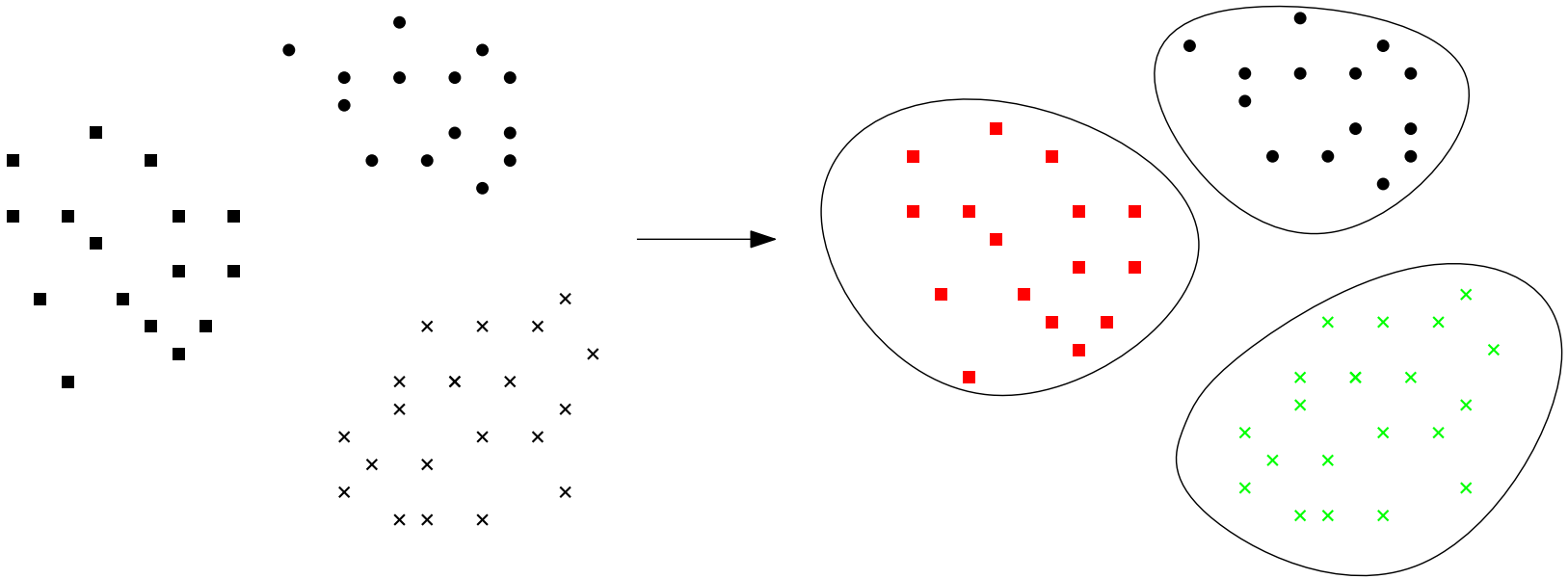
Support Vector Machines - Basics

- training set $\{\mathbf{x_i}, i = 1, \ldots, n\}$, with corresponding labels $y = \pm 1$.
- optimal hyperplane $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ for linearly separable patterns
- decision rule

$$y_i(\mathbf{w}^T\mathbf{x} + w_0) > 0 \text{ for all } i$$

- SVMs maximize the margin around the separating hyperplane

Clustering Algorithms



K-means pseudocode

1. initialize cluster centroids $\mu_1, \ldots, \mu_k \in \mathrm{R}^n$ randomly
2. repeat until convergence {

   for every i, set

$$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2$$

   for every j, set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

Clustering algorithm - Partitioning Around Medoids

Algorithm 1 PAM algorithm

Input: Number of clusters k, set of data points T
Output: k clusters

1: Initialize: randomly select k data points from T to become the medoids
2: Assign each data point to its closest medoid
3: **for all** cluster **do**
4: identify the observation that would yield the lowest average distance if it were to be re-assigned as the medoid
5: **if** the observation is not current medoid **then**
6: make this observation the new medoid
7: **end if**
8: **end for**
9: **if** at least one medoid has changed **then**
10: **go to** step 2
11: **else**
12: end the algorithm.
13: **end if**

Model Evaluation

- Model evaluation consists of two steps:
  1. Model selection
     - **Training** - estimate the parameters of the model.
     - **Validation** - select the model with the best results.
  2. **Testing** - evaluate the selected model in order to assess the real performance of the model on new unseen data
- We split data into training set and a testing set. The training set is used to estimate the parameters of the model (training) and also for model selection (validation).
- The testing set is then used to assess the performance of the model.

Evaluation metrics

Evaluation metrics are used to measure the performance of the classification models. In a binary classification problem, the following classical quantities are employed:

- **True Positive** (TP) represents the number of malicious samples classified as malware
- **True Negative** (TN) represents the number of benign samples classified as benign
- **False Positive** (FP) represents the number of benign samples classified as malware
- **False Negative** (FN) represents the number of malicious samples classified as benign

Confusion matrix:

		predicted	
		positive	negative
true	positive	TP	FN
	negative	FP	TN

Evaluation metrics

- The most intuitive and commonly used evaluation measure in Machine Learning is Accuracy (ACC):

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{5}$$

- It is defined on a given test set as the percentage of correctly classified instances.
- Alternative for accuracy is an error rate, it is defined as

$$\mathrm{ERR} = 1 - \mathrm{ACC}. \tag{6}$$

Evaluation metrics

- True Positive Rate (TPR) (or recall), is defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{7}$$

- TPR is the percentage of truly malicious samples that were classified as malware.
- False Positive Rate (FPR) is defined as follow:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{8}$$

- FPR is the percentage of benign samples that were wrongly classified as malware.

Evaluation metrics

- Another metric is precision, and it is defined as follows:

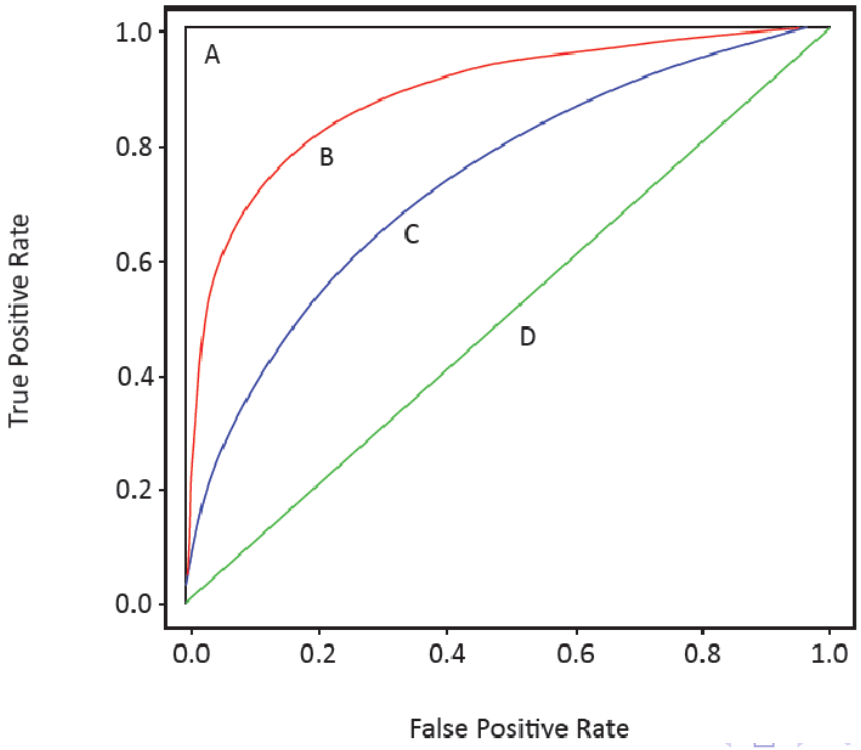
$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{9}$$

- Precision is the percentage of samples classified as malware that are truly malware.
- F_1-measure is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{10}$$

Evaluation metrics

- Another metrics are the ROC (Receiver Operating Characteristic) curve and the AUC (Area Under the ROC Curve)
- The ROC curve is used only for binary classifiers and plots the FPR against the TPR for all possible classification threshold values.



Problems - overfitting

- A training set is not a representative set of a population and it usually contains some noise elements.
- A model that captures structure in the data together with the noise is "over-fitted".
- Classifiers have then high performance on a training set, but low generalization ability on the testing set.

Problems - imbalanced data

- Imbalanced data usually reflects an uneven distribution of classes within a dataset (eg, there is significantly less malware than benign files).
- Resampling is used to solve this problem, which includes two methods: undersampling and oversampling.
- Undersampling is the process of randomly deleting some data from a majority class to achieve a size comparable to the data size of a minority class.
- Oversampling is a synthetic data generation process that attempts to randomly generate samples of data from a minority class.

Challenges (for AV vendors or attackers)

- **Minimizing of reaction time** - The majority of new malicious samples are generated by malware generators that need to input some parameters. Since attackers have all antivirus products at their disposal, a common strategy of attackers is to change the setting of malware generators and generate samples as long as none of the antiviruses detect them. New malware is then spread among users by some infection vector. The time period between spreading the malware and creating detection rule, referred to as reaction time, should be minimal since users are not protected during this time period. New malware may spread quickly and infect thousands of computers worldwide before any AV software makers can isolate and analyze it and issue an update, and certainly before most users can download, install, and run this update.

Challenges (for AV vendors or attackers)

- **Prediction of future variants of malware** - This problem is related to the previous problem of minimizing reaction time. The mentioned attackers' strategy can be applied regularly, for example, every second, which is, unfortunately, a shorter time than the reaction time of antiviruses. If it would be possible to predict future variants of malware, then the reaction time could be reduced by preparing detection rules ahead.

Challenges (for AV vendors or attackers)

- **Interpretability of the models** - Most of the models used at the present time are treated as a black box. This black box is given an input X and it produces an output Y through a sequence of operations hardly understandable to a human. This could pose a problem in cybersecurity applications when a false alarm occurs as analysts would like to understand why it happened. The interpretability of the model determines how easily the analysts can manage and assess the quality and correct the operation of a given model.

Challenges (for AV vendors or attackers)

- **Adversarial learning** - Machine learning models are vulnerable to adversarial attacks that can fool the models. For instance, an adversary can craft malware that has a similar feature vector to some benign file's feature vector. As a result, the training set may have different statistical distribution than the distribution of the testing set. The goal is to propose defense techniques in order for machine learning algorithms can resist such adversarial attacks.

Challenges (for AV vendors or attackers)

- **Anti-analysis techniques** - Malware developers want to avoid analyzing their samples, so they devise and refine several anti-analysis techniques that are effective in hindering the reverse engineering of executables. Various obfuscation techniques such as packing, encryption, etc., are effective for static analysis. Such a kind of anti-analysis techniques can be overcome by using dynamic analysis to make the sample unveil hidden information and load them in memory, where they can then be extracted by creating a dump. However, such a process is computationally and time consuming.

Challenges (for AV vendors or attackers)

- **Reducing huge training datasets** - Training datasets usually consist of millions of samples. With the growth of samples' amounts, classification algorithms have become more and more expensive. The problem is how to quickly reduce the huge number of samples in the training dataset without reducing the accuracy. *Instance selection algorithms* could significantly reduce the training dataset size, and even a slightly better detection rate can be achieved. However, they are not applicable for large-size data sets due to high computational complexity.