OPTIMIZING MACHINE LEARNING ALGORITHMS APPLIED TO THE CONTEXT OF INTELLIGENT VEHICLES

RAMAROLAHIVONJITIANA Frédéric Joël[1], RANDRIAMITANTSOA Paul Auguste[2], RANDRIAMITANTSOA Andry Auguste[3]

[1] *PhD student, TASI, ED-STII, Antananarivo, Madagascar*
[2] *Thesis director, TASI, ED-STII, Antananarivo, Madagascar*
[3] *Thesis co-director, TASI, ED-STII, Antananarivo, Madagascar*

ABSTRACT

Securing open networks, optimizing the performance of datacenters, and even getting more value from data are all part of the future of information technology. An important issue in its own right is the use and / or valuation of data produced and exchanged between these connected objects due to the number of nodes considered, the heterogeneity of the technologies for their connectivity, and the exponential evolution of needs expressed, given the volume of data available. Controlling and mastering connected and autonomous objects are becoming major societal and economic concerns. This has sparked intense activity in academia and industry in the area of machine learning research. Inside intelligent vehicle, machine learning algorithms are implemented in the detection, perception and decision subsystems to ensure localization, object recognition, trajectory planning, action prediction until obstacle avoidance. With the ultimate goal of optimizing these algorithms, we implemented cost functions, also called lost function, for a predictive model of a desired trajectory and motion planning. Moreover, several technics like dimensionality reduction, function approximation and model selection can be employed. We were able to determine the cost function of a cyberattack. Also, we used the dimensionality reduction technique for motion planning optimization to map the collision.

Keywords : *Machine learning, Cost function, Lost function, Dimensionality reduction, Intelligent vehicles*

1. INTRODUCTION

Connected objects intelligence experienced important progress due to combination of market demand and technological progress. Exploitation of available data is becoming a major concern for society and the economy. This concern has prompted academia and industry to engage in intense activity in the field of artificial intelligence research. Machine learning would therefore be in pole position for the intelligibility of Big data. In our daily lives, with the advent of connected objects, machine learning is the basis of recommendation and prediction systems. Machine learning algorithms are evaluated on their ability to correctly classify or predict to achieve specific goals. For systems optimization, the work will identify cost function also called loss function. In this paper, we will highlight loss minimization and the criteria for model selection with application in the context of intelligent vehicles. First, we will define different cost functions and demonstrate techniques such as dimensionality reduction, covariate shift and function approximation. Next, we will see the overall architecture of a connected and autonomous vehicle and the implementation of machine learning.

2. Optimizing machine learning

2.1 Cost function

The cost function, also known as the loss function, is the function used to determine the loss. The cost or loss of a prediction y ', when the correct value is y, is a measure of the relative utility of that prediction given that correct value. A common loss function used with classification learning is zero-one loss. Zero-one loss assigns 0 to loss for a correct classification and 1 for an incorrect classification. Cost-sensitive classification assigns different costs to different forms of misclassification. A common loss function used with regression is error squared. It is the square of the difference between the predicted and true values. [1]

For the measure of success, the error of a classifier is defined as the probability that it does not predict the correct label on a random data point generated by the underlying distribution. In other words, the error of h is the probability to draw a random instance x, according to the distribution *D*, such that h(x) does not equal *f(x)*. [2]

Formally, given a domain subset, $A \subset X$, the probability distribution *D* assigns a number $D(A)$, which determines how likely it is to observe a point $x \in A$. In many cases, we refer to *A* as an event and express it using a function $\pi: X \to \{0,1\}$, namely $A = \{x \in X : \pi(x) = 1\}$. In this case, we also use the notation $\mathbb{P}_{x \sim D}[\pi(x)]$ to express $D(A)$.

We define the error of a prediction rule, $h: X \to Y$, to be:

$$L_{D,f}(\mathrm{h}) \stackrel{\text{def}}{=} \mathbb{P}_{x \sim D}[h(x) \neq f(x)] \stackrel{\text{def}}{=} D(\{x: h(x) \neq f(x)\}) \tag{1}$$

2.2 Dimensionality reduction

Dimensionality reduction is an unsupervised learning type. The objective is also to reveal a particular data structure, which is different than groupings. For example, although data can be represented in a high dimensional space, it can be located around a lower dimensional subspace or a manifold. These methods are very important in machine learning for compressed representations or for computational reduction reasons. Learning the lower dimensional structure associated with a given set of data is gaining in importance in the context of big data processing and analysis.

2.2.1 Linear transformation

Let $x_1, \ldots, x_m$ *be* m vectors in $\mathbb{R}^d$.

We would like to reduce the dimensionality of these vectors using a linear transformation. A matrix $W \in \mathbb{R}^{n,d}$, *where* $\mathrm{n} < \mathrm{d}$, induces a mapping $\mathrm{x} \mapsto \mathrm{Wx}$ where $\mathrm{Wx} \in \mathbb{R}^n$ is the lower dimensionality representation of x. Then, a second matrix $U \in \mathbb{R}^{d,n}$ can be used to (approximately) recover each original vector x from its compressed version. That is, for a compressed vector y = Wx, where y is in the low dimensional space $\mathbb{R}^n$, we can construct $\tilde{\mathbf{x}} = \mathbf{Uy}$, so that $\tilde{\mathbf{x}}$ is the recovered version of x and resides in the original high dimensional space $\mathbb{R}^d$.

2.2.2 Principal component analysis (PCA)

In PCA, we find the compression matrix W and the recovery matrix U so that the total squared distance between the original and recovered vectors is minimal; namely, we aim to solve the problem: [2] [3]

$$\underset{W \in \mathbb{R}^{n,d}, U \in \mathbb{R}^{d,n}}{argmin} \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2 \tag{2}$$

To solve this problem, we first show that the optimal solution takes a specific form.
Let (U, W) be a solution of equation (2). Then the columns of U are orthonormal (namely, $U^T U$ is the identity matrix of $\mathbb{R}^n$) and $W = U^T$.
We have the following theorem: [2]
Let $x_1, \dots, x_m$ be arbitrary vectors in $\mathbb{R}^d$, let $\mathbf{A} = \sum_{i=1}^{m} x_i {x_i}^T$, and let $u_1, \dots, u_n$ n eigenvectors of the matrix A corresponding to the largest n eigenvalues of A. Then, the solution to the PCA optimization problem given in equation (2) is to define U to be the matrix whose columns are $u_1, \dots, u_n$ and to set $W = U^T$.

2.2.3 Random projection

This section shows that reducing the dimension by using a random linear transformation leads to a simple compression scheme with surprisingly low distortion. The transformation x⟼Wx, when W is a random matrix, is often referred to as a random projection. In particular, we provide a variant of a famous lemma due to Johnson and Lindenstrauss, showing that random projections do not distort Euclidean distances too much. [2]

Let x_1, x_2 be two vectors in $\mathbb{R}^d$. A matrix W does not distort too much the distance between x_1 and x_2 if the ratio:

$$\frac{\|W_{x_1} - W_{x_2}\|}{\|x_1 - x_2\|} \text{ is close to 1} \tag{3}$$

In other words, the distances between x_1 and x_2 before and after the transformation are almost the same. To show that $\|W_{x_1} - W_{x_2}\|$ is not too far away from $\|x_1 - x_2\|$, it suffices to show that W does not distort the norm of the difference vector $x = x_1 - x_2$. Therefore, from now on, we focus on the ratio $\frac{\|W_x\|}{\|x\|}$.

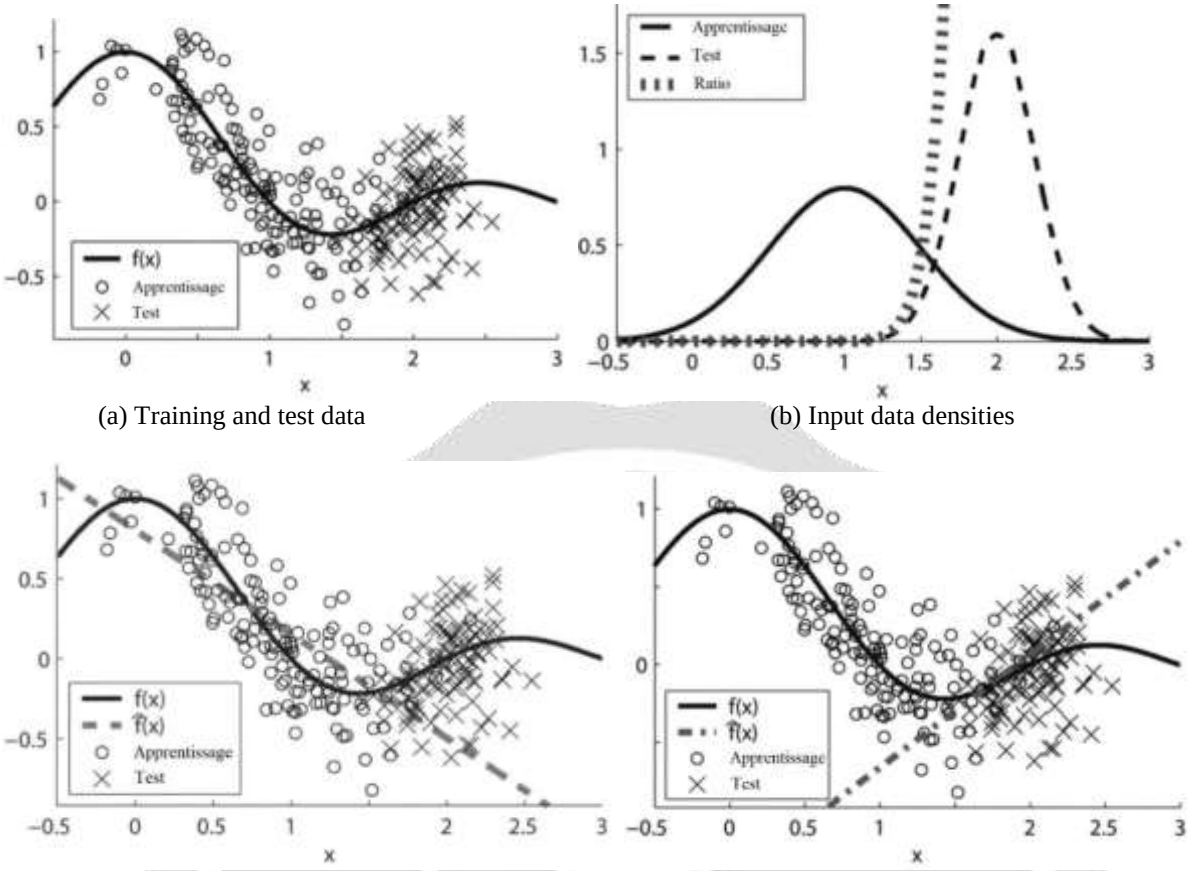
2.3 Covariate shift

If there is no connection between the training data and the test data, nothing will be learned about the test data from the training samples. This means that a reasonable assumption is needed to relate the training samples to the test data. Covariate shift is one of these hypotheses. [4]

2.3.1 Covariate shift adaptation

Consider a regression problem of learning a function f(x) from its samples $\{(x_i^a, y_i^a)\}_{i=1}^{n_a}$. Once a good approximation function $\hat{f}(x)$ is obtained, we can predict the output value y^t at an unseen test entry point x^t by means of $\hat{f}(x^t)$. [4]

In the regression example shown in Figure 01(a), training samples are located in the left-hand side of the graph and test samples are distributed in the right-hand side. Thus, this is an extrapolation problem where the test samples are located outside the training region. The probability densities of the training and test entry points, $p_a(x)$ and $p_t(x)$, are plotted in Figure 01(b).



(a) Training and test data (b) Input data densities

(c) Function learned by ordinary least squares (d) Function learned by importance-weighted least squares

Fig -1: A regression example with covariate shift

Let consider straight-line function fitting by the method of least squares:

$$min_{\theta_1,\theta_2}\left[\sum_{i=1}^{n_a}(\hat{f}(x_i^a)-y_i^a)^2\right] \quad (4)$$

where

$\hat{f}(x) = \theta_1 x + \theta_2$

This ordinary least squares gives a function that goes through the training samples well, as illustrated in Figure 01(c). However, the function learned by least squares is not useful for predicting the output values of the test samples located on the right-hand side of the graph.

Intuitively, training samples that are far from the test region (say, training samples with x <1 in Figure 01(a)) are less informative for predicting the output values of the test samples located on the right-hand side of the graph. This gives the idea that ignoring such less informative training samples and learning only from the training samples that are close to the test region (say, training samples with x > 1,2 in Figure 01(a)) is more promising. The key idea of covariate shift adaptation is to (softly) choose informative training samples in a systematic way, by considering the

importance of each training sample in prediction test output values. More specifically, we use the ratio of training and test input densities (see Figure 01(b)),

$$\frac{p_t(x_i^a)}{p_a(x_i^a)},$$

as a weight for the i-th training sample in the least-squares fitting:

$$min_{\theta_1,\theta_2}\left[\sum_{i=1}^{n_a}\frac{p_t(x_i^a)}{p_a(x_i^a)}(\hat{f}(x_i^a)-y_i^a)^2\right] \quad (5)$$

Then we can obtain a function that extrapolates the test samples well (see Figure 01(d)). Note that test samples are not used for obtaining this function. In this example, the training samples located on the left-hand side of the graph (say, x <1.2) have almost zero importance (see Figure 01(b)). Thus, these samples are essentially ignored in the above importance-weighted least-squares method, and informative samples in the middle of the graph are automatically selected by importance weighting.As illustrated above, importance weights play an essential role in covariate shift adaptation. Below, the problem of covariate shift adaptation is formulated more formally.

2.3.2 Function Learning from examples

Let us consider the supervised learning problem of estimating an unknown input-output dependency from training samples. Let

$$\{(x_i^a, y_i^a)\}_{i=1}^{n_a}$$

be the learning samples, where the training input point of learning

$$x_i^a \in \mathcal{X} \subset \mathbb{R}^d, i = 1,2,\ldots,n_a$$

is an independent and identically distributed (i.i.d.) sample following a probability distribution $P_a(x)$ with density $p_a(x)$:

$$\{x_i^a\}_{i=1}^{n_a} \overset{i.i.d.}{\sim} P_a(x) \quad (6)$$

The training output value:

$$y_i^a \in \mathcal{Y} \subset \mathbb{R}, i = 1,2,\ldots,n_a$$

follows a conditional probability distribution $P(y|x)$ with conditional density $p(y|x)$.

$$y_i^a \sim P(y|x = x_i^a) \quad (7)$$

$P(y|x)$ may be regarded as the superposition of the true output f(x) and the noise ϵ:

$$y = f(x) + \epsilon \quad (8)$$

We assume that the noise ϵ has mean 0 and variance σ^2. Then the function f(x) coincides with the conditional mean of y given x.

2.3.3 Loss Functions

Let loss(x, y, $\hat{y}$) be the loss function which measures the discrepancy between the true output value y at an input point x and its estimate $\hat{y}$. In the regression scenarios where Y is continuous, the squared loss is often used [4].

$$loss(x, y, \hat{y}) = (\hat{y} - y)^2 \tag{9}$$

On the other hand, in the binary classification scenarios where Y = {+ 1, -1}, the following 0/1-loss is a typical choice since it corresponds to the misclassification rate.

$$loss(x, y, \hat{y}) = \begin{cases} 0 \; if \; sgn(\hat{y}) = y, \\ 1 \; otherwise \end{cases} \tag{10}$$

where $sgn(\hat{y})$ denotes the sign of $\hat{y}$:

$$sgn(\hat{y}) \begin{cases} +1 & si\ \hat{y} > 0 \\ 0 & si\ \hat{y} = 0 \\ -1 & si\ \hat{y} < 0 \end{cases}$$

Although the above loss functions are independent of x, the loss can generally depend on x.

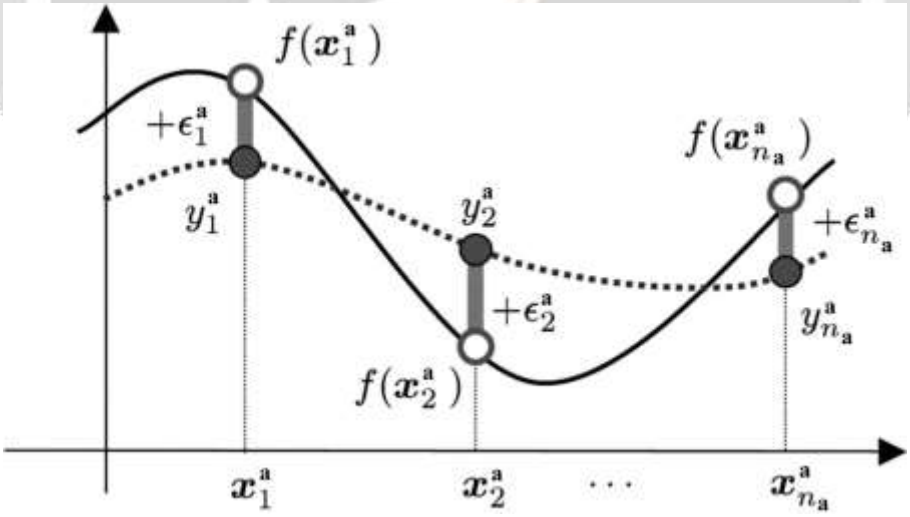


Fig -2: Framework of supervised learning

2.4 Function Approximation

We introduce learning methods that can cope with covariate shift. We employ a parameterized function $\hat{f}(x; \theta)$ for approximating a target function f(x) from training samples $\{x_i^a, y_i^a\}_{i=1}^{n_a}$.

A standard method to learning the parameter θ would be empirical risk minimization (ERM): [4]

$$\hat{\theta}_{ERM} \coloneqq argmin_{\theta}\left[\frac{1}{n_a}\sum_{i=1}^{n_a} loss\left(x_i^a, y_i^a, \hat{f}(x_i^a;\theta)\right)\right] \tag{11}$$

where $loss(x, y, \hat{y})$ is a loss function. If $P_a(x) = P_t(x)$, $\hat{\theta}_{ERM}$ is known to be consistent. Under covariate shift where $P_a(x) \neq P_t(x)$, however, the situation differs: ERM still gives a consistent estimator if the model is correctly specified, but it is no longer consistent if the model is misspecified:

$$\operatorname*{plim}_{n_a\to\infty}\left[\hat{\theta}_{ERM}\right] \neq \theta^* \tag{12}$$

where "plim" denotes the convergence in probability, and θ^* is the optimal parameter in the model:

$$\theta^* \coloneqq argmin_{\theta}[Gen] \tag{13}$$

Gen is the generalization error defined as:

$$Gen = \mathbb{E}_{x^t}\mathbb{E}_{y^t}\left[loss(x^t, y^t, \hat{f}(x^t;\theta))\right] \tag{14}$$

2.4.1 Importance-Weighted ERM

The failure of the ERM method comes the fact that the training input distribution is different from the test input distribution. Importance sampling is a standard technique to compensate for the difference in distributions. The following identity shows the essential idea of importance sampling. For a function g,

$$\mathbb{E}_{x^t}[g(x^t)] = \int g(x)p_t(x)dx \tag{15}$$

$$\mathbb{E}_{x^t}[g(x^t)] = \int g(x)\frac{p_t(x)}{p_a(x)}p_a(x)dx \tag{16}$$

$$\mathbb{E}_{x^t}[g(x^t)] = \mathbb{E}_{x^a}\left[g(x^a)\frac{p_t(x^a)}{p_a(x^a)}\right] \tag{17}$$

where $\mathbb{E}_{x^a}$ and $\mathbb{E}_{x^t}$ denote the expectation on x taken from $p_a(x)$ and $p_t(x)$ respectively.

The amount:

$$\frac{p_t(x)}{p_a(x)}$$

is called importance. The above identity shows that the expectation of a function g over x^t can be computed by the importance-weighted expectation of the function over x^a. Thus, the difference in distributions can be systematically adjusted by importance weighting.

Under covariate shift, importance-weighted ERM (IWERM),

$$\hat{\theta}_{IWERM} \coloneqq argmin_{\theta}\left[\frac{1}{n_a}\sum_{i=1}^{n_a}\frac{p_t(x_i^a)}{p_a(x_i^a)} loss\left(x_i^a, y_i^a, \hat{f}(x_i^a;\theta)\right)\right]$$

is shown to be consistent even for misspecified models, that is, it satisfies:

$$\operatorname*{plim}_{n_a \to \infty} \left[\hat{\theta}_{IWERM}\right] = \theta^*$$

2.4.2 Adaptive IWERM

As shown above, IWERM gives a consistent estimator. However, it also can produce an unstable estimator, and therefore IWERM may not be the best possible method for finite samples. In practice, a slightly stabilized variant of IWERM would be preferable, that is, one achieved by slightly "flattening" the importance weight in IWERM. We call this variant Adaptive IWERM (AIWERM):

$$\hat{\theta}_\gamma := argmin_\theta \left[\frac{1}{n_a}\sum_{i=1}^{n_a}\left(\frac{p_t(x_i^a)}{p_a(x_i^a)}\right)^\gamma loss\left(x_i^a, y_i^a, \hat{f}(x_i^a;\theta)\right)\right] \tag{18}$$

where γ (0≤γ≤1) is called the flattening parameter.

2.4.3 Regularized IWERM

Instead of flattening the importance weight, we can add a regularizer to the empirical risk term. We call this regularized IWERM:

$$\hat{\theta}_\lambda := argmin_\theta \left[\frac{1}{n_a}\sum_{i=1}^{n_a}\frac{p_t(x_i^a)}{p_a(x_i^a)} perte\left(x_i^a, y_i^a, \hat{f}(x_i^a;\theta)\right) + \lambda R(\theta)\right] \tag{19}$$

where R(θ) is a regularization function and λ (≥0) is the regularization parameter that controls the strength of the regularization.

3. Machine learning and intelligent vehicles

3.1 Autonomous driving technologies overview

Autonomous driving is not a single technology, but rather a highly complex system that consist of many subsystems. Let us break it down into three major components: algorithms, including sensing, perception, and decision (which requires reasoning for complex cases); client systems, including the operating system and the hardware platform; and the cloud platform, including high-definition (HD) map, deep learning model training, simulation and data storage. [5] [6]

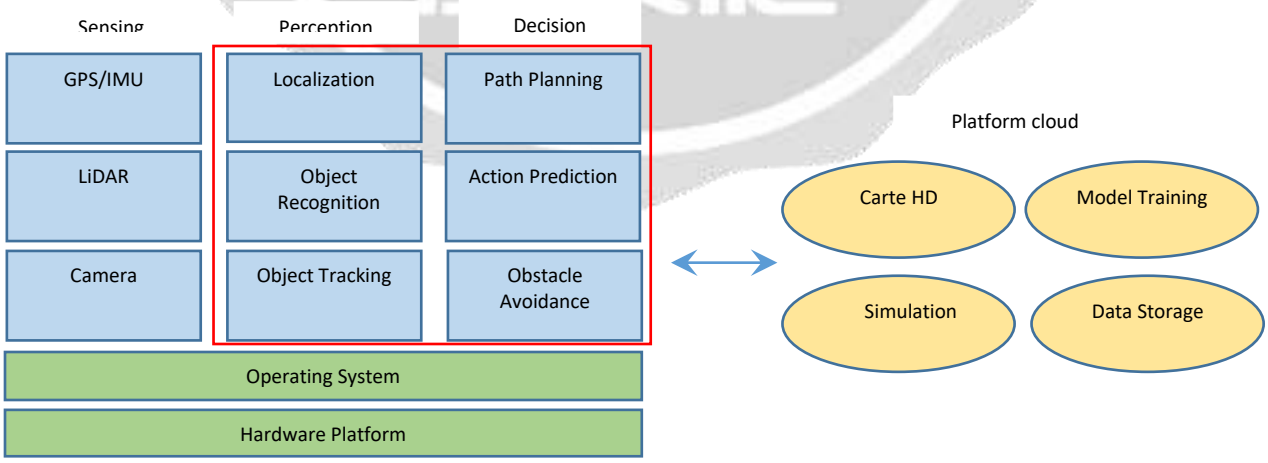


Fig -3: Autonomous driving system architecture overview

The algorithm subsystem extracts meaningful information from the sensor raw data to understand its environment and make decisions about its future actions. The client systems integrate these algorithms to meet real-time and reliability requirements. For example, if the camera is generating data at 60Hz, the client systems need to make sure that the longest stage of the processing pipeline takes less than 16ms to complete. The cloud platform provides offline computing and storage capabilities for autonomous cars. With the cloud platform, we are able to test new algorithms, update HD map and train better recognition, tracking and decision models.

3.2 Autonomous driving algorithms

The algorithms component consists of sensing, that is extracting meaningful information from sensor raw data; perception, which is to localize the vehicle and to understand the current environment; and decision, in other words taking action so as to reliably and safely reach target destinations.

3.2.1 Sensing

Normally, an autonomous car consists of several major sensors. Indeed, each type of sensor presents advantages and drawbacks, in autonomous vehicles, the data from multiple sensors must be combined for increased reliability and safety. They can include the following: [6]

- **GPS/IMU**: The GPS/IMU system helps the autonomous vehicle localize itself by reporting both inertial updates and a global position estimate at a high rate, e.g., 200 Hz. GPS is a fairly accurate localization sensor, but its update rate is slow, at about only 10 Hz, and thus not capable of providing real-time updates. However, IMU errors accumulate over time, leading to a corresponding degradation in the position estimates. Nonetheless, an IMU can provide updates more frequently, at or higher than 200 Hz. This should satisfy the real-time requirement. By combining both GPS and IMU, we can provide accurate and real-time updates for vehicle localization.
- **LiDAR**: LiDAR is used for mapping, localization, and obstacle avoidance. It works by bouncing a beam off surfaces and measures the reflection time to determine distance. Due to its high accuracy, LiDAR can be used to produce HD maps, to localize a moving vehicle against HD maps, to detect obstacle ahead, etc. Normally, a LiDAR unit, such as Velodyne 64-beam laser, rotates at 10 Hz and takes about 1.3 million readings per second.
- **Cameras**: Cameras are mostly used for object recognition and object tracking tasks such as lane detection, traffic light detection, and pedestrian detection, etc. To enhance autonomous vehicle safety, existing implementations usually mount eight or more 1080p cameras around the car, such that we can use cameras to detect, recognize, and track objects in front of, behind, and on both sides of the vehicle. These cameras usually run at 60 Hz, and, when combined, would generate around 1.8 GB of raw data per second.
- **Radar and Sonar**: The radar and sonar system is mostly used for the last line of defense in obstacle avoidance. The data generated by radar and sonar shows the distance as well as velocity from the nearest object in front of the vehicle's path. Once we detect that an object is not far ahead, there may be a danger of a collision, then the autonomous vehicle should apply the brakes or turn to avoid the obstacle. Therefore, the data generated by radar and sonar does not require much processing and usually is fed directly to the control processor, and thus not through the main computation pipeline, to implement such "urgent" functions as swerving, applying the brakes, or pre-tensioning the seatbelts.

3.2.2 Perception

The sensor data is then fed into the perception stage to provide an understanding of the vehicle's environment. The three main tasks in autonomous driving perception are localization, object detection, and object tracking.
GPS/IMU can be used for localization, and, as mentioned above, GPS provides fairly accurate localization results but with a comparatively low update rate, while an IMU provides very fast updates at a cost of less accurate results. We can thus use Kalman Filter techniques to combine the advantages of the two and provide accurate and real-time position updates. As shown in Figure 4, it works as follows: the IMU updates the vehicle's position every 5 ms, but the error accumulates with time. Fortunately, every 100 ms, a GPS update is received, which helps correct the IMU error. By running this propagation and update model, the GPS/IMU combination can generate fast and accurate localization results. Nonetheless, we cannot solely rely on this combination for localization for three reasons: (1) the accuracy is only about one meter; (2) the GPS signal has multipath problems, meaning that the signal may bounce off buildings, introducing more noise; and (3) GPS requires an unobstructed view of the sky and would thus not work in environments such as tunnels. [6]

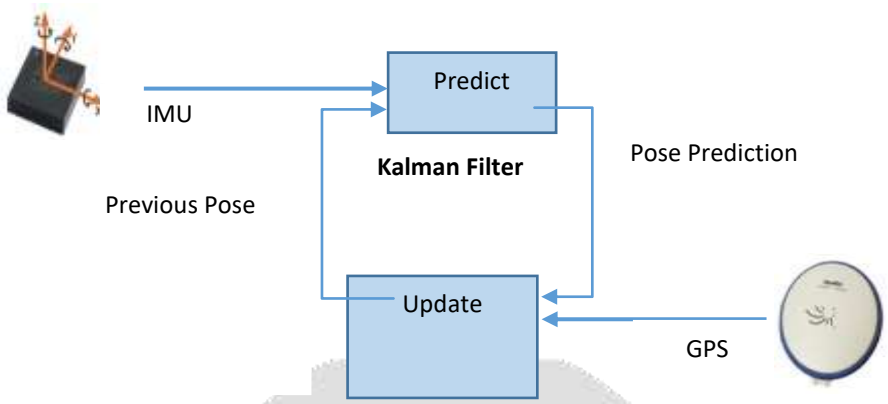


Fig -4: GPS/IMU localization

3.2.3 Environmental variability

Lane detection and traffic sign recognition depend on a number of parameters, such as the rapidly changing road morphology and lighting conditions, making it a not trivial problem. Designing effective lane and traffic sign detection and recognition systems requires the development of complex and specific algorithms taking care of many parameters and issues, with in many cases limited computational resources.
Figure 5 shows some situations making lane detection difficult to handle, while Figure 6 shows some traffic signs recognition scenarios that are difficult to handle.

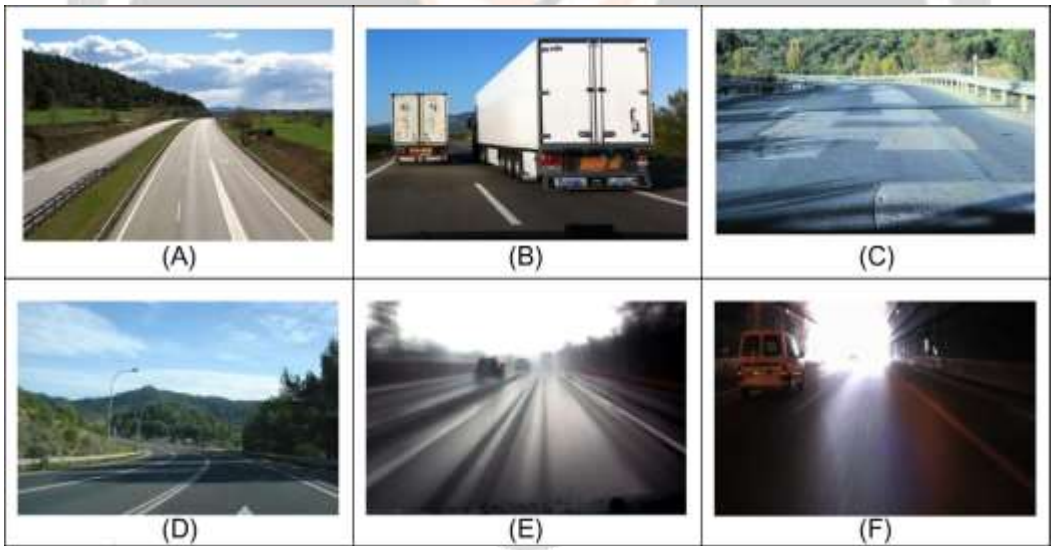


Fig -5: Examples of some extreme situation to handle by lane detection systems.
(A) Different lane marking; (B) Lane occlusion; (C) Change of pavement texture; (D) Lanes with shadow; (E) Rainy road; (F) Saturated image at tunnel exit.

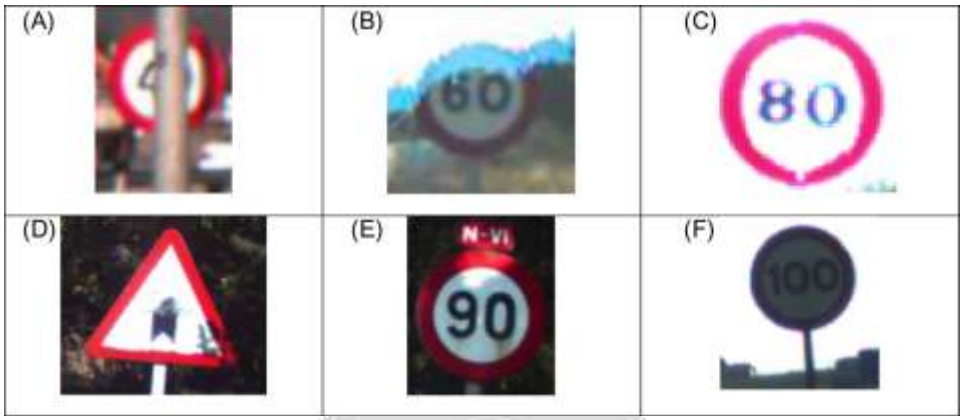


Fig -6: Examples of situations difficult to handle by traffic sign recognition systems.
(A) Occlusion; (B) Smearing; (C) Saturation; (D) Degradation; (E) Shadows; (F) Backlight.

4. Cost functions optimization in the context of intelligent vehicles

4.1 Prediction model and cost function

Consider the lateral movement control. The lateral control problem is complex due to the longitudinal and lateral coupled dynamics as well as the tire behavior.
The different predictive control algorithms differ in the models used to describe the system and in the cost function to be minimized [7]. Mathematical expressions for the prediction and the cost function are:

$$\hat{y} = \mathrm{Gu} + \mathrm{f} \tag{20}$$

with u, proportional to the lateral torque:

$$J = \sum_{j=1}^{p} [\hat{y}(t+j|t) - w(t+j)]^2 + \sum_{j=1}^{m} \lambda [\Delta u(t+j-1)]^2 \tag{21}$$

where $\hat{y}$ is a vector with dimension equal to the prediction horizon containing the predicted outputs until the prediction horizon p, w is the future output desired value, u is a vector with dimension equal to the control horizon m containing the future control actions, G is the dynamic matrix control of the locally linearized system, and f is the vector of free response, with dimension equal to the prediction horizon.
The free response is the prediction of how the system will behave if the command keeps constant and equal to the last command calculated. The λ parameter allows to carry out the weighing of the path tracking errors and the control efforts separately, in the way that we could design a controller to try to adjust to the desired trajectory, regardless of usage command, or on the other hand the controller could be more permissive with the path tracking errors and has a more soft use of command, saving energy in the control.

Interpretations:

- The model of process is used to predict the future outputs using information of the past input signals, past control commands, as well as the future control actions calculated by an optimizer;
- To calculate the future control signals, the optimizer uses the cost function mentioned previously. With this explanation in mind, the model process is fundamental to the correct functioning of the system.

Note that if in the optimization process it doesn't include the restrictions of the physical model, it is possible to obtain the minimization of the next cost function analytically:

$$J = \mathrm{ee}^T + \lambda \mathrm{uu}^T \tag{22}$$

where e is the vector of predicted errors until the prediction horizon and u is the vector of future signal control increments until the control horizon.
The mathematical expression to calculate the future commands is obtained taking the derivative of J and equating to zero:

$$u = (\mathrm{G}^T \mathrm{G} + \lambda I)^{-1} \mathrm{G}^T (\mathrm{w} - \mathrm{f}) \tag{23}$$

The optimizer will be able to calculate the steer angle in the way to minimize the differences between the free response and the desired trajectory. In other words, the optimizer will calculate the steer angle in order to produce the best path tracking.

Interpretations:

- The software reads the sensors and sets the values of the internal states of the system in each iteration. These states are the position, the orientation, and the velocity of the vehicle. With these values the step response of the system is calculated. The parameters of the step response form the dynamic matrix G.
- The prediction of vehicle behavior is calculated using the read values of the sensors at the beginning of the iteration. The prediction of vehicle movement is compared with the desired trajectory from the point closest to the prototype.
- The future errors vector is the result of the previous comparison. The future commands are obtained using the equation, but only the first term of future commands vector is applied, keeping in mind the concept of sliding horizon. Finally, the variables of the algorithm are updated.

4.2 Motion planning cost function

The task of motion planning is to generate a trajectory and send it to the feedback control for the execution of the physical command of the vehicle. The planned path is usually specified and represented as a sequence of planned path points. Each of these points contains attributes like location, time, speed, curvature, etc.

4.2.1 Motion planning with longitudinal planning and lateral planning

Instead of doing path planning and then speed planning, proposed the idea of doing longitudinal and lateral planning. A desired motion planning trajectory should be smooth, which is usually represented mathematically in continuity of pose with its derivatives in a dimension (eg, position, speed, and acceleration). And ease and comfort can be best measured by jerk, which is the change rate in acceleration.
Consider the jerk-optimal trajectory connecting a start state $P_0 = [p_0, \dot{p}_0, \ddot{p}_0]$ and a end state $P_1 = [p_1, \dot{p}_1, \ddot{p}_1]$, where the pose P could be either the longitudinal s-pose or lateral l-pose, in a time frame of $T := t_1 - t_0$. Note the integral of the squared jerk to be: $S_t(p_t) := \int^{t_1} \dddot{p}^2(\tau) d\tau$. An important proposition that we have is that, for any cost function with the form [6]:

$$J = K_j S_t + K_t g(T) + K_p h(p_1) \tag{24}$$

where g and h are arbitrary functions and $K_j, K_t, K_p > 0$, the optimal solution for minimizing the above cost function is a quintic polynomial. Intuitively, the cost function penalizes the high jerk integral S_t along the trajectory, and also considers the time T and end pose $h(p_1)$.

4.2.2 Lateral planning

We start planning with the lateral L dimension. Then the start state P_0 becomes $D_0 = [d_0, \dot{d}_0, \ddot{d}_0]$, and we set this start state according to the actual end state of the previous planning frame, such that the continuities are maintained. The end state P_1 are chosen from a set of possible candidate lateral offset with $\dot{d}_1 = \ddot{d}_1 = 0$ since we prefer to move parallel to the central reference line direction (s-direction). Functions g and h are chosen with g(T)=T

and $h(d_1) = d_1^2$. We can see that slow convergence is penalized as well as any lateral difference with d=0 at the end state. With the optimal solution taking the form of a quintic polynomial:

$$l_{optimal}(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \quad (25)$$

which minimizes the cost function of:

$$J_l = K_jS_t(l(t)) + K_tT + K_ld_1^2 \quad (26)$$

The coefficients $a_5, a_4, a_3, a_2, a_1, a_0$ could be calculated with boundary conditions at $D_0 = [d_0, \dot{d}_0, \ddot{d}_0]$ and $D_0 = [d_1, \dot{d}_1, \ddot{d}_1]$. We could choose a set of d_i and compute a set of best candidate one-dimensional trajectories as $\text{Solution}_{i,j}$ from $[d_1, \dot{d}_1, \ddot{d}_1, T]_{i,j} = [d_1, 0, 0, T_j]$.

Interpretations:

- Each candidate solution will have a cost. For each candidate, we check if this solution will be consistent with upstream behavioral decision outputs and make this solution in valid if any violations/collisions is found. The remaining valid trajectories make a candidate set of lateral dimension and will be utilized in computing the planned trajectory on the 2D dimension.
- The method described above works well for high-speed trajectories where longitudinal move and lateral move could be chosen independently.

4.3 Cyberattack cost function

The significant growth of autonomous systems involves a wide range of cyber-physical systems (CPS), and thus exposes the threat of cyberattacks due to their close integration of computational resources, physical processes and communication capabilities.
Consider a discrete-time linear CPS model subjected to a priori-unknown cyberattacks: [8]

$$x_a(k+1) = Ax_a(k) + Bu(k) + B_aa(k), \qquad x_a(0) = x_0 \quad (27)$$

where

- the index "a" designates the system under attack;
- $x_a(k) \in \mathbb{R}^n et\ u(k) \in \mathbb{R}^p\ are\ the\ CPS's state\ and\ input$
- A and B are the system matrices of appropriate dimensions;
- k ∈ N designates the discrete-time index, taking values from the time horizon (possibly infinite) N={0,1,2,…, N}

Various attack sequences a(k) ∈ $\mathbb{R}^s$ can be injected into the CPS with the attack matrices B_a of compatible dimension. It is assumed that the system matrix pairs (A, B) satisfy the controllability condition. Practically, the injected attacks are realized through the physical structure of actuator components. Thus, the attack matrix B_a should satisfy [8]:

$$span\{B_a\} \subseteq span\{B\} \quad (28)$$

where "span" denotes the columns space of the matrix.
For simplicity, let us use the following notation:

$$x_{[\tau_1,\tau_2]} := [x^T(\tau_1)x^T(\tau_1+1) \ldots x^T(\tau_2)]^T, \qquad 0 \le \tau_1 \le \tau_2 \le N$$

With perfect state measurements, the state history of the CPS, $x_{a[0,k]}$, and past attacks, $a_{[0,k-1]}$, are available for each time step k, though the future attack signals for $a_{[k,N]}$ are unknown.

To measure the impact of cyberattacks on the CPS, we introduce a cost function $J(u_{[\tau_1,\tau_2]}, a_{[\tau_1,\tau_2]})$. The cost function during the time interval $[\tau_1, \tau_2]$. is defined by the following quadratic form [8]:

$$J(u_{[\tau_1,\tau_2]}, a_{[\tau_1,\tau_2]}) := \sum_{k=\tau_1}^{\tau_2} \left(x_a^T(k) Q x_a(k) + u^T(k) R u(k)\right) + x_a^T(\tau_2 + 1) Q x_a(\tau_2 + 1) \quad (29)$$

where Q≥0 and R> 0 are design parameters including the information about scaling factors, security weights, control efforts, etc.

Interpretations:

- Note that the cost function J can be established according to the CPS's operational objective, and thus motivating different control strategies. It can be seen that J simultaneously evaluates the control performance and attack severity over the time interval $[\tau_1, \tau_2]$.

4.4 Dimensionality reduction

Ubiquitous computing devices have changed the acquisition of mobility data: the high penetration rate and their ability to capture and share information on a continuous basis. This applies to geolocation information, GPS position, the operational mobile phone data, exchanged frequently with the network, and also social network crowdsourced information. Additionally, under the umbrella of the Internet of Things trend, the deployment of the connected vehicle (or Car-as-a-sensor) concept, supported by advanced V2X communications, provides massive data volume. [7]

Motion planning, first for mobile robots and then for autonomous vehicles, has been widely studied in recent decades. The resulting strategies were designed to respond, under different assumptions, to a variety of kinematic, dynamic and environmental constraints. The notion of costmap can be considered at this stage.

A costmap is a fundamental concept in autonomous robotics. It represents the cost (difficulty) of crossing different areas of the map.

The costmap maintains information about occupied/free areas in the map in the form of an occupancy grid. It uses sensor data and information from the static map to store and update information about obstacles in the world, which are marked in the map (or cleared, if they are no longer there). [7]

Costmap computation is supported on a layered costmap, which will be used for the integration of the different information sources into a single-monolithic costmap. At each layer, information on occupied/free areas in the surrounding of the vehicle is maintained in the form of an occupancy grid, using the different observation sources as input. Using this information, both dynamic and static obstacles are marked in the map. For example, let us suppose each cell in the map can have 255 different cost values. Then, at each layer, costmap is represented as follows:

- A value of 255 means that there is no information available about a specific cell in the map;
- 254 means that a sensor has marked this specific cell as occupied. This is considered a lethal cell, so the vehicle should never enter there;
- The rest of the cells are considered as free, but with different cost levels depending on an inflation method relative to the size of the vehicle and its distance to the obstacle.

Cost values decrease with the distance to the nearest occupied cell using the following expression:

$$C(i,j) = \exp(-1 \cdot \alpha \cdot (\|c_{ij} - \vec{o}\| - \rho_{\text{inscrit}})) \cdot 253$$

In this expression, α is a scaling factor that allows increasing or decreasing the decay rate of the cost of the obstacle. $\|c_{ij} - \vec{o}\|$ is the distance between cell $c_{ij} \in C$ (where C is the set of cells of the costmap) and the obstacle. Finally, ρ_{inscrit} is the inscribed radius, which is the inner circle of the limits of the vehicle.

5. CONCLUSIONS

This paper has allowed us to focus on cost function for optimizing machine learning algorithms in intelligent vehicle context. We have opened our writing with the different optimization options such as cost function, dimensionality reduction, covariate shift and function approximation.

We did then focus mainly on applying a cost function on a predictive controller. Thus, the model can ensure the proper functioning of the system to predict future outputs using information from past input signals, past control commands, as well as future control actions calculated by an optimizer. We could see Bayesian optimizer result that uses the performance information to iteratively update the dynamic model to improve performance. Cost function can also be applied to the motion planning, whether longitudinal or lateral. It makes it possible to see solution consistency with the behavioral decision outputs upstream and thus validate trajectories if violations or collisions are found.

Given connectivity importance for an autonomous and connected vehicle, we were able to define the cyberattack cost function. The result is the simultaneous assessment of the control performance and the attack severity over a time period.

Finally, we work on the dimensionality reduction given the volume of data handled and processed. We were able to use the costmap concept from which we could develop the collision map. Thus, the different danger levels will be mapped.

6. REFERENCES

[1] C. Sammut, G. I. Webb, « *Encyclopedia of Machine Learning and Data Mining* », Springer, 2017

[2] S. Shalev-Shwartz, S. Ben-David, « *Understanding machine learning: from theory to algorithms* », Cambridge University Press, 2014

[3] S Theodoridis, « *Machine Learning, A Bayesian and Optimization Perspective* », Academic Press, 2020

[4] M Sugiyama, M. Kawanabe, « *Machine Learning in Non–Stationary Environments – Introduction to Covariate Shift Adaptation* », MIT PRESS, 2012

[5] X. Zhang, M. Mansoor Khan, *« Principles of Intelligent Automobiles »*, Springer, 2019

[6] S. Liu, L. Li, J. Tang, S. Wu, J.-L. Gaudiot, « *Creating Autonomous Vehicle Systems* », Morgan & Claypool Publishers, 2018

[7] F Jiménez, « *Intelligent Vehicles* », Elsevier, 2018

[8] H. Yu, X. Li, R. M. Murray, S. Ramesh, C. J. Tomlin, « *Safe, Autonomous and Intelligent Vehicles* », Springer, 2019