Alma Mater Studiorum · Università di Bologna

**Scuola di Scienze**
**Corso di Laurea in Fisica**

# Predicting CMS datasets popularity with Machine Learning

**Relatore:**
**Prof. Daniele Bonacorsi**

**Correlatore:**
**Prof. Valentin Kuznetsov**

**Presentata da:**
**Luca Giommi**

**Sessione II**
**Anno Accademico 2014/2015**

*Ita fac, mi Lucili: vindica te tibi, et tempus quod adhuc aut auferebatur aut subripiebatur aut excidebat collige et serva. [...] Fac ergo, mi Lucili, quod facere te scribis, omnes horas complectere; sic fiet ut minus ex crastino pendeas, si hodierno manum inieceris. Dum differtur vita transcurrit. Omnia, Lucili, aliena sunt, tempus tantum nostrum est; in huius rei unius fugacis ac lubricae possessionem natura nos misit, ex qua expellit quicumque vult.*
*[Seneca, Epist. ad Lucil. I]*

# Contents

# Sommario

Questa tesi discute l'architettura, lo sviluppo e l'utilizzo di una tecnica di "supervised Machine Learning (classification)" al fine di soddisfare le necessità dell'esperimento CMS in termini di predizione della "popolarità" dei dataset di CMS su Grid.

L'esperimento CMS ha completato il suo primo periodo di presa dati a LHC (Run-1). Dopo il long shutdown (LS1), CMS sta ora raccogliendo dati di collisioni p-p a 13 TeV come energia nel centro di massa in Run-2. L'esperienza nelle operazioni di calcolo di CMS che è stata sviluppata negli ultimi anni è enorme, come è assai ampio il volume dei metadati raccolti nei database di CMS e relativi alle operazioni relative ai workflow di CMS condotte sui Tiers della Worldwide LHC Computing Grid. Tali dati sono stati raramente oggetto di campagne di data mining, ma sono cruciali per una migliore comprensione delle modalità con cui sono state condotte computing operations di grande successo in Run-1. Essi sono fondamentali per costruire una modellizzazione adattiva del sistema di calcolo di CMS, che permetta ottimizzazioni dettagliate e puntuali nonchè predizioni sul comportamento futuro dei sistemi. In CMS è stato lanciato un progetto di Data Analytics e, all'interno di esso, un'attività specifica pilota che mira a sfruttare tecniche di Machine Learning per predire la popolarità dei dataset di CMS. Si tratta di un'osservabile molto delicata, la cui eventuale predizione premetterebbe a CMS di costruire modelli di data placement più intelligenti, ampie ottimizzazioni nell'uso dello storage a tutti i livelli Tiers, e formerebbe la base per l'introduzione di un solito sistema di data management dinamico e adattivo. Questa tesi descrive il lavoro fatto sfruttando un nuovo prototipo pilota chiamato DCAFPilot, interamente scritto in python, per affrontare questa sfida.

**Il Capitolo 1** offre un'introduzione alla Fisica delle Altre Energie a LHC.

**Il Capitolo 2** descrive il Modello di Calcolo di CMS con maggiore attenzone al settore del data management, introducendo e discutendo anche il concetto di popolarità.

**Il Capitolo 3** offre una breve introduzione a concetti basilari di Machine Learning.

**Il Capitolo 4** descrive il contesto del progetto CMS Data Analytics, nonchè l'architettura e le funzionalità del prototipo pilota DCAFPilot.

**Il Capitolo 5** presenta e discute i risultati ottenuti. Un riassunto di essi e i prossimi passi in tale attività sono presentati nelle Conclusioni.

# Abstract

This thesis presents the design, development and exploitation of a supervised Machine Learning classification system aimed at attacking the very concrete need of the prediction of the "popularity" of the CMS datasets on the Grid.

The CMS experiment has completed its first data taking period at the LHC (Run-1). After a long shutdown (LS1), CMS is now collecting data on p-p collisions at 13 TeV of centre-of-mass energy in Run-2. The amount of experience collected in CMS computing operations during the last few years is enormous, and the volume of metadata in CMS database systems which describes such experience in operating all the CMS workflows on all the Worldwide LHC Computing Grid Tiers is huge as well. Data mining efforts into all this information have rarely been done, but are of crucial importance for a better understanding of how CMS did successful operations, and to reach an adequate and adaptive modelling of the CMS operations, in order to allow detailed optimizations and eventually systems behaviour predictions. A Data Analytics project has been launched in CMS and, within this area of work, a specific activity on exploiting machine learning techniques to predict dataset popularity has been launched as a pilot project. The popularity of a dataset is an important observable to predict, as its control would allow a more intelligent data placement, large optimizations in the storage utilization at all Tiers levels, and would form the basis of a solid, self-tuning, adaptive dynamic data management system. This thesis describes the work done exploiting a new pilot prototype called DCAFPilot, entirely written in python, to attack this kind of challenge.

**Chapter 1** gives an introduction to High Energy Physics at the LHC.

**Chapter 2** describes the CMS Computing Model, with main focus on the data management sector, introducing and discussing also the concept of popularity.

**Chapter 3** offers a brief introduction to basic machine learning concepts.

**Chapter 4** describes the context of the CMS Data Analytics project, and the architecture and functionalities of the DCAFPilot prototype.

**Chapter 5** presents and discuss the results obtained. A summary and the next steps foreseen in this activity are presented in the Conclusions.

# Chapter 1

# High Energy Physics at LHC

## 1.1 Overview of the LHC accelerator

The Large Hadron Collider (LHC) [1, 2] is a two-ring particle accelerator and collider (Figure 1.1) built by the European Organization for Nuclear Research (CERN). It is located beneath the Franco-Swiss border near Geneva in Switzerland where the Large Electron-Positron collider (LEP) previously existed [3]. The purpose of the LHC is to give scientists an experimental apparatus that would enable them to test theories in high energy physics, such as the existence of the Higgs boson, the search for supersymmetries, the search for particles that would indicate the existence of dark matter etc . One of its recent results was the discovery of the Higgs boson, publicly announced on July 4th 2012, predicted by the Standard Model [4]. LHC consists of a 27 km long circular ring, designed to accelerate protons and heavy ions at high energies . LHC is characterised by two accelerated beams travel in opposite directions inside different channels in the same pipe at ultrahigh vacuum.

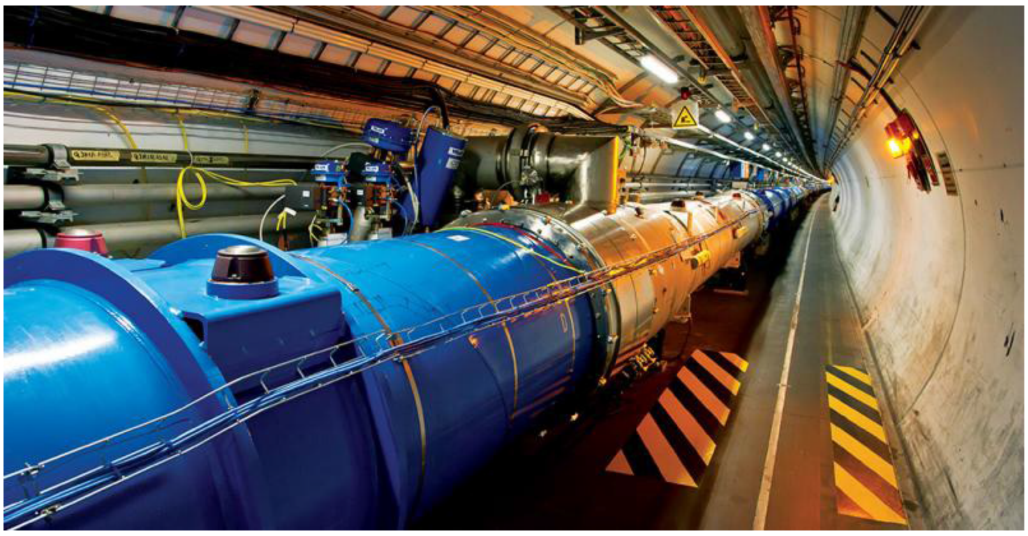


**Figure 1.1:** The LHC tunnel.

The acceleration process for protons is done in five steps (see Figure 1.2).

Initially, hydrogen atoms are ionized in order to produce protons and then they are injected in the LINAC 2, a linear accelerator. When protons reach an energy of 50 MeV they subsequently enter the Booster where their energy goes up to 1.4 GeV. After that, they enter the Proton Synchrotron (PS) where 277 electromagnets push the protons to 99,9% the speed of light: at this point, each proton has an energy of 25 GeV . Then, proton bunches are accelerated in the Super Proton Synchrotron (SPS), a circular particle accelerator with a circumference of 7 km. After protons have reached an energy of 450 GeV, they are injected into the LHC in two separate pipes in which they move in opposite directions; here, through magnets, the particles can be accelerated up to their maximum designed energy of 7 TeV. Currently this is the maximum energy of each proton beam reachable after the last improvement completed in March 2015 from which began RUN-2, the second LHC data taking period. The two LHC channels intersect in the four caverns (where the four detectors are installed). Here protons can collide and the products of the collisions can be revealed.

The vacuum system is necessary so the particles do not lose energy in the acceleration process due to impacts with the molecules that constitute air. The LHC vacuum system is made up of three individual vacuum systems: the insulation vacuum for cryomagnets, the insulation vacuum for helium distribution, and the beam vacuum.

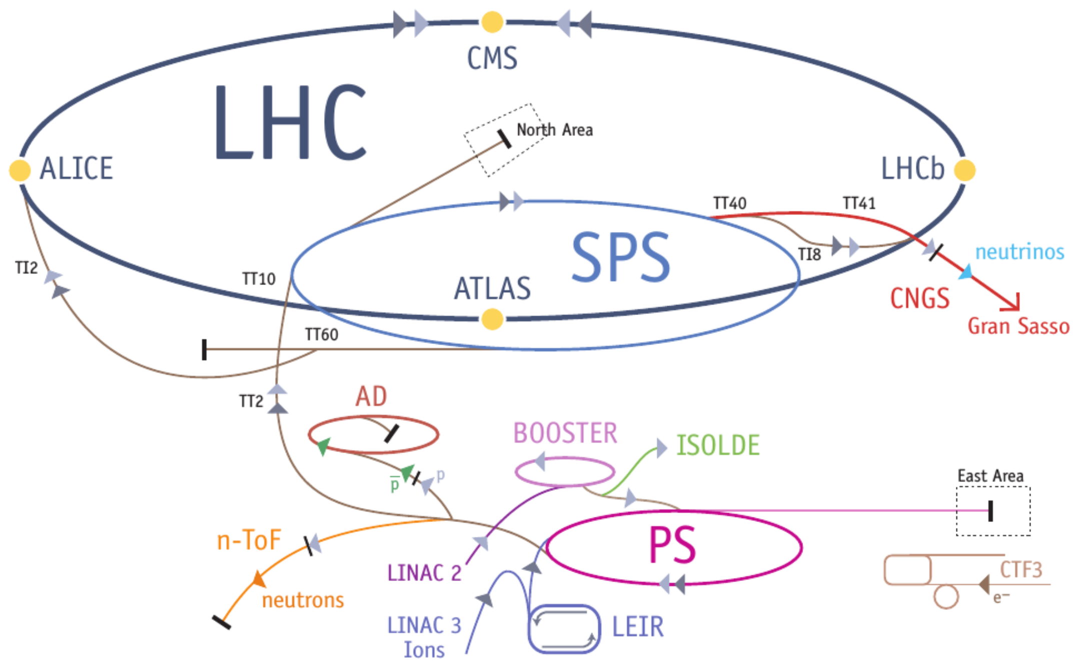


**Figure 1.2:** The LHC accelerator complex at CERN.

### 1.1.1 Electromagnets

There is a large variety of magnets in the LHC (dipoles, quadrupoles, sextupoles, octupoles, decapoles) giving a total of about 9600 magnets. Each type of magnet contributes to optimizing a particle's trajectory: dipoles have the function to maintain the beams in their circular orbit, while quadrupoles are used to focus the beam down to the smallest possible size at the collision points, thereby maximizing the chance of two protons smashing head-on into each other. The dipoles of the LHC represented the most important technological challenge in LHC design. In a proton accelerator like the LHC, the maximum energy that can be achieved is directly proportional to the strength of the dipole field, given a specific acceleration circumference. At the LHC the dipole magnets are superconducting electromagnets, which are able to provide the very high field of 8.3 T over their length. None solution could have been designed using 'warm' magnets instead of superconducting ones. The LHC dipoles use niobium-titanium (NbTi) cables, which become superconducting below a temperature of 10 K, that is, they conduct electricity without resistance: indeed LHC operates at 1.9 K, even lower than the temperature of outer space, 2.7 K).

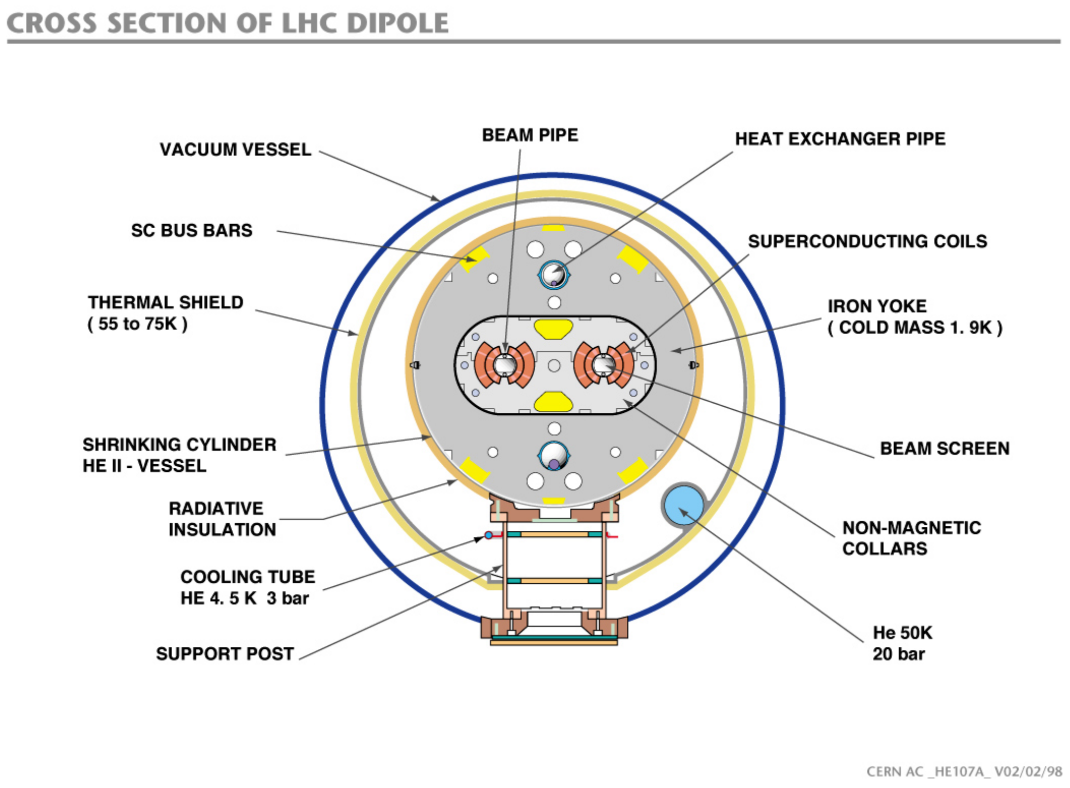

CROSS SECTION OF LHC DIPOLE
VACUUM VESSEL
BEAM PIPE
HEAT EXCHANGER PIPE
SC BUS BARS
SUPERCONDUCTING COILS
THERMAL SHIELD ( 55 to 75K )
IRON YOKE ( COLD MASS 1. 9K )
SHRINKING CYLINDER HE II - VESSEL
BEAM SCREEN
RADIATIVE INSULATION
NON-MAGNETIC COLLARS
COOLING TUBE HE 4. 5 K 3 bar
SUPPORT POST
He 50K 20 bar
CERN AC _HE107A_ V02/02/98

**Figure 1.3:** Cross section of LHC dipole.

### 1.1.2 Radiofrequency cavities

Along LHC length there are radiofrequency (RF) cavities that are metallic chambers containing an electromagnetic field. Their primary purpose is to accelerate charged particles and to divide protons in packages keeping them grouped. To prepare an RF cavity to accelerate particles, an RF power generator supplies an electromagnetic field. The RF cavity is molded to a specific size and shape so that electromagnetic waves become resonant and build up inside the cavity (see Figure 1.4). Charged particles passing through the cavity feel the overall force and direction of the resulting electromagnetic field, which transfers energy to push them forwards along the accelerator. The field in an RF cavity is made to oscillate (switch direction) at a given frequency, so timing the arrival of particles is important. On the Large Hadron Collider (LHC), each RF cavity is tuned to oscillate at 400 MHz. The ideally timed proton, with exactly the right energy, will see zero accelerating voltage when the LHC is at full energy. Protons with slightly different energies arriving earlier or later will be accelerated or decelerated so that they stay close to the energy of the ideal particle. In this way, the particle beam is sorted into discrete packets called "bunches". During the energy-boosting process the protons in the bunches shift collectively to see an overall acceleration on each passage through the cavities, picking up the energy needed to keep up with the increasing field in the LHC powerful magnets. Top energy is reached in around 15 minutes, the bunches having passed the cavities around 1 million times. The 16 RF cavities on the LHC are housed in four cylindrical refrigerators called cryomodules – two per beam – which keep the RF cavities working in a superconducting state, without losing energy to electrical resistance.

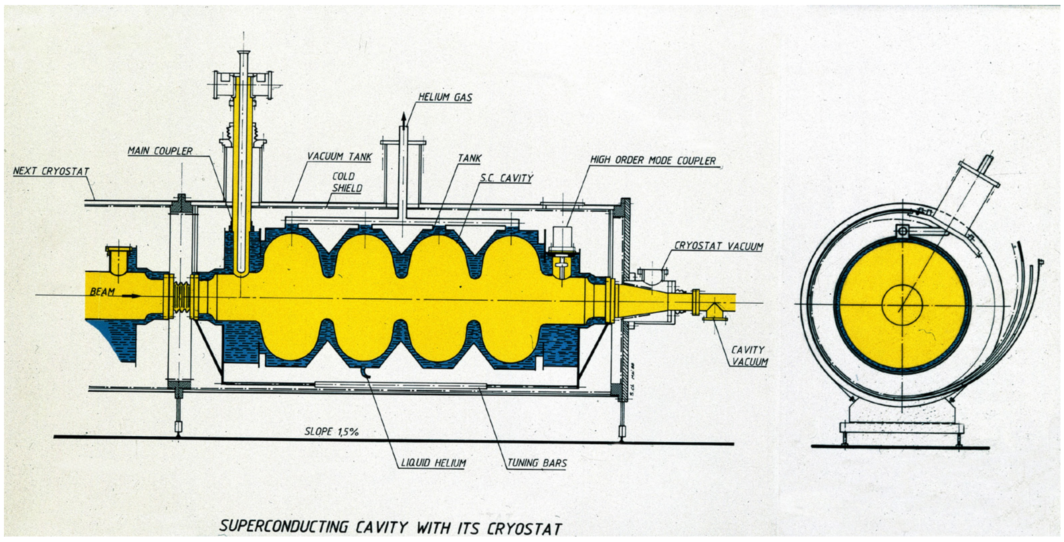


**Figure 1.4:** Schematic drawing of a superconducting cavity.

In LHC, at operation conditions, each proton beam is divided into 2808 bunches, each one of them has about $10^{11}$ protons. Their dimension is not constant along the circumference when they travel far from the collision point: their size is about few centimetres length and 1 millimeter width. Otherwise in the collision points protons are collimated and the packages are compressed to about 16 nm. At full luminosity the packages are separated by about 7m (25 ns).

### 1.1.3 Luminosity and other characteristic parameters

An important parameter which characterizes a particle accelerator is the machine luminosity ($L$) defined as:

$$L = \frac{f_{rev} n_b N_b^2 \gamma_r}{4\pi\epsilon_n \beta^*} F \tag{1.1}$$

where $f_{rev}$ is the revolution frequency, $n_b$ is the number of bunches per beam, $N_b$ is the number of particles in each colliding beam, $\epsilon_n$ is the normalized transverse beam emittance, $\beta^*$ is the beta function at the collision point, $\gamma_r$ is a relativistic factor, and $F$ the geometric luminosity reduction factor. The number of events that occur each second is:

$$N_{event} = L\sigma_{event} \tag{1.2}$$

Following are listed some of the most relevant parameters of LHC.

**Table 1.1:** LHC main technical parameters.

| **Quantity** | **value** |
|---|---|
| **Circumference** (m) | 26 659 |
| **Magnets working temperature** (K) | 1.9 |
| **Number of magnets** | 9593 |
| **Number of dipoles** | 1232 |
| **Number of quadrupoles** | 392 |
| **Number of radiofrequency cavities per beam** | 8 |
| **Protons nominal energy** (TeV) | 7 |
| **Ions nominal energy** (TeV/nucleon) | 2.76 |
| **Maximum intensity of the magnetic field** (T) | 8.33 |
| **Design luminosity** ($\mathrm{cm^{-2}\,s^{-1}}$) | $10 \times 10^{34}$ |
| **Number of proton bunches per beam** | 2808 |
| **Number of protons per bunch** | $1.1 \times 10^{11}$ |
| **Minimum distance between bunches** (m) | $\sim 7$ |
| **Revolutions per second** | 11 245 |
| **Collisions per second** (milions) | 600 |

## 1.2 The experiments at LHC

Four experiments (and others smaller) at the Large Hadron Collider (LHC) use detectors to analyse the myriad of particles produced by collisions in the accelerator. These experiments are run by large collaborations of scientists from institutes all over the world. Each experiment is distinct, and characterized by detectors and subdetectors. The main four experiments are ALICE, ATLAS, CMS and LHCb while among other secondary experiments are for example TOTEM, LHCf and MoEDAL. The main experiments are installed in four caverns built around the four collision points.

### 1.2.1 ALICE

The Large Ion Collider Experiment (ALICE) [5, 6] is a general-purpose, heavy-ion detector which studies the strong interaction, in particular quark-gluon plasma at extreme values of energy density and temperature during the collision of heavy nuclei (Pb). Collisions in the LHC generate temperatures more than 100000 times hotter than the centre of the Sun. For part of each year the LHC provides collisions between lead ions, recreating in the laboratory conditions similar to those just after the Big Bang. Under these extreme conditions protons and neutrons "melt", freeing the quarks from their bonds with the gluons. This is a state of matter called quark-gluon plasma. ALICE studies as it expands and cools, observing how it progressively gives rise to the particles that constitute the matter of our universe today. In order to study quark-gluon plasma the ALICE collaboration uses a 10,000-tonne detector which is 26 m long, 16 m high, and 16 m wide. The detector sits in a vast cavern 56 m below ground close to the village of St Genis-Pouilly in France, receiving beams from the LHC. The collaboration counts more than 1000 scientists from over 100 physics institutes in 30 countries.

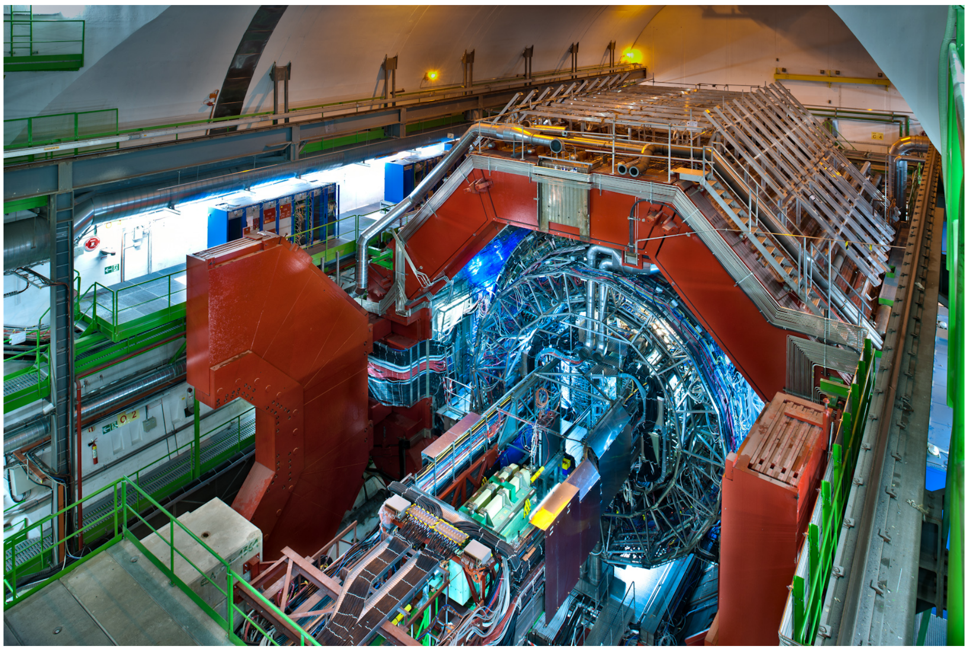


**Figure 1.5:** The ALICE detector.

### 1.2.2 ATLAS

A Toroidal LHC ApparatuS (ATLAS) [7, 8] is an experiment whose main purpose is to investigate new physics beyond the Standard Model, exploiting the extremely high energy at the LHC. Besides it searches the existence of dark matter and extra dimensions. The detector is made by four main layers: the magnet system, that bends the trajectories of charged particles; the Inner Detector, which measures the path of charged particles; the calorimeters, which identify photons, electrons, and jets; the Muon Spectrometer, which recognises the presence of muons.

The interactions in the ATLAS detectors create an enormous flow of data. To digest it, ATLAS uses an advanced "trigger" system to tell the detector which events to record and which to ignore. A complex data-acquisition and computing systems are then used to analyse the collision events recorded. At 46 m long, 25 m high and 25 m wide, the 7000-tonne ATLAS detector is the largest volume particle detector ever constructed. It sits in a cavern 100 m below ground near the main CERN site, close to the village of Meyrin in Switzerland. More than 3000 scientists from 174 institutes in 38 countries work on the ATLAS experiment (February 2012).

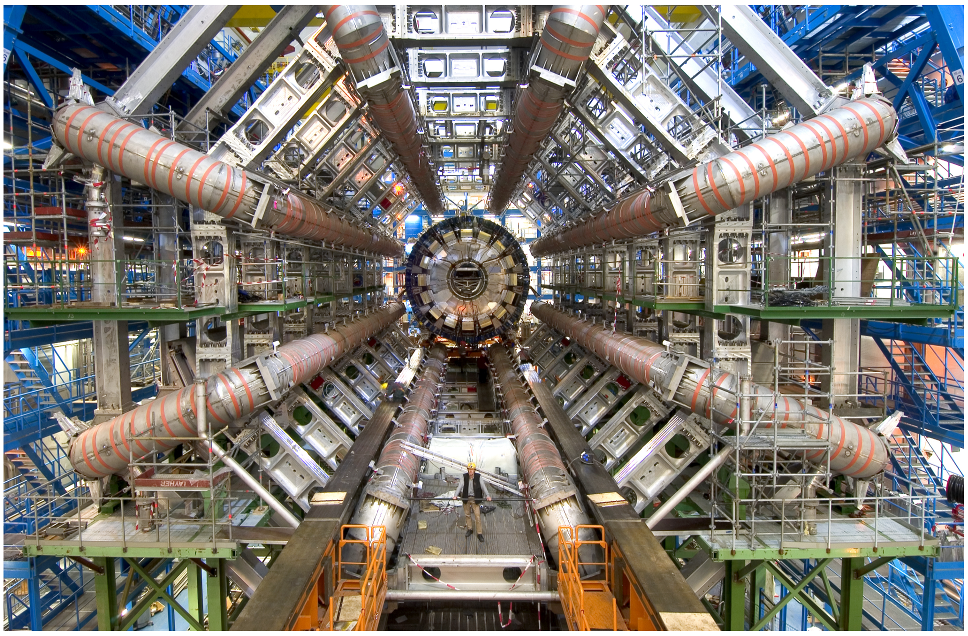


**Figure 1.6:** The ATLAS detector.

### 1.2.3 CMS

The Compact Muon Solenoid (CMS) [9, 10] is a general-purpose detector at the Large Hadron Collider (LHC). It has a broad physics programme ranging from studying the Standard Model, including the Higgs boson, to searching for extra dimensions and particles that could make up dark matter. Although it has the same scientific aims as the ATLAS experiment, it uses different technical solutions and a different magnet-system design. The CMS detector is built around a huge solenoid magnet. This takes the form of a cylindrical coil of superconducting cable that generates a field of 4 Tesla, that is about 100,000 times the magnetic field of the Earth. The field is confined by a steel "yoke" that forms the bulk of the detector's 14,000-tonne weight. An unusual feature of the CMS detector is that instead of being built in-situ, like the other giant detectors of the LHC experiments, it was constructed in 15 sections at ground level before being lowered into an underground cavern near Cessy in France and reassembled. The whole detector is 21 metres long, 15 metres wide and 15 metres high. The CMS experiment is one of the largest international scientific collaborations in history, involving 4300 particle physicists, engineers, technicians, students and support staff from 182 institutes in 42 countries (February 2014).

### 1.2.4 LHCb

The Large Hadron Collider beauty (LHCb) [11, 12] experiment is specialized in investigating the slight differences between matter and antimatter by studying the

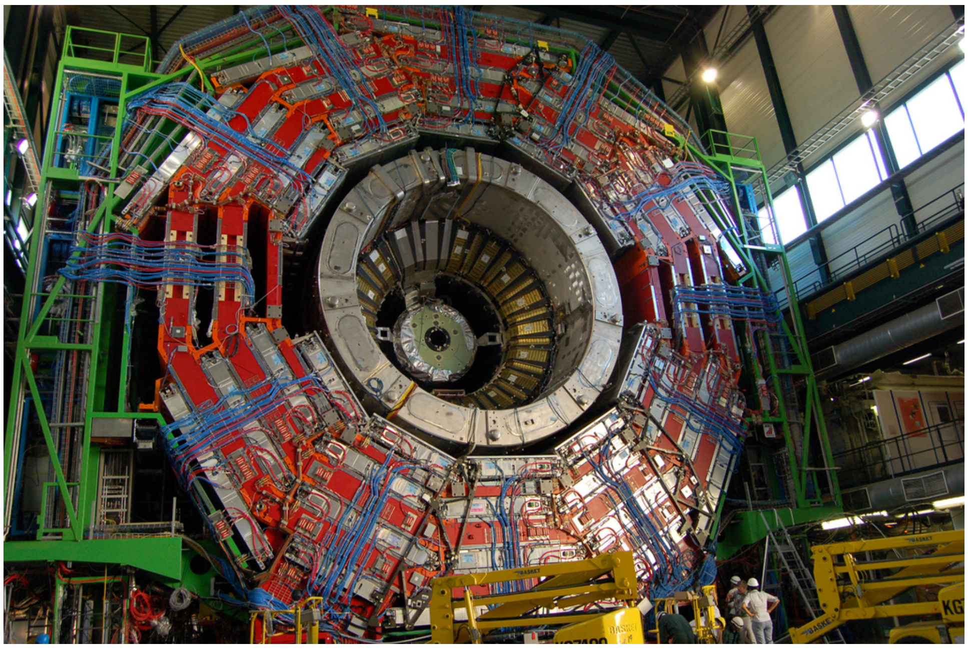


**Figure 1.7:** The CMS detector.

"beauty quark". Instead of surrounding the entire collision point with an enclosed detector as do ATLAS and CMS, the LHCb experiment uses a series of subdetectors to detect mainly forward particles, those thrown forwards by the collision in one direction. The first subdetector is placed close to the collision point, with the others following one behind the other over a length of 20 metres. The 5600-tonne LHCb detector is made up of a forward spectrometer and planar detectors. It is 21 metres long, 10 metres high and 13 metres wide, and sits 100 metres below ground near the village of Ferney-Voltaire, France. About 700 scientists from 66 different institutes and universities make up the LHCb collaboration (October 2013).

Other experiments at LHC include LHCf, TOTEM, and MoEDAL which are briefly presented in the following paragraphs.

### 1.2.5 Other experiments at LHC

The Large Hadron Collider forward (LHCf) [13, 14] experiment uses particles thrown forward by collisions in the Large Hadron Collider as a source to simulate cosmic rays in laboratory conditions. Cosmic rays are naturally occurring charged particles from outer space that constantly bombard the Earth's atmosphere; they collide with nuclei in the upper atmosphere, triggering a cascade of particles that reaches ground level.

When protons meet head-on at the Large Hadron Collider (LHC), the collisions provide a micro-laboratory to investigate many phenomena, including the protons themselves and their cross section. This is the physics that the TOTEM [15, 16] experiment is designed to explore, by taking precise measurements of protons as they emerge from collisions at small angles. This region is known as the 'forward' direction and is inaccessible by other LHC experiments. As CERN's 'longest' experiment, TOTEM detectors are spread across almost half a kilometre around

**Figure 1.8:** The LHCb detector.

the CMS interaction point.

In 2010 the Large Hadron Collider (LHC) approved its seventh experiment: the Monopole and Exotics Detector at the LHC (MOEDAL) [17]. The prime motivation of MOEDAL is to search directly for the magnetic monopole, a hypothetical particle with a magnetic charge. This detector has deployed around the same intersection region as the LHCb detector.

## 1.3 The CMS experiment

The CMS is a general-purpose detector at the LHC. It is designed to investigate a wide range of physics, including the search for the Higgs boson, extra dimensions and particles that could show the presence of dark matter. CMS is also designed to measure the properties of already discovered particles with unprecedented precision and be on the lookout for completely new, unpredicted phenomena. Although it has the same scientific goals as the ATLAS experiment, it uses different technical solutions and a different magnet-system. The CMS detector is built around a huge solenoidal magnet. This takes the form of a cylindrical coil of superconducting cable that generates a field of 4 T which is confined by a steel "yoke" that forms the bulk of the detector's 12,500-tonne weight. When protons will collide at their maximum designed energy ($\sqrt{s} = 14\text{TeV}$ ), 109 collisions/s will occur and therefore the online selection process has to trigger only 100 events/s to be saved.

CMS is built around a huge superconducting solenoid and different layers of detectors measure the different particles and use this key data to build up a picture of events at the heart of the collision. The detector is like a giant filter where each layer is designed to stop, track or measure a different type of particle emerging from proton-proton and heavy ion collisions; in finding the energy and momentum of a particle are obtained clues to its identity and particular patterns of particles or "signatures" are indications of new and exciting physics. The detector consists of layers of different material that exploit the properties of particles to catch and measure the energy and momentum of each one. In order to work correctly, CMS needs:

- a high quality central tracking system to give accurate momentum measurements;
- a high resolution method to detect and measure electrons and photons (through an electromagnetic calorimeter);
- a "hermetic" hadron calorimeter, designed to entirely surround the collision point and prevent particles from escaping;
- a high performance system to detect and measure muons.

Therefore the detector is made up of five concentric layers (see Figure 1.9): the tracker, the electromagnetic calorimeter (ECAL), the magnet, the hadronic calorimeter (HCAL), and the muon detector.

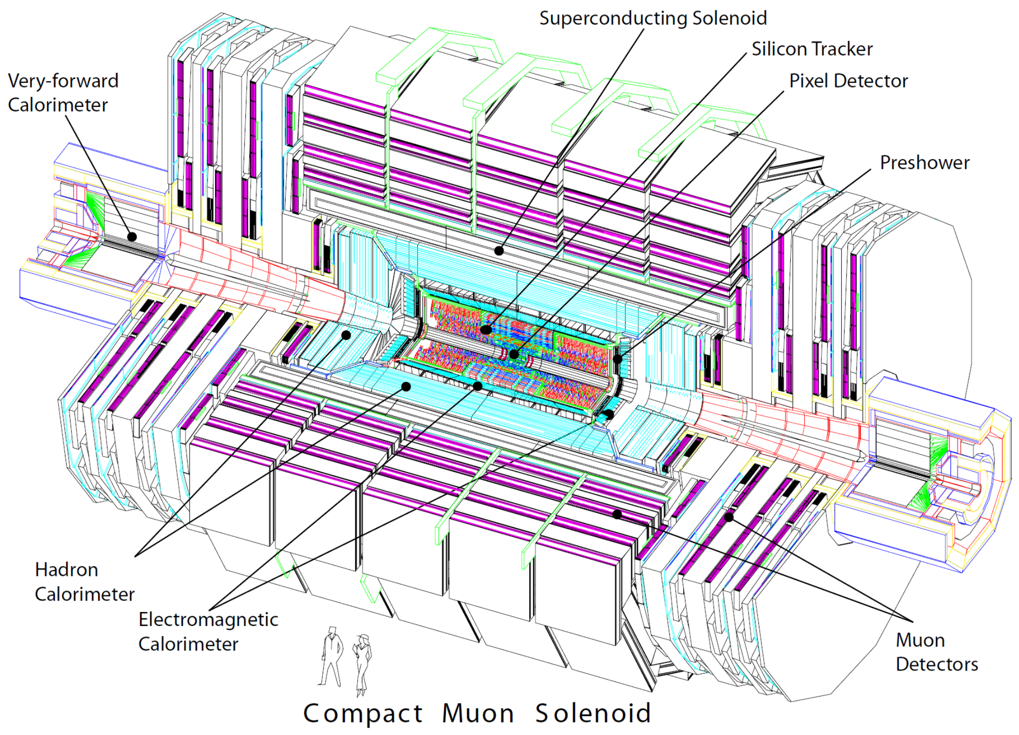


**Figure 1.9:** Section of the CMS detector.

Particles emerging from collisions first meet a tracker, made entirely of silicon, that charts their positions as they move through the detector, allowing scientists to measure their momentum; outside the tracker are calorimeters that measure the energy of particles. In measuring the momentum, the tracker should interact with the particles as little as possible, whereas the calorimeters are specifically designed to stop particles in their tracks. Besides muon tracks are measured by four layers of muon detectors, while the neutrinos escape from CMS undetected, although their presence can be indirectly inferred from the "missing transverse energy" in the event.

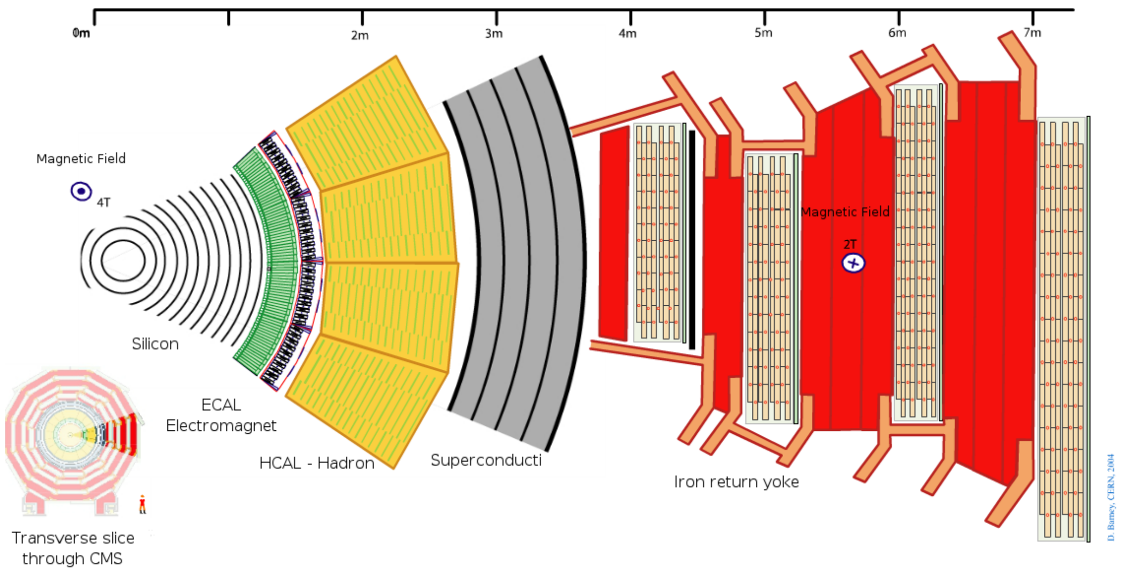


**Figure 1.10:** Section of the CMS detector.

### 1.3.1 The magnet

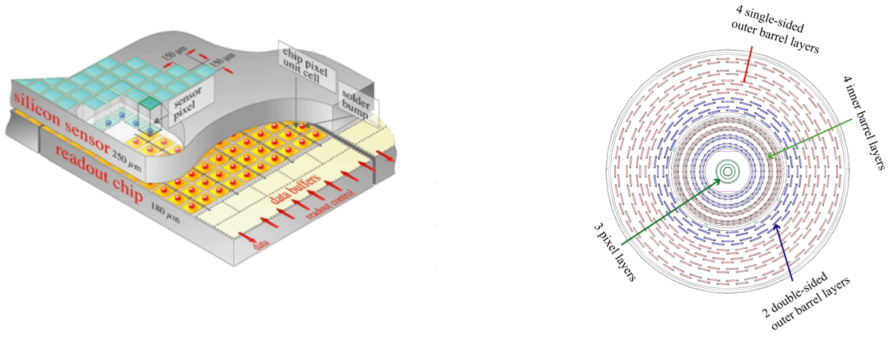
The CMS magnet is a solenoid, that is a coil of superconducting wire, and creates a magnetic field when electricity flows through it. In CMS the solenoid has an overall length of 13 m and a diameter of 7 m and creates a magnetic field of about 4 T. It is the largest magnet of its type ever constructed and allows the tracker and calorimeter detectors to be placed inside the coil, resulting in a detector that is overall "compact", compared to detectors of similar weight.

### 1.3.2 The tracker

The momentum of particles is crucial in helping us to build up a picture of events at the heart of the collision. One method to evaluate the momentum of a particle is to track its path through a magnetic field and the CMS tracker records the paths taken by charged particles by finding their positions at a established number of key points. The tracker can reconstruct the paths of high-energy muons, electrons and hadrons as well as it can see tracks coming from the decay of very short-lived particles such as b-quarks. Moreover, it needs to record particle paths accurately in order to disturb particles as little as possible. It does this by taking position measurements so accurate that tracks can be reliably reconstructed using just a few measurement points and each measurement is accurate to 10 µm. It is also the most inner layer of the detector and so receives the highest volume of particles: the construction materials were therefore carefully chosen to resist radiation. The final design consists of a tracker made entirely of silicon: internally, at the core of the detector, there are three pixels levels (pixel detector) and, after these, particles pass through ten microstrip detectors levels, up to reach a 130 cm ray from the beam pipe. As particles travel through the tracker the pixels and microstrips produce tiny electric signals that are amplified and detected.

The pixel detector (about the size of a shoebox) contains 65 million pixels, allowing to track the paths of particles emerging from the collision with extreme accuracy. It is also the closest detector to the beam pipe, with cylindrical layers at 4, 7 and 11 cm and disks at either end, and so it is crucial in reconstructing tracks of very short-lived particles. However, being so close to the collision means that the number of particles passing through is huge: indeed the rate at 8 cm from the beam line amounts to about 10 million particles per square centimetre per second. Despite this huge number of particles passing through, the pixel detector is able to disentangle and reconstruct all the tracks. When a charged particle passes through, it gives enough energy to electrons to be ejected from the silicon atoms, creating electron-hole pairs. Each pixel uses electric current to collect these charges on the surface as a small electric signal which is then amplified.

After the pixels and on their way out of the tracker, particles pass through ten layers of silicon strip detectors, reaching out to a radius of 130 centimetres; this part of the tracker contains 15,200 highly sensitive modules with a total of 10 million detector strips read by 80,000 microelectronic chips. Each module consists of three elements: a set of sensors, its mechanical support structure and readout electronics.



**(a)** CMS Silicon pixel detector

**(b)** Tracker layers seen by normal plane to beams

**Figure 1.11:** Schematic views of the CMS Tracker design.

## 1.3.3 ECAL

The Electromagnetic Calorimeter (ECAL) measures the energy of photons and electrons. In order to find them with the necessary precision in the very strict conditions of the LHC -high magnetic field, high levels of radiation and only 25 nanoseconds between collisions - requires very particular detector materials. The lead tungstate crystal is made primarily of metal and it is heavier than stainless steel, but with a touch of oxygen in this crystalline form it is highly transparent and “scintillates” when electrons and photons pass through it. This means the cystal produces light in proportion to the particle's energy. Photodetectors are glued onto the back of each of the crystals to detect the scintillation light and convert it to an electrical signal that is amplified and processed. The ECAL, made up of a barrel section and two ”endcaps”, forms a layer between the tracker and the HCAL. The cylindrical “barrel” consists of 61,200 crystals formed into 36 “supermodules”, each weighing around three tonnes and containing 1700 crystals. The flat ECAL endcaps seal off the barrel at either end and are made up of almost 15,000 further crystals. For extra spatial precision, the ECAL also contains Preshower detectors that sit in front of the endcaps. These allow CMS to distinguish between single high-energy photons (often signs of exciting physics) and the less interesting close pairs of low-energy photons.

### 1.3.4 HCAL

The Hadron Calorimeter (HCAL) measures the energy of “hadrons”, particles made of quarks and gluons (for example protons, neutrons, pions and kaons); additionally it provides indirect measurement of the presence of non-interacting, uncharged particles such as neutrinos.

Measuring these particles is important as they can tell us if new particles such as the Higgs boson or supersymmetric particles (much heavier versions of the standard particles we know) have been formed.

As these particles decay they may produce new particles that do not leave record of their presence in any part of the CMS detector and to spot these the HCAL must be “hermetic”, that is make sure it captures, to the extent possible, every particle emerging from the collisions. This way if we see particles shoot out one side of the detector, but not the other, with an imbalance in the momentum and energy (measured in the sideways “transverse” direction relative to the beam line), we can deduce that we're producing “invisible” particles.

The HCAL is a sampling calorimeter, meaning it finds a particle's position, energy and arrival time using alternating layers of “absorber” and fluorescent “scintillator” materials that produce a rapid light pulse when the particle passes through. Special optic fibres collect up this light and feed it into readout boxes where photodetectors amplify the signal. When the amount of light in a given region is summed up over many layers of tiles in depth, called a “tower”, this total amount of light is a measure of a particle's energy.

### 1.3.5 The muon detectors

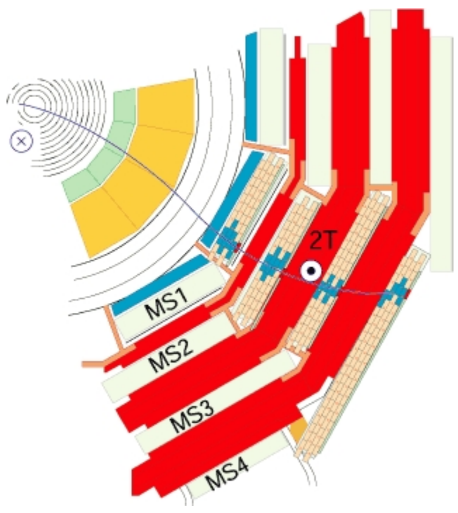
As the name “Compact Muon Solenoid” suggests, detecting muons is one of CMS's most important tasks. Muons are charged particles that are just like electrons and positrons, but are 200 times heavier; we expect them to be produced in the decay of a number of potential new particles: for instance, one of the clearest "signatures" of the Higgs boson is its decay into four muons.

Because muons can penetrate several metres of iron without interacting, unlike most particles they are not stopped by any of CMS calorimeters. Therefore, chambers to detect muons are placed at the very edge of the experiment where they are the only particles likely to register a signal.

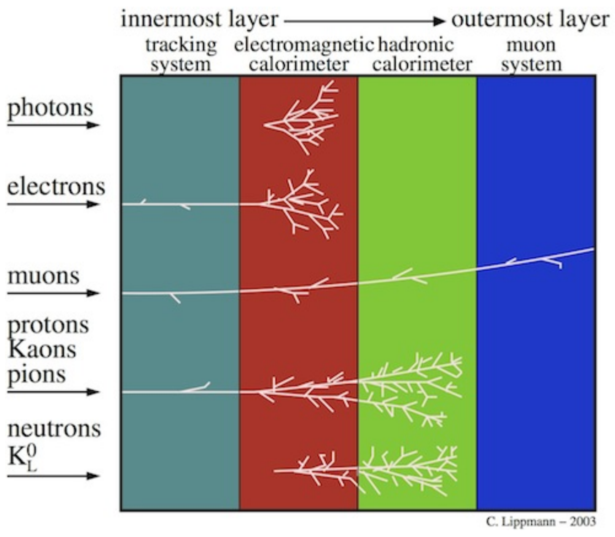
A particle is measured by fitting a curve to hits among the four muon stations, which sit outside the magnet coil and are interleaved with iron "return yoke" plates (shown in red in Figure 1.12a). By tracking its position through the multiple layers of each station, combined with tracker measurements, the detectors precisely trace a particle's path. This gives a measurement of its momentum: we know that particles travelling with more momentum bend less in a magnetic field. As a consequence, the CMS magnet is very powerful so we can bend even the paths of very high-energy muons and calculate their momenta.

In total there are 1400 muon chambers: 250 drift tubes (DTs) and 540 cathode strip chambers (CSCs) track the particles' positions and provide a trigger, while 610 resistive plate chambers (RPCs) form a redundant trigger system, which quickly decides to keep the acquired muon data or not. Because of the many layers of detector and different specialities of each type, the system is naturally robust and able to filter out background noise.

DTs and RPCs are arranged in concentric cylinders around the beam line ("the barrel region") whilst CSCs and RPCs, make up the "endcaps" disks that cover the ends of the barrel.



(a)



(b)

**Figure 1.12:** Pictorial view of a muon passing through the CMS muon detectors (a) and paths of different particles traversing different CMS subdetectors (b).

### 1.3.6 Data Acquisition and Trigger

When CMS is performing at its peak, about one billion proton-proton interactions will take place every second inside the detector. There is no way that data from all these events could be read out, and even if they could, most would be less likely to reveal new phenomena: they might be low-energy glancing collisions for instance, rather than energetic, head-on interactions.

We therefore need a "trigger" that can select the potentially interesting events (such as those which will produce a Higgs particle or a supersymmetric particle) and reduce the rate to just a few hundred “events” per second, which can be read out and stored on computer disk for subsequent analysis.

However, with groups of protons colliding 40 million times per second there are only ever 25 ns before the next lot arrive. The solution is to store the data in pipelines that can retain and process information from many interactions at the same time. To not confuse particles from two different events, the detectors must have very good time resolution and the signals from the millions of electronic channels must be synchronised so that they can all be identified as being from the same event.

There is a trigger system organized in two levels.

The first one is based on hardware (Level-1 (L1) Trigger), based on a data selection process extremely fast and completely automatic that selects data according to a physics interest, for example an high energy value or a unusual combination of interacting particles in a event. This kind of trigger acts asynchronously in the phase of signals reception and performs a first selection that reduce the frequency of the acquiring data to some hundreds of events per second. Then they are transferred and saved in storage spaces for following analysis on storage, and passed on to the next step.

The second level of trigger is based on software (High-Level Trigger (HLT)). It is a software system implemented in a filter farm of about one thousand commercial processors. The HLT has access to the complete read-out data and can therefore perform complex calculations similar to those made in the analysis off-line software, if required for specially interesting events. The HLT contains many trigger paths, each corresponding to a dedicated trigger (such as a single-electron trigger or a 3-jets-with-MET trigger). A path consists of several steps (software modules), each module performing a well-defined task such as the reconstruction of physics objects (electrons, muons, jets, MET, etc).

The rate reduction capability is designed to be at least a factor of $10^6$ for the combined L1 Trigger and HLT. Events frequency that exits from the second level of trigger is about 100 Hz.

# Chapter 2

# The CMS Computing Model

During LHC operation, the accelerator produces a huge amount of data that has to be stored and later analysed by scientists. The frequency of collisions *p*-*p* or of heavy ions in each detector is $10^9$ Hz that produce roughly 1 PB per second of data for each detector. As described in previous chapter, the flux of data is selected by a trigger system that reduces the data stack produced to some hundreds of MB (MegaBytes) per second. To deal with all this data, a complex computing infrastructure has been designed and deployed, characterized by computing centers distributed worldwide known as the Worldwide LHC Computing Grid.

Including data produced by physics simulations and by detectors, LHC Computing has to manage roughly 15 PB of data each year in operating conditions. In addition to this, each computing model at LHC has to guarantee the access and the possibility to run jobs on a computer center somewhere on the globe, even if the user isn't at CERN. Basic requirements necessary to build a computing system like this are:

- ability to store and manage original and derived data
- performant reconstruction code (to allow frequent re-reconstruction)
- Streamed Primary Datasets (to allow priority driven distribution and processing)
- distribution of Raw and Reconstructed data together (to allow easy access to raw detector information)
- compact data formats (to allow hosting multiple copies at multiple sites)
- consistent production reprocessing and bookkeeping

Additionally the *computing environment* has to manage also analysis process, in what is referred to *chaotic user analysis*

The software that allows users to access computers distributed across the network is called "middleware" because it sits between the operating systems of the computers and the physics-applications software that can solves a user's particular problem.

Each project acting on Grid must be equipped with a solid *Computing Model.* It includes the description of the whole hardware, software components that are developed to supply to the collection, distribution and analysis of the huge of data produced, the management and interaction of each components through a certain number of tools and services in real time.

Each HEP experiment, such as the LHC experiments (and therefore also CMS), uses a set of components so-called “middleware-layer", but each experiment also adds some software autonomously projected and developed to carry out functions of specific interest of that experiment: this software represents all that we call "application layer". Common solutions for more experiments, and also for the single experiment, also have to operate in coherent way on resources that coexist in computing worldwide centers.

## 2.1 WLCG and the CMS Computing Tiers

The Worldwide LHC Computing Grid (WLCG) project [18, 19] is a global collaboration for building and maintaining a data storage and analysis infrastructure required by the experiments at the LHC. The main purpose of this infrastructure is to provide computing resources to store, distribute and analyse the data produced by LHC to all the users of the collaboration regardless of where they might be. This idea of a shared computing infrastructure is at the basis of the concept of the Grid. The WLCG cooperates with several Grid projects such as the European Grid Infrastructure (EGI) [20] and Open Science Grid (OSG) [21]. As mentioned in the previous section, the middleware projects provide the software layers on top from which the experiments add their own (different) application layer. At the middleware layer, the main building blocks that make up the infrastructure are the logical elements of a Grid site, namely:

- the Computing Element (CE) service, that manages the user’s requests for computational power at a Grid site;
- the Worker Node (WN), where the computation actually happens on a site farm;
- the Storage Element (SE), that gives access to storage and data at a site. Data are stored on tapes and disks: tapes are used as long-term secure storage media whereas disks are used for quick data access for analysis;
- the User Interface (UI), the machine on which users interact with the Grid;
- central services, that help users access computing resources. Some examples are data catalogues, information systems, workload management systems, and data transfer solutions.

WLCG provides access to computing resources which include data storage capacity, processing power, sensors, visualization tools and more. Users make job requests from one of the many entry points into the system. A job request can include anything: storage, processing capacity, or availability of analysis software, for example. The computing Grid establishes the identity of the user, checks their credentials and searches for available sites that can provide the resources requested.

Users do not have to worry about where the computing resources are coming from, they can tap into the Grid computing power and access storage on demand.

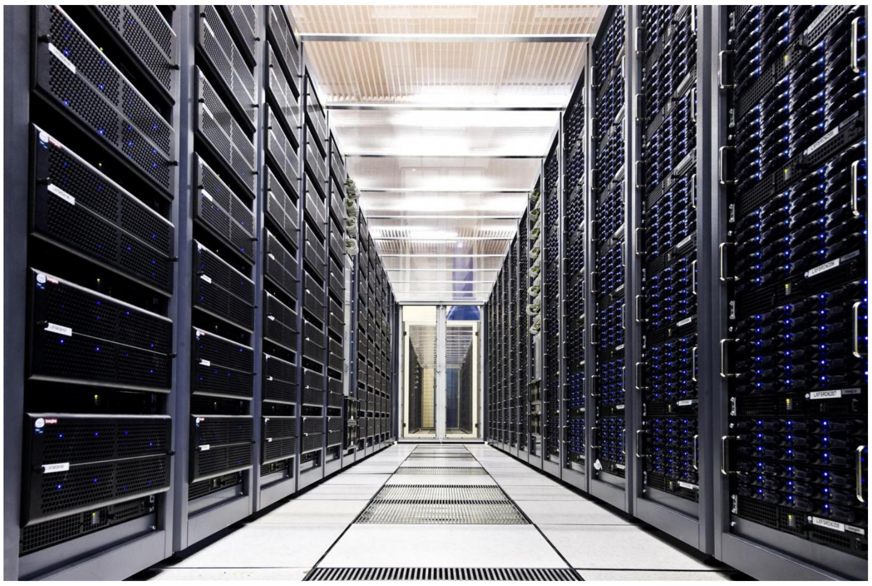


**Figure 2.1:** Servers at the CERN Data Centre from Tier 0 of the Worldwide LHC Computing Grid.

The Worldwide LHC Computing Grid is composed by four levels, or “Tiers”, called 0, 1, 2 and 3. Each Tier is made up of several computer centres and provides a specific set of services. Between them the tiers process, store and analyse all the data from the Large Hadron Collider. The Tier is a computing center that provides storage capacity, CPU power and network connectivity. The structure made by Tiers was formalized by the MONARC model, that foresees a rigid hierarchy. The number associated to the Tier means that, minor is the number and more are services and functionality offered, hence a bigger site in terms of storage, CPU, network and of the so-called *availability* required. A computer center that take part to WLCG for one or more LHC experiments could theoretically cover different roles and perform different functions for each experiments: indeed for instance a Tier could have the function of Tier 1 for Alice and of Tier 2 for CMS.

### 2.1.1 Tier-0 and CMS-CAF

Tier-0 and CMS CERN Analysis Facility (CAF) are located at CERN. From 2012 Tier-0 was extended linking it with Wigner Research Centre fo Physics in Budapest, that works remotely and ensure more *availability* allowing T-0 to be operative also in case of problems in main seat. The role of Tier-0 is to receive RAW data from detectors and store them into tapes, collecting them in data fluxes and starting a first quick rebuilding of events. All of the data from the LHC passes through this central hub, but it provides less than 20 of the Grid’s total computing capacity. Tier 0 distributes the raw data and the reconstructed output to Tier-1s, and reprocesses data when the LHC is not running. The T-0 of CMS classifies rebuilt data (collected in RECO) in additional 50 primary datasets and it make them available to be transferred to T-1s. The mechanism through which *integrity*

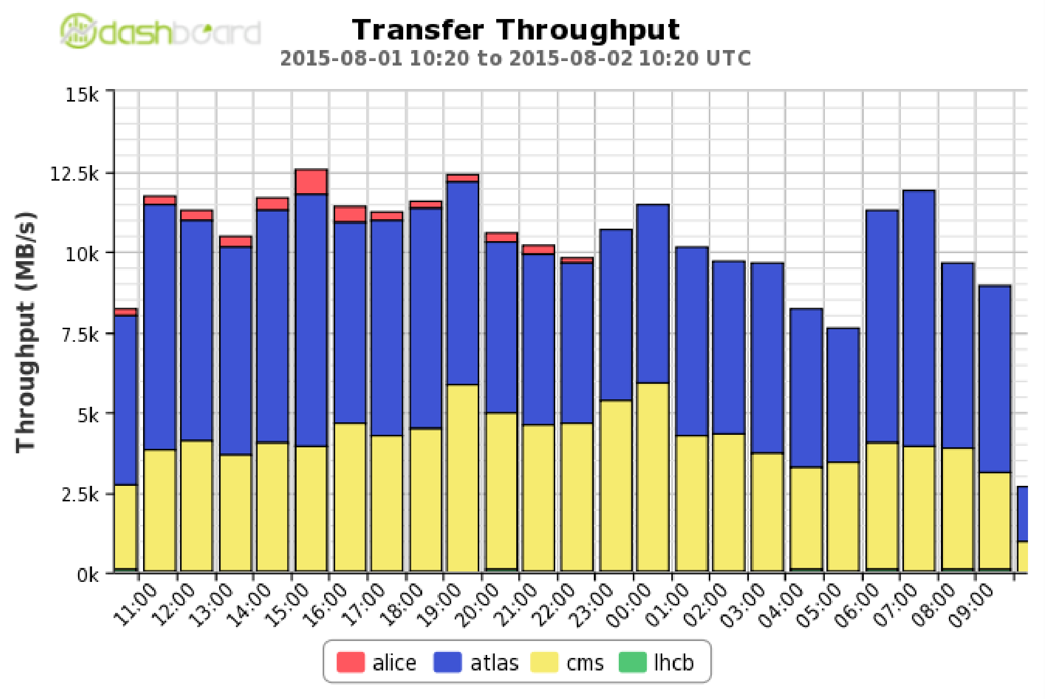


**Figure 2.2:** Transfer throughput, or transmission capacity actually used of a communication channel.

and *availability* are applied provide that data are stored in secure way in two copies: one is saved at CERN (*cold copy*) and one is distribute to Tier-1s (*hot copy*). The task to send copies to Tier-1s is always of T-0: to this aim the transfer capacity of data is fundamental and it was created an infrastructure of a dedicated fiber-optic network that allowsto reach the rate of 120 Gbps with Tier-1s.

CMS-CAF has the aim to provide support to all activities that require a very short time latency and a very quick asynchronous access to RAW data from T-0, such as the diagnostic detector, services related to performance management of trigger or calibrations.

### 2.1.2 Tier-1s

Tier-1 consists of 13 computer centres in different part of the world (Italy, Germany, France, Spain, USA, Russia, England and others) large enough to store LHC data. They are responsible for storing data and their reprocessing. Their main role is to make available physical space where it is possible to store the "hot copies" of real and simulated data of CMS, in order to accede it quickly. Besides it is necessary that they have a relevant computing power and a quick cache disk. This allows that data can be reprocessed at each recalibration, selected in a way to choose which of them are more useful to physic analysis and quickly transferred to T-2s. Another aim of Tier-1s is to store simulated data that the Tier-2s produce. Optical-fibre links working at 10 gigabits per second connect CERN to each of the 13 Tier-1 centres around the world.

### 2.1.3 Tier-2s and Tier-3s

Tier 2s are typically universities and other scientific institutes that can store sufficient data and provide adequate computing power for specific analysis tasks. They handle a proportional share of the production and reconstruction of simulated events. There are around 155 Tier 2 sites around the world. Individual scientists can access the Grid through local (or Tier 3) computing resources, which can consist of local clusters in a university department or even an individual PC; there is no formal engagement between WLCG and Tier 3 resources.

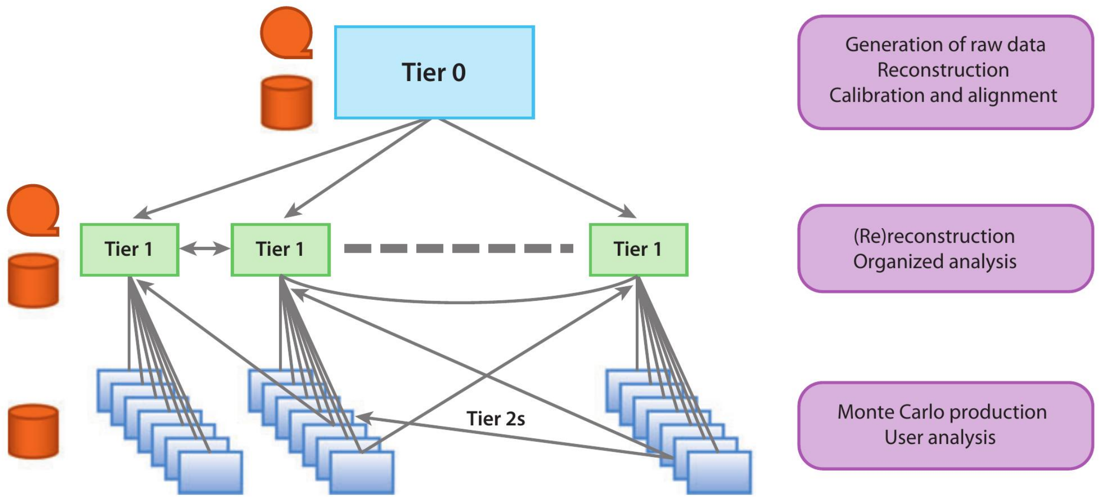


**Figure 2.3:** Structure with Tiers of CMS computing model.

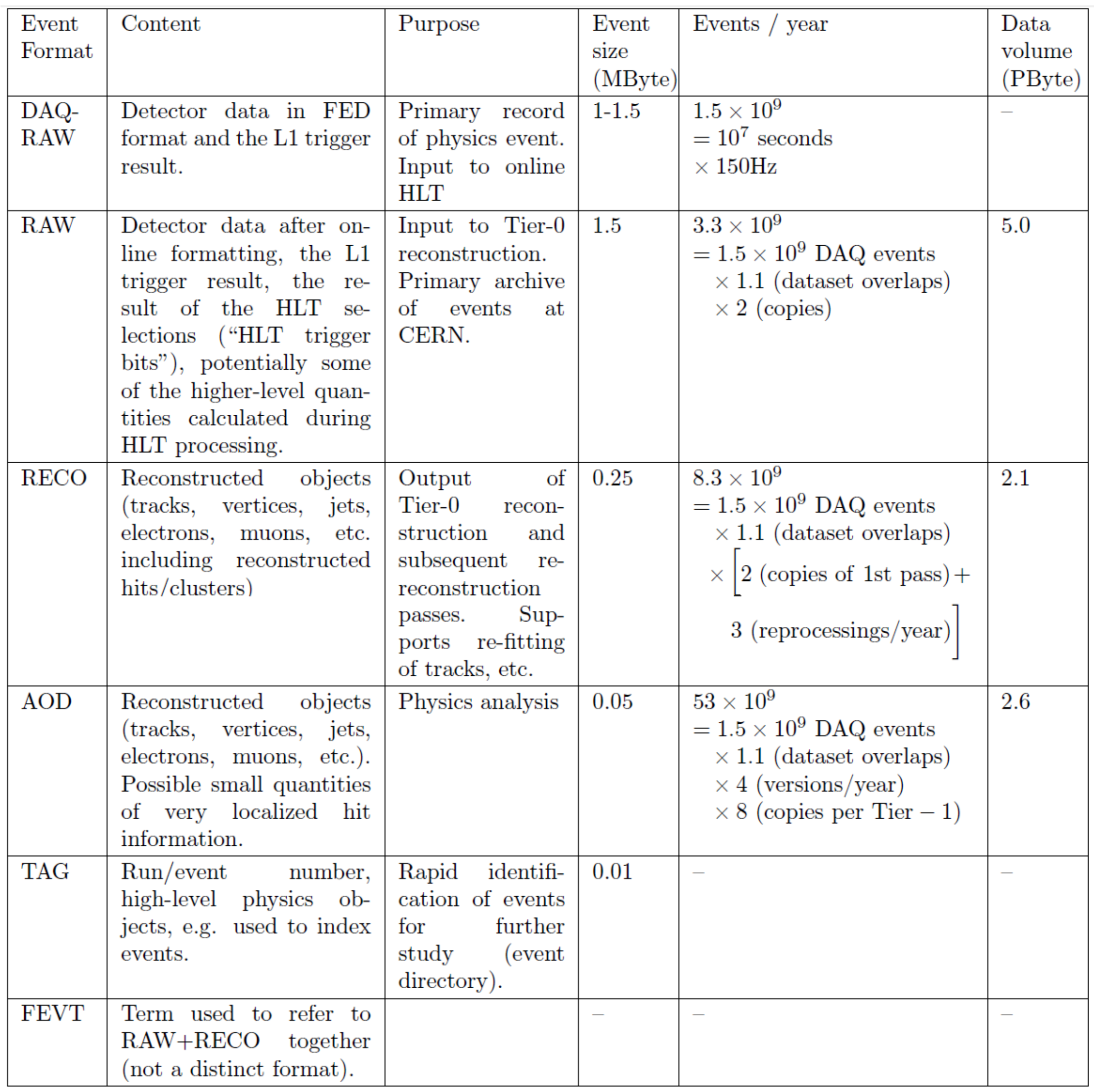
## 2.2 The CMS data and simulation model

CMS data is arranged into a hierarchy of data tiers. Each physics event is written into each data tier, where each tier contains different levels of information about the event. The three main data tiers written in CMS are:

- **RAW**: full event informations from the Tier-0, containing 'raw' detector informations (detector particles hits, etc). RAW data are not directly used for analysis.

- **RECO** ("RECOnstructed data"): the output from first-pass processing by the Tier-0. This layer contains reconstructed physics objects, but it's still very detailed. RECO data can be used for analysis, but are too big for frequent use when CMS has collected a substantial data sample.

- **AOD** ("Analysis Object Data"): this is a "distilled" version of the RECO event information, and is expected to be used for most analyses. AOD provides a trade-off between event size and complexity of the available information in order to optimize flexibility and speed for analyses.



| Event Format | Content | Purpose | Event size (MByte) | Events / year | Data volume (PByte) |
|---|---|---|---|---|---|
| DAQ-RAW | Detector data in FED format and the L1 trigger result. | Primary record of physics event. Input to online HLT | 1-1.5 | $1.5 \times 10^9$ $= 10^7$ seconds $\times$ 150Hz | – |
| RAW | Detector data after online formatting, the L1 trigger result, the result of the HLT selections ("HLT trigger bits"), potentially some of the higher-level quantities calculated during HLT processing. | Input to Tier-0 reconstruction. Primary archive of events at CERN. | 1.5 | $3.3 \times 10^9$ $= 1.5 \times 10^9$ DAQ events $\times$ 1.1 (dataset overlaps) $\times$ 2 (copies) | 5.0 |
| RECO | Reconstructed objects (tracks, vertices, jets, electrons, muons, etc. including reconstructed hits/clusters) | Output of Tier-0 reconstruction and subsequent re-reconstruction passes. Supports re-fitting of tracks, etc. | 0.25 | $8.3 \times 10^9$ $= 1.5 \times 10^9$ DAQ events $\times$ 1.1 (dataset overlaps) $\times \left[2 \text{ (copies of 1st pass)} + 3 \text{ (reprocessings/year)}\right]$ | 2.1 |
| AOD | Reconstructed objects (tracks, vertices, jets, electrons, muons, etc.). Possible small quantities of very localized hit information. | Physics analysis | 0.05 | $53 \times 10^9$ $= 1.5 \times 10^9$ DAQ events $\times$ 1.1 (dataset overlaps) $\times$ 4 (versions/year) $\times$ 8 (copies per Tier $-$ 1) | 2.6 |
| TAG | Run/event number, high-level physics objects, e.g. used to index events. | Rapid identification of events for further study (event directory). | 0.01 | – | – |
| FEVT | Term used to refer to RAW+RECO together (not a distinct format). | | – | – | – |

**Figure 2.4:** CMS data tiers and their characteristics.

The workflow of named "data reconstruction" in CMS consists in the passage from informations contained in RAW data, through subsequent reprocessing stages, up to formats containing objects of interest for physics analysis.

The workflow of named "simulation reconstruction", foresees that a first step is executed ("kinematics") based on several Monte Carlo generator events, following by a second step ("simulation") that simulates the detector answer when generated interactions occur, and finally there is a third step ("reconstruction") where, to simulate a real bunch crossing, the single interaction is combined with pile-up events and then rebuilt. In the end of this phase, the CMS data model expected that reference formats are called AOD and AODSIM respectively.

Therefore CMS uses a number of event data formats with varying degrees of detail, size, and refinement. Starting from RAW data produced from the online system, successive degrees of processing refine this data, apply calibrations and create higher level physics objects.

The table in Figure 2.4 describes the various CMS event formats. It is important to note that this table corresponds to the LHC startup period and assumes a canonical luminosity of $L = 2 \cdot 10^{33} cm^{-2} s^{-1}$. The determinations of the data volume include the effects of re-processing steps with updated calibrations and software and the copying of data for security and performance reasons.

### 2.2.1 CMS data organization

To extract a physics information useful for a high energy physics analysis, a physicist has to combine a variety of blocks:

- reconstructed information from the recorded detector data, specified by a combination of trigger paths and possibly further selected by cuts on reconstructed quantities,
- Monte Carlo samples which simulate the physics signal under investigation,
- background samples (specified by the simulated physics process).

These informations are stored into datasets and event collections. Dataset constitutes a set of structured data in related form, or a matrix made by data in which each column represents a variable and each row represents a member of the dataset. Their dimension may vary into range 1-100 TB. Datasets are split off at the T-0 and distributed to the T-1s. An event collection is the smallest unit within a dataset that a user can select. Typically, the reconstructed information needed for the analysis would all be contained in one or a few event collection(s). The expectation is that the majority of analyses should be able to be performed on a single primary dataset. Data are stored as ROOT files. The smallest unit in computing space is the file block which corresponds to a group of ROOT files likely to be accessed together. This requires a mapping from the physics abstraction of the event to the file location. CMS has a global data catalogue called the Dataset Bookkeeping System (DBS) [22] which provides mapping between the physics abstraction (dataset or event collection) and the list of fileblocks corresponding to this abstraction. The locations of these fileblocks within the CMS grid (several centers can provide access to the same fileblock) are resolved by the PhEDEx (the Physics Experiment Data EXport service) [23, 24]. PhEDEx is responsible for transporting data around the CMS sites, and keeps track of which data exists at which site.

### 2.2.2 Workflows in CMS Computing

A workflow can be described simply as "what we do to the data". There are three principle places where workflows are executed in CMS:

- At Tier-2 Centres: Monte Carlo events are generated, detector interactions simulated.
- At the Tier-0 Center: data is received from the CMS detector experiment and it is "repacked" (i.e. events from the unsorted online streams are sorted into physics streams of events with similar characteristics). Reconstruction

algorithms are run, AOD is produced, and RAW, RECO and AOD are exported to Tier-1 sites.

- Half of the T-2 resources (CPU and disk storage) is devoted to Monte Carlo simulation and half to distributed analysis. Network connection with T-1s and among the T-2s is vital to data exchange. This is where most of the user activities happen. The user: prepares analysis code, sends code to site where there is appropriate data, then run your code on the data and collect the results. The process of finding the sites with data and CPU, running the jobs, and collecting the results is all managed for you (via the grid) by CRAB.

The management of grid jobs is handled by a series of systems. The aim is to schedule jobs onto resources according to the policy and priorities of CMS, to assist in monitoring the status of those jobs, and to guarantee that site-local services can be accurately discovered by the application once it starts executing in a batch at the site. These issues should be invisible to the user.

The essential elements of the flow of real physics data through the hardware tiers are:

- T0 to T1:
  - scheduled, time-critical, continuous during data-taking periods
  - reliable transfer needed for fast access to new data
- T1 to T1:
  - redistributing data, generally after reprocessing (e.g. processing with improved algorithms)
- T1 to T2:
  - Data for analysis at Tier-2s

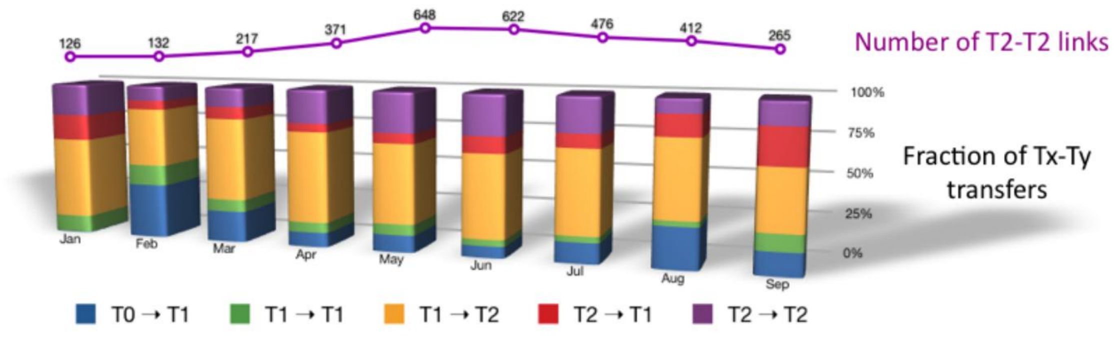


**Figure 2.5:** Fraction of data traffic in different T2-T2 routes (example from 2010).

Monte Carlo generated data is typically produced at a T-2 center, and stored at its associated T-1 and made available to the whole CMS collaboration (Figure 2.5).

For completeness, it is worth adding that these are the main data flows as from the original CMS computing model. As from Run 1 and Long Shutdown 1, the model has been evolved and expanded to support also other relevant data flows (e.g. among T-2s). An example is shown in Figure 2.5. This level of details in the model evolution is anyway only briefly cited here, and not deeply discussed in this thesis.

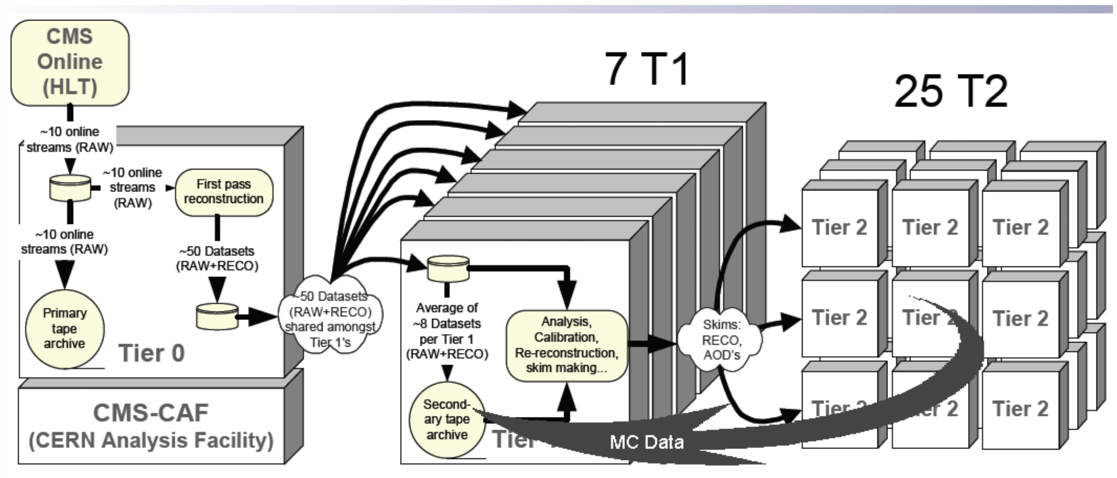

CMS Online (HLT)
~10 online streams (RAW)
~10 online streams (RAW)
First pass reconstruction
~10 online streams (RAW)
~50 Datasets (RAW+RECO)
Primary tape archive
Tier 0
~50 Datasets (RAW+RECO) shared amongst Tier 1's
CMS-CAF (CERN Analysis Facility)
7 T1
Average of ~8 Datasets per Tier 1 (RAW+RECO)
Analysis, Calibration, Re-reconstruction, skim making...
Secondary tape archive
MC Data
Skims: RECO, AOD's
25 T2
Tier 2

**Figure 2.6:** Graphic representation of CMS dataflows (as from the Computing Model effective in Run-1).

### 2.2.3 CMS data location

CMS does not use a central catalogue to store informations about location of each file in each site. Each file is biunivocally linked to a *Logical File Name*, that is the name used to refer to the physical file in all statements [25]. It is possible to know the existence and the position of a file through a *mapping* of this LFN with sites in which exists, carried out and stored by *central services* of Grid. Each site that contains replies has inside a local catalogue, the *Trivial File Catalog* (TFC), that in turn contains a local mapping of the LFN through the *Physical File Name* (PFN). The advantages linked to this implementation and managing of data are due to the freedom given to the several sites regard to the manage of their storage: indeed it is not necessary to contact the *central services* of Grid for internal operations at the site.

## 2.3 CMS services and operations

### 2.3.1 Grid computing

The integration of the resources at CMS Computing Centres into a single coherent system relies upon Grid middleware which presents a standardised interface to storage and CPU facilities at each WLCG site. The Grid allows remote job submission and data access with security. A number of CMS-specific distributed computing services operate above the generic Grid layer, allowing higher-level data and workload management functions. These services require CMS-specific software agents to run at some sites, in addition to generic Grid services. CMS also provides specialised user-intelligible interfaces to the Grid for job submission, and tools to monitor a large-scale data production and processing. An overview of the CMS Computing Services components is shown in Figure 2.7.

The CMS Computing Model executes a pattern in which jobs move towards data and it is possible to identify two distinct systems, even if interacting: the *CMS Data Management System*, to which is given the data managing task, and the *CMS Workload Management System*, that has the aim to manage jobs flux.

### 2.3.2 Data management

CMS requires tools to catalogue the data which exist, to track the location of the corresponding physical data files on site storage systems, and to manage and monitor the flow of data between sites. In order to simplify the data management problem, the data management system therefore defines higher-level concepts including: *dataset*, a logical collection of data grouped by physical-meaningful criteria; *event collection*, roughly corresponding to an experiment "run" for a given dataset definition. To provide the connection between abstract datasets and physical files, a multi-tiered catalogue system is used. The Dataset Bookkeeping System provides a standardised and queryable means of cataloguing and describing event data. It is the principle means of data discovery for the user, answering the question "which data of this type exists in the system?" A second catalogue system, the Data Location Service, provides the mapping between file blocks to the particular sites at which they are located, including the possibility of replicas at multiple sites. Local File Catalogues at each site map logical files onto physical files in local storage. The data transfer and placement system is responsible for the physical movement of fileblocks between sites on demand; it is currently implemented by the PhEDEx system. This system must schedule, monitor and verify the movement of data in conjunction with the storage interfaces at CMS sites, ensuring optimal use of the available bandwidth.

### 2.3.3 Workload management

Processing and analysis of data at sites is typically performed by submission of jobs to a remote site via the Grid workload management system. These jobs are based on the CMS *software framework* (CMSSW) and the output of such jobs is stored by the *Data Management System.* A standard job performs the necessary setup, executes a CMSSW application upon data are present on local storage at the site, arranges for any produced data to be made accessible via Grid data management tools, and provides logging information. This process is supported by several CMS-specific services. A parameter set management system, implemented with either global or local scope according to the application, allows the storage and tracking of the configuration of CMSSW applications submitted to the Grid. A job bookkeeping and monitoring system allows users to track, monitor, and retrieve output from jobs currently submitted to and executing at remote sites.

Some main characteristics of the WMS of CMS are: the reading access optimization, the minimization of job dependence from the Worker Nodes (WN) of

Grid, the job distribution according to the policy that they have to satisfy and to the priority inside the Virtual Organization.

### 2.3.4 Distributed analysis

For a generic CMS physicist is available a dedicated tool (CRAB) for workflow management. It allows to submit user-specific jobs to a remote computing element which can access data previously transferred to a close storage element. CRAB takes care of interfacing with the user environment, it provides data-discovery and data-location services, and also Grid infrastructure. It also manages the status reporting, monitoring, and user job output which can be put on a user-selected storage element. Via a simple configuration file, a physicist can thus access data available on remote sites as easily as he can access local data: all infrastructure complexities are hidden to him as much as possible. There is also a client-server architecture available, so the job is not directly submitted to the Grid but to a dedicated CRAB server, which, in turn, handles the job on behalf of the user, interacting with the Grid services.

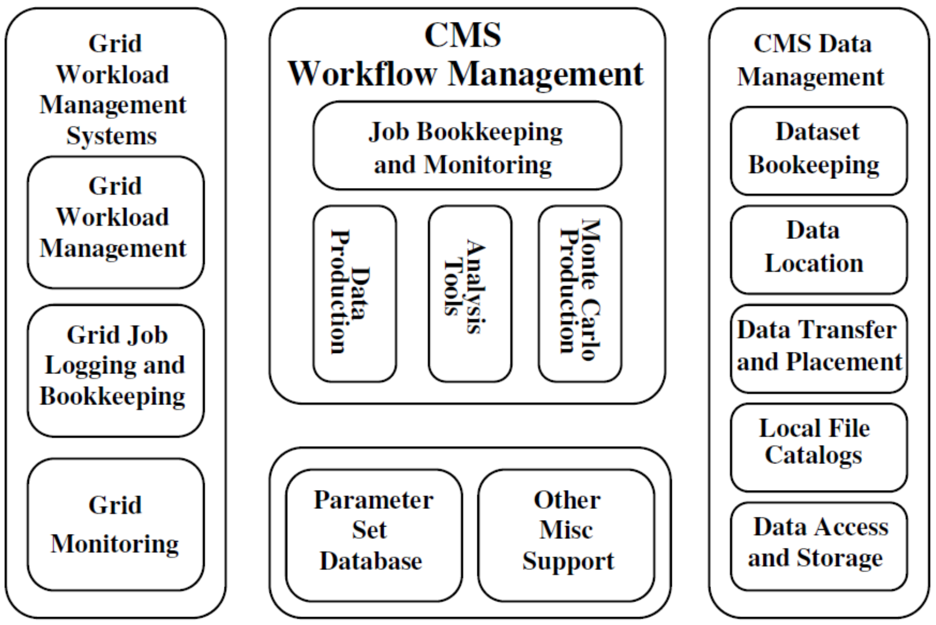
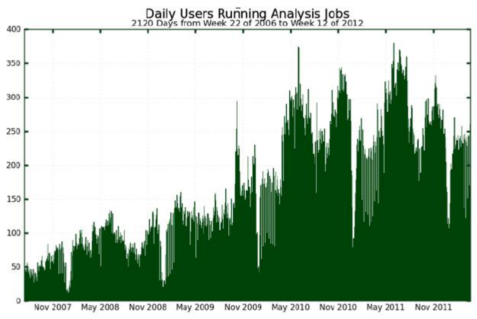


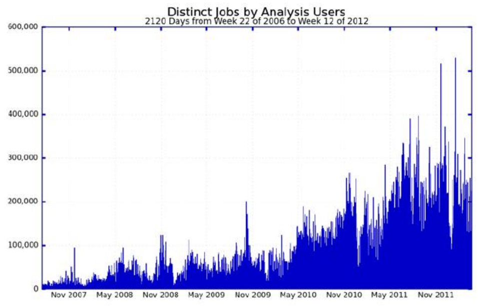
**Figure 2.7:** Overview of the CMS Computing Services.

## 2.4 A crucial metric: the CMS data popularity

One of the requirements for an experiment as CMS that does a large resources use, is to create computing models based on solutions that permits to optimize the net usage and the available storage. As shown in Figure 2.8, hundreds of users submit everyday about 200000 jobs, providing to the Grid a workload that executes jobs of the order of ten million. At the same time, the resources managing regards tens of PB (PetaByte) of data, quantity that will increase in Run-2.



**(a)** Daily users on Grid



**(b)** Analysis jobs per users

**Figure 2.8:** Two example of quantities monitored by the CMS Dashboard, related to the CMS distributed analysis on Grid.

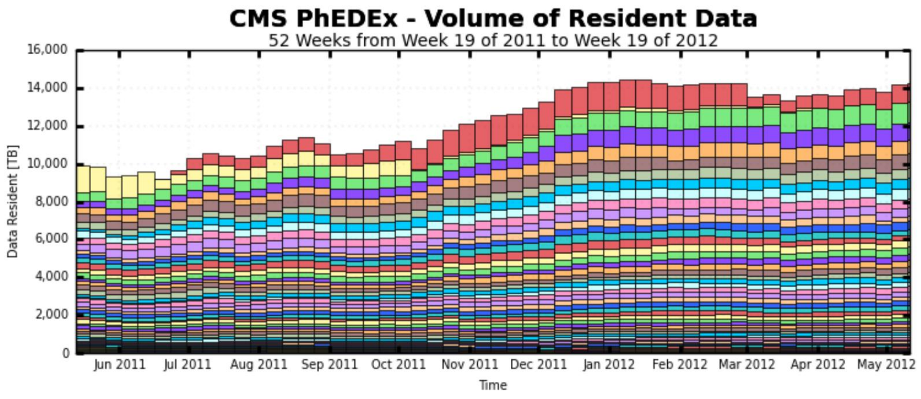


**Figure 2.9:** Volume of data resident on CMS Tiers in 2012 (from PhEDEx).

Regarding the scale of the CMS Grid infrastructure, the control and the optimization of storage resources through automated procedures is a complex task. Currently, CMS does not have a system that can localize worldwide inefficiencies, in terms of allocated and unused space. This causes in computing operations of computing model a progressive reduction of the actually available storage space, of difficult identification. For this and with the target to reach a dynamic data placement which optimizes resources allocation, the CMS popularity service was developed, originally inspired by a similar experience of the ATLAS experiment. In this way is introduced in CMS the concept of the "data popularity", that is an observable able to quantify the interest of the analysis community for data samples or Monte Carlo simulations, on the base of number of access, local or remote, successful or failed, to files by users' job. The CMS Popularity Service aim is to monitor which data are more used, measuring their popularity in time, so tracking time evolution of:

- dataset name
- number of access
- outcome of reading access (success or failure)
- CPU hours spent in accessing
- number of unique users that execute the access

The wotk on CMS data popularity is currently at a stage where such information can be used to trigger ad-hoc cancellation of unaccessed replicas and trigger ad-hoc replication of frequently access datasets. This system is in production since almost 1 year. Despite extremely useful for CMS Computing Operations, a substantial limitation of the system is that its intelligence is static, i.e. relies on a fixed algorithm that does not learn for experience and does not evolve/improve/auto-tune over time. The work presented in the following aims to demonstrate the effectiveness of a prototype that could equip CMS with a simple data popularity prediction system, whose impact on the resource utilization optimization is potentially very large.

# Chapter 3

# Machine Learning

## 3.1 Introduction

Machine Learning [26] is one of the most exciting recent technologies. People probably use a learning algorithm dozens of times a day without knowing it. Every time we use a web search engine like Google or Bing, one of the reasons that works so well is because a learning algorithm, one implemented by Google or Microsoft, has learned how to rank web pages. Every time we use Facebook or Apple's photo typing application and it recognizes our friends' photos, that is also machine learning. Every time we read our email and our spam filter saves you from having to wade through tons of spam email, that is also a learning algorithm. The dream is that someday somebody will build machines as intelligent as us. We are a long way away from that goal, but many researchers believe that the best way towards that goal is through learning algorithms that try to mimic how the human brain learns.

There is autonomous robotics, computational biology, tons of disciplines that machine learning is having an impact on. There are some other examples of machine learning.

- **Database mining** is one of the reasons machine learning has so pervaded is the growth of the web and the growth of automation. All this means that we have much larger data sets than ever before. So, for example tons of Silicon Valley companies are today collecting web click data, also called *clickstream data*, and are trying to use machine learning algorithms to mine this data to understand the users better and then to serve them better. With the advent of automation, we now have electronic medical records, so if we can turn medical records into medical knowledge, then we can start to understand disease better. With automation again, biologists are collecting lots of data about gene sequences, DNA sequences, and so on, and machines running algorithms are giving us a much better understanding of the human genome, and what it means to be human. And in engineering as well, in all fields of engineering, we have larger and larger data sets that we are trying to understand using learning algorithms.

- A second range of **machinery applications** is ones that **we cannot program by hand**. So for example, some researchers worked on autonomous helicopters for many years. They just did not know how to write a computer program to make this helicopter fly by itself. The only thing that worked was having a computer learn by itself how to fly this helicopter (helicopter whirling).

- Learning algorithms are also widely used for **self-customizing programs**. Every time you go to Amazon or Netflix or iTunes Genius, and it recommends the movies or products and music to you, that is a learning algorithm. If you think about it they have million users; there is no way to write a million different programs for your million users. The only way to have software give these customized recommendations is to learn by itself to customize itself to users' preferences.

- Finally learning algorithms are being used today to **understand human learning and to understand the brain**. Researches are using these to make progress towards the big AI (Artificial Intelligence) dream.

Nowadays, as we have seen, the adoption of Machine Learning techniques is speeding up due to a growing number of use cases in Big Data Analytics and Data Science domains.

## 3.2 What is Machine Learning?

Several specialists gave correct and valuable definition of what Machine Learning actually consists of. One of the most adequate (and operational) definition comes from Professor Tom Mitchell, from Carnegie Mellon University:

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*

Following this sentence, the key that connects past behaviour to future behaviour is the experience "E". Focusing on a task "T" (e.g. the car traffic pattern at an intersection), the experience "E" contains the data about all the traffic patterns that happened in the past: this is fed to a Machine Learning algorithm, which indeed learn about "T" from "E" and it will improve – as measured by "P", the metric chosen as performance measure - in predicting future traffic patterns.

In this sense, Machine Learning can hence solve problems that cannot be solved by numerical means alone. Another way of stating the same is that Machine Learning is the discipline that gives computers the ability to learn without being explicitly programmed.

There are several different types of learning algorithms. The main two types are what we call supervised learning and unsupervised learning. In supervised learning the idea is that we are going to teach the computer how to do something, whereas in unsupervised learning we are going to let it learn by itself. In the following section will be described in general way this two different kind of learnings.

## 3.3 Supervised Learning

In order to a better understanding of supervised learning, we give an example [27]. Suppose that a student collects data sets about American house price and wants to plot it. In Figure 3.1 on the horizontal axis, there is the size of different houses in square feet, and on the vertical axis, the price of different houses in thousands of dollars.

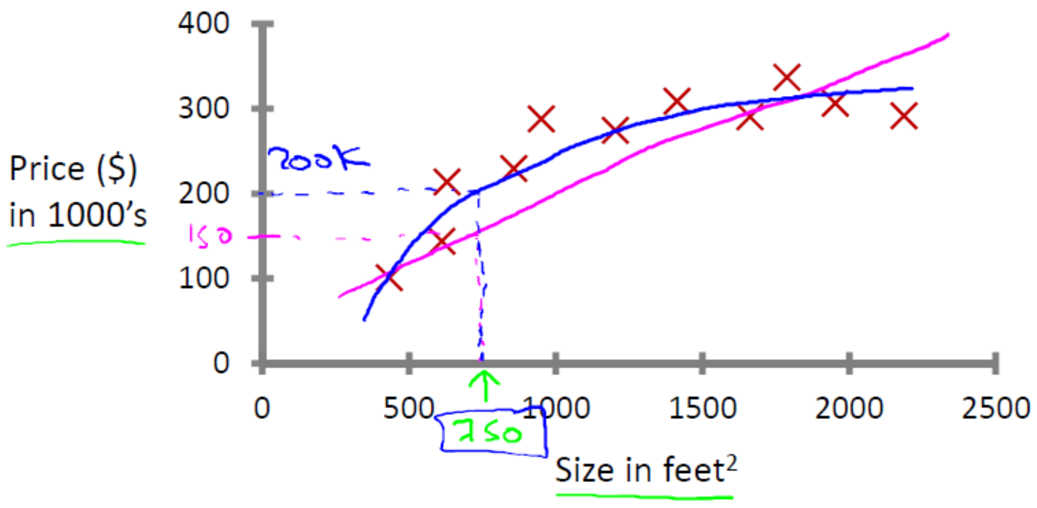


**Figure 3.1:** First example of supervised learning as discussed in the text [27].

Given this data, let's say you have a friend who owns a house for example of 750 square feet and he wants to know how much he can get for the house. So how can the learning algorithm help us? One thing a learning algorithm might be able to do is put a straight line through the data or to fit a straight line to the data and, based on that, it looks like maybe the house can be sold for maybe about $150,000. But maybe this is not the only learning algorithm we can use. There might be a better one. For example, instead of sending a straight line to the data, we might decide that it is better to fit a quadratic function or a second-order polynomial to this data. And if we do that, and make a prediction here, maybe we can sell the house for closer to $200,000. Each of these two different possibilities would be a fine example of a learning algorithm. So this is an example of a supervised learning algorithm. And the term supervised learning refers to the fact that we gave the algorithm a data set in which the "right answers" were given. That is, we gave it a data set of houses in which for every example in this data set, we told it what is the right price so what is the actual price that, that house sold for and the toss of the algorithm was to just produce more of these right answers for new houses. To define with a bit more terminology this is also called a regression problem and for regression problem we mean we are trying to predict a continuous value output namely the price. So technically we guess prices can be rounded off to the nearest cent. So prices are actually discrete values, but usually we think of the price of a house as a real number, as a scalar value, as a continuous value number and the term regression refers to the fact that we are trying to predict the sort of continuous values attribute.

There is also another supervised learning example. Imagine that we want to look at medical records and try to predict of a breast cancer as malignant or benign. Let's see a collected data set and suppose in your data set you have on your horizontal axis the size of the tumor and on the vertical axis we put one or zero, yes or no, whether or not these are examples of tumors we have seen before are malignant–which is one–or zero if not malignant or benign.

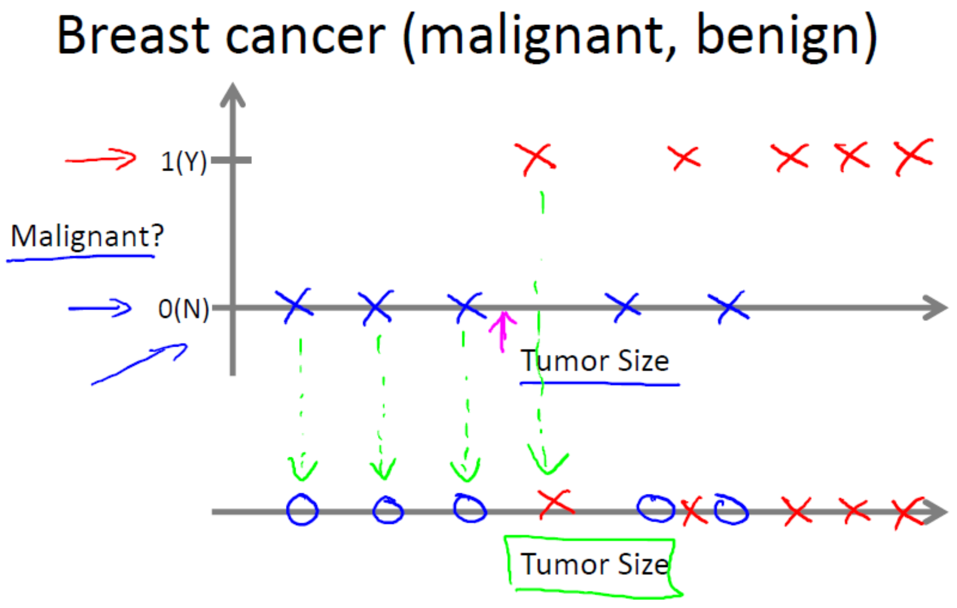


**Figure 3.2:** Second example of supervised learning as discussed in the text [27].

In Figure 3.2 we have five examples of benign tumors shown down, and five examples of malignant tumors shown with a vertical axis value of one.

And let's say we have a friend who tragically has a breast tumor, and we know her breast tumor size. The machine learning question is, can you estimate what is the probability, what is the chance that a tumor is malignant versus benign? To introduce a bit more terminology this is an example of a classification problem. The term classification refers to the fact that here we are trying to predict a discrete value output: zero or one, malignant or benign. And it turns out that in classification problems sometimes you can have more than two values for the two possible values for the output. As a concrete example maybe there are three types of breast cancers and so you may try to predict the discrete value of zero, one, two, or three with zero being benign. But this would also be a classification problem, because this other discrete value set of output corresponding to no cancer, or cancer type one, or cancer type two, or cancer type three. In classification problems there is another way to plot this data. We might use a slightly different set of symbols to plot this data. We might use different symbols to denote benign and malignant, or negative and positive examples. So instead of drawing crosses, we draw O's for the benign tumors and we use X's to denote malignant tumors (as shown in figure 3.2). All we did was we took these, the data set on top and we just mapped it down and started to use different symbols, circles and crosses, to denote malignant versus benign examples. Now, in this example we use only one feature or one attribute, mainly, the tumor size in order to predict whether the tumor is malignant or benign. In other machine learning problems we could have more than one feature, more than one attribute, for example we can add to tumor size, the age of the patients. Now let's say that instead of just knowing the tumor size, we know both the age of the patients and the tumor size. In that case maybe our data set will look like those in figure 3.3.

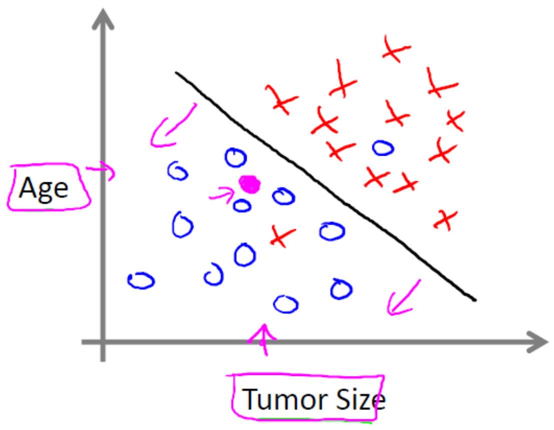


**Figure 3.3:** Further elaboration of the second example of supervised learning as discussed in the text [27].

And a different set of patients, they look a little different, whose tumors turn out to be malignant, as denoted by the crosses. So given a data set like that in figure 3.3, what the learning algorithm might do is throw the straight line through the data to try to separate out the malignant tumors from the benign ones and, so the learning algorithm may decide to throw the straight line like that to separate out the two classes of tumors. In this example we had two features, namely, the age of the patient and the size of the tumor. In other machine learning problems we will often have more features, and remaining in the previous example we can have clump thickness of the breast tumor, uniformity of cell size of the tumor, uniformity of cell shape of the tumor, and so on. And it turns out the case of great interest of learning algorithms that can deal with, not just two or three or five features, but an infinite number of features. So how do we deal with an infinite number of features? How do we even store an infinite number of things on the computer when your computer is gonna run out of memory? It turns out that when we talk about an algorithm called the Support Vector Machine", there will be a neat mathematical trick that will allow a computer to deal with an infinite number of features.

## 3.4 Unsupervised Learning

In this section, we will talk about the second major type of machine learning problem, called Unsupervised Learning. In Unsupervised Learning, we are given data that does not have any labels or that all has the same label.

It is possible to find some structure in the data on Figure 3.4? Given this data set, an Unsupervised Learning algorithm might decide that the data lives in two

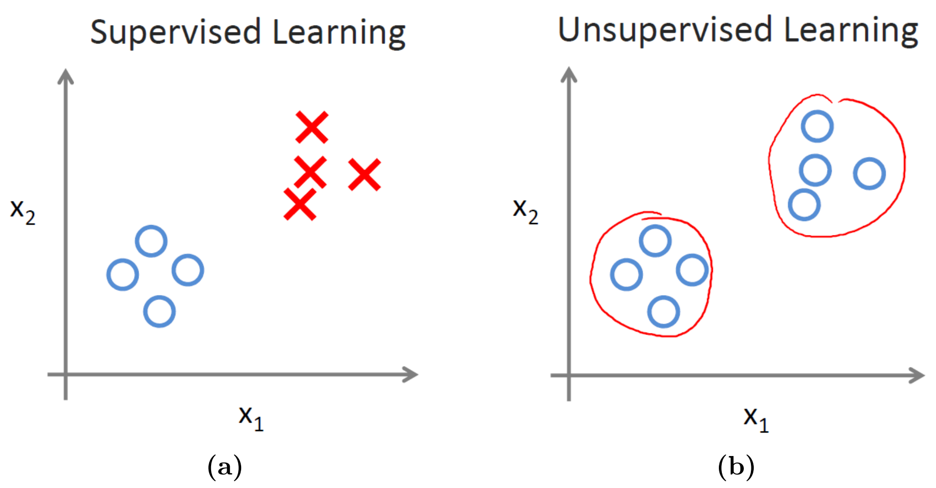
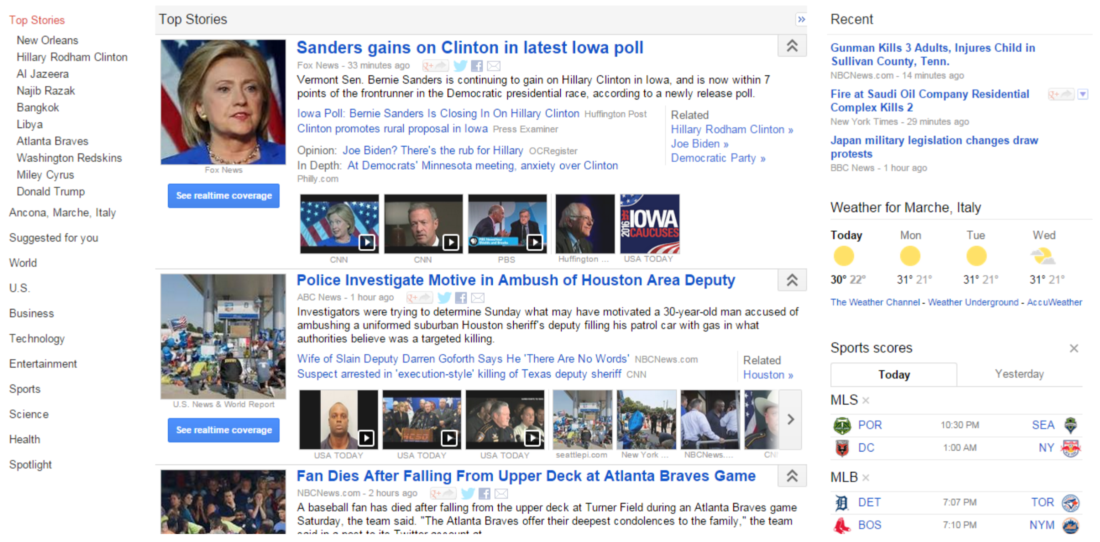


**Figure 3.4:** More on supervised vs unsupervised learning (see text for explanaton) [27].

different clusters. And also Supervised Learning algorithm may break these data into these two separate clusters. So this is called a clustering algorithm. And this turns out to be used in many places. One example where clustering is used is in Google News (Figure 3.5)


Top Stories
New Orleans
Hillary Rodham Clinton
Al Jazeera
Najib Razak
Bangkok
Libya
Atlanta Braves
Washington Redskins
Miley Cyrus
Donald Trump
Ancona, Marche, Italy
Suggested for you
World
U.S.
Business
Technology
Entertainment
Sports
Science
Health
Spotlight

Top Stories

**Sanders gains on Clinton in latest Iowa poll**
Fox News - 33 minutes ago
Vermont Sen. Bernie Sanders is continuing to gain on Hillary Clinton in Iowa, and is now within 7 points of the frontrunner in the Democratic presidential race, according to a newly release poll.
Iowa Poll: Bernie Sanders Is Closing In On Hillary Clinton Huffington Post
Clinton promotes rural proposal in Iowa Press Examiner
Opinion: Joe Biden? There's the rub for Hillary OCRegister
In Depth: At Democrats' Minnesota meeting, anxiety over Clinton Philly.com
Related: Hillary Rodham Clinton » Joe Biden » Democratic Party »
Fox News — See realtime coverage
CNN CNN PBS Huffington ... USA TODAY

**Police Investigate Motive in Ambush of Houston Area Deputy**
ABC News - 1 hour ago
Investigators were trying to determine Sunday what may have motivated a 30-year-old man accused of ambushing a uniformed suburban Houston sheriff's deputy filling his patrol car with gas in what authorities believe was a targeted killing.
Wife of Slain Deputy Darren Goforth Says He 'There Are No Words' NBCNews.com
Suspect arrested in 'execution-style' killing of Texas deputy sheriff CNN
Related: Houston »
U.S. News & World Report — See realtime coverage
USA TODAY USA TODAY USA TODAY seattlepi.com New York ... NBCNews.... CN

**Fan Dies After Falling From Upper Deck at Atlanta Braves Game**
NBCNews.com - 2 hours ago
A baseball fan has died after falling from the upper deck at Turner Field during an Atlanta Braves game Saturday, the team said. "The Atlanta Braves offer their deepest condolences to the family," the team

Recent
**Gunman Kills 3 Adults, Injures Child in Sullivan County, Tenn.**
NBCNews.com - 14 minutes ago
**Fire at Saudi Oil Company Residential Complex Kills 2**
New York Times - 29 minutes ago
**Japan military legislation changes draw protests**
BBC News - 1 hour ago

Weather for Marche, Italy

| Today | Mon | Tue | Wed |
|---|---|---|---|
| 30° 22° | 31° 21° | 31° 21° | 31° 21° |

The Weather Channel - Weather Underground - AccuWeather

Sports scores
Today | Yesterday

| MLS | | |
|---|---|---|
| POR | 10:30 PM | SEA |
| DC | 1:00 AM | NY |

| MLB | | |
|---|---|---|
| DET | 7:07 PM | TOR |
| BOS | 7:10 PM | NYM |

**Figure 3.5:** Example of Google News [28], where ‘clustering’ is used (see text).

What Google News does is everyday it goes and looks at tens of thousands or hundreds of thousands of new stories on the web and it groups them into cohesive news stories. So for each story we can read the same news from different websites. So what Google News has done is look for tens of thousands of news stories and automatically cluster them together. So, the news stories that are all about the same topic get displayed together.

It turns out that clustering algorithms and Unsupervised Learning algorithms are used in many other problems as well. Below is explained one on understanding genomics, an example of DNA microarray data.

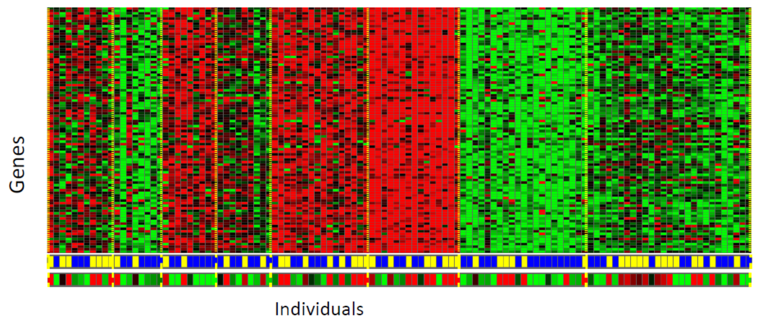


**Figure 3.6:** DNA microarray data.

The idea is put a group of different individuals and for each of them, you measure how much they do or do not have a certain gene. Technically you measure how much certain genes are expressed. So these colors, red, green, gray and so on, they show the degree to which different individuals do or do not have a specific gene (Figure 3.6). And what you can do is then run a clustering algorithm to group individuals into different categories or into different types of people.

So this is Unsupervised Learning because we are not telling the algorithm in advance that these are type 1 people, those are type 2 persons, those are type 3 persons and so on and instead what we are saying is here is a bunch of data. I do not know what is in this data. We do not know who is and what type. We do not even know what the different types of people are, but can we automatically find structure in the data, form automatically clusters of individuals that belongs to different types that we do not know in advance? Because we are not giving the algorithm the right answer for the examples in our data set, this is Unsupervised Learning.

Unsupervised Learning or clustering is used for a bunch of other applications. It is used to organize large computer clusters. In large data centers, that is large computer clusters, researchers try to figure out which machines tend to work together and if you can put those machines together, you can make your data center work more efficiently.

This second application is on social network analysis. So given knowledge about which friends we email the most or given your Facebook friends, can we automatically identify which are cohesive groups of friends, also which are groups of people that all know each other?

Another application is on market segmentation caused by the fact that companies have huge databases of customer information.

So, can we look at customer data set and automatically discover market segments and automatically group our customers into different market segments so that we can automatically and more efficiently sell or market our different market segments together? Again, this is Unsupervised Learning because we have all this customer data, but we do not know in advance what are the market segments and for the customers in our data set, you know, we do not know in advance who is in market segment one, who is in market segment two, and so on. But we have to let the algorithm discover all this just from the data.

Finally, it turns out that Unsupervised Learning is also used for astronomical data analysis and these clustering algorithms gives interesting useful theories of how galaxies are born.

All of these are examples of clustering, which is just one type of Unsupervised Learning. Another type can be explained through the so called "cocktail party problem". So, there is a party, room full of people, all sitting around, all talking at the same time and there are all these overlapping voices because everyone is talking at the same time, and it is almost hard to hear the person in front of you. So maybe at a cocktail party with two people, two people talking at the same time, and it is a somewhat small cocktail party. And we are going to put two microphones in the room so there are microphones, and because these microphones are at two different distances from the speakers, each microphone records a different combination of these two speaker voices. Maybe speaker one is a little louder in microphone one and maybe speaker two is a little bit louder on microphone 2 because the 2 microphones are at different positions relative to the 2 speakers, but each microphone would cause an overlapping combination of both speakers' voices. So we can do, is take these two microphone recorders and give them to an Unsupervised Learning algorithm called the cocktail party algorithm, and tell the algorithm find structure in this data for you. And what the algorithm will do is listen to these audio recordings and say: it sounds like the two audio recordings are being added together or that have being summed together to produce these recordings that we had. Moreover, what the cocktail party algorithm will do is separate out these two audio sources that were being added or being summed together.

## 3.5 Supervised Learning in more detail

In this thesis, which has a naturally limited scope in the overall efforts in applying Machine Learning techniques to complex problem, we focus on an application of a "supervised" Machine Learning. In this approach, a program is "trained" on a predefined set of "training examples", which then facilitates its ability to reach an accurate conclusion when given new data.

Our problem may be formalized as follows. We have multiple data points x, and a "predictor" h(X) (i.e. some metric of interest about x). This predictor may be written as simple as:

$$h(x) = \theta_0 + \theta_0 x \tag{3.1}$$

Training examples (x-train) are used to optimize *h(x)*: for each x-train, we have a y known in advance, so each *h(x-train)* instead given by my model may be compared with the true value y with a simple difference, i.e. y-*h(x-train)*. If we have a large set of x-train values, each with a known corresponding value of true y, we may use the model to compute all *h(x-train)* and estimate the "wrongness" of *h(x)* itself by computing all values of y-*h(x-train)*. I can then tweak $\theta_0$ and $\theta_1$ – thus tweaking *h(x)* – to make *h(x)* "less wrong". This process may be repeated over and over until the system has converged on the "best" values of Theta-0 and Theta-1. In this way, the predictor becomes "trained" and it is ready to challenge itself against some "real world" predicting. Note however that the goal of Machine Learning is never to make "perfect" guesses. The "best" values – as written above – i.e. how "good" or "better" than others they are, depend essentially on which level of precision in the prediction one's problem needs, which quality of Machine Learning models we can afford to apply, e.g. also taking into account the non-infinite amount of computing resources we may have available to run a Machine Learning model. Ultimately, the thing to remember is that the goal of any successful Machine Learning effort is never to reach perfection, but only and always to make guesses that are good enough to be useful for the problem under study.

In the directions to follow to build a successful Machine Learning model, much care must be given to the choice of the training data. Regardless of how such data is operationally divided into in the model implementation (e.g. training vs validation vs test sub-samples) and considering instead "training data" the whole set of data used to build up the model, it is worth underlying that this must be a statistically significant random sample – as Machine Learning builds heavily on statistics. If the sample is not random, the price to pay is that the machine learns patterns in the data that are not actually there. If the sample is not large enough, the price to pay is that the machine will not learn enough, or (even worse) reach inaccurate conclusions. In terms of how making sure that $\theta_0$ and $\theta_1$ are better at each step of this iterative process, the "wrongness" (as quoted above) must be minimized. The key ingredient here is to have a "cost (loss) function", i.e. find $\theta_0$ and $\theta_1$ for our predictor *h(x)* such that our cost function is as small as possible.

As a final remarks, at this stage of the introduction of Machine Learning approaches, it must be reminded that a supervised Machine Learning system can be either a "regression" system or a "classification" system. In a regression system, the value being predicted falls somewhere on a continuous spectrum (e.g. question to answer could be "how many/much?"). In a classification system, the value being predicted is discrete, a yes-or-no (e.g. question to answer could be "is this guess true or false?"). Major differences exist in the design of the predictor *h(x)* and of the cost (loss) function. In particular, all the logic behind the latter is of particular interest in this thesis. In a Machine Learning classification system, the most important question to ask ourselves is "what is the cost of a guess to be wrong?". If I guessed 0 and it turned out to be 1, or viceversa, I was completely wrong, and as one cannot be more wrong than completely wrong, the penalty (cost) in this case must be maximum. Alternatively, if the correct value was 0/1 and we guessed 0/1, this is good and the selected cost function should not add any cost for each time this happens. But there are other cases to consider: if the guess was right but one was not completely confident (e.g. y=1 but *h(x)* $= 0.75$) this should have a cost but it should also be relatively low; on the other hand, if the guess was wrong and one was not completely confident (e.g. y=1 but *h(x)* $= 0.25$), this should come with some significant cost, but still not as much as if one was completely wrong.

When evaluating a classifier, there are different ways of measuring its performance. For supervised learning with two possible classes, all measures of performance are based on four variables obtained from applying the classifier to the test set: TP (true positive), TN (true negative), FP (false positive), FN (false negative). True Positive are those which we guess true (1) and really are true, True Negative are those which we guess false (0) and really are false, False Positive are those which we guess true and really are false, False Negative are those which we guess false and really are true. In predictive analytics, a *table of confusion* (sometimes also called a *confusion matrix* (Figure 3.7), is a table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives. The entries in a confusion matrix are counts, i.e. integers.

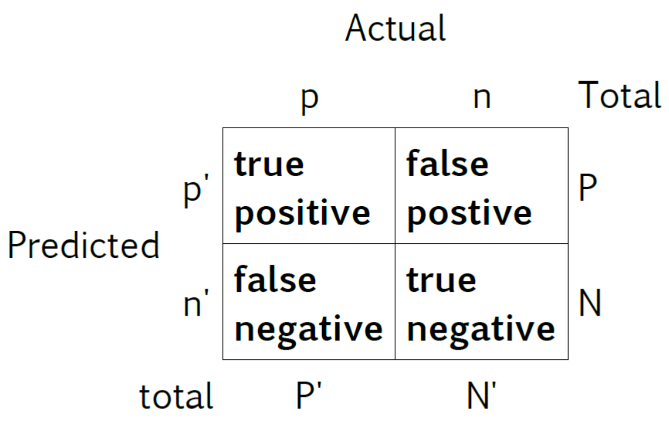


**Figure 3.7:** Confusion matrix.

The total of the four entries TP+TN+FP+FN=n is the number of train examples. In this thesis true and false classifier is restrict to the concept of data popularity. As it will be shown in following pages, we focus on DCAFPilot that is a pilot project that has the goal of creating a system (for CMS) which can predict which datasets will become popular even before datasets will be available on Grids for further analysis. The concept of popularity is complicated and will be explain better in a later chapter, but at this point, once the concept of popularity is defined, the classifier has the goal to predict if a dataset will be popular or unpopular in the future. In order to evaluate the classifiers and find the best way to apply them, there are some estimators that help us to choose the best. For example there are: *Accuracy*, *Precision* (or *Positive predictive value*, that is the proportion of predicted positives which are actual positive), *Recall* (that is the proportion of actual positives which are predicted positive) and *F1* (that is the harmonic mean of precision and sensitivity(or true positive rate)). They are calculated as:

- **Accuracy** = (TP+TN)/n
- **Precision** = TP/(TP+FP)
- **Recall** = TP/(TP+FN)
- **F1** = 2TP/(2TP+FP+FN)

Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the number of samples in different classes vary greatly). For example, if there were 95 elements of type A and only 5 elements of type B in the data set, the classifier could easily be biased into classifying all the samples as cats. The overall accuracy would be 95%, but in practice the classifier would have a 100% recognition rate for the A class but a 0% recognition rate for the B class.

For our purposes, in particular, the *FPR* (False Positive Rate or FP/(FP+TN)) is relevant. In general both *FP* and *FN* data are relevant because, as the name suggests, they are wrong predictions but the first one is heavier as error because it cause the erroneously popular classification of a data that involves a series of concauses on the system such as the production of more copies of the data. In this way, more storage capacity is needed and economically involves an higher expense. Otherwise *FN* data don't involve more storage capacity because they are classify as unpopular but they are not. So here the problem is that there will be less copies on Grid and at the same time their access will be elevated, due to the fact that they are popular and this causes transfer latencies. The disadvantage is that, when a user request a FN data, he has to wait more time to really have it, but sooner or later will arrive. So the problem is not economically but is the time. Because of everything is said our next goal mainly will be reduce *FPR*.

In the following, a supervised Machine Learning classification system designed and used to attack the CMS dataset popularity problem, will be presented and discussed.

# Chapter 4

# Big Data Analytics techniques in CMS: the DCAFPilot

## 4.1 Introduction to Analytics in CMS

The CMS experiment at the LHC accelerator at CERN designed and implemented a Computing model that allowed successful Computing operations in Run-1 and gave a crucial contribution to the discovery of the Higgs boson by the ATLAS and CMS experiments. The workflow management and data management sectors of the model have been operated at full capacity exploiting WLCG resources for years. Around the massive volume of original and derived physics data from proton-proton and heavy-ions collisions in CMS, plenty of other data and metadata about the performances of the computing operations have been also collected and rarely (or never) examined. This latter sample is a wild mixture of non-physics heterogeneous data, both structured and unstructured, which well fits to deeper investigation with Big Data analytics approaches. In the context of CMS R&D activities, exploratory projects have been started to extract some values from this dataset and to seek for patterns, correlations as well as ways to simulate the Computing Model itself.

The CMS experiment launched a Data Analytics project, whose goal is manyfold and depends on the timeline. As a long-term goal (2-3 years), the project aims to build adaptive datadriven models of CMS Data Management (DM) and Workload Management (WM) activities - as part of the overall CMS Computing Model - with the target to be able to predict future behaviours of the CMS systems in operations from the detailed measurements of their performances in the past. As a medium-term goal (hopefully within LHC Run-2 already, aiming for incremental improvements), the project aims to improve the use of CMS computing resources. As a short-term goal (within Run-2), the projects aim to concretely support the CMS Computing Operations team as much as possible through deeper understanding of the CMS data collected over the years. In this sense, understanding the "data" - by which we mean any (meta-)data produced by any Computing Operations activity since Run-1 started - is extremely valuable in itself. The reason for such modelling to be adaptive is that models elaborated in the past aren't going to apply to the future for long, and only adaptive modelling itself will give CMS confidence and predictive power in the long term.

### 4.1.1 Approach to data and metadata

During Run-I CMS built an operations model that worked and successfully met all requirements. It served well the CMS physics program, according to all possible metrics. But apart from Run-1 success we cannot claim that we fully understand the system and neither know how efficient we were during this time of operations. For example, the data transfers to Tier-2 sites exceed expectations as from the MONARC model, moreover we have no model to shape this kind of traffic. Another example, researchers filled the disk storage at the Tier-2 level with plenty of data useful for physics analyses, but a large fraction of AOD/AODSIM storage is left un-accessed for long periods of time which accounts for inefficient resource utilisation. These are only a few examples which strongly suggest that deep understanding of all system components is required to improve its utilisation. In the Computing operations during Run-1 and the first long shutdown (LS1) period, all computing systems collected plenty of data about operations themselves, e.g. monitoring data, accounting information, machine logs, etc. All of them were archived, but rarely (or never) accessed by anyone. This data comprises information about transfers, job submissions, site efficiencies, release details, infrastructure performance, analysis throughput, and much more (Figure 4.1).

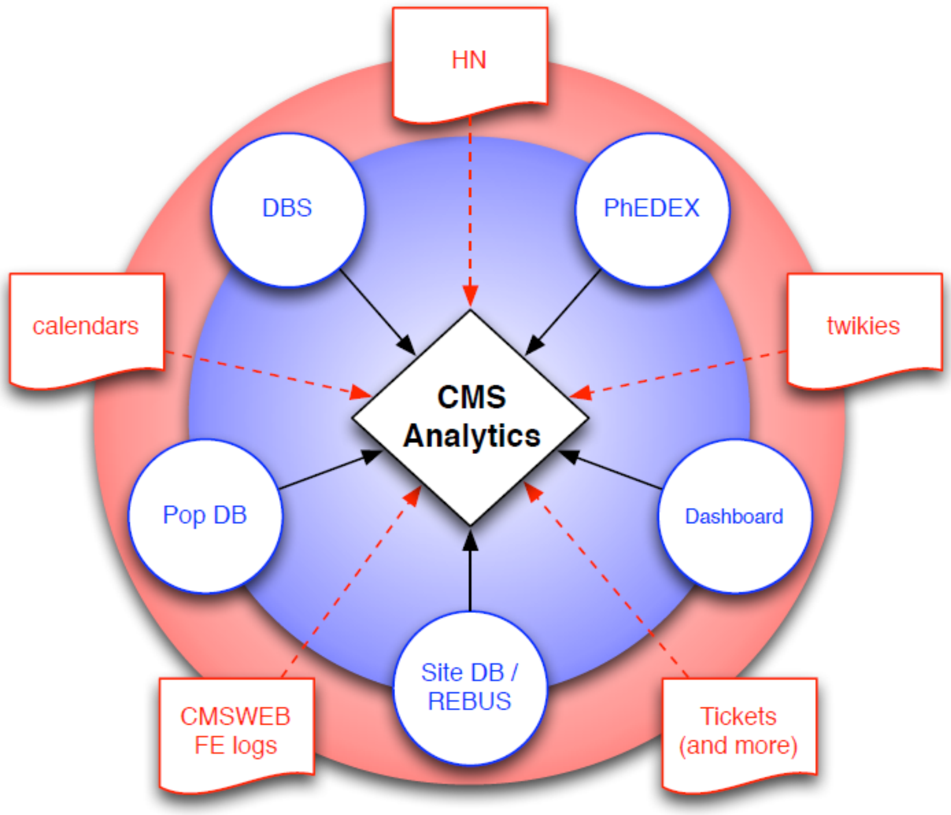


**Figure 4.1:** Source of structured and unstructured data to the CMS Analytics project.

This precious data set is left unanalysed so far because researchers mainly monitor their systems in near-time for debugging purposes, rather than analyse what happened in the past and study in depth systems behaviour. Additionally, they never fixed holes in their monitoring data and validated (most of) them with decent care. Therefore such data can be considered as incomplete and not suitable for further analysis. The quality of the data, and a careful work on data preparation before the analysis, is one of the main components which leads to success stories in any Big Data analytics project. In our case we must pay significant attention to the four big V's: the Volume (scale of the data), Velocity (analysis of streaming data), Variety (different forms of data), Veracity (uncertainty of data). The data Volume here is not negligible in itself, but definitely manageable with respect to the LHC collisions data we deal with. The Velocity is partially relevant, i.e. we aim to a quick availability of analytics results, but having a real-time feedback from the data is not actually a requirement. The Variety is very relevant, as we deal with a very irregular data set, consisting of structured, semi-structure and unstructured data (see next section). The Veracity is also extremely delicate, as the data integrity and the ability to trust the data analysis outcome to make decisions is crucial.

### 4.1.2 Structured data and beyond

Structured information (see Figure 4.1, blue color) represents a variety of CMS Computing activities which are stored across multiple data services. For example, the DBS system is the CMS source for physics meta-data; the PhEDEx transfer management database offers data transfer service and data replica catalog functionalities; the Popularity Database (PopDB) collects dataset user access information (e.g. access frequency, which replicas are accessed on Tiers of the WLCG, the amount of CPU used); SiteDB (and REBUS, eventually) gives authoritative information about site pledges, deployed resources, and manpower onsite, while CERN Dashboard stands as a massive repository of details on Grid jobs (and beyond). In addition to structured data, plenty of data in the CMS Computing ecosystem is completely unstructured (see Figure 4.1, red color). This type of information is hard to collect and process, but it represents potentially very rich content.

For instance, the CMS HyperNews system offers today more than 400 different fora, representing de-facto a reference on several years of user activities (announcements, information on user activities, insight into change of focus in the physics interests of individuals/groups over time, hot topics, etc.). But to be useful, it requires social data mining efforts on several aspects of collaboration-level activities.

The tickets offers a view on infrastructure issues reporting/tracking, via different tracking systems used over the years (e.g. Savannah, GGUS), and complemented by activity-based ELOGs, topical e-groups, etc.

The CERN-based twikies offer a content that stands as a knowledge graph that could be mapped to user activities and physics interests, and help to model their evolution over the time.

Hot periods of CMS physics analysis can be tracked via the CMS calendar of events (as well as non-CMS calendars), the CMS sub-projects planning information, list of major conferences and workshops, etc. Such information can also lead to identification of seasonal cycles within different physics communities. To this extent, also the Vidyo logs may turn up to be useful, in terms of knowing who regularly attends specific meetings.

Finally, CMSWEB cluster front-end logs (actually, semi-structured data in this case) serve all data sources to users, thus might be mined to extract valuable information on user activities.

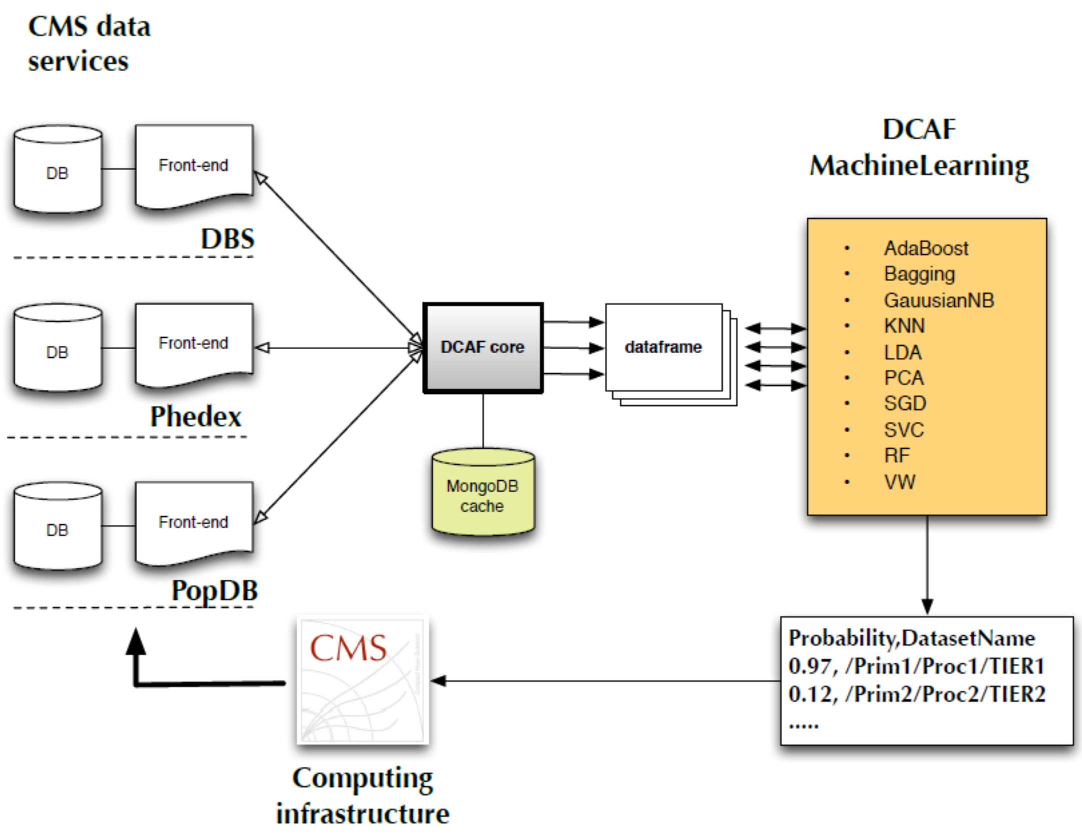


**Figure 4.2:** Descripton of the DCAFPilot workflow and components.

### 4.1.3 A use-case: CMS data popularity and the DCAFPilot

In CMS the Dynamic Data Placement [29] team is relying on historical information of datasets popularity to add (remove) replicas of existing datasets that appear to be most (least) desired by end-users. The goal of this activity is to balance resource utilisation at different sites by replicating more replicas of popular dataset. But this approach has one flaw, it reacts to spikes of the dataset popularity after the fact. Therefore it would be desired to have a system in place which can predict which datasets will become popular even before datasets will be available on Grids for further analysis.

The DCAFPilot (Data and Computing Analysis Framework Pilot) [30] is a pilot project to understand the metrics, the analysis workflow and necessary tools (with possible technology choices) needed to attack this problem. The framework has been recently exploited to investigate the feasibility of this analytics approach for the CMS data popularity use-case.

In a nutshell, the pilot architecture is shown in Figure 4.2. Data is collected from CMS data services by a DCAF core that uses MongoDB for its internal cache. So far, informations are collected from the following CMS structured data-services: DBS, PhEDEx, PopDB, SiteDB, Dashboard (see [31] for more details on the CMS systems). A data-frame generator toolkit has been developed to collect and transform data from CMS data services, and to extract necessary bits for a subset of popular and un-popular datasets. The data-frame is fed to machine learning algorithms (both python and R code used) for data analysis. A quantitative estimate of the popularity is given for specific types of datasets, which may be fed back to the CMS computing infrastructure as a useful input to daily operations and strategical choices.

The data collection flows, in some more details, works as follows. All datasets from DBS are collected into the internal cache. Popular datasets are queried from PopDB with a weekly granularity. For all of these datasets more information is also extracted from DBS, PhEDEx, SiteDB and the Dashboard. This information is complemented with random set of unpopular datasets (to avoid bias in later stage machine learning algorithms). All such information is stored in different data-frame files, that can be fed to any ML library for whatever purpose one may have. At the moment, all data from 2013 and 2014 years have already been pre-processed and are available for analysis. At this stage, a prediction of which dataset(s) may become popular is given, in the form of their probability versus each dataset name.

Some statistics from a dry run of the machinery are reported below. Five data services were queried (4 DBS instances used) and 10 APIs were used. The internal MongoDB cache was fed with about 220k datasets, 900+ release names, 500+ SiteDB entries, 5k people's DNs. In total, about 800k queries were placed. Anonymisation of potentially sensible information is done via the internal cache. The final data-frame is constructed out of 78 variables, and made out of 52 dataframe files, roughly 600k row in total. Each file is worth 1 week of CMS meta-data (approximately 600kB gzipped), and it has about 1k popular datasets with a roughly 1:10 ratio of popular vs unpopular samples randomly mixed. The data-frame can be visualised real-time in terms of live data, correlations, and also exploring different data popularity metrics (e.g. number of users accessing a dataset versus total CPU used to process it).

Once the data collection is finalised, the actual analysis can start. First of all, a data transformation is needed to transform the data into a suitable format for machine learning techniques. Then, a specific machine learning approach must be chosen, e.g. classification (it allows only to classify into categories, e.g. popular or unpopular) versus regression (it allows to predict real values of the chosen metrics, e.g. number of accesses) vs online learning techniques (so far a classification approach has been adopted). The following step is to train and validate the machine learning model. This is done by splitting the data into train and validation sets. The model chosen as the best one is then applied to such new data to make predictions, and such predictions are regularly verified with PopDB fresh data once metrics become available.

Within the DCAFPilot project we have completed a major milestone to build-up the machinery, i.e. collect data on regular intervals, transform them into ML data-format, run various ML algorithms and yield and compare predictions. At this moment several ML algorithms are adopted within DCAFPilot project: a regular set of scikit-learn classifiers [32], e.g. *Random Forest*, *SGDClassifier*, *SVC*, etc., the *online learning algorhtm*, *Vowpal Wabbit*, by Yahoo, and gradient boosting tree solution (*xgboost*, the *eXtreme Gradient Boosting*).

The preliminary results are quite encouraging. For instance we are able to predict with reasonable accuracy a portion of the 2014 data set. All the results have to be taken as preliminary, but they already give interesting indications, and once they will be considered solid they will offer valuable information to tune CMS computing operations. A few examples may clarify how. A few false-positive popularity predictions would imply a little wasted bandwidth and disk space for a while, but not much. On the other hand, a false-negative can mean a few days delay in a specific analysis. The experience of the Higgs announcement - in which data taken two weeks earlier was included, with plots produced that same morning - is teaching us that in hot periods a delay of few days could be important.

So far DCAFPilot project is capable of collecting the data, transforming them into ML suitable format, run various ML algorithms and make a predictions. The final predictions can be verified posteriorly by comparing them with data collected in PopDB. With such machinery in place we are ready to start full analysis. Our approach is the following: collect historical data on weekly basis, run them through transformation and modeling steps, compare different classifiers and built best predictive model, and, finally apply this model to new set of data we expect to have. The latter can be collected from Request Manager and DBS CMS data-system by the time data-ops team start processing the datasets.

## 4.2 The DCAFPilot components and functionalities

This section lists and described the DCAFPilot python components and how they are run.

There are several tools which are available in DCAFPilot project:

- *dataframe*, to collect information from various CMS data-providers and represent it in anonymized form of the dataset suitable for data analysis;
- *merge_ csv*, a tool to merge multiple CSV files into single one;
- *transform_ csv*, a tool to transform generic dataframe into classification problem;
- *model*, a tool to train ML algorithm;
- *pred2dataset*, a tool to convert prediction into human/CMS data format;
- *check_prediction*, a tool to test predicted values against provided data;
- *popular_ datasets*, a tool that get popular datasets from popularity DB;
- *verify_predictions*, a tool to cross-check predictions against popDB data.

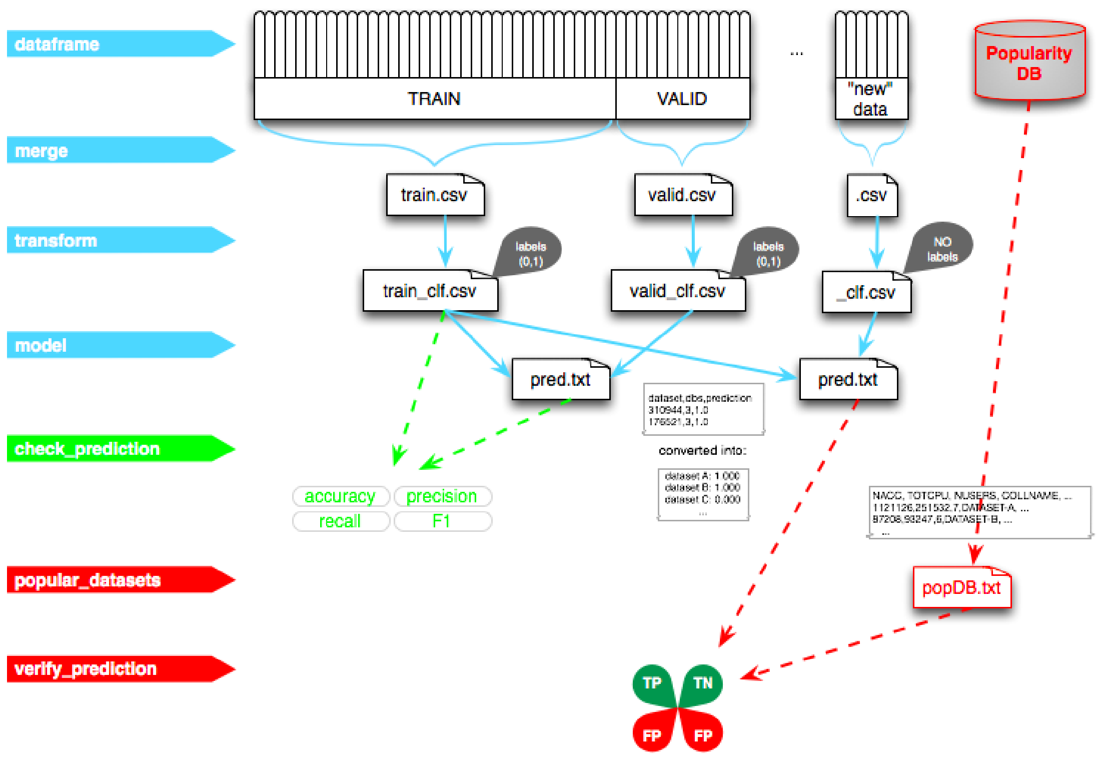


**Figure 4.3:** Graphical representation of possible workflows in DCAFPilot. A full description is in the text.

### 4.2.1 dataframe

```
dataframe --start=20150101 --stop=20150108 --newdata
--verbose=1 --fout=new-20150101-20150108.csv
```

Through *dataframe* tool we keep dataframe from DBS giving, as `--start` and `--stop` arguments, the beginning and the end of temporal window of datasets that we want to catch, putting them into a csv file. So in dataframe we have all data in DBS presents on the start day of week and all dataframes created during the week. The dataframes that we create are always relative to one week. Besides in dataframe we know for example how many times a dataset is accessed in that week, but this dataset could be created the first day of the weak but also the last.

### 4.2.2 merge_csv

```
merge_csv --fin=path_to_data --fout=2014.csv.gz
```

This is a tool that has the aim of collect together a given number of dataframe creating a csv file. It has 78 variables such as *id, dataset, size, tier, naccess, nusers, totcpu*, etc. So as `--fin` argument we pass the dataframes path that we want to merge and as `--fout` we pass the name of the file that the tool will produce after it has run (in this case it produces a file that contains all 2014 dataframes). This process is applied two times, in order to produce train and valid sets.

### 4.2.3 transform_csv

```
transform_csv --fin=train.csv.gz --fout=train_clf.csv.gz
--target=naccess--target-thr=100
--drops=nusers,totcpu,rnaccess,rnusers,rtotcpu,nsites,
s_0,s_1,s_2,s_3,s_4,wct
```

The aim of this tool is to transform a file into a classification one, into a suitable format for Machine Learning. So in this passage, starting from *train.csv.gz* as `--fin` argument, is created a new classification file (*train_ clf.csv.gz*) in which is add a column (target) where the datasets have 0 (zero) or 1 (one) value according to the fact that in fixed target column (in this example is *naccess*) they have a value higher than a fixed one (target threshold). In addition to this some column of the original file are deleted. The level of cut, given by the target threshold, is linked to the concept of popularity. Indeed in target column we have the answer to the question: this dataset is popular? If the answer is true, then target column has 1, otherwise has 0.

As described in the previous section, DCAFPilot focuses on the concept of data popularity, trying to understand from a huge amount of data which are popular and unpopular and from what this build a predictive model in order to optimize the storage use and job latencies on Grid. As will be discussed in the next chapter, the definition of popularity itself in this context needs to be tailored to allow its adequate prediction. We will see that the concept of popularity builds from the possibilities of CMS to afford storage space and to accept latencies, joint to a good reliability of the model.

So we can use different target column (such as *naccess, nusers, totcpu*) with different threshold, creating our definition of popularity.

`transform` script can be used both to create a classification file for train file and for valid file.

### 4.2.4 model

```
model --learner=RandomForestClassifier--idcol=id
--target=target --trainfile=train_clf.csv.gz
--scaler=StandardScaler --newdata=valid_clf.csv.gz
--predict=pred.txt
```

This tool creates a model that gives us prediction about popularity. So after having chosen a classifier (passed as `--learner` argument), in this example we pass as arguments to model *train_ clf.csv.gz* as train file, and *valid_ clf.csv.gz* as newdata. Here the train set is used to build our model, a validation set is used to tune our model parameters, e.g. run cross-validation, etc. The option `--newdata` will accept both either validation set or totally new data. The difference between then is the following. In validation set we know our labels, i.e. if dataset was popular or not, while in new data we do not know them. The former case then can be used to test the prediction of our model (since we know the labels), while in later case we will get predictions but we will have no other information to know if they are good or bad.

Besides we ask model script to yield prediction into *pred.txt* file. This file contains *dataset_id,dbs,prediction* triplets.

Through *pred2dataset* we convert prediction into human/CMS data format giving a path of the data with type of it (AOD, AODSIM, USER...) and the prediction.

### 4.2.5 check_prediction

```
check_prediction --fin=valid_clf.csv.gz --fpred=pred.txt
--scorer=accuracy,precision,recall,f1,tp,tn,fp,fn
```

With `check_prediction` we verify the reliability of the model giving some parameters (*accurancy, precision, recall, F1, TP, TN, FP, FN*) comparing in this case the valid data with *pred.txt.* The validation set here contains labels so when we will run a model over these data it will make predictions and you can compare predictions with real labels. Usually the quality of the fit is measured by statistical parameters that are those in `--scorer` arguments.

### 4.2.6 popular_datasets

```
popular_datasets --start=20150101
--stop=20150108 > popdb-20150101-20150108.txt
```

`popular_datasets` is a tool to extract popular datasets from popDB. *popdb-YYYYMMDD1-YYYYMMDD2.txt* is a file with historical information about dataset usage. It contains NACC, TOTCPU, NUSERS, COLLNAME, RNACC, RNUSERS, RTOTCPU columns. The difference between this and `dataframe` tool is that here every time we run it we obtain different values of *naccess, totcpu, nusers*, etc., because they are related to informations updated to the run moment of `popular_datasets`, informations about dataframes, that exist during the period passed as argument, but up to the present; conversely `dataframe` tool create a csv in which there are informations of dataframes, that exist during the period passed as argument, but related to the week itself. So if we use `popular_datasets` we obtain different values on time, they change with time, while when we use `dataframe` tool we obtain static informations that don't depend on the moment in which we run the script.

### 4.2.7 verify_prediction

```
verify_predictions --pred=pred.txt.out
--popdb=popdb-20150101-20150108.txt
```

This tool will compare prediction pairs from *pred.txt.out* file (that is produced by the model) and popdb one. We have to underline an important thing: popdb file and the file produced by `popular_dataset` (*popdb-XXX-YYY.txt*) that here behaves as valid file to build *pred.txt.out*, must to be related to the same temporal window. The `verify_prediction` will only look at matches and tell us how many datasets we predicted as popular and unpopular. This time we don't know what is FP/FN rates and can't know *accuracy, precision* metrics, we only have a prediction. But we can tell that our model has certain FP/FN rates because I studied it before with some validation data.

## 4.3 How to use DCAFPilot: build, tune, run a model

There are two clear example about we can use the model: in the first one we compare predictions given by the model with the valid set (that has labels); in the second one we compare prediction with popdb file which not contains labels but only informations (such as *naccess, nusers, totcpu*) up to the run moment. So we will present all steps of this two different proceedings.

When we have data and build a model, to fit them we want to check how good or bad our model is. For that purpose usually a user split data into training set and test set. The train set is used to build a model and test set is used to verify the model. Usually the train set is also split into two subsets where one is used for finding fit parameters and another for *cross-validation* [33] (train and valid subsets). Here 70%/30% splitting for train/valid is common practice, but any other split level close to the previous is fine too. The point is how much data we want to use for fit and how much we want to do *cross-validation.*

The validation set here contains labels so when we will run a model over these data it will make predictions and we can compare predictions with valid labels itself. Usually the quality of the fit is measured by statistical parameters, such as *accuracy, precision, recall*, etc. Now we back to definitions we used. The data file(s), *dataframe-XXX.csv*, is what we collect from CMS data-services. Then for example we merge them into *2014.csv.* Then we split merged file into *train.csv* and *valid.csv.* But yet these files need to be transformed into suitable format for ML. This is done via `transform_csv` script. Once this is done we can start train our model, e.g.

```
model --learner=RandomForestClassifier --idcol=id
--target=target --scaler=StandardScaler
--predict=pred.txt--train-file=train_clf.csv.gz
--newdata=valid_clf.csv.gz
```

Here we told model script to use *train_ clf* file and give *valid_ clf* file as "newdata". We also ask model script to yield prediction into *pred.txt* file. This file contains *dataset_ id,dbs,prediction* triplets. Using this file we can invoke `check_prediction` script to find out some statistical metrics which can tell us how my model performed over data, e.g.

```
check_prediction --fin=valid_clf.csv.gz --fpred=pred.txt
```

```
--scorer=accuracy,precision,recall,f1
```

But as we can see we need to supply *valid_ clf* file in order to extract labels and compare them with prediction ones (which come from *pred.txt*). Using these informations we can find out *accuracy, precision*, etc. metrics. So this script just check my model using file *valid_ clf* and *pred.txt* files. In `check_prediction` we use labels in valid sets, conversely we crate a model through valid set but without using labels.

The `pred2dataset` script is used only to convert *pred.txt* file (which is again *dataset_ id,dbs,prediction triplets*) into human readable format as prediction and dataset name.

All this is shown in the left side of Figure 4.3. Now we pass on the right side.

To get new data we invoke the following script

```
dataframe --start=XXX --stop=YYY
--newdata --fout=new-XXX-YYY.csv
```

We only specify some time window for the dataframe and supplied `--newdata` flag. The newly created file *new-XXX-YYY.csv* will contain the same meta-data, but it does not have any labels, since it is new data.

Now we will need to transform it, we drop some attributes, etc.

```
transform_csv --fin=new-XXX-YYY.csv
--fout=newdata-XXX-YYY.csv
--target=naccess --target-thr=10
--drops=nusers,totcpu,rnaccess,rnusers,rtotcpu,nsites,
s_0,s_1,s_2,s_3,s_4,wct
```

Now, we have *newdata-XXX-YYY.csv* which is suitable for ML.

Then we can run our model script again

```
model --learner=RandomForestClassifier --idcol=id
--target=target --scaler=StandardScaler
--train-file=train_clf.csv.gz
--newdata=newdata-XXX-YYY.csv --predict=pred.txt
```

Here we supplied the same options as before, but for `--newdata` we put a file which we just created. We also asked to write out *pred.txt* file. This file will again contains *dataset_ id,dbs,prediction* triplets.

We can convert it into human readable format, like

```
pred2dataset --fin=pred.txt --fout=pred.txt.out
```

and we will have prediction and dataset pairs but apart from *valid_ clf* which we discussed above we don't have labels in this case. Therefore we can't apply `check_prediction` script here since we don't have labels, but what we can do is "wait" once data will be appear in Pop DB. If we choose time interval XXX-YYY as one week, we will need to wait another week and some data will appear in Pop DB. Therefore, we will wait, e.g. one week, and then invoke new script

```
popular_datasets --start=XXX
--stop=YYY > popdb-XXX-YYY.txt
```

This time we can invoke another script `verify_prediction` which will compare prediction pairs from *pred.txt.out* file and popdb one. This script will only look at matches and tell us how many datasets we predicted as popular and unpopular. This time we don't know what is FP/FN rates and we don't know *accuracy, precision* metrics. We only have prediction. But we can tell that my model has certain FP/FN rates because we studied it before with some validation data.

# Chapter 5

# Analysis and discussion of results

In this chapter the steps that have brought to choose the best model to predict popularity of new CMS datasets will be described. Before going into the details of the analysis and the discussion of the results, it is worth to make a point regarding what the actual goal of the measurement is, which is/are the observable(s) chosen to measure it, and what is the rationale behind its/their choice.

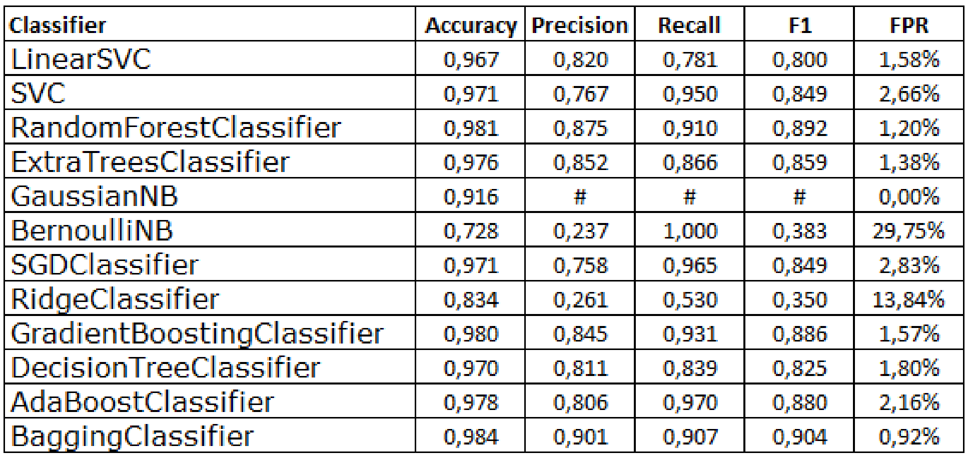
As discussed previously, the goal is to perform popularity predictions that would allow an a-priori intelligent data placement instead of the existing a-posteriori datasets cancellations or additional replication according to a static algorithm. In this sense, the work documented in this thesis has a goal that can be quantified in terms of the FP rate (FPR) alone, i.e. "how much can we afford to produce a high number of FP?". In fact, a FP means that we have predicted as popular a dataset that turns out not to be actually popular at all. This “mistake” gives an overhead in data placement, because this dataset will be unnecessarily replicated worldwide. The key point here is to quantify which is the cost that I am willing to accept for my “mistake”. In order to estimate this, this simple exercise was made. DBS was queried for the average number of newly registered datasets per week in 2014, and this number turned out to be 565$\pm$300, so we can assume for simplicity to that 500 new datasets are created every week. Given the average dataset size is 2 TB, this corresponds to 1 PB per week. If we allowed 1% of this amount to be FP, it would imply that every week 5 datasets (10 TB) will be transferred and never accessed. As we move few PBs per week, this amount is indeed very acceptable. So a FPR as low as roughly 1% is a more than acceptable price to pay. This parameter just drives everything. This parameters will tell you which algorithm is more acceptable, how good it must be, how much you can spend to tune the model, etc. E.g. if a new hardware architecture would allow you to run DCAFPilot in a “better” way (in any sense) and will offer you a FPR reduction that is sensible, probably it will be worth doing it, otherwise it will not be. So, FPR will be our main driving factor in the following. Of course it is not the only one. For example, the FN rate (FNR) is also interesting. A FN means that we have predicted as unpopular a dataset that turns out to be actually popular. By doing this different kind of “mistake”, we do not replicate such dataset, so the price we pay is in terms of increase in the latency of CRAB jobs which can run on fewer sites (or will have to access the dataset remotely). But, thinking about it, this “mistake” case is just like doing nothing, i.e. it is as the system is today. While the FPR implies a transfer, a cost you pay, the FNR is not a price paid for sure, as the dataset may not be requested by anyone, and if it is, the price may be lower, just a slightly increased latency. So, indeed, FPR gives the higher price and will be our main driver, and FNR will be a second, lower-priority driver in the discussions in the following part of this chapter.

As a result of the discussion above, we will focus on FPR, that is calculated as $FPR = FP/(FP + TN)$, on FNR , that is calculated as $FNR = FN/(FN + TP)$, having a look at TP also. Additionally we want a model that has high levels of *accuracy*, *precision*, *recall* and *F1*, that means that our model fits very well the data.

## 5.1 Choice of the best classifier

As already explained in the previous chapter, the model is built on the basis of a ML classifier. A classifier is an algorithm that implements the ML classification, details of which are reported in Chapter 3. There are several different classifiers and in this thesis we have tried some of them. Before starting, it is worth reminding that all tests in this section were done using dataframes of the whole 2014 split in 70% for train and 30% for valid sets, and we use `check_prediction` comparing predictions to valid set. Below results for different *naccess* cuts are shown, in order to better understand the behaviour of different classifiers.

We have tried to study the best suited classifier by applying two different cuts, namely *naccess>10* and *naccess>60*. It is possible to see through Tables 5.1, 5.3 and Figures 5.2, 5.4, 5.5 that there are some differences between values of the two different cuts.



| Classifier | Accuracy | Precision | Recall | F1 | FPR |
|---|---|---|---|---|---|
| LinearSVC | 0,967 | 0,820 | 0,781 | 0,800 | 1,58% |
| SVC | 0,971 | 0,767 | 0,950 | 0,849 | 2,66% |
| RandomForestClassifier | 0,981 | 0,875 | 0,910 | 0,892 | 1,20% |
| ExtraTreesClassifier | 0,976 | 0,852 | 0,866 | 0,859 | 1,38% |
| GaussianNB | 0,916 | # | # | # | 0,00% |
| BernoulliNB | 0,728 | 0,237 | 1,000 | 0,383 | 29,75% |
| SGDClassifier | 0,971 | 0,758 | 0,965 | 0,849 | 2,83% |
| RidgeClassifier | 0,834 | 0,261 | 0,530 | 0,350 | 13,84% |
| GradientBoostingClassifier | 0,980 | 0,845 | 0,931 | 0,886 | 1,57% |
| DecisionTreeClassifier | 0,970 | 0,811 | 0,839 | 0,825 | 1,80% |
| AdaBoostClassifier | 0,978 | 0,806 | 0,970 | 0,880 | 2,16% |
| BaggingClassifier | 0,984 | 0,901 | 0,907 | 0,904 | 0,92% |

**Figure 5.1:** Results of performance studies of different ML classifiers upon applying a *naccess>10* cut, in terms of *Accuracy, Precision, Recall* and *F1* scorers.

The choice of *naccess* as a cut to compare classifiers among each other is quite a natural one, as the definition of popularity - to be discussed later - will definitely be connected to the application of a cut (eventually, a combined cut) that must include at least this observables. On the basis of the Tables and Figures shown above, it can be seen that some scorers - like *GaussianNB* and *Ridge Classifier* - yield very high FPR. These must be excluded in this study, but there is no strong reason

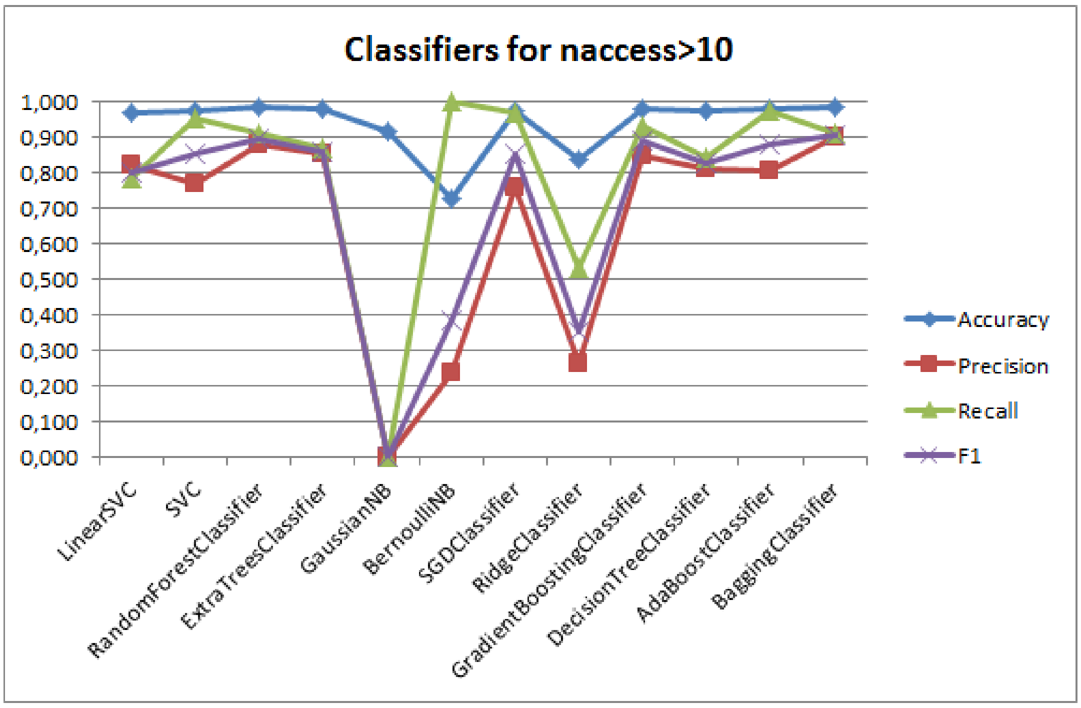
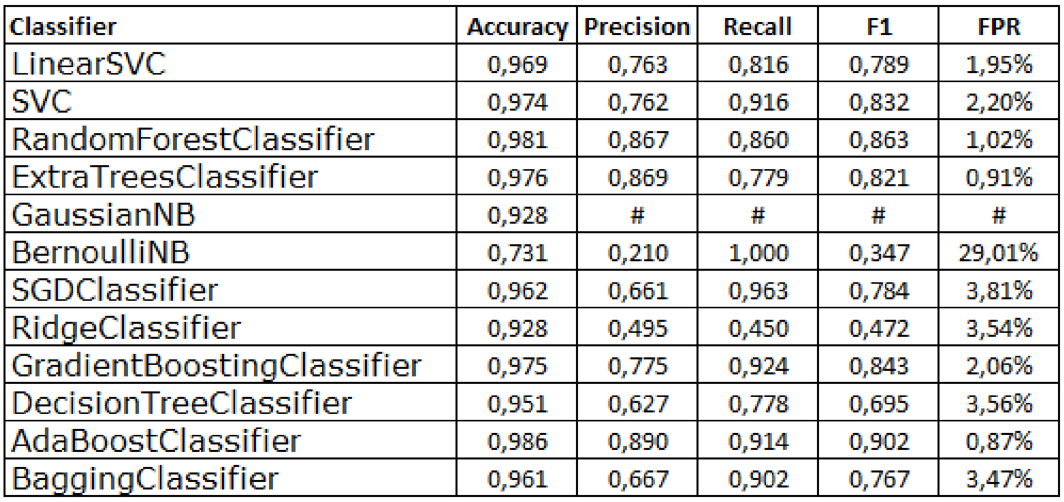


**Figure 5.2:** Graphical view of the results from the performance studies of different ML classifiers upon applying a *naccess>10* cut (same data as from the corresponding Table), in terms of different scorers.



| Classifier | Accuracy | Precision | Recall | F1 | FPR |
|---|---|---|---|---|---|
| LinearSVC | 0,969 | 0,763 | 0,816 | 0,789 | 1,95% |
| SVC | 0,974 | 0,762 | 0,916 | 0,832 | 2,20% |
| RandomForestClassifier | 0,981 | 0,867 | 0,860 | 0,863 | 1,02% |
| ExtraTreesClassifier | 0,976 | 0,869 | 0,779 | 0,821 | 0,91% |
| GaussianNB | 0,928 | # | # | # | # |
| BernoulliNB | 0,731 | 0,210 | 1,000 | 0,347 | 29,01% |
| SGDClassifier | 0,962 | 0,661 | 0,963 | 0,784 | 3,81% |
| RidgeClassifier | 0,928 | 0,495 | 0,450 | 0,472 | 3,54% |
| GradientBoostingClassifier | 0,975 | 0,775 | 0,924 | 0,843 | 2,06% |
| DecisionTreeClassifier | 0,951 | 0,627 | 0,778 | 0,695 | 3,56% |
| AdaBoostClassifier | 0,986 | 0,890 | 0,914 | 0,902 | 0,87% |
| BaggingClassifier | 0,961 | 0,667 | 0,902 | 0,767 | 3,47% |

**Figure 5.3:** Results of performance studies of different ML classifiers upon applying a *naccess>60* cut, in terms of *Accuracy, Precision, Recall* and *F1* scorers.

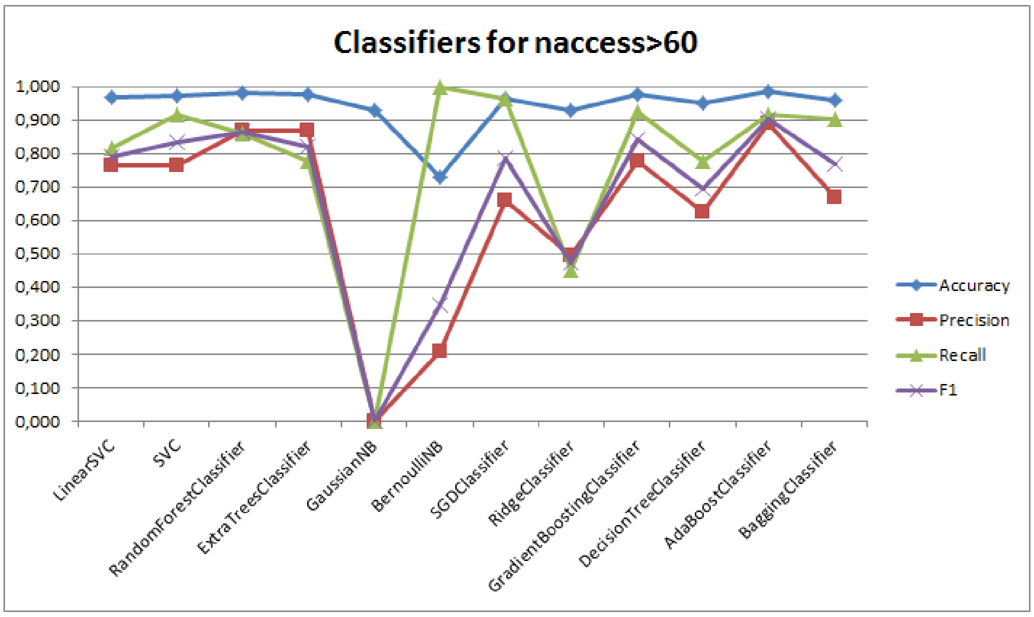


**Figure 5.4:** Graphical view of the results from the performance studies of different ML classifiers upon applying a *naccess>60* cut (same data as from the corresponding Table), in terms of different scorers.

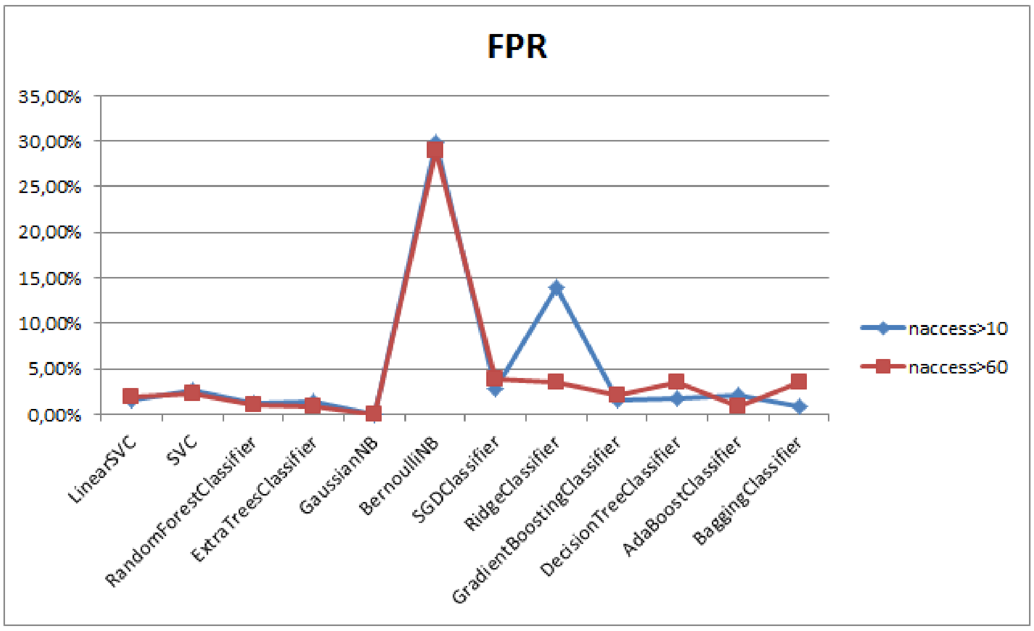


**Figure 5.5:** FPR variability as a function of different classifiers, displayed for both *naccess>10* (blue dots) and *naccess>60* (red squares) cuts.

in favour of adopting one or another of the remaining ones. Given the amount of positive feedback in the literature on *RandomForestClassifier*, and given it seems one of the most adequate given the time of the problem we study and the statistics we can rely on, we decided to adopt *RandomForestClassifier* in this study. All the results shown later in this chapter are hence obtained with such classifier.

## 5.2 Choice of the best dataframes splitting

In order to create a model and to verify its reliability, we have to split our datasets into three different sets:

- **training** set is used to build the model. It contains a set of data that has preclassified target and predictor variables;
- **validation** set is used to adjust the model, tuning internal model parameters;
- **test** set is used to apply our chosen prediction algorithm on it, in order to see how it is performing and thus have an idea about our algorithm's performance on data the algorithm has never seen before.

It is not easy to choose which is the best way to split our overall set of data into training vs validation vs test sets. Two important ML concepts need to be taken into account: *overfitting* and *cross validation*, which are briefly discussed in the following subsections.

### 5.2.1 Overfitting

In statistics and in Machine Learning, overfitting [34] occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as have too many parameters against the number of observations. A model that has been overfit will generally have poor predictive performance, as it can exaggerate in minor fluctuations in the data. The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. Indeed, a model is typically trained by maximizing its performance on some set of training data, while its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to "memorize" training data rather than "learning" to generalize from trend.

The potential for overfitting depends not only on the number of parameters and data but also the conformability of the model structure with the data shape, and the magnitude of model error compared to the expected level of noise or error in the data.

However, especially in cases where learning was performed too long or where training examples are rare, the learner may adjust to very specific random features of the training data, that have no causal relation to the target function. In this process of overfitting, the performance on the training examples still increases while the performance on unseen data becomes worse.

Both in statistics and in Machine Learning, in order to avoid overfitting, it is necessary to implement some particular techniques, such as *cross-validation*, that shows when a further training does not produce a better generalization.

### 5.2.2 Cross-validation

Cross-validation [33] is a statistics technique used in cases in which there is a sufficiently large number of available dataframes. In particular the *k-fold cross-validation* consists on subdivision of the whole dataset in k parts of the same numerosity and, at each passage, the (1/k)th part of dataset become the *validation dataset*, while the remaining part become the *training dataset*. In this way for each k-part, the model will be trained, avoiding overfitting problems, but also asymmetric sampling (and so affected by bias) of *training dataset*, typical of dataset subdivision in only two parts (or training and validation datasets). In other words, the data sample is divided in groups of the same dataframes quantity, excluding iteratively a group at time, trying to predict it through non-excluded groups.

### 5.2.3 Cross validation on DCAFPilot

So there are two possibilities in order to split dataframes. Either we split datasets into 60%/20%/20% for training/validation/test sets or we use the same group of data for validation and test through a 70%/30% split. Because of the fact that we use only 2014 dataframes, it is better to use 70%/30% split in order to have more statistics for training and validation, using anyway cross-validation to avoid overfitting. Thus subsequently will be shown which is the behaviour of *Accuracy*, *Precision*, *Recall*, *F1* and *FPR* through a table (Figure 5.6) and plots (Figure 5.7, Figure 5.8), starting from using 2014 and split it into first 2014 four weeks for train set and the rest of 2014 for valid set. The next passages is obtained adding 4 weeks at time to the train set, subtracting them from the valid set. In previous table and plots we refer to the percentage of 2014 dataframes belonging to train and valid sets. Values are also referred to *naccess>100* cut. Observing all the parameters, it is possible to say that increasing dataframes into train set *Accuracy* slowly oscillates around 0.95%, *Precision* overcoming 50%/50% subdivision decreases, *Recall* and *F1* increases, FPR starting from 53.8% train increases very quickly. Because of we are interested not only on FPR (that behind has economic motivation) but above all on a good level of predictability, something close to 70%/30% split for train and valid sets seems to be the choice that optimizes as much as possible all the parameters.

As from this study, we opted for a 70% training set size and a 30% valid set size, and we stick to this throughout all the studies presented in the rest of the work of this thesis.

| | Accuracy | Precision | Recall | F1 | FPR |
|---|---|---|---|---|---|
| Train 7,7%, Valid 92,3% | 0,953 | 0,948 | 0,362 | 0,524 | 0,15% |
| Train 15,4%, Valid 84,6% | 0,948 | 0,980 | 0,283 | 0,439 | 0,04% |
| Train 23,1%, Valid 76,9% | 0,948 | 0,953 | 0,300 | 0,456 | 0,11% |
| Train 30,8%, Valid 69,2% | 0,953 | 0,917 | 0,395 | 0,552 | 0,28% |
| Train 38,5%, Valid 61,5% | 0,959 | 0,916 | 0,489 | 0,637 | 0,36% |
| Train 46,2%, Valid 53,8% | 0,964 | 0,956 | 0,531 | 0,683 | 0,19% |
| Train 53,8%, Valid 46,2% | 0,959 | 0,937 | 0,463 | 0,619 | 0,24% |
| Train 61,5%, Valid 38,5% | 0,976 | 0,869 | 0,756 | 0,809 | 0,82% |
| Train 69,2%, Valid 30,8% | 0,982 | 0,869 | 0,849 | 0,859 | 0,91% |
| Train 76,9%, Valid 23,1% | 0,981 | 0,864 | 0,850 | 0,857 | 0,96% |
| Train 84,6%, Valid 15,4% | 0,964 | 0,746 | 0,671 | 0,706 | 1,57% |
| Train 92,3%, Valid 7,7% | 0,974 | 0,741 | 0,807 | 0,772 | 1,66% |

**Figure 5.6:** Values of *Accuracy, Precision, Recall, F1* and FPR in cases of different 2014 datasets split into train and valid sets.

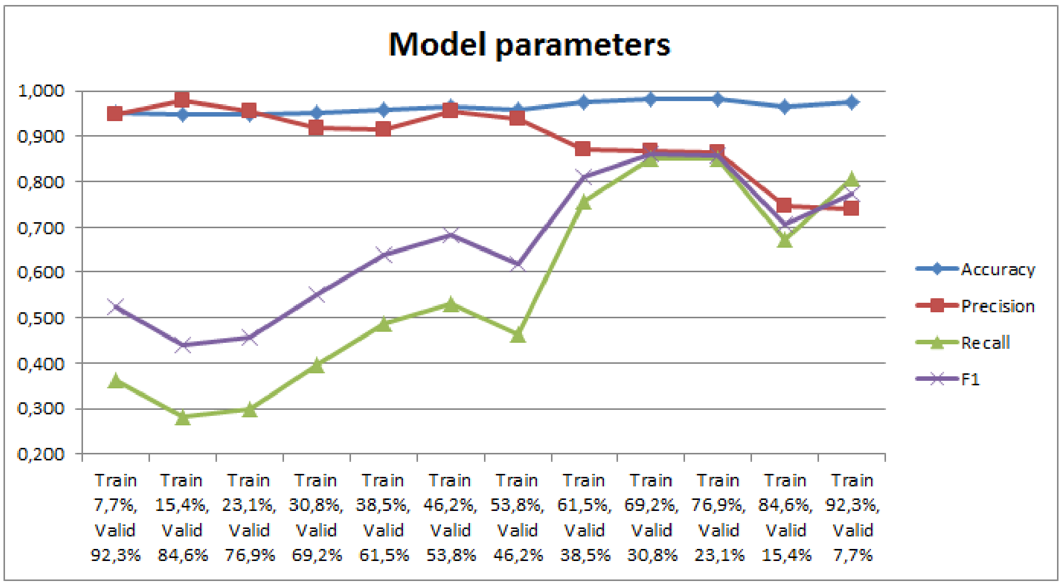


**Figure 5.7:** *Accuracy, Precision, Recall, F1* trend in cases of different 2014 datasets split into train and valid sets.

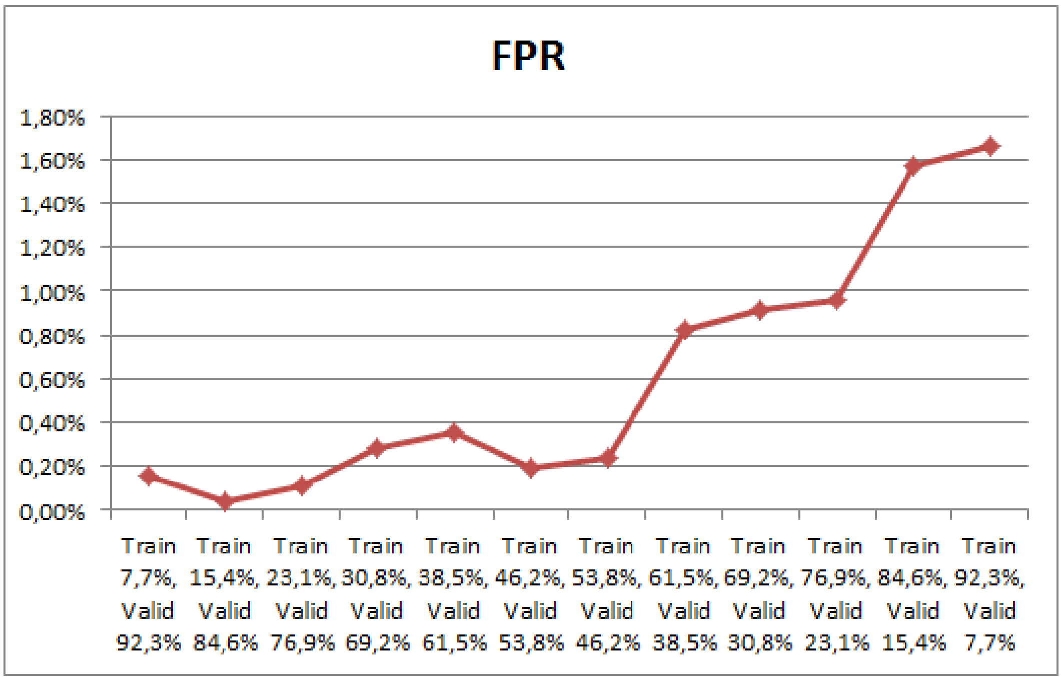


**Figure 5.8:** FPR trend in cases of different 2014 dataset split into train and valid sets.

## 5.3 Definition of popularity metrics

In this section we refer to "cuts" as the selection applied to the available set of variables in order to apply a certain definition of data popularity. The available variables that are primarily used in this analysis are:

- **naccess**: number of accesses to a CMS dataset
- **totcpu**: total CPU time (in hours) spent on running on a CMS dataset
- **nusers**: number of users accessing a CMS dataset

One may define the popularity in different ways. From a CMS data management standpoint, a dataset may be defined "popular" if it attracts a lot of accesses. From a computing facilities standpoint, a dataset may be defined "popular" if a lot of CPU hours are kept busy while serving this dataset to analysis users. From a user community perspective, a dataset may be defined "popular" if a lot of users access it via the CMS distributed analysis tools. So, depending on which is the choice of the actual popularity metrics, its definition may be different, and different subsets of CMS datasets may be popular according to one definition, and less popular according to another one. But it is crucial to mention at this point that, from the standpoint of a ML-based analysis of the problem - despite it may sound contradictory - the actual chosen definition of popularity does not matter much in itself: of course it needs to be reasonable, but what matter is that a definition is taken that allows the DCAFPilot system to predict – on the basis of that definition - what will be popular or not and keep a low FPR while doing it. To reach this goal, it is important to study the characteristic of the aforementioned variables and the effects of different popularity definition cuts in some details. This will be done in the following sections, exploiting three main popularity metrics, using a set of dataframes that corresponds to roughly the metadata from the first three quarters of year 2014 for the part of general explanation of datasets behaviour relative to each metric. At the same time we analize single cuts in which now we use a set of dataframes that corresponds to whole 2014 metadata, splitted into 70%/30% for train/valid sets using *RandomForestClassifier* as classifier. Here we pay attention first of all to minimize FPR but at the same time we want that *%TP* (where %TP is equal to ratio between TP and the number of whole valid datasets) is enough high so that our model also predicts correctly significant quantities of data. Secondly we also want that *FNR* (where $FNR = FN/(FN + TP)$) is quite low because also FN has a cost, albeit in a reduced way, and latencies.

### 5.3.1 Studies of single cuts

#### Number of access

The number of accesses (*naccess* in the following) is a golden variable in our set as it counts the individual accesses to the CMS datasets and it is among the most easy and natural metric to be used in order to quantify the actual popularity of a dataset. Its distribution in terms of probability density is shown in Figure 5.9.

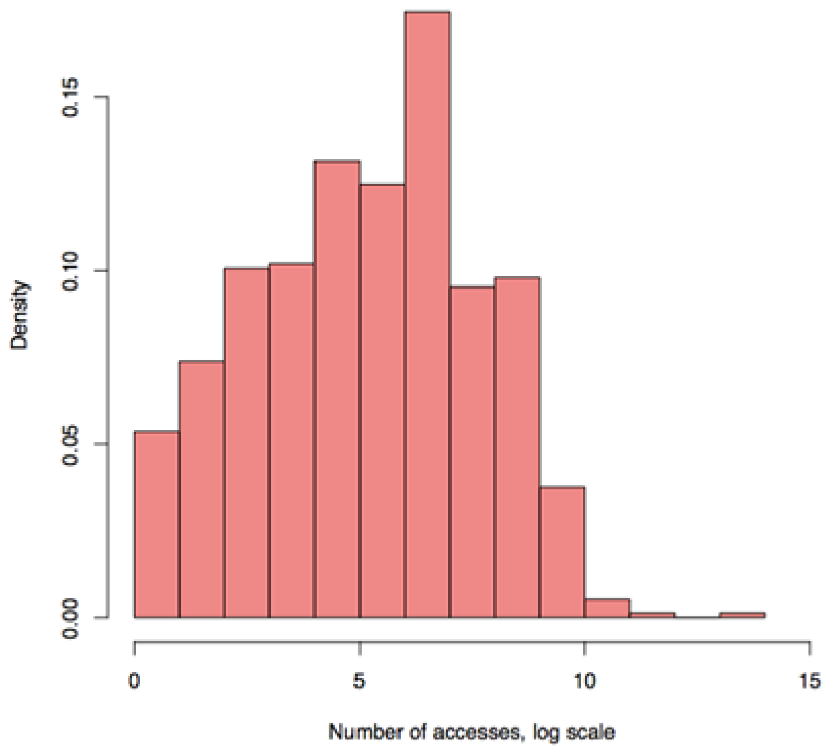
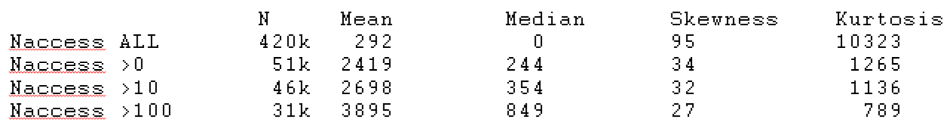


**Figure 5.9:** *naccess* distribution in terms of probability density.

Over the 420k datasets quoted above (corresponding to datasets of the first three quarters of year 2014), *naccess* had mean $\sim 292$, median 0, skewness 95 and kurtosis 10323 (we do not quote errors because is not relevant for the discussion).

The global distribution is right-skewed (positive skewness indicates that the mean of the data values is larger than the median) and leptokurtic (positive kurtosis indicates a peaked distribution). As can be seen for the median value, indeed, the vast majority of the datasets were not accessed at all during the indicated period. In fact, only 50834 datasets out of 420k recorded at least 1 access, and – if datasets with *naccess=0* are excluded from the pool – the distribution becomes one with a larger mean ($\sim$ 2419), larger median (244), smaller skewness (34) and smaller kurtosis (1265), thus indicating a less peaked distribution. As expected, when applying more stringent cuts on *naccess*, the mean and median increase, the skewness decreases (the distribution becomes more symmetric) and the kurtosis decreases (the distribution becomes flatter). The value of all such statistical moments for no cuts on *naccess*, as well as for *naccess>0, >10* and *>100* are shown in Figure 5.10.



| | N | Mean | Median | Skewness | Kurtosis |
|---|---|---|---|---|---|
| Naccess ALL | 420k | 292 | 0 | 95 | 10323 |
| Naccess >0 | 51k | 2419 | 244 | 34 | 1265 |
| Naccess >10 | 46k | 2698 | 354 | 32 | 1136 |
| Naccess >100 | 31k | 3895 | 849 | 27 | 789 |

**Figure 5.10:** Statistical moments for several *naccess* cuts.

In terms of popularity metric definition, it is tempting to declare that *naccess* for a dataset must exceed an high value to justify that such dataset is indeed “popular”. But there is nothing solid apart from a random way to fix a threshold for this. Now using whole 2014 dataframes in the way previously explained, the DCAF model was run for a wide set of reasonable choices of *naccess* cuts. Fixing n=178462, that represents the number of datasets contained into validation set, we represent in the following plots %TP (that is %TP=TP/n, Figure 5.15), %TN (that is %TN=TN/n, Figure 5.16), %FP (that is %FP=FP/n, Figure 5.17), %FN (that is %FN=FN/n, Figure 5.18), FPR (Figure 5.19) and FNR (Figure 5.20).

We can see that the majority of datasets are correctly predicted as TN. Indeed, we know from CMS Operations that most of the CMS datasets are residing on disks at Computing centres and are not accessed so frequently, so this is correct. The same information, as it is obviously correlated, is shown in the %TP plot (Figure 5.15). The results show that if the *naccess* cut ($naccess > n$) moves to larger n, we are defining as “popular” those datasets which have a high number of accesses, thus resulting in a lower %TP and a higher %TN (Figure 5.16). On the other hand, whenever we cut on *naccess*, the fraction of FP (Figure 5.17) is stable around 0.6-1.4%, and similar for %FN (around 0.6-1.8, Figure 5.18): this shows that the fraction of datasets for which the model gives a wrong predictions (in both ways) is only slightly sensible to the *naccess* threshold.

In Figure 5.19 and 5.20, the FPR, FNR rates are shown. As explained earlier, we focus primarily on the FPR, with an eye also on the FNR. In terms of the *naccess* cut alone, we observed that FPR oscillates between 0.6 and 1.5% for different values of n. This would indicate that any value of n would leave us with a relatively “small” cost to pay in terms of FPR, which is good - so we can choose our preferred operational point. On the other hand, looking at FNR, we observe than larger values of n cause FNR to raise from about 8% to up to 30%. Despite this is not our primary metric of interest, given that any value of n is safe from a FPR standpoint, it is hence preferable to choose a smaller value of n so that it minimizes the FNR as well. Looking at both plots altogether, each cut under n=40 (or similar) seems a reasonable choice.

### Number of users

The number of users (*nusers* in the following) is another interesting variable to study, which counts the individuals who used their own Grid certificate through the CRAB distributed analysis toolkit to access the CMS datasets. Its distribution in terms of probability density is shown in Figure 5.11.

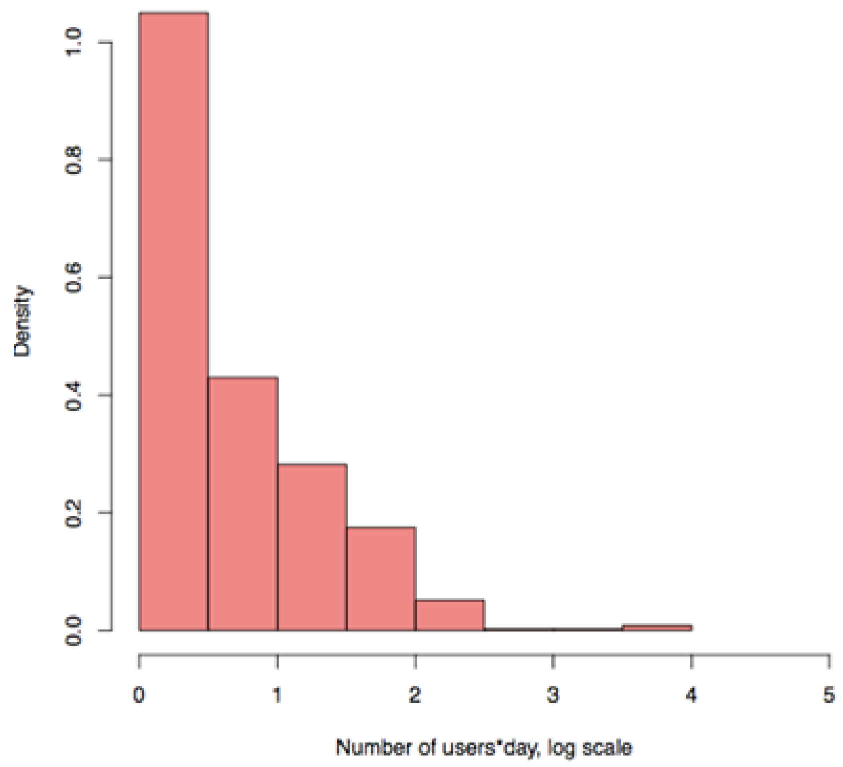
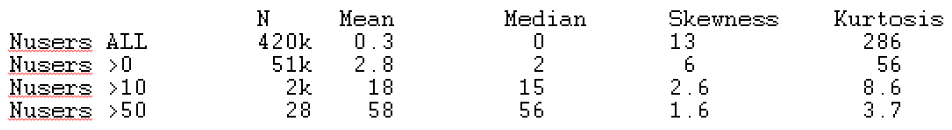


**Figure 5.11:** *nusers* distribution in terms of probability density.

It seems obvious to think of an high number of users accessing a dataset as a reasonable measure of the actual popularity of such dataset. On the other hand, it is sometimes not so straightforward: in CMS Physics groups it is quite frequent that very few "power users" submit jobs to the Grid for the entire team: for example, a large fraction of massive Grid jobs submissions for the analyses in the Higgs working group were performed by just few PhD students or young researchers to collect pre-analyzed data to share with all their colleagues in the team. In other words, also in this case choosing a good threshold may not be as easy as one may think.

Checking over the total of 420k datasets in the whole set of dataframes used in this study (first three quarters of 2014), *nusers* had mean $\sim 0.3$, median 0, skewness 13 and kurtosis 286. Of course, the vast majority of *naccess=0* in the previous section is correlated with *nusers=0* in this section. In fact, only $\sim 51k$ datasets out of 420k recorded at least 1 individual accessing the dataset, and – if datasets with *nusers=0* are excluded from the pool – the distribution has a larger mean and median, and a smaller skewness and kurtosis. Applying more stringent cuts on *nusers*, the mean and median increase further, and the skewness and the kurtosis decrease further. In this specific case, though, it does not make much sense to increase n in *nusers>n* a lot, as the dataset which recorded the maximum amount of users accessing it is around 83 individuals: a threshold at 50 was investigated, resulting in a too low statistics samples to be actually meaningful. The values of the statistical moments for no cuts on *nusers*, as well as for *nusers>0, >10* and *>50* are shown in Figure 5.12.



| | N | Mean | Median | Skewness | Kurtosis |
|---|---|---|---|---|---|
| Nusers ALL | 420k | 0.3 | 0 | 13 | 286 |
| Nusers >0 | 51k | 2.8 | 2 | 6 | 56 |
| Nusers >10 | 2k | 18 | 15 | 2.6 | 8.6 |
| Nusers >50 | 28 | 58 | 56 | 1.6 | 3.7 |

**Figure 5.12:** Statistical moments for several *nusers* cuts.

We move now to analyse the *nusers* cut alone in terms of popularity metric definition, as we did so far with the *naccess* cut alone (using whole 2014 datasets). We see that all trends (Figures 5.21, 5.22, 5.23, 5.24, 5.25, 5.26)becomes very stable if n is greater than some units. It is evident that the *nusers* metrics may not be the appropriate one to cut against, as it usually peaks at very low values of n, and becomes flat soon after (even if FNR in Figure 5.26 increases regularly with n). In this study at this stage it seems that we may not be using *nusers* as a variable in a combined cut, despite some combined plot are shown (see later for more details).

## Total CPU hours

The total number of CPU hours spent to analyse a CMS dataset (*totcpu* in the following) is a third and last interesting variable to consider. This gives the computing facilities perspective of popularity, namely a dataset is more popular than others if the CPUs at Grid sites have been used to access it more than others. Its distribution in terms of probability density is shown in Figure 5.13.

Again, an high value of *totcpu* stands a reasonable choice to reflect the actual popularity of a dataset. Over the total of 420k datasets in the whole set of first three quarters 2104 dataframes used in this study, *totcpu* had mean 226, median 0, skewness 29 and kurtosis 1231. As in the two previous studies, the vast majority of *naccess=0* and *nusers=0* are correlated with *totcpu=0* in this section. Looking at actual figures, fact, only $\sim 47k$ datasets out of 420k caused at least some CPU hours to be spent in analyzing them, – if datasets with *totcpu=0* are excluded from the overall sample – the distribution has a much larger mean, a larger median, and smaller skewness and kurtosis. Applying more stringent cuts on *totcpu*, as observed before, the mean and median increase further, and the skewness and the kurtosis decrease further – as expected. In this specific case, though, it does make sense to increase n in *totcpu>n* quite a bit, as even after a cut at n=500, still about 11k datasets are left in the sample. The values of the statistical moments for no cuts on *totcpu*, as well as for *totcpu>0, >10, >50, >100, >200* and *>500* are shown in Figure 5.14.

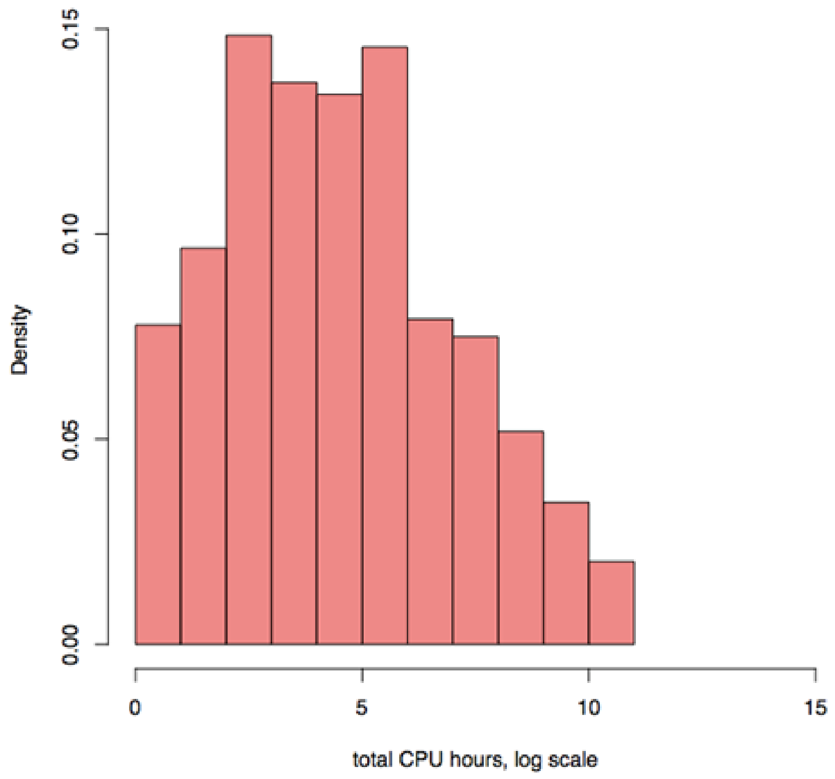
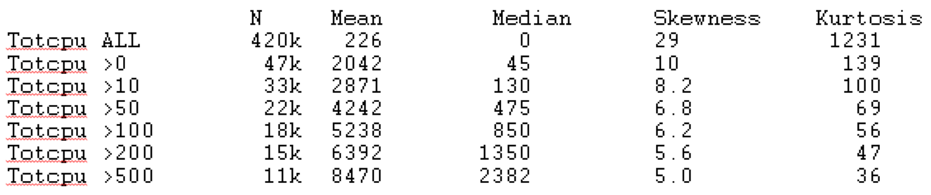


**Figure 5.13:** *totcpu* distribution in terms of probability density.



| | N | Mean | Median | Skewness | Kurtosis |
|---|---|---|---|---|---|
| Totcpu ALL | 420k | 226 | 0 | 29 | 1231 |
| Totcpu >0 | 47k | 2042 | 45 | 10 | 139 |
| Totcpu >10 | 33k | 2871 | 130 | 8.2 | 100 |
| Totcpu >50 | 22k | 4242 | 475 | 6.8 | 69 |
| Totcpu >100 | 18k | 5238 | 850 | 6.2 | 56 |
| Totcpu >200 | 15k | 6392 | 1350 | 5.6 | 47 |
| Totcpu >500 | 11k | 8470 | 2382 | 5.0 | 36 |

**Figure 5.14:** Statistical moments for several *totcpu* cuts.

Despite a larger set of tests is needed in this case, in terms of popularity metric definition a similar study previously done for *naccess* and *nusers* will be applied as well for *totcpu* using whole 2014 datasets. As the case of *naccess*, %TP (Figure 5.27) decrease and %TN (Figure 5.28) increase with n, but here %FP (Figure 5.29) decreases quickly from 2.5% up to about 0.25% starting from about *totcpu>50* and then it oscillates. This oscillation at high n values is not too significant because it is only a statistic effect (long tails of distribution where a reduced datasets fraction is acceded using a lot of CPU, but in general their contribution is not relevant). Besides %FN (Figure 5.30)increases from about 0.2% up to about 1.4% quite quickly with some oscillations and then from about *naccess>100* cut %FN decrease. At the same time %FPR and %FNR(Figures 5.19, 5.32) has a trend quite similar to %FP and %FN. Analysing all these trends we can say that from %TP plot for example each cut with *totcpu* lower than 40 is quite good (even if %TP became half of its value at *totcpu>0* cut), from %FPR plot cut at high value of *totcpu* is better, from %FNR plot cut at low value of *totcpu* is better. So in order to use *totcpu* as a popularity metric, a reasonable choice of the cut is that each cut below *totcpu>40* could be fine.

### 5.3.2 Study of combined cuts

In order to find out the best cut to apply to the machinery, in this section the results about combined cuts will be exposed.

#### naccess & nusers

In this section we study a combined on *naccess* and *nusers*. The trend of %TP(Figure 5.33), FPR (Figure 5.34) and FNR (Figure 5.35) does not depends on *naccess>n* cut for n greater than 7. For any cut on *nusers>n* with n>=2, whatever *naccess* cut is applied %TP goes under 1.5% and so there is no reason to do this kind of cut due to the too low level of datasets predicted as popular. At the same time FPR goes under 2% (that is a quite good target) but FNR goes over 40%. So, as we have already discussed in the single cut on *nusers*, a combined cut with *nusers* is not a good choice because, as we can see from these plots, there is not a cut that at the same time give us good %TP, FPR, FNR levels.

#### nusers & totcpu

In this combined cut, overcome about *nusers>3* cut for %TP (Figure 5.36), *nusers>6* cut for FPR (Figure 5.37)the trend does not depend on *totcpu* cut, while for FNR (Figure 5.38) the trend depends, although weakly, by *totcpu* cut. For any cut on *nusers>n* with n>=2 %TP goes under about 1.5% and so there is no reason to do this kind of cut due to the too low level of datasets predicted as popular. At the same time FPR goes under 2% (that is a quite good target) but FNR goes over about 40%. So, as we have already discussed in the single cut on *nusers* and into *naccess* & *nusers* cut, a combined cut with *nusers* is not a good choice because, as we can see from these plots, there is not a cut that at the same time give us good %TP, FPR, FNR levels.

#### naccess & totcpu

As we can see in the FPR plot (Figure 5.40) if we apply a *naccess>10* cut or each other cut with *naccess>10* (i.e. *naccess>20*, *naccess>30* etc.), *totcpu>10*, *totcpu>15* and *totcpu>20* cuts goes under 1.20% and they are about always under FPR level of *totcpu>0* and *tocpu>5* cuts. Instead in FNR plot (Figure 5.41), we can see that between the three previous choice *totcpu>10* has always the lowest level of FNR up to *naccess>50* cut. As we want also the value of %TP (Figure 5.39) as high as possible, it is better to do a lower cut on *naccess*, and for each *naccess* cut we have that *totcpu>10* cut has an higher %TP value with respect to the others *totcpu>15* and *totcpu>20* cuts.

So through all plots of combined cuts it is possible to summarize that **naccess>10 & totcpu>10** cut is reasonably the best choice that give us the best configuration of %TP, FPR, FNR levels. Then the values of *Accuracy*, *Precision*,

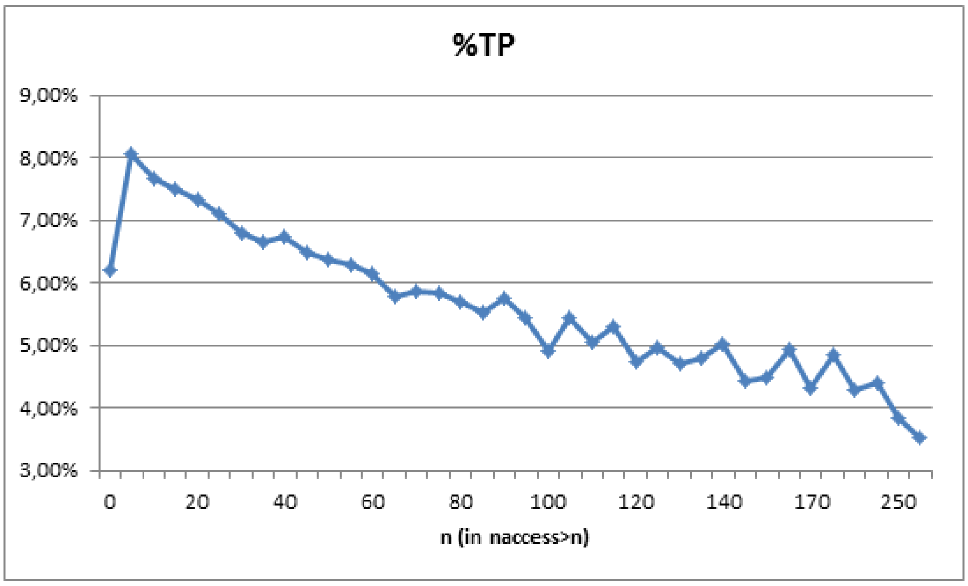


**Figure 5.15:** Fraction of all CMS datasets which are true positive (TP) for the *naccess* cut. This plot and the following ones always relate to positive and negative in terms of our machine learning study of dataset popularity.

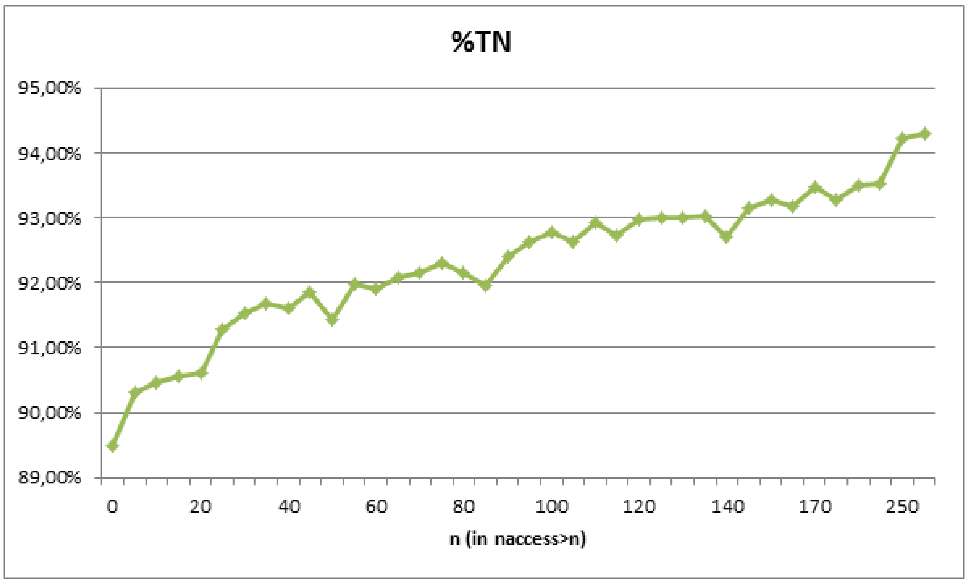


**Figure 5.16:** Fraction of all CMS datasets which are true negative (TN) for the *naccess* cut.

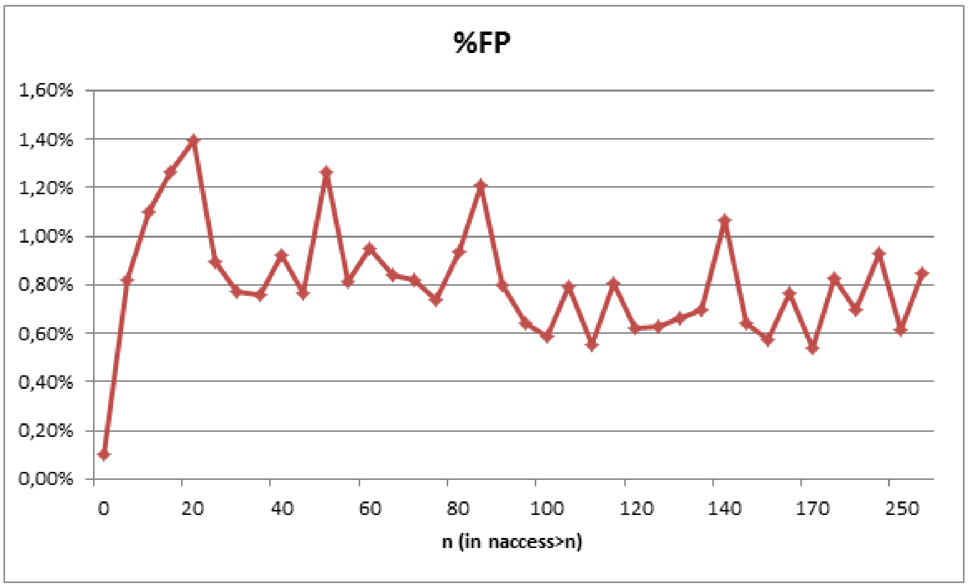


**Figure 5.17:** Fraction of all CMS datasets which are false positive (FP) for the *naccess* cut.

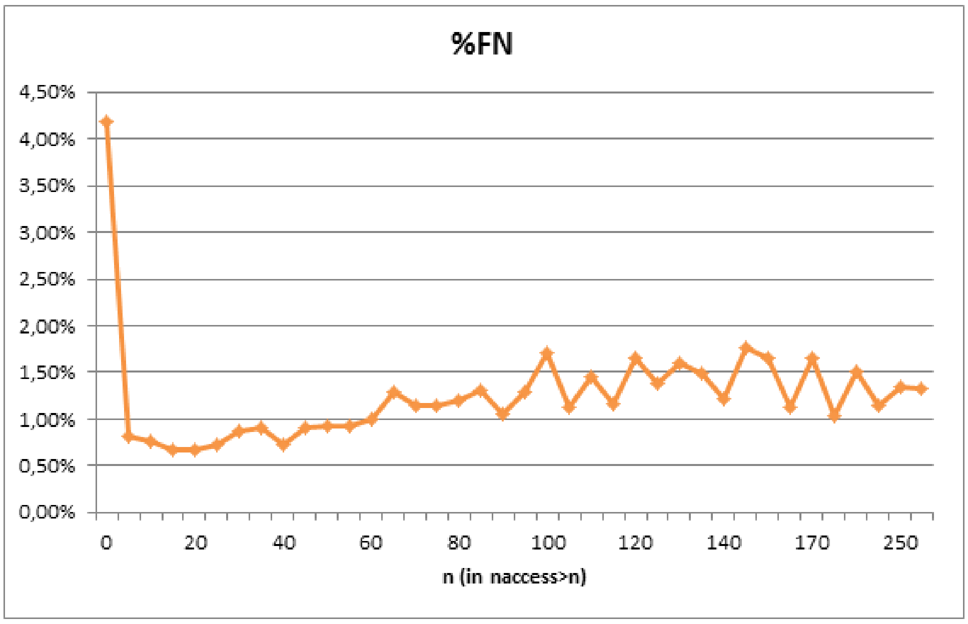


**Figure 5.18:** Fraction of all CMS datasets which are false negative (FN) for the *naccess* cut.

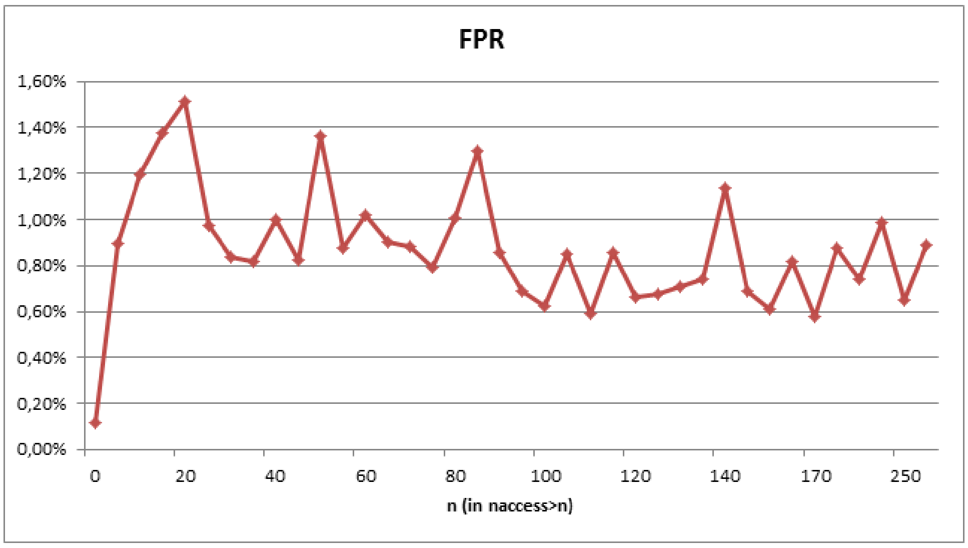


**Figure 5.19:** False positive rate (FPR) for the *naccess* cut.

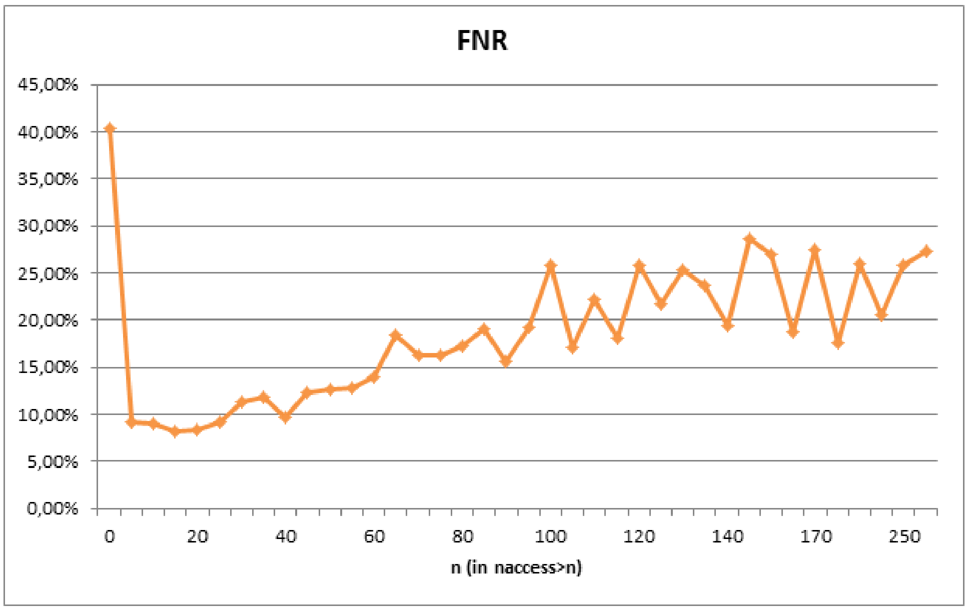


**Figure 5.20:** False negative rate (FNR) for the *naccess* cut.

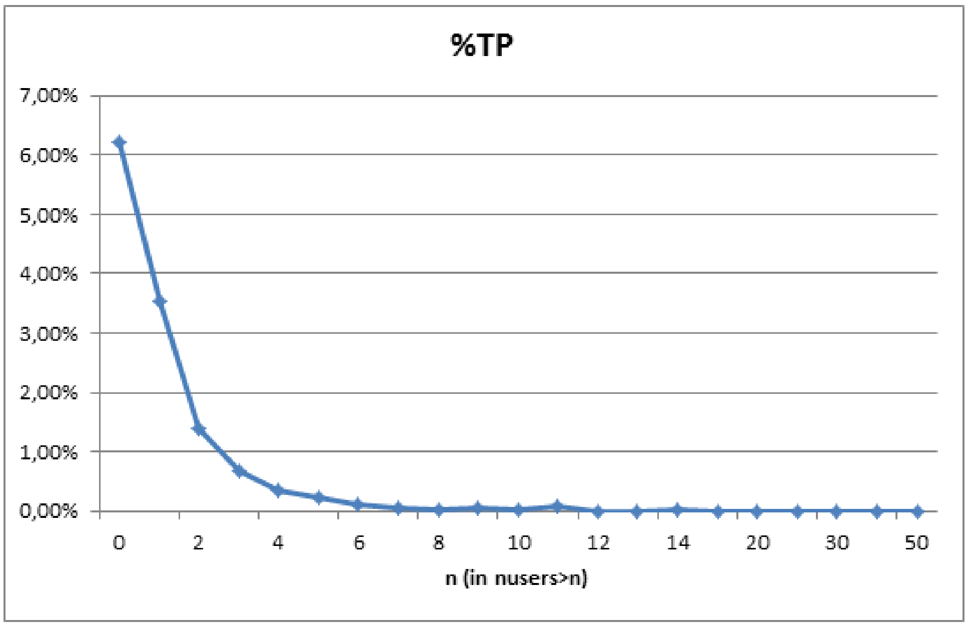


**Figure 5.21:** Fraction of all CMS datasets which are true positive (TP) for the *nusers* cut.

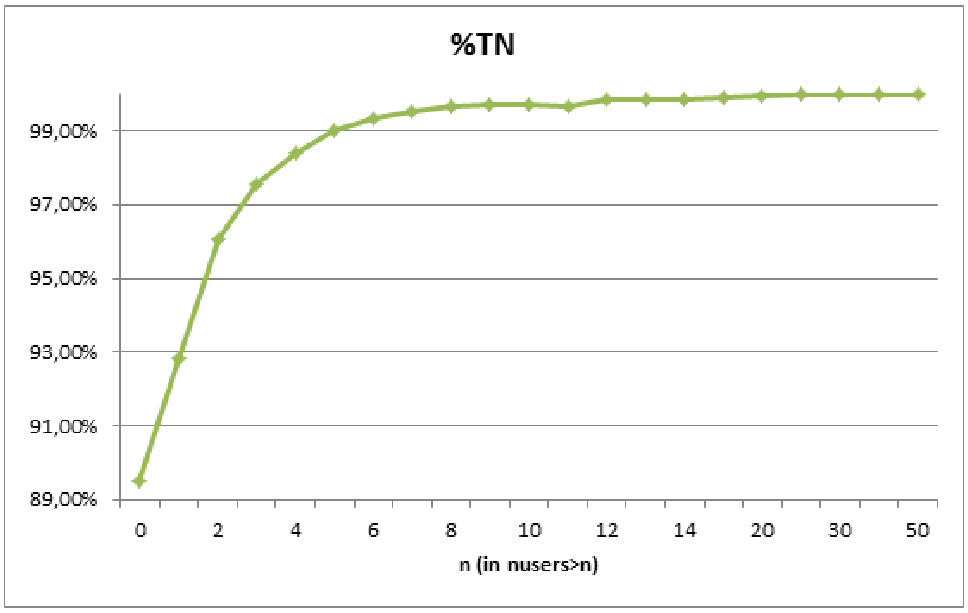


**Figure 5.22:** Fraction of all CMS datasets which are true negative (TN) for the *nusers* cut.

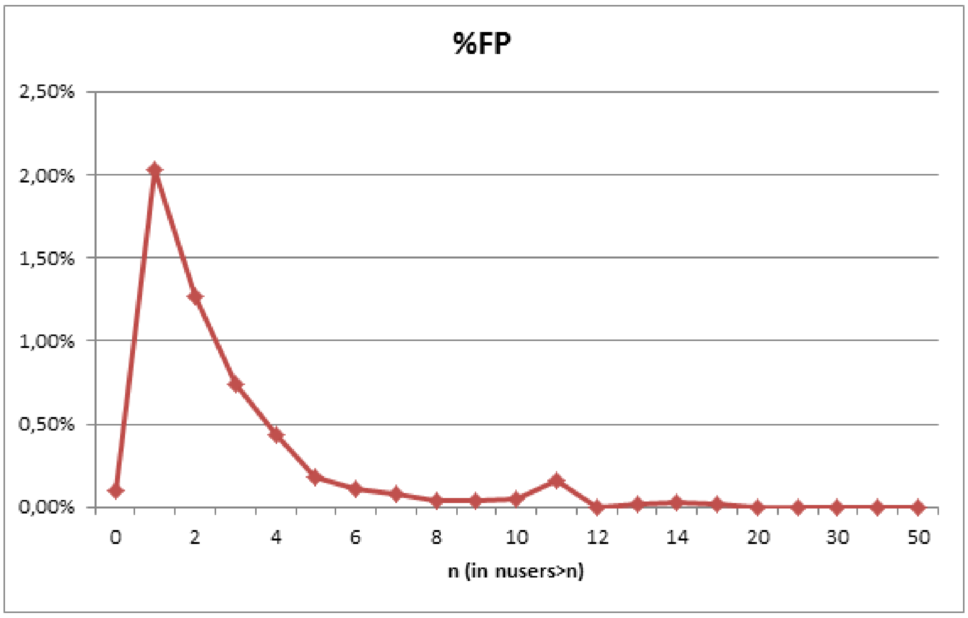


**Figure 5.23:** Fraction of all CMS datasets which are false positive (FP) for the *nusers* cut.

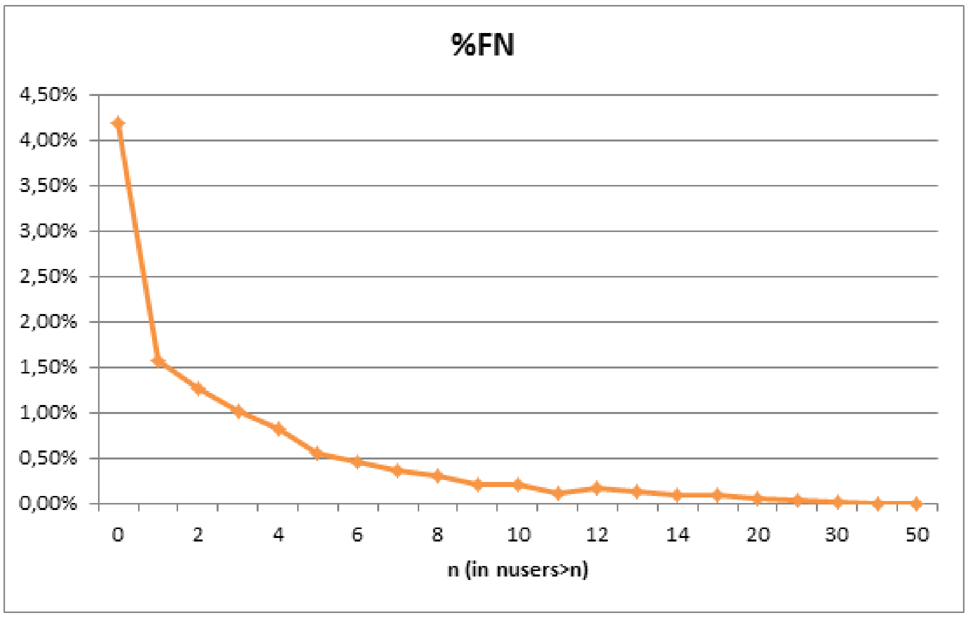


**Figure 5.24:** Fraction of all CMS datasets which are false negative (FN) for the *nusers* cut.

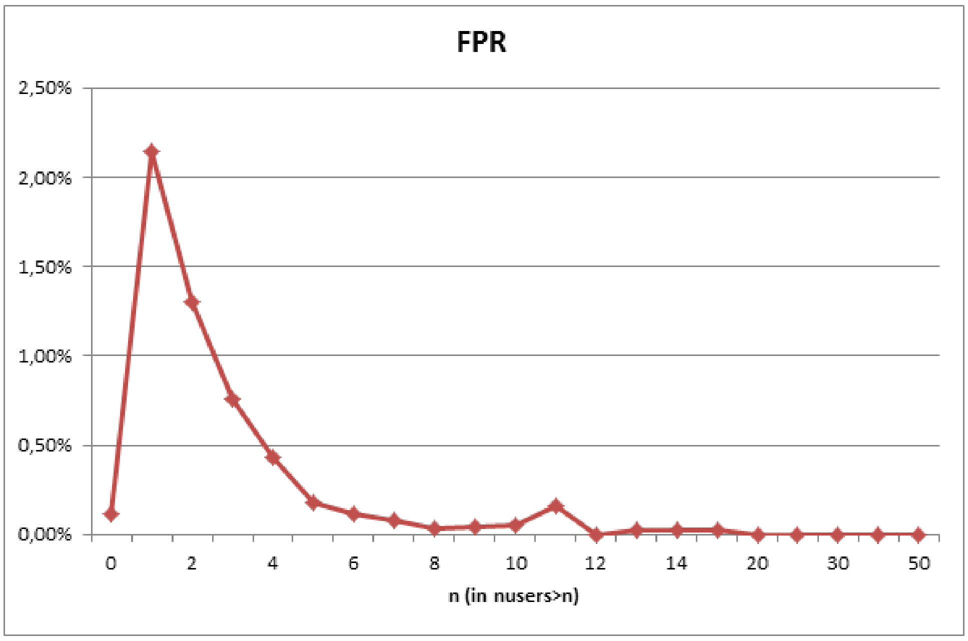


**Figure 5.25:** False positive rate (FPR) for the *nusers* cut.

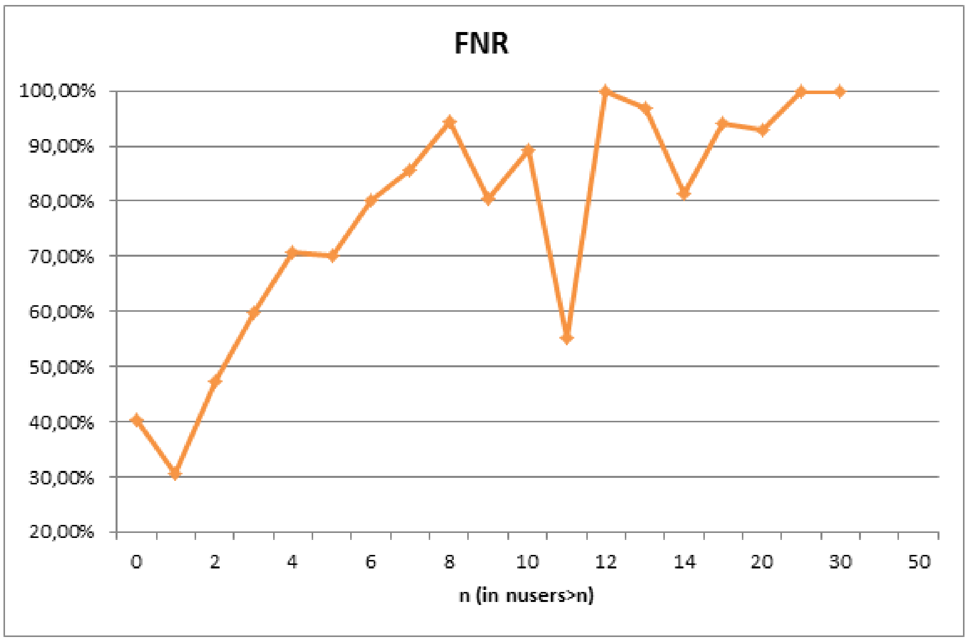


**Figure 5.26:** False negative rate (FNR) for the *nusers* cut.

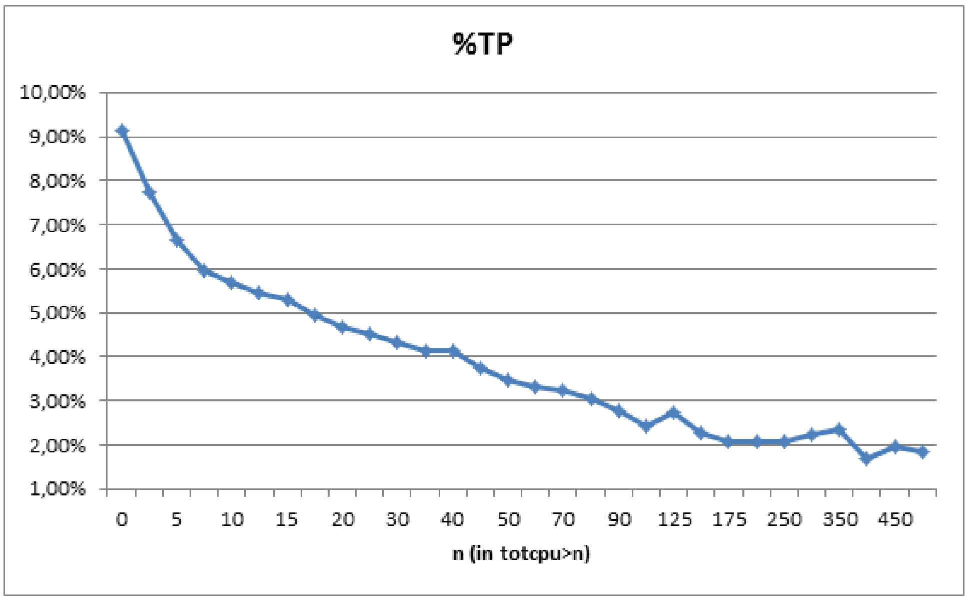


**Figure 5.27:** Fraction of all CMS datasets which are true positive (TP) for the *totcpu* cut.

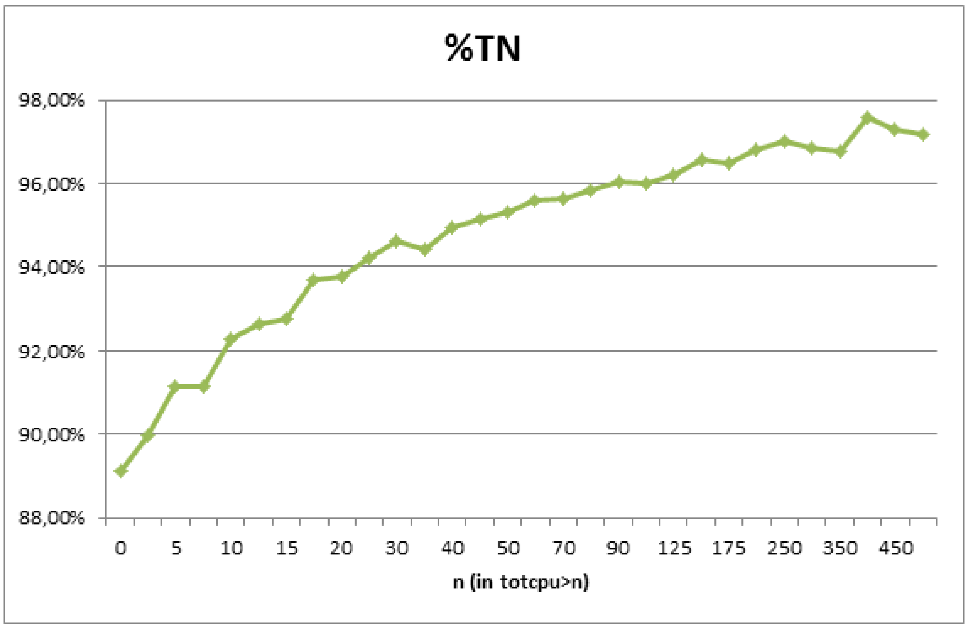


**Figure 5.28:** Fraction of all CMS datasets which are true negative (TN) for the *totcpu* cut.

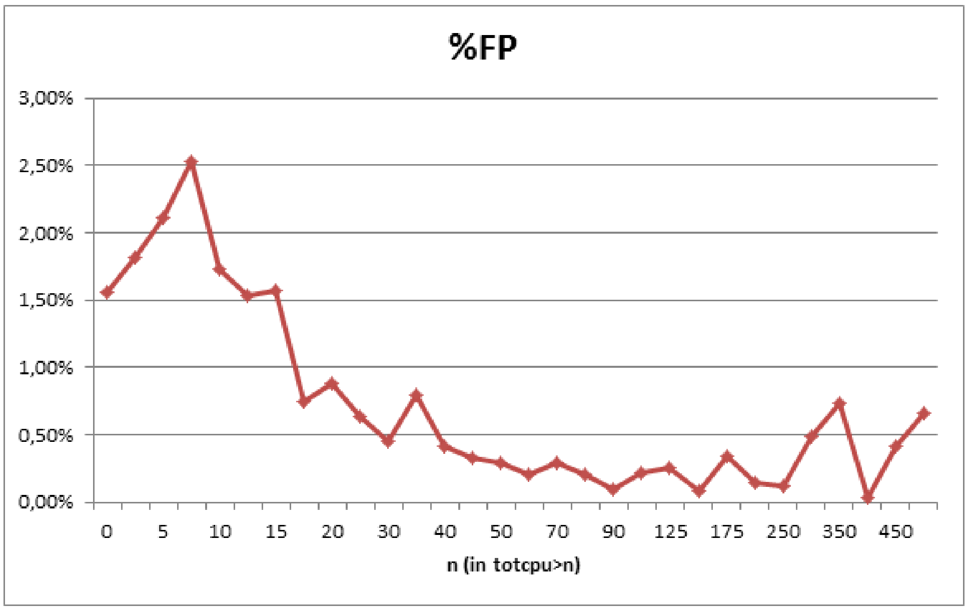


**Figure 5.29:** Fraction of all CMS datasets which are false positive (FP) for the *totcpu* cut.

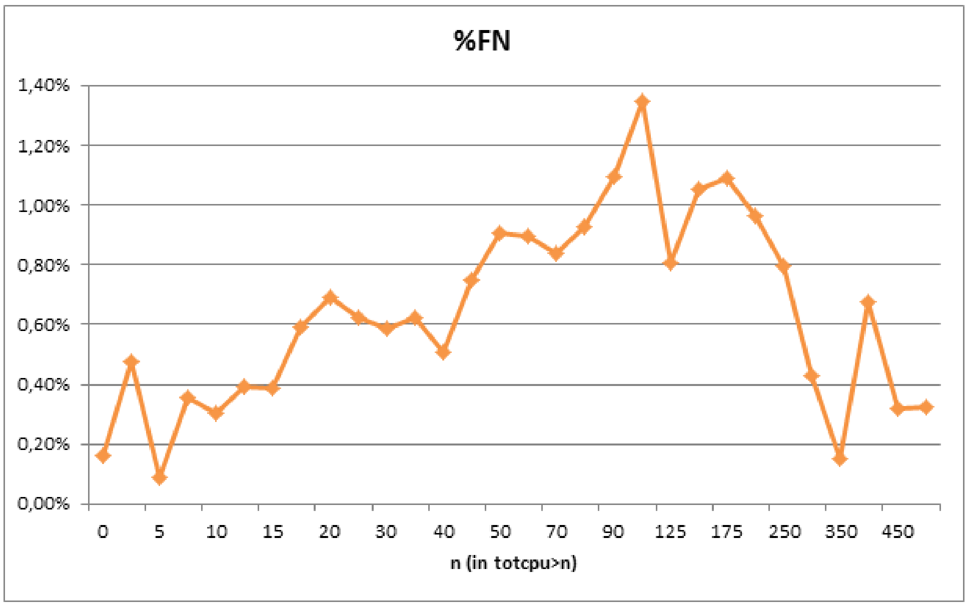


**Figure 5.30:** Fraction of all CMS datasets which are false negative (FN) for the *totcpu* cut.

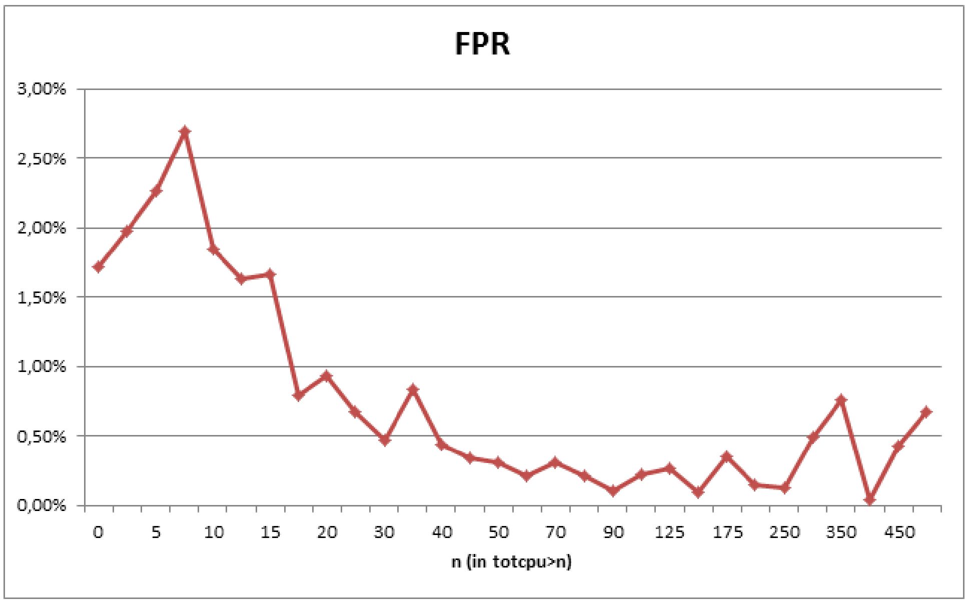


**Figure 5.31:** False positive rate (FPR) for the *totcpu* cut.

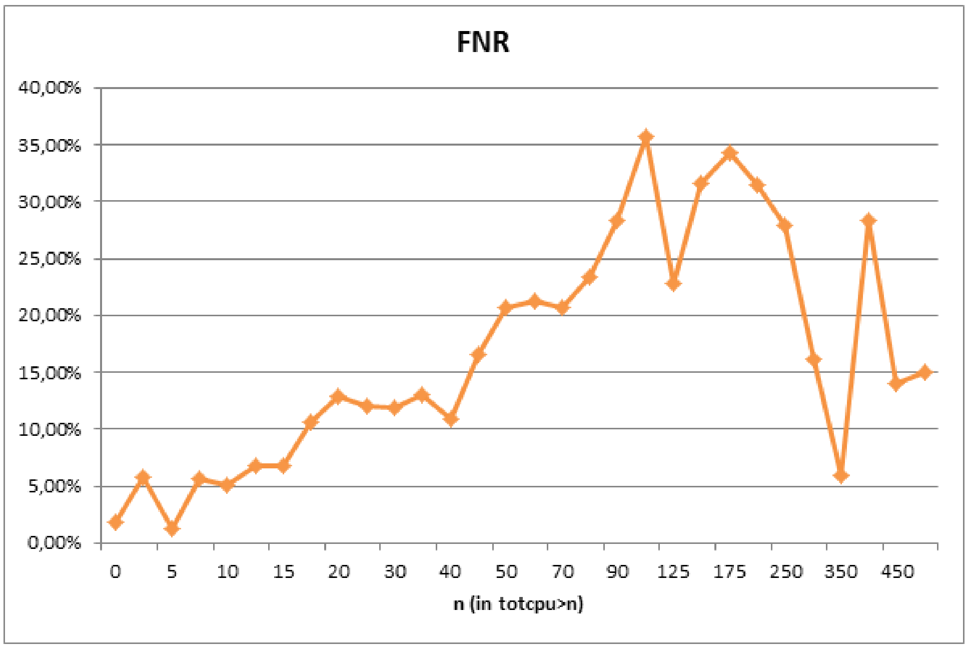


**Figure 5.32:** False negative rate (FNR) for the *totcpu* cut.

*Recall*, *F1*, *%TP*, *FPR* and *FNR* of the model using this combined cut, are reported in Figure 5.42.

From now on, we will apply a combined cut as follows: ***naccess>10* & *naccess>10***.

It is also interesting to see which is the behaviour of *nusers* before and after the cut for whole 2014 dataframes. We know that in 2014 there are 587636 datasets and after the *naccess>10* & *totcpu>10* cut 42005 dataframes remain, only the 7.15% on the total, and then the highest value of *nusers* is 83. In Figure 5.43 the percentage of dataframes related to a *nusers* value greater than a fixed value, calculated over the whole 2014 dataframes or over 2014 dataframes that has *naccess>10* and *totcpu>10* values are shown.

## 5.4 Study of different datatiers

So far, we studied all CMS datasets altogether. In this section we study different data tiers separately. In fact, as we have already discussed in previous chapters, there is a variety of different data tiers in CMS (AOD, AODSIM, RAW, RECO,USER, etc.). In order to understand which are the data tiers that are more interesting for a deeper study, Tables 5.44 and 5.45 show a summary of many features of all data tiers.

It is possible to see that the most present datatier in 2014 datasets is USER (with the 44.6% on the total) and then there is AODSIM (with the 12.9%). All the other datatiers are under 8% and in general (also for USER and AODSIM) the percentage of datasets non accessed in very high, with an average of 88.45%.

So just from this observation, it is evident that a vast majority of the storage in CMS is filled but not accessed in reading mode by anyone for long times. Of course, a data not accessed today may not imply the lack of an access sometime soon in the next future, but the probability that such a no-access pattern continues is actually very high.

The results of our model applied to all datatiers individually are shown in Figure 5.46 and 5.47.

Figure 5.47 shows that there are some peaks of FPR (the most relevant are RAWRECOSIMHLT with 30% and AODSIM with 9.5%) which have both an accessing percentage lower than the mean, where in the first case this difference is weaker than the second case but AODSIM has much and much more datasets than RAWRECOSIMHLT, so this two combined things can lead to an high FPR. Another aspect is that we have run our model on single datatier, but the model originally is built starting from whole 2014 datasets and so it could be not able to predict unpopular datatiers or popular ones but in exiguous number.

Figure 5.47 also shows that FPR is quite high for some data tiers (but not for the others) when applying a combined cut as explained in a previous section. This may imply that such combined cut may not be the optimal one for all data tiers, and special tuning of the cuts would be adequate for specific data tiers. This is a wide investigation in itself and was not pursued further in the scope of this thesis, and it has been left as a possible study to perform in the future.

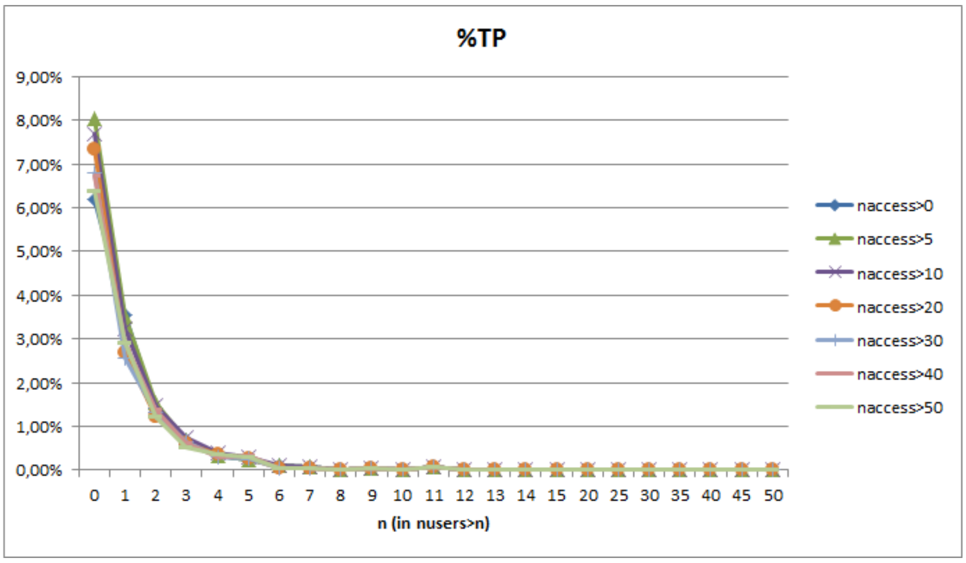


**Figure 5.33:** Fraction of all CMS datasets which are true positive (TP) for the *naccess* & *nusers* cut. This plot and the following ones always relate to positive and negative in terms of our machine learning study of dataset popularity.

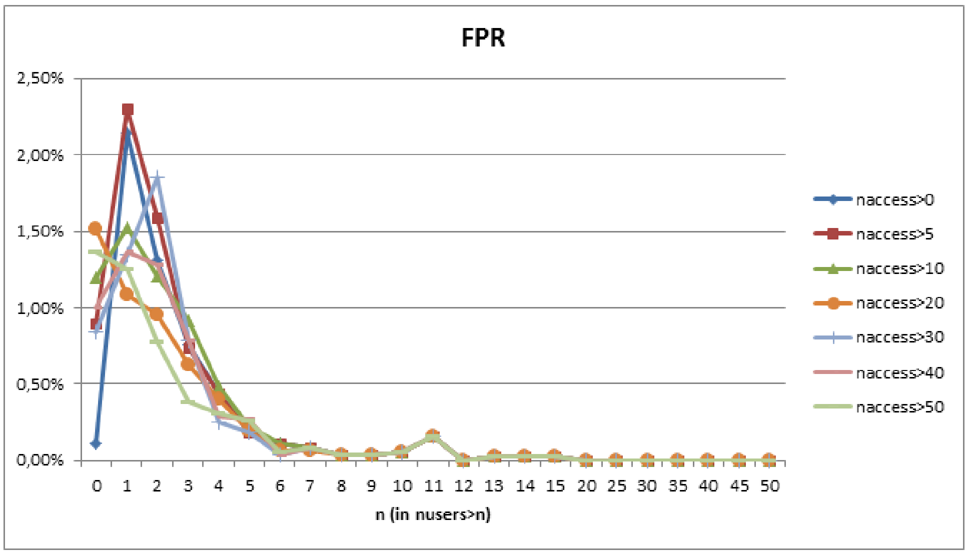


**Figure 5.34:** False positive rate (FPR) for the *naccess* & *nusers* cut.

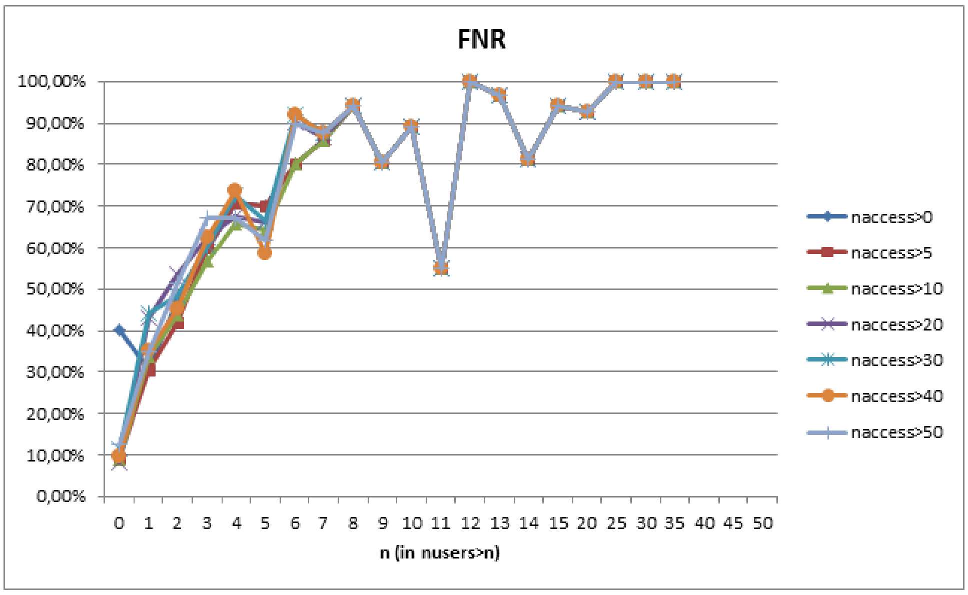


**Figure 5.35:** False negative rate (FNR) for the *naccess* & *nusers* cut.

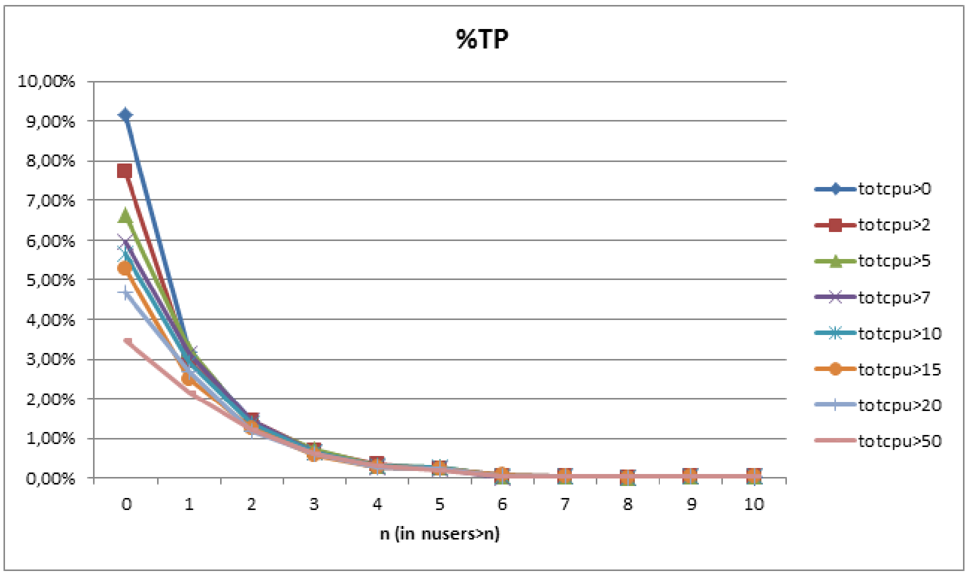


**Figure 5.36:** Fraction of all CMS datasets which are true positive (TP) for the *nusers* & *totcpu* cut.

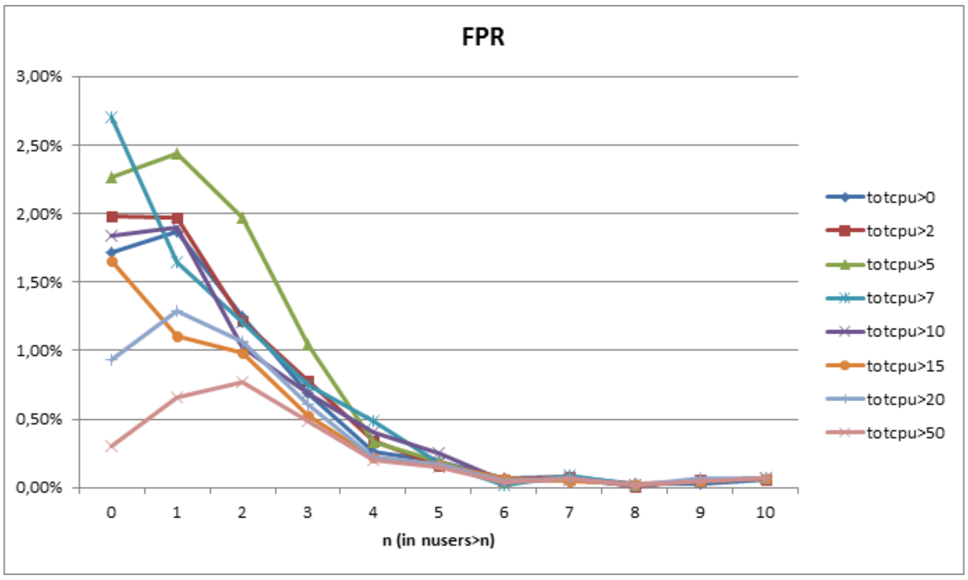


**Figure 5.37:** False positive rate (FPR) for the *nusers* & *totcpu* cut.

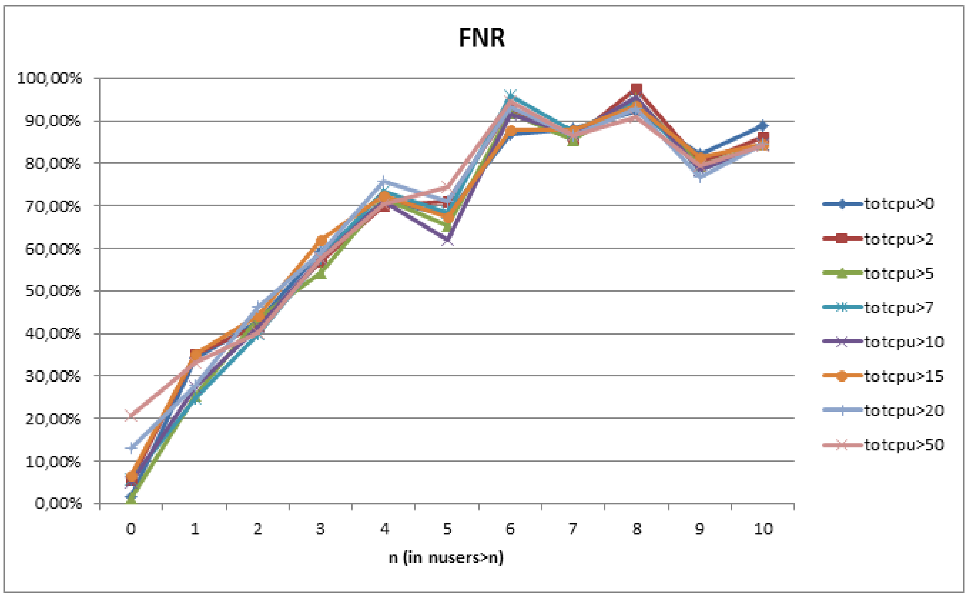


**Figure 5.38:** False negative rate (FNR) for the *nusers* & *totcpu* cut.

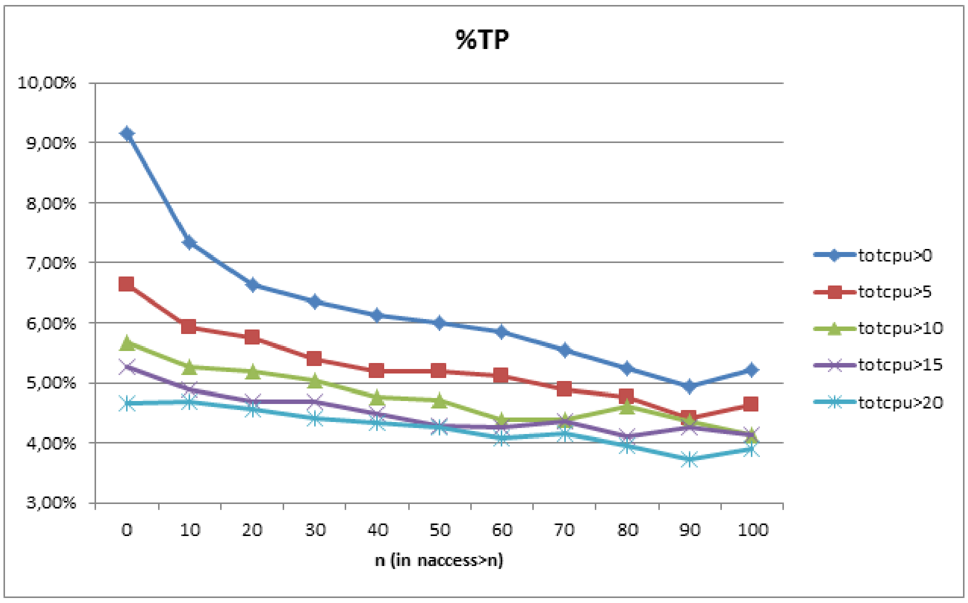


**Figure 5.39:** Fraction of all CMS datasets which are true positive (TP) for the *naccess* & *totcpu* cut.

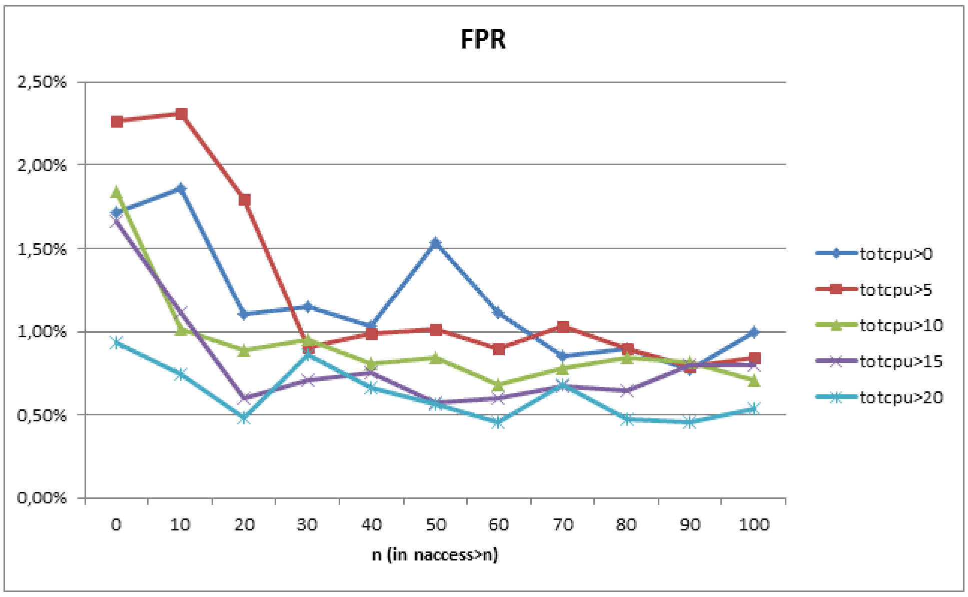


**Figure 5.40:** False positive rate (FPR) for the *naccess* & *totcpu* cut.

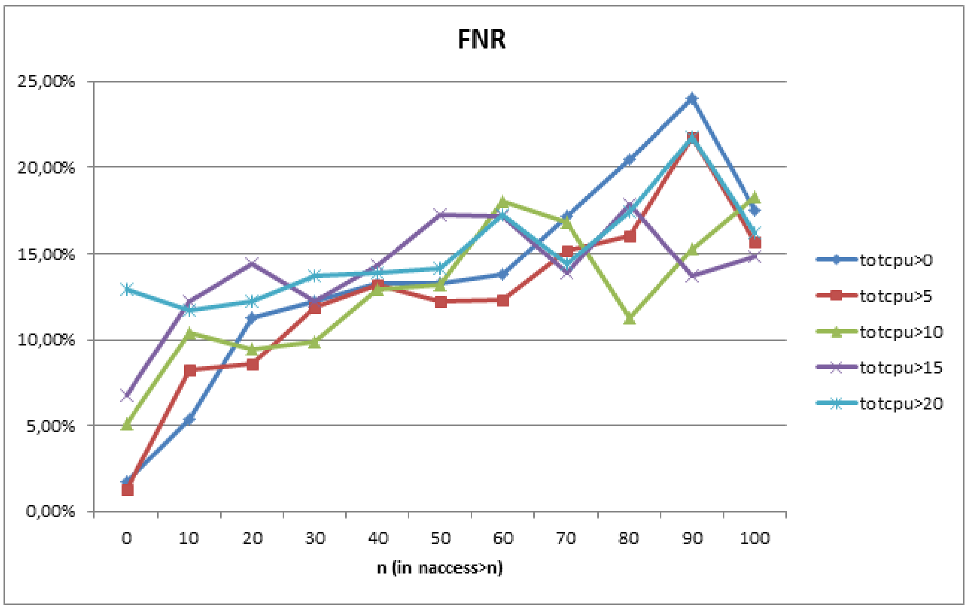


**Figure 5.41:** False negative rate (FNR) for the *naccess* & *totcpu* cut.

| Accuracy | Precision | Recall | F1 | %TP | FPR | FNR |
|---|---|---|---|---|---|---|
| 0,984 | 0,846 | 0,896 | 0,871 | 5,26% | 1,01% | 10,36% |

**Figure 5.42:** Measured value of scorers for a combined cut on *naccess>10* and *totcpu>10*.

| nusers> | datasets | % on the whole 2014 dataframes | % on the whole dataframes after the cut |
|---|---|---|---|
| 0 | 42005 | 7,15% | 100,00% |
| 1 | 28490 | 4,85% | 67,83% |
| 2 | 17446 | 2,97% | 41,53% |
| 3 | 11831 | 2,01% | 28,17% |
| 4 | 8425 | 1,43% | 20,06% |
| 5 | 6263 | 1,07% | 14,91% |
| 6 | 4807 | 0,82% | 11,44% |
| 7 | 3615 | 0,62% | 8,61% |
| 8 | 2949 | 0,50% | 7,02% |
| 9 | 2431 | 0,41% | 5,79% |
| 10 | 2051 | 0,35% | 4,88% |
| 15 | 900 | 0,15% | 2,14% |
| 20 | 458 | 0,08% | 1,09% |
| 30 | 150 | 0,03% | 0,36% |
| 50 | 28 | 0,00% | 0,07% |
| 70 | 1 | 0,00% | 0,00% |

**Figure 5.43:** Fraction of datasets after applying a *nusers* cut.

## 5.5 Studies of time-series

In this section we try to study which is the best time window of the data on which we build and train the model, in view of its application for prediction purpose in subsequent time windows.

In all studies in this work, we must define and use a time window whose dataframes are exploited to build and tune the model, and then we apply such model to a "future" window to get a prediction. But, it is not given for granted that there is an always-good choice to choose such time window. A possible approach is e.g. to make the time window grow as time passes (this approach is called *rolling* in this section). Another possible approach is e.g. to make the time window split as time passes by keeping its size fixed (this approach is called *translational* in this section).

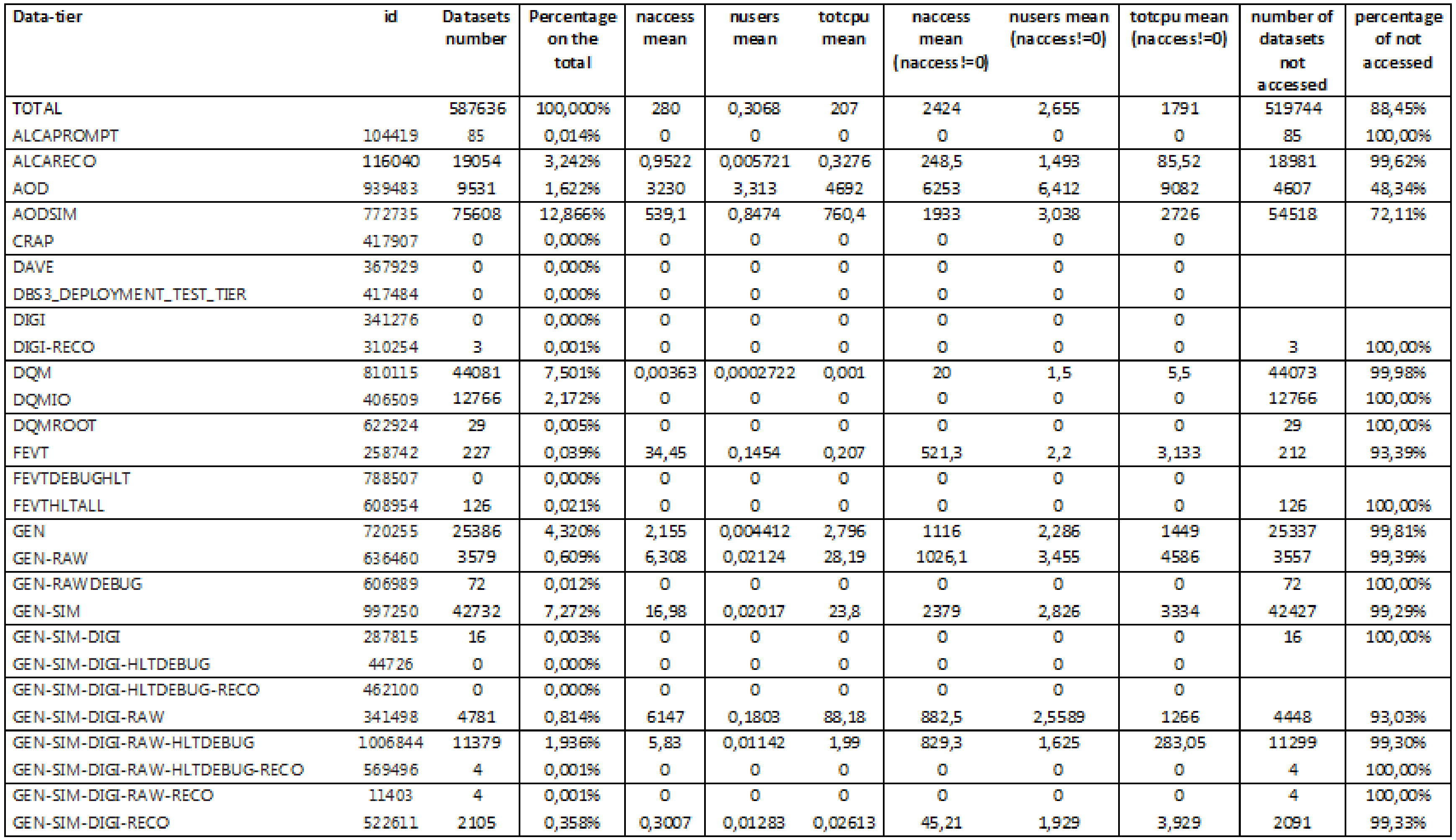
Figure 5.48 shows FPR results for a rolling approach: starting from 70%/30% datasets subdivision (of 2014) for train/valid sets, we add each time a dataframe week to the global set of datasets, maintaining as much as possible the initial proportion between train and valid. The x-axis shows the number of week that constitutes the whole group of train plus valid datasets. From the plot we can see that FPR oscillates and goes from a minimum of about 1% up to a maximum of about 3.1%. This shows that FPR correctly does not stray too much from the FPR value obtained with all 2014 datasets (that represents the minimum).

Figure 5.49 shows FPR results for a translational approach: starting from 70%/30% datasets subdivision (of 2014, so 12 months) for train/valid sets, we translate each time the group of datasets forward of a week, in order to maintain unchanged the number of dataframe weeks (but the number of datasets could change). The x-axis shows the number of dataframe weeks contained in valid set, since that at each step in which we add a new week, a new dataframe week will be added to valid set at the expense of a 2014 dataframe week that will be removed (the first one of the previous valid set). Also here it is possible to see that FPR oscillates and goes from a minimum of about 1% up to a maximum of about 3.25%. This shows that FPR correctly does not stray too much from the FPR value obtained with original 2014 datasets (that represents the minimum).

Figure 5.50 shows the same kind of exercise of the previous plot but now initially we consider not all 2014 datasets but only the datasets of the 2014 first half. So train set is made by 18 weeks, instead valid set is made by 8 weeks. The x-axis shows the number of the starting train week, so for example the point 25 is linked to a train set that starts from the 25-th week and ends to the 25+18=43-th week, while valid set starts from the 44-th week and ends to the 44+8=52-th week of 2014. So from this point valid set starts to contain some 2015 dataframes week. In this plot the result is different from the previous time-series exercises: FPR is very low for small x and starting from x=13 there is a quite steep increase in FPR that culminates with a peak at x=29 with FPR about equal to 4% (that corresponds to train start=2014 29-th week, train stop=2014 47-th week, valid start=2014 48-th week, valid stop= 2015 3-th week). After this point FPR decreases at the same rate to low values. So problems start when valid set contains the period near Christmas: indeed train set is made by data up to November 2014, so a normal working period at CMS, but valid set is made by December datasets and first three 2015 weeks datasets (over the last November week), in which there are festivities and there is much less work at CMS, with a low level of datasets access. This fact affects just FPR because the model, trained through datasets of a normal working period, tends to predict a popular dataset in train set as popular dataset in the period of valid set, but it is wrong and the validation action is not sufficient to prevent this fact.

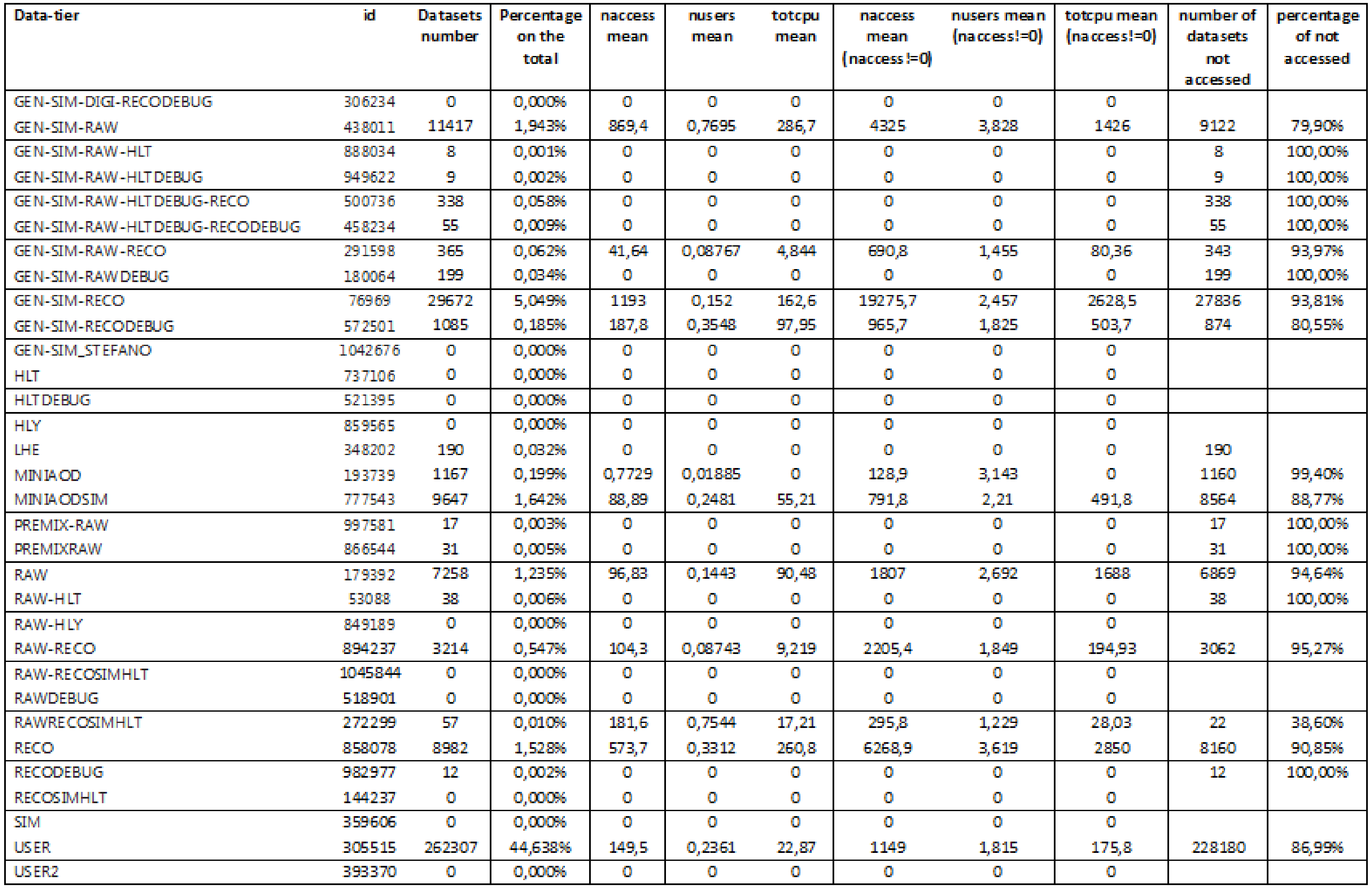
In conclusion, in order to have more statistics it is better in order to create a good predictive model, but not too big because otherwise overfitting problem may occur. Overfitting seems not to exist in case of one year or a little more of datasets period.

As a final note, it is worth noting that some of the results obtained in this thesis work have already been presented to an International Conference, and the proceedings have been accepted and are being processed for publication [35].



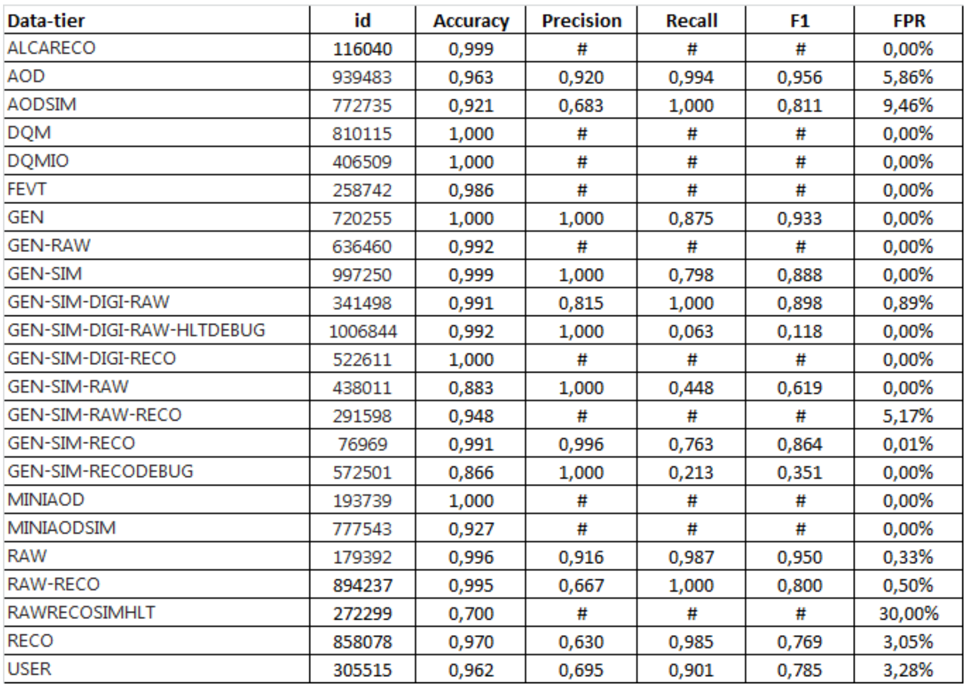
| Data-tier | id | Datasets number | Percentage on the total | naccess mean | nusers mean | totcpu mean | naccess mean (naccess!=0) | nusers mean (naccess!=0) | totcpu mean (naccess!=0) | number of datasets not accessed | percentage of not accessed |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TOTAL | | 587636 | 100,000% | 280 | 0,3068 | 207 | 2424 | 2,655 | 1791 | 519744 | 88,45% |
| ALCAPROMPT | 104419 | 85 | 0,014% | 0 | 0 | 0 | 0 | 0 | 0 | 85 | 100,00% |
| ALCARECO | 116040 | 19054 | 3,242% | 0,9522 | 0,005721 | 0,3276 | 248,5 | 1,493 | 85,52 | 18981 | 99,62% |
| AOD | 939483 | 9531 | 1,622% | 3230 | 3,313 | 4692 | 6253 | 6,412 | 9082 | 4607 | 48,34% |
| AODSIM | 772735 | 75608 | 12,866% | 539,1 | 0,8474 | 760,4 | 1933 | 3,038 | 2726 | 54518 | 72,11% |
| CRAP | 417907 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| DAVE | 367929 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| DBS3_DEPLOYMENT_TEST_TIER | 417484 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| DIGI | 341276 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| DIGI-RECO | 310254 | 3 | 0,001% | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 100,00% |
| DQM | 810115 | 44081 | 7,501% | 0,00363 | 0,0002722 | 0,001 | 20 | 1,5 | 5,5 | 44073 | 99,98% |
| DQMIO | 406509 | 12766 | 2,172% | 0 | 0 | 0 | 0 | 0 | 0 | 12766 | 100,00% |
| DQMROOT | 622924 | 29 | 0,005% | 0 | 0 | 0 | 0 | 0 | 0 | 29 | 100,00% |
| FEVT | 258742 | 227 | 0,039% | 34,45 | 0,1454 | 0,207 | 521,3 | 2,2 | 3,133 | 212 | 93,39% |
| FEVTDEBUGHLT | 788507 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| FEVTHLTALL | 608954 | 126 | 0,021% | 0 | 0 | 0 | 0 | 0 | 0 | 126 | 100,00% |
| GEN | 720255 | 25386 | 4,320% | 2,155 | 0,004412 | 2,796 | 1116 | 2,286 | 1449 | 25337 | 99,81% |
| GEN-RAW | 636460 | 3579 | 0,609% | 6,308 | 0,02124 | 28,19 | 1026,1 | 3,455 | 4586 | 3557 | 99,39% |
| GEN-RAWDEBUG | 606989 | 72 | 0,012% | 0 | 0 | 0 | 0 | 0 | 0 | 72 | 100,00% |
| GEN-SIM | 997250 | 42732 | 7,272% | 16,98 | 0,02017 | 23,8 | 2379 | 2,826 | 3334 | 42427 | 99,29% |
| GEN-SIM-DIGI | 287815 | 16 | 0,003% | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 100,00% |
| GEN-SIM-DIGI-HLTDEBUG | 44726 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| GEN-SIM-DIGI-HLTDEBUG-RECO | 462100 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| GEN-SIM-DIGI-RAW | 341498 | 4781 | 0,814% | 6147 | 0,1803 | 88,18 | 882,5 | 2,5589 | 1266 | 4448 | 93,03% |
| GEN-SIM-DIGI-RAW-HLTDEBUG | 1006844 | 11379 | 1,936% | 5,83 | 0,01142 | 1,99 | 829,3 | 1,625 | 283,05 | 11299 | 99,30% |
| GEN-SIM-DIGI-RAW-HLTDEBUG-RECO | 569496 | 4 | 0,001% | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 100,00% |
| GEN-SIM-DIGI-RAW-RECO | 11403 | 4 | 0,001% | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 100,00% |
| GEN-SIM-DIGI-RECO | 522611 | 2105 | 0,358% | 0,3007 | 0,01283 | 0,02613 | 45,21 | 1,929 | 3,929 | 2091 | 99,33% |

**Figure 5.44:** Some features of all datatier kinds.



| Data-tier | id | Datasets number | Percentage on the total | naccess mean | nusers mean | totcpu mean | naccess mean (naccess!=0) | nusers mean (naccess!=0) | totcpu mean (naccess!=0) | number of datasets not accessed | percentage of not accessed |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GEN-SIM-DIGI-RECODEBUG | 306234 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| GEN-SIM-RAW | 438011 | 11417 | 1,943% | 869,4 | 0,7695 | 286,7 | 4325 | 3,828 | 1426 | 9122 | 79,90% |
| GEN-SIM-RAW-HLT | 888034 | 8 | 0,001% | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 100,00% |
| GEN-SIM-RAW-HLTDEBUG | 949622 | 9 | 0,002% | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 100,00% |
| GEN-SIM-RAW-HLTDEBUG-RECO | 500736 | 338 | 0,058% | 0 | 0 | 0 | 0 | 0 | 0 | 338 | 100,00% |
| GEN-SIM-RAW-HLTDEBUG-RECODEBUG | 458234 | 55 | 0,009% | 0 | 0 | 0 | 0 | 0 | 0 | 55 | 100,00% |
| GEN-SIM-RAW-RECO | 291598 | 365 | 0,062% | 41,64 | 0,08767 | 4,844 | 690,8 | 1,455 | 80,36 | 343 | 93,97% |
| GEN-SIM-RAWDEBUG | 180064 | 199 | 0,034% | 0 | 0 | 0 | 0 | 0 | 0 | 199 | 100,00% |
| GEN-SIM-RECO | 76969 | 29672 | 5,049% | 1193 | 0,152 | 162,6 | 19275,7 | 2,457 | 2628,5 | 27836 | 93,81% |
| GEN-SIM-RECODEBUG | 572501 | 1085 | 0,185% | 187,8 | 0,3548 | 97,95 | 965,7 | 1,825 | 503,7 | 874 | 80,55% |
| GEN-SIM_STEFANO | 1042676 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| HLT | 737106 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| HLTDEBUG | 521395 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| HLY | 859565 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| LHE | 348202 | 190 | 0,032% | 0 | 0 | 0 | 0 | 0 | 0 | 190 | |
| MINIAOD | 193739 | 1167 | 0,199% | 0,7729 | 0,01885 | 0 | 128,9 | 3,143 | 0 | 1160 | 99,40% |
| MINIAODSIM | 777543 | 9647 | 1,642% | 88,89 | 0,2481 | 55,21 | 791,8 | 2,21 | 491,8 | 8564 | 88,77% |
| PREMIX-RAW | 997581 | 17 | 0,003% | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 100,00% |
| PREMIXRAW | 866544 | 31 | 0,005% | 0 | 0 | 0 | 0 | 0 | 0 | 31 | 100,00% |
| RAW | 179392 | 7258 | 1,235% | 96,83 | 0,1443 | 90,48 | 1807 | 2,692 | 1688 | 6869 | 94,64% |
| RAW-HLT | 53088 | 38 | 0,006% | 0 | 0 | 0 | 0 | 0 | 0 | 38 | 100,00% |
| RAW-HLY | 849189 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| RAW-RECO | 894237 | 3214 | 0,547% | 104,3 | 0,08743 | 9,219 | 2205,4 | 1,849 | 194,93 | 3062 | 95,27% |
| RAW-RECOSIMHLT | 1045844 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| RAWDEBUG | 518901 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| RAWRECOSIMHLT | 272299 | 57 | 0,010% | 181,6 | 0,7544 | 17,21 | 295,8 | 1,229 | 28,03 | 22 | 38,60% |
| RECO | 858078 | 8982 | 1,528% | 573,7 | 0,3312 | 260,8 | 6268,9 | 3,619 | 2850 | 8160 | 90,85% |
| RECODEBUG | 982977 | 12 | 0,002% | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 100,00% |
| RECOSIMHLT | 144237 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| SIM | 359606 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |
| USER | 305515 | 262307 | 44,638% | 149,5 | 0,2361 | 22,87 | 1149 | 1,815 | 175,8 | 228180 | 86,99% |
| USER2 | 393370 | 0 | 0,000% | 0 | 0 | 0 | 0 | 0 | 0 | | |

**Figure 5.45:** Some features of all datatier kinds.



| Data-tier | id | Accuracy | Precision | Recall | F1 | FPR |
|---|---|---|---|---|---|---|
| ALCARECO | 116040 | 0,999 | # | # | # | 0,00% |
| AOD | 939483 | 0,963 | 0,920 | 0,994 | 0,956 | 5,86% |
| AODSIM | 772735 | 0,921 | 0,683 | 1,000 | 0,811 | 9,46% |
| DQM | 810115 | 1,000 | # | # | # | 0,00% |
| DQMIO | 406509 | 1,000 | # | # | # | 0,00% |
| FEVT | 258742 | 0,986 | # | # | # | 0,00% |
| GEN | 720255 | 1,000 | 1,000 | 0,875 | 0,933 | 0,00% |
| GEN-RAW | 636460 | 0,992 | # | # | # | 0,00% |
| GEN-SIM | 997250 | 0,999 | 1,000 | 0,798 | 0,888 | 0,00% |
| GEN-SIM-DIGI-RAW | 341498 | 0,991 | 0,815 | 1,000 | 0,898 | 0,89% |
| GEN-SIM-DIGI-RAW-HLTDEBUG | 1006844 | 0,992 | 1,000 | 0,063 | 0,118 | 0,00% |
| GEN-SIM-DIGI-RECO | 522611 | 1,000 | # | # | # | 0,00% |
| GEN-SIM-RAW | 438011 | 0,883 | 1,000 | 0,448 | 0,619 | 0,00% |
| GEN-SIM-RAW-RECO | 291598 | 0,948 | # | # | # | 5,17% |
| GEN-SIM-RECO | 76969 | 0,991 | 0,996 | 0,763 | 0,864 | 0,01% |
| GEN-SIM-RECODEBUG | 572501 | 0,866 | 1,000 | 0,213 | 0,351 | 0,00% |
| MINIAOD | 193739 | 1,000 | # | # | # | 0,00% |
| MINIAODSIM | 777543 | 0,927 | # | # | # | 0,00% |
| RAW | 179392 | 0,996 | 0,916 | 0,987 | 0,950 | 0,33% |
| RAW-RECO | 894237 | 0,995 | 0,667 | 1,000 | 0,800 | 0,50% |
| RAWRECOSIMHLT | 272299 | 0,700 | # | # | # | 30,00% |
| RECO | 858078 | 0,970 | 0,630 | 0,985 | 0,769 | 3,05% |
| USER | 305515 | 0,962 | 0,695 | 0,901 | 0,785 | 3,28% |

**Figure 5.46:** Values of some model parameters in *naccess>10* cut & *totcpu>10* cut for some datatiers.

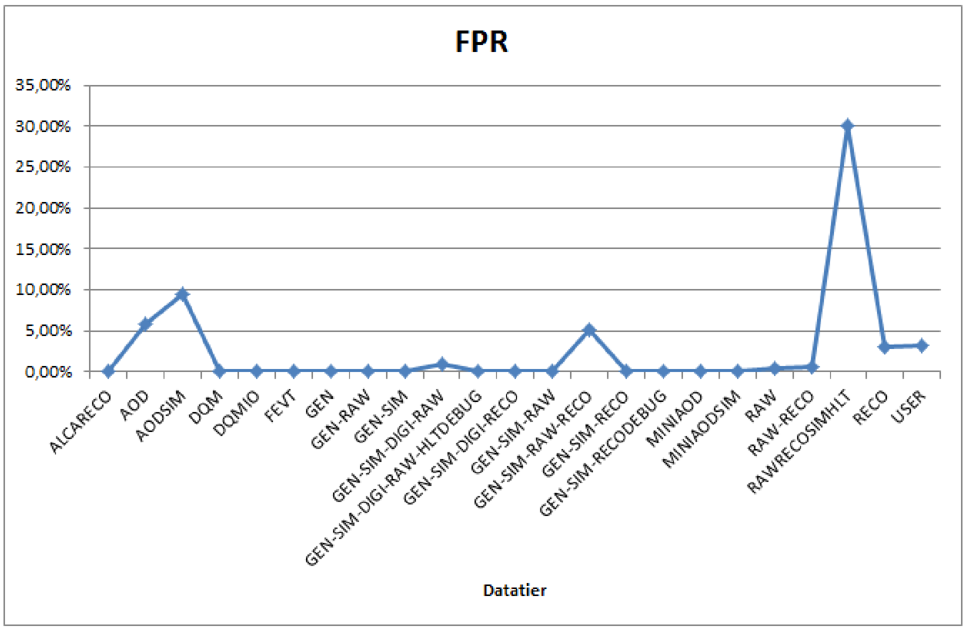


**Figure 5.47:** FPR in *naccess>10* cut & *totcpu>10* cut for some datatiers.

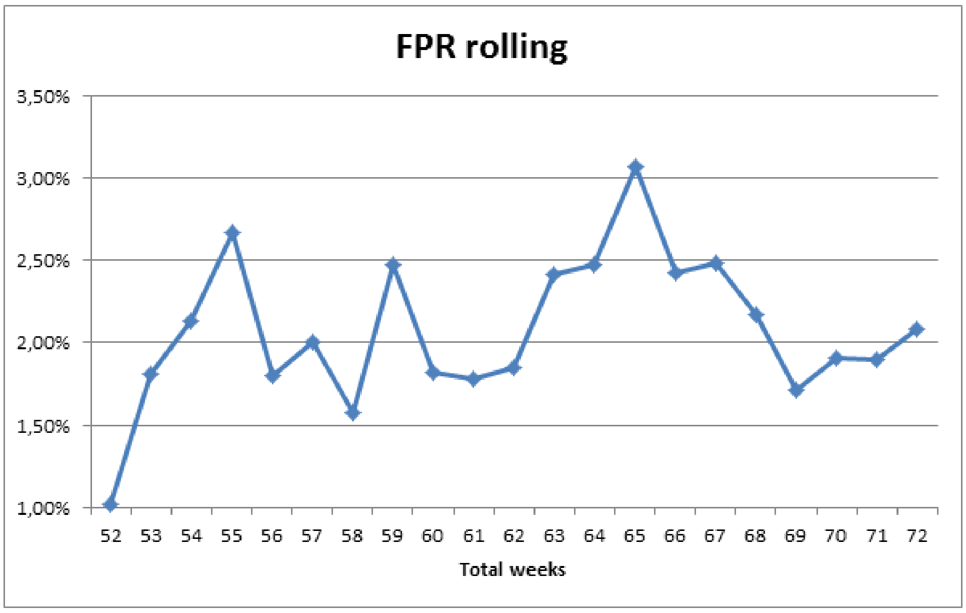


**Figure 5.48:** FPR trend for the *rolling* approach in the time-series analysis, with a variable time window.

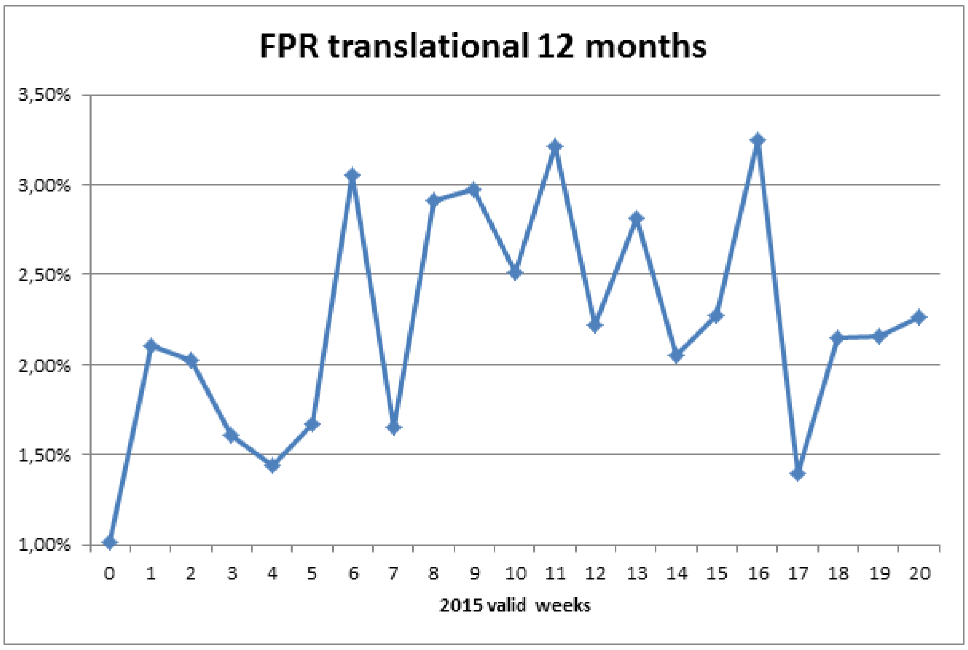


**Figure 5.49:** FPR trend for the *translational* approach in the time-series analysis, with a time window of 12 months.

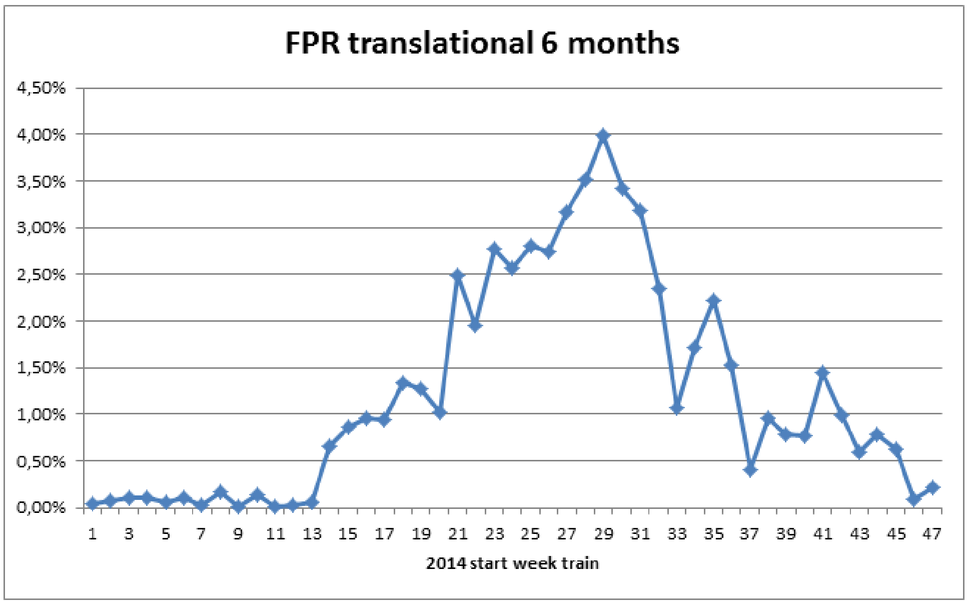


**Figure 5.50:** FPR trend for the *translational* approach in the time-series analysis, with a time window of 6 months.

# Conclusions

This thesis presented and discussed the design, development and exploitation of a supervised Machine Learning classification system aimed at attacking the very concrete need of the prediction of the “popularity” of the CMS datasets on the Grid.

The DCAFPilot prototype has been largely exploited to perform this task, helping to find and fix fragilities and hence to strenghten the code itself. The popularity problem has been explored and analysed from a Machine Learning perspective, and a quick cost analysis has been performed to identify the “false positive rate” (FPR) as the main metric to be kept under control. More than a dozen of different machine learning classifiers have been tested, and *RandomForestClassifier* has been chosen as adequate. Different split proportions of the data used to train and validate the model have been tested, and a 70%/30% split has been chosen as optimal. The definition of popularity has been explored, in terms of studies of cuts on 3 different variables, namely *naccess, nusers, totcpu* (see Chapter 5 for details). The impact of cuts on each of the 3 variables alone have been studied, yielding indications on the thresholds to set for the cuts on each variable individually. Combined cuts on 2 out of these 3 variables have also been studied, showing that *nusers* can be dropped as the least useful one, and identifying the combined cut on *naccess* and *totcpu* as the one that offers an acceptably low FPR. The golden combined cut was identified to be *naccess>10* & *totcpu>10* (see Chapter 5 for details). These studies, done on all CMS datasets altogether, were also prepared to be fully repeated on different data tiers separately (this part of the work would have required a large amount of time, well beyond the scope of this thesis, so it was prepared but not brought to completion). The problem of identifying the best time window of the data on which we build and train a model, in view of its application for prediction purposes in a subsequent time window, was also attacked in a “time series” study. About the latter, a “rolling" approach - that allows the time window of the model training grow with time - as well as a “translational" approach - that makes the time window slip as time passes by keeping its size fixed - were both set-up and run. In the latter approach, a working point was found between two extremes, i.e. using too few data to train the model - and hence have a useless model with no predictive power - and using too much data to train the model - and hence experience overfitting issues thus resulting again in a useless model: this working point was found in using basically 1 year worth of data to train and validate the model (a solution that also allows to avoid seasonal effects). All this studies and set-up have allowed to prepare DCAFPilot for a dry run on popularity that allowed to achieve a prediction with *Accuracy* 0.984, *Precision* 0.846, *Recall* 0.896, *F1* 0.871, FPR 1.01%.

Plans for the short term are to run this prototype as a production service for CMS and regularly perform predictions of the CMS datasets popularity in an automated manner, and eventually feed this information to CMS Computing Operations teams to drive their choices and optimize their operational actions. Plans for the longer term are to evaluate other areas (aside from the popularity) where the same DCAFPilot approach could be exploited as effectively as in this case.

Some of the results of this work have been presented to an International Conference on Grid and Distributed Computing topics, and the proceedings of which I am co-author have been submitted and accepted for publication ([35]).

# Bibliography

[1] Oliver Sim Brüning et al. *LHC Design Report*. Ed. by CERN library copies. Vol. 1, 2, 3. 2012. URL: http://ab-div.web.cern.ch/ab-div/Publications/LHC-DesignReport.html (cit. on p. 1).

[2] Lyndon Evans and Philip Bryant. "LHC Machine". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08001. URL: http://iopscience.iop.org/1748-0221/3/08/S08001 (cit. on p. 1).

[3] S. Myers R. ABmann M. Lamont. "A Brief History of the LEP Collider". In: Suppl. 109, 17-31 (2002). Ed. by Nucl.Phys. B (cit. on p. 1).

[4] S. Chatrchyan et al. [CMS Collaboration]. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Phys. Lett. B* 716 (2012). Ed. by IOPscience, p. 30 (cit. on p. 1).

[5] The ALICE Collaboration et al. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08002. URL: http://iopscience.iop.org/1748-0221/3/08/S08002 (cit. on p. 5).

[6] *The Alice Collaboration*. URL: http://aliceinfo.cern.ch (cit. on p. 5).

[7] The ATLAS Collaboration et al. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08003. URL: http://iopscience.iop.org/1748-0221/3/08/S08003 (cit. on p. 6).

[8] *The Atlas Collaboration*. URL: http://atlas.web.cern.ch/Atlas/Collaboration (cit. on p. 6).

[9] The CMS Collaboration et al. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008), S08004. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08004 (cit. on p. 7).

[10] *The CMS Collaboration*. URL: http://cms.web.cern.ch (cit. on p. 7).

[11] The LHCb Collaboration et al. "The LHCb Detector at the LHC". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08005. URL: http://iopscience.iop.org/1748-0221/3/08/S08005 (cit. on p. 7).

[12] *The LHCb Collaboration*. URL: http://lhcb.web.cern.ch/lhcb (cit. on p. 7).

[13] The LHCf Collaboration et al. “The LHCf detector at the CERN Large Hadron Collider”. In: *Journal of Instrumentation* 3.S08006 (2008). Ed. by IOPscience. URL: http://iopscience.iop.org/1748-0221/3/08/S08006 (cit. on p. 8).

[14] *The LHCf experiment*. URL: http://home.web.cern.ch/about/experiments/lhcf (cit. on p. 8).

[15] The TOTEM Collaboration et al. “The TOTEM Experiment at the CERN Large Hadron Collider”. In: *Journal of Instrumentation* 3.S08007 (2008). Ed. by IOPscience. URL: http://iopscience.iop.org/1748-0221/3/08/S08006 (cit. on p. 8).

[16] *The TOTEM Collaboration*. URL: http://totem.web.cern.ch/Totem/ (cit. on p. 8).

[17] *The MOEDAL Collaboration*. URL: http://home.web.cern.ch/about/experiments/moedal (cit. on p. 9).

[18] J. D. Shiers. “The Worldwide LHC Computing Grid (worldwide LCG)”. In: *ComputerPhysics Communications* 219-223 (2007) (cit. on p. 18).

[19] *WLCG Project*. URL: http://www.cern.ch/lcg (cit. on p. 18).

[20] *European Grid Infrastructure (EGI)*. URL: http://www.egi.eu/ (cit. on p. 18).

[21] *Open Science Grid (OSG)*. URL: http://www.opensciencegrid.org (cit. on p. 18).

[22] M Giffels, Y Guo, and D Riley. “Data Bookkeeping Service 3 – Providing event metadata in CMS”. In: *Journal of Physics: Conference Series*. Conference Series 513.4 (2014), p. 042022. URL: http://stacks.iop.org/1742-6596/513/i=4/a=042022 (cit. on p. 23).

[23] Tony Wildish et al. “From toolkit to framework - the past and future evolution of PhEDEx”. In: *Journal of Physics*. Conference Series 396.3 (2012). Ed. by IOPscience, p. 032118. URL: http://iopscience.iop.org/1742-6596/396/3/032118 (cit. on p. 23).

[24] J. Rehn et al. “PhEDEx high-throughput data transfer management system”. In: *CHEP06* (2006). Ed. by GridPP. URL: http://www.gridpp.ac.uk/papers/chep06_tuura.pdf (cit. on p. 23).

[25] *CRAB*. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab (cit. on p. 25).

[26] *Machine Learning*. URL: http://www.wikiwand.com/en/Machine_learning (cit. on p. 31).

[27] *Machine Learning tutorial*. URL: https://class.coursera.org/ml-005/lecture (cit. on pp. 33–36).

[28] *Google News*. URL: https://news.google.it/ (cit. on p. 36).

[29] *Dynamic Data Placement*. URL: https://twiki.cern.ch/twiki/bin/viewauth/CMS/DynData (cit. on p. 46).

[30] *DCAF pilot.* URL: https://github.com/dmwm/DMWMAnalytics/tree/master/Popularity/DCAFPilot (cit. on p. 46).

[31] V. Kuznetsov N. Magini M. Giffels Y. Guo and T. Wildish. "The CMS Data Management System". In: *J.Phys* (2014) (cit. on p. 47).

[32] *scikit-learn.* URL: http://scikit-learn.org/stable/" (cit. on p. 47).

[33] *Cross-validation.* URL: http://www.wikiwand.com/en/Cross-validation_(statistics) (cit. on pp. 52, 60).

[34] *Overfitting.* URL: http://www.wikiwand.com/en/Overfitting (cit. on p. 59).

[35] Kuznetsov V. Wildish T. Bonacorsi D. Giommi L. "Exploring patterns and correlations in CMS Computing operations data with Big Data analytics techniques". In: *oral presentation at International Symposium on Grids and Clouds (ISGC), Academia Sinica (ASGC), Taipei, Taiwan* (). Ed. by SISSA Proceedings Of Science. URL: http://pos.sissa.it/archive/conferences/239/008/ISGC2015_008.pdf (cit. on pp. 85, 92).