People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Control**

Option: **Control**

Title:

Prediction of Remaining Useful Lifetime (RUL) of Turbofan Engine Using Machine Learning

Presented by:

- **ABDELLAOUI Abdelkader.**
- **MENASRIA Hafidh.**

Supervisor:

Dr. BOUSHAKI Razika.

Registration Number:……../2018

Dedication

To Mom and Dad, it's impossible to thank you adequately for everything you've done, from loving me unconditionally to raising me in a stable household. To my brothers and sisters, to my grandfather and my grandmothers and my future wife. To all my friends. I love all of you.

ABDELLAOUI Abdelkader

Dedication

I dedicate this moderate work solely to my mother and father.

MENASRIA Hafidh

Acknowledgement

We would like to thank Dr. Boushaki for her support until we finished this project. We also want to thank Mr. Yacine Aribi for his continuous help during this work, as well as Fares Bencheikh and all other friends. We would also like to thank our families for supporting us throughout our Bachelors and Masters.

Abstract

One of the most important factors in the field of flying is the maintenance of the aircraft engine, that is because of the accidents that happened more than once. This requires a knowledge not only in the system of the aircraft engines, but also how they work and how their performance degrades over time. This drives us to the prediction field where machine learning plays an important role in analyzing and the data measurements from the equipment and attempt to predict any failure that could happen.

In this thesis a study of prediction of the remaining useful life (RUL) of aircraft's turbo fan engine has been investigated by bringing a dataset from turbo fan engine from the Prognostics Data Repository of NASA and using Principal Component Analysis (PCA) and Independent Component Analysis (ICA) techniques for data analytics and preprocessing, then selecting two machine learning algorithms Random Forest and Gradient Boosted Machine (GBM) so that a model can be trained.

The idea is to develop a model to estimate the remaining useful life of the functionality of the turbofan engine and predict failure before it actually happens.

Key words: Principal Component Analysis (PCA), Independent Component Analysis (ICA), Random Forest, Gradient Boosted Machine (GBM).

Table of Contents

List of Tables

List of Figures

List of Figures

Nomenclature

Abbreviations

ATC	Air Traffic Control
BSS	Blind Source Separation
CA	Correlation analysis
CCA	Canonical Correlation Analysis
EGT	Exhaust Gas Temperature
EOF	Empirical Orthogonal Functions
EVD	Eigenvalue Decomposition
FD	Fault Detection
FDD	Fault Detection and Diagnosis
GBM	Gradient Boosted Machine
GUI	Graphical User Interface
ICA	Independent Components Analysis
LEMs	Linear Engine Models
MSE	Mean Squared Error
OOB	Out-of-bag Data
PC	Principal Component
PCA	Principal Components Analysis
POD	Proper Orthogonal Decomposition
RF	Random Forest
RUL	Remaining Useful Life
SPE	Squared Prediction Error
SVD	Singular Value Decomposition
TRA	Throttle-Resolver Angle

Symbols

$X \in \mathbb{R}^{N \times m}$	Original data matrix with N observations and m variables
x_i	The i^{th} Data Variable
D	The Diagonal Matrix of Eigenvalues
λ_i	The i^{th} eigenvalue
$\rho_{x,y}$	The population correlation coefficient between two random variables X and Y
$\Psi(y, f)$	Loss Function
$h(x, \theta)$	Parameterized "base-learner" function
μ	The Mean
σ	The Standard Deviation
$\hat{\theta}$	The parameter estimates
$J(\theta)$	Empirical loss function
$\nabla J(\theta)$	Gradient of the loss function
L2	Squared loss function

General Introduction

Whenever we hear about an accident that happens in space or crashing in the sea, it comes to our minds what is the cause behind that? Even the modern airplanes with turbofan engines has been crashed in the space. Here we found ourselves asking more questions if it is safe to travel by plane, engineers must reconsider the functionality and the performance of the airplane systems, including the past and present status of its turbofan engines. This leads us to think about the prognostic field where we can predict the time of failure in our systems and hence the time left of the functionality of turbofan engines often referred to as the remaining useful life (RUL).

Most modern airliners use turbofan engines to generate the required thrust in order to move through the air. Hence prediction of the remaining useful life (RUL) is critical for scheduling aircraft maintenance and stop the failure [1] [2] [3] [4]. Additionally, predictive maintenance can save lot of time and cost as well as energy by reducing the unnecessary maintenances that can be done in the urgent cases [1][5].

Nowadays predictive maintenance has been very popular in modern industrial processes in which we can do it in two ways. The first one is the classification approach where it can predict a failure in next n-steps. Or it can be regression approach where it predicts the time left before the next failure, called Remaining Useful Life (RUL) [6][8].

The RUL estimation is now a standard problem for any system and is the main focus of several organizations, including the *National Aeronautics and Space Administration (NASA)* and the *Prognostics & Health Management (PHM) Society* who have promoted the field by publishing several datasets open to the public, providing the free, unrestricted access to PHM knowledge, and promoting collaboration [7].

This thesis is organized as follows: Chapter 1 discusses the principal of airplane turbofan engine operation and malfunctions, as well as the cause of the malfunctions, and we also discussed the C-MAPSS tool which stands for Commercial Modular Aero-Propulsion System Simulation which is used as a tool for simulation due to the rare failures of the turbofan engines. In chapter 2, we introduced two machine learning techniques, Principal Component Analysis (PCA) and Independent Component Analysis (ICA) and the use of them as tools to extract the main features and reducing the dimension of the data. Chapter 3 discusses ensemble machine learning as a concept of solving supervised learning problems by training multiple models using the same

algorithm, in this case we discussed the two machine learning algorithms Random Forest (RF) and Gradient Boosted Machine (GBM) and the difference between them in the bagging and boosting methods for building a model.

Chapter 1

Airplane Turbofan Engine Operation and Malfunctions

1.1 General Principals

1.1.1 Introduction

Today's modern airplanes are powered by turbofan engines. These engines are quite reliable, providing years of trouble- free service. Because of the rarity of turbofan engine malfunctions and the limitations of simulating these malfunctions, many flight crews have felt unprepared to diagnose actual engine malfunctions that have occurred. The purpose of this chapter is to provide basics of airplane engine operational theory.

This chapter will also provide pertinent information about malfunctions and failures that may be encountered during the operation of turbofan powered airplanes that cannot be simulated well and may cause the flight crew to be startled or confused as to what the actual malfunction is. While simulators have greatly improved pilot training, many may not have been programmed to simulate the actual noise, vibration and aerodynamic forces that certain malfunctions cause. In addition, it appears that the greater the sensations, the greater the startle factor, along with greater likelihood the flight crew will try to diagnose the problem immediately instead of flying the airplane.

1.1.2 Propulsion

Propulsion is the net force that results from unequal pressures. Gas (air) under pressure in a sealed container exerts equal pressure on all surfaces of the container; therefore, all the forces are balanced and there are no forces to make the container move.

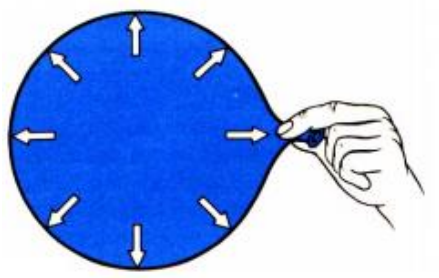

Figure 1-1 Showing balloon with no escape path for the air inside. All forced are balanced

If there is a hole in the container, gas (air) cannot push against that hole and thus the gas escapes. While the air is escaping and there is still pressure inside the container, the side of the container opposite the hole has pressure against it. Therefore, the net pressures are not balanced and there is a net force available to move the container. This force is called ***thrust.***

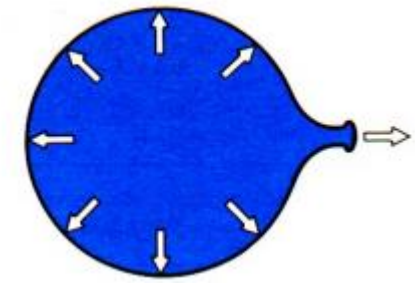

Figure 1-2 Showing balloon with released stem.

1.1.3 The Simplest Propulsion Engine

The simplest propulsion engine would be a container of air (gas) under pressure that is open at one end. A diving SCUBA tank would be such an engine if it fell and the valve was knocked off the top. The practical problem with such an engine is that, as the air escapes out the open end, the pressure inside the container would rapidly drop. This engine would deliver propulsion for only a limited time.

1.1.4 The Turbine Engine

A turbine engine is a container with a hole in the back end (tailpipe or nozzle) to let air inside the container escape, and thus provide propulsion. Inside the container is turbomachinery to keep the container full of air under constant pressure.

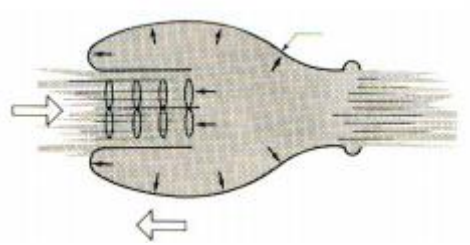

Figure 1-3 showing our balloon with machinery in front to keep it full as air escapes out the back of continuous thrust.

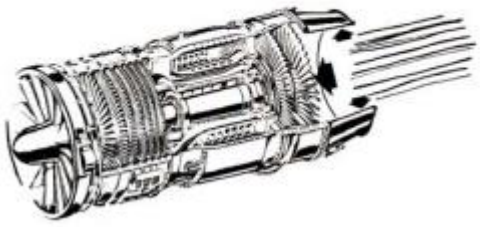

Figure 1-4 showing turbine engine as a cylinder of turbomachinery with unbalanced forces pushing forward.

1.1.5 Components of a Turbine Engine

The turbomachinery in the engine uses energy stored chemically as fuel. The basic principle of the airplane turbine engine is identical to any and all engines that extract energy from chemical fuel. The basic 4 steps for any internal combustion engine are:

1) Intake of air (and possibly fuel)
2) Compression of the air (and possibly fuel)
3) Combustion, where fuel is injected (if it was not drawn in with the intake air) and burned to convert the stored energy.
4) Expansion and exhaust, where the converted energy is put to use.

These principles are exactly the same ones that make lawn mower or automobile engine go. In the case of a piston engine such as the engine in cars or lawn mower, the intake, compression, combustion, and exhaust steps occur in the same place (cylinder head) at different times as the piston goes up and down.

In the turbine engine, however, these same four steps occur at the same time but in different places. As a result of this fundamental difference, the turbine has engine sections called:

1) The inlet section
2) The compressor section
3) The combustion section
4) The exhaust section.

1.1.6 The Practical Axial Flow Turbine Engine

The turbine engine in an airplane has the various sections stacked in a line from front to back. As a result, the engine body presents less drag to the airplane as it is flying. The air enters the front of the engine and passes essentially straight through from front to back. On its way to the back, the air is compressed by the compressor section. Fuel is added and burned in the combustion section, then the air is exhausted through the exit nozzle.

The laws of nature will not let us get something for nothing. The compressor needs to be driven by something in order to work. Just after the burner and before the exhaust nozzle, there is a turbine that uses some of the energy in the discharging air to drive the compressor. There is a long shaft connecting the turbine to the compressor ahead of it.

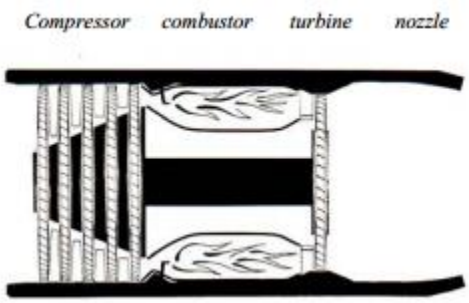

Figure 1-5 showing turbine engine as a cylinder of turbomachinery with unbalanced forces pushing forward.

1.1.7 Machinery Details

From an outsider's view, the flight crew and passengers rarely see the actual engine. What is seen is a large elliptically-shaped pod hanging from the wing or attached to the airplane fuselage toward the back of the airplane. This pod structure is called the *nacelle* or *cowling*. The engine is inside this nacelle.

The first nacelle component that incoming air encounters on its way through an airplane turbine engine is the inlet cowl. The purpose of the inlet cowl is to direct the incoming air evenly across the inlet stages of the engine. The shape of the interior of the inlet cowl is very carefully designed to guide this air.

The compressor of an airplane turbine engine has quite a job to do. The compressor has to take in an enormous volume of air and compress it to 1/10th or 1/15th of the volume it had outside the engine. This volume of air must be supplied continuously, not in pulses or periodic bursts.

The compression of this volume of air is accomplished by a rotating disk containing many airfoils, called blades, set at an angle to the disk rim. Each blade is close to the shape of a miniature propeller blade, and the angle at which it is set on the disk rim is called the angle of attack. This angle of attack is similar to the pitch of a propeller blade or an airplane wing in flight. As the disk with blades is forced to rotate by the turbine, each blade accelerates the air, thus pumping the air behind it. The effect is similar to a household window fan.

After the air passes through the blades on a disk, the air will be accelerated rearward and also forced circumferentially around in the direction of the rotating disk.

Figure 1-6 showing compressor rotor disk.

Any tendency for the air to go around in circles is counterproductive, so this tendency is corrected by putting another row of airfoils behind the rotating disk. This row is stationary and the airfoils are at an opposing angle.

What has just been described is a single stage of compression. Each stage consists of a rotating disk with many blades on the rim, called a rotor stage, and, behind it, another row of airfoils that is not rotating, called a stator. Air on the backside of this rotor/stator pair is accelerated rearward, and any tendency for the air to go around circumferentially is corrected.

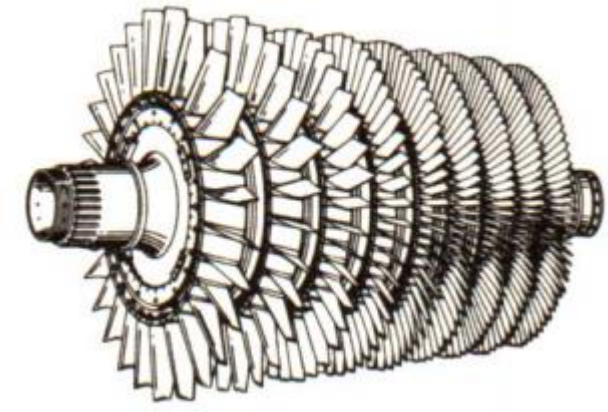

Figure 1-7 showing 9 stages of a compressor rotor

A single stage of compression can achieve perhaps 1.5:1 or 2.5:1 decrease in the air's volume. In order to achieve the 10:1 to 15:1 total compression needed for the engine to develop adequate power, the engine is built with many stages of compressors stacked in a line. Depending upon the engine design, there may be 10 to 15 stages in the total compressor.

As the air is compressed through the compressor, the air increases in velocity, temperature, and pressure. Air does not behave the same at elevated temperatures, pressures, and velocities as it does toward the front of the engine before it is compressed. In particular, this means that the speed that the compressor rotors must have at the back of the compressor is different than at the front of the compressor. If we had only a few stages, this difference could be ignored; but, for 10 to 15 stages of compressor, it would not be efficient to have all the stages go at the same rotating speed. The most common solution to this problem is to break the compressor in two. This way, the front 4 or 5 stages can rotate at one speed, while the rear 6 or 7 stages can rotate at a different, higher, speed. To accomplish this, we also need two separate turbines and two separate shafts.

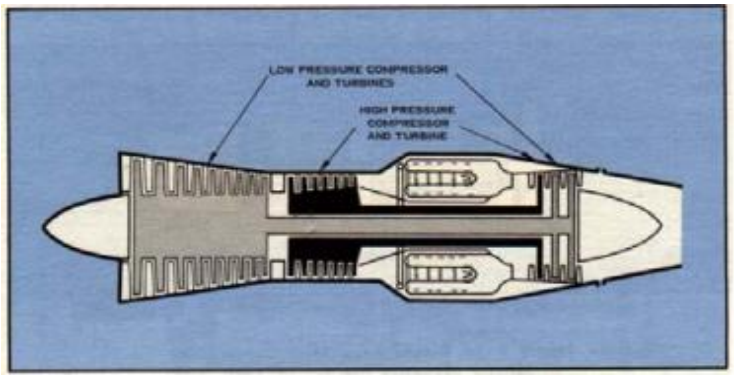

Figure 1-8 showing layout of a dual rotor airplane

Most of today's turbine engines are dual-rotor engines, meaning there are two distinct sets of rotating components. The rear compressor, or high-pressure compressor, is connected by a hollow shaft to a high-pressure turbine. This is the high rotor. In some literature, the rotors are called spools, such as the "high spool." In this text, we will use the term rotor. The high rotor is often referred to as N2 for short.

The front compressor, or low-pressure compressor, is in front of the high-pressure compressor. The turbine that drives the low-pressure compressor is behind the turbine that drives the high-pressure compressor. The low-pressure compressor is connected to the low-pressure turbine by a shaft that goes through the hollow shaft of the high rotor. The low-pressure rotor is called N1 for short.

The N1 and N2 rotors are not connected mechanically in any way. There is no gearing between them. As the air flows through the engine, each rotor is free to operate at its own efficient speed. These speeds are all quite precise and are carefully calculated by the engineers who designed the engine. The speed in RPM of each rotor is often displayed on the engine flight deck and identified by gages or readouts labeled N1 RPM and N2 RPM. Both rotors have their own redline limits.

1.1.8 The Turbofan Engine

In some engine designs, the N1 and N2 rotors may rotate in opposite directions, or there may be three rotors instead of two. Whether or not these conditions exist in any particular engine are engineering decisions and are of no consequence to the pilot.

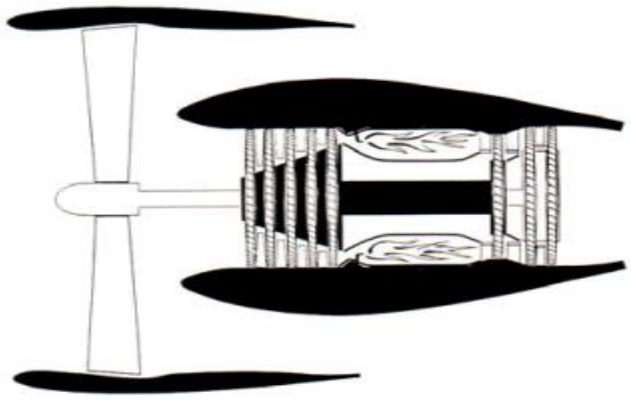

Figure 1-9 showing schematic of fan jet engine. In this sketch, the fan is the low-pressure compressor.

Chapter 1 Airplane Turbofan Engine Operation and Malfunctions

A turbofan engine is simply a turbine engine where the first stage compressor rotor is larger in diameter than the rest of the engine. This larger stage is called the fan. The air that passes through the fan near its inner diameter also passes through the remaining compressor stages in the core of the engine and is further compressed and processed through the engine cycle. The air that passes through the outer diameter of the fan rotor does not pass through the core of the engine, but instead passes along the outside of the engine. This air is called bypass air, and the ratio of bypass air to core air is called the bypass ratio.

The air accelerated by the fan in a turbofan engine contributes significantly to the thrust produced by the engine, particularly at low forward speeds and low altitudes. In large engines such as the engines that power the B747, B757, B767, A300, A310, etc., as much as three quarters of the thrust delivered by the engine is developed by the fan.

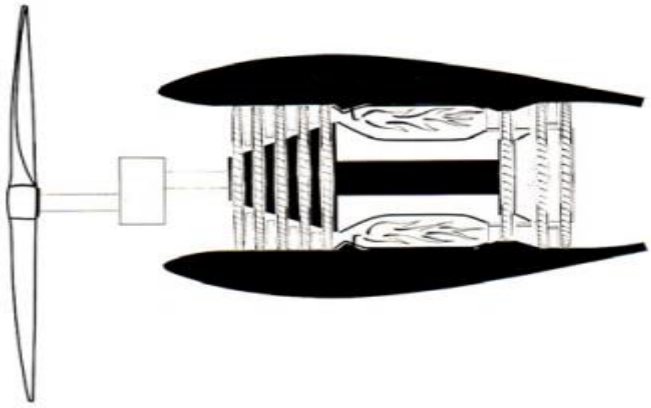

Figure 1-10 showing schematic of a turboprop.

The fan is not like a propeller. On a propeller, each blade acts like an airplane wing, developing lift as it rotates. The "lift" on a propeller blade pulls the engine and airplane forward through the air.

In a turbofan engine, thrust is developed by the fan rotor system, which includes the static structure (fan exit guide vanes) around it. The fan system acts like the open balloon in our example at the start of this discussion, and thus pushes the engine, and the airplane along with it, through the air from the unbalanced forces.

1.2 Engine Malfunctions

To provide effective understanding of and preparation for the correct responses to engine in-flight malfunctions, this chapter will describe turbofan engine malfunctions and their consequences in a manner that is applicable to almost all modern airplane turbofan-powered aircraft.

1.2.1 Compressor Surge

It is most important to provide an understanding of compressor surge. In modern turbofan engines, compressor surge is a rare event. If a compressor surge (sometimes called a compressor stall) occurs during high power at takeoff, the flight crew will hear a very loud bang, accompanied by yaw and vibration. The bang will likely be far beyond any engine noise, or other sound, the crew may have previously experienced in service.

A surge from a turbofan engine is the result of instability of the engine's operating cycle. Compressor surge may be caused by engine deterioration, it may be the result of ingestion of birds or ice, or it may be the final sound from a “severe engine damage” type of failure. The operating cycle of the turbine engine consists of intake, compression, ignition, and exhaust, which occur simultaneously in different places in the engine. The part of the cycle susceptible to instability is the compression phase.

In a turbine engine, compression is accomplished aerodynamically as the air passes through the stages of the compressor, rather than by confinement, as is the case in a piston engine. The air flowing over the compressor airfoils can stall just as the air over the wing of an airplane can. When this airfoil stall occurs, the passage of air through the compressor becomes unstable and the compressor can no longer compress the incoming air. The high-pressure air behind the stall further back in the engine escapes forward through the compressor and out the inlet.

This escape is sudden, rapid and often quite audible as a loud bang similar to an explosion. Engine surge can be accompanied by visible flames forward out the inlet and rearward out the tailpipe. Instruments may show high EGT and EPR or rotor speed changes, but, in many stalls, the event is over so quickly that the instruments do not have time to respond.

Once the air from within the engine escapes, the reason (reasons) for the instability may selfcorrect and the compression process may reestablish itself. A single surge and recovery will occur quite rapidly, usually within fractions of a second. Depending on the reason for the cause of the compressor instability, an engine might experience:

-1) A single self-recovering surge

-2) Multiple surges prior to self-recovery

-3) Multiple surges requiring pilot action in order to recover

-4) A non-recoverable surge.

The actual cause for the compressor surge is often complex and may or may not result from severe engine damage. Rarely does a single compressor surge cause severe engine damage, but sustained surging will eventually over-heat the turbine, as too much fuel is being provided for the volume of air that is reaching the combustor. Compressor blades may also be damaged and fail as a result of repeated violent surges; this will rapidly result in an engine which cannot run at any power setting. Additional information is provided below regarding single recoverable surge, self-recoverable after multiple surges, surge requiring flight crew action, and non- recoverable surge. In severe cases, the noise, vibration and aerodynamic forces can be very distracting. It may be difficult for the flight crew to remember that their most important task is to fly the airplane.

1.2.1.1 Single Self-Recoverable Surge

The flight crew hears a very loud bang or double bang. The instruments will fluctuate quickly, but, unless someone was looking at the engine gage at the time of the surge, the fluctuation might not be noticed. For example: During the surge event, Engine Pressure Ratio (EPR) can drop from takeoff (T/O) to 1.05 in 0.2 seconds. EPR can then vary from 1.1 to 1.05 at 0.2-second intervals two or three times. The low rotor speed (N1) can drop 16% in the first 0.2 seconds, then another 15% in the next 0.3 seconds. After recovery, EPR and N1 should return to pre-surge values along the normal acceleration schedule for the engine.

1.2.1.2 Multiple Surge Followed by Self-Recovery

Depending on the cause and conditions, the engine may surge multiple times, with each bang being separated by a couple of seconds. Since each bang usually represents a surge event as described above, the flight crew may detect the "single surge" described above for two seconds, then the engine will return to 98% of the pre-surge power for a few seconds. This cycle may repeat two or three times. During the surge and recovery process, there will likely be some rise in EGT. For example: EPR may fluctuate between 1.6 and 1.3, Exhaust Gas Temperature (EGT) may rise 5 degrees C/second, N1 may fluctuate between 103% and 95%, and fuel flow may drop 2% with

no change in thrust lever position. After 10 seconds, the engine gages should return to pre-surge values.

1.2.1.3 Surge Recoverable After Flight Crew Action

When surges occur as described in the last paragraph, but do not stop, flight crew action is required to stabilize the engine. The flight crew will notice the fluctuations described in "recoverable after two or three bangs," but the fluctuations and bangs will continue until the flight crew retards the thrust lever to idle. After the flight crew retards the thrust lever to idle, the engine parameters should decay to match thrust lever position. After the engine reaches idle, it may be re-accelerated back to power. If, upon re-advancing to high power, the engine surges again, the engine may be left at idle, or left at some intermediate power, or shutdown, according to the checklists applicable for the airplane. If the flight crew takes no action to stabilize the engine under these circumstances, the engine will continue to surge and may experience progressive secondary damage to the point where it fails completely.

1.2.1.4 Non-Recoverable Surge

When a compressor surge is not recoverable, there will be a single bang and the engine will decelerate to zero power as if the fuel had been chopped. This type of compressor surge can accompany a severe engine damage malfunction. It can also occur without any engine damage at all.

EPR can drop at a rate of .34/sec and EGT rise at a rate of 15 degrees C/sec, continuing for 8 seconds (peaking) after the thrust lever is pulled back to idle. N1 and N2 should decay at a rate consistent with shutting off the fuel, with fuel flow dropping to 25% of its pre-surge value in 2 seconds, tapering to 10% over the next 6 seconds

1.2.2 Flame Out

A flameout is a condition where the combustion process within the burner has stopped. A flameout will be accompanied by a drop in EGT, in engine core speed and in engine pressure ratio. Once the engine speed drops below idle, there may be other symptoms such as low oil pressure warnings and electrical generators dropping off line – in fact, many flameouts from low initial power settings

are first noticed when the generators drop off line and may be initially mistaken for electrical problems. The flameout may result from the engine running out of fuel, severe inclement weather, a volcanic ash encounter, a control system malfunction or unstable engine operation (such as a compressor stall). Multiple engine flameouts may result in a wide variety of flight deck symptoms as engine inputs are lost from electrical, pneumatic and hydraulic systems. These situations have resulted in pilots troubleshooting the airplane systems without recognizing and fixing the root cause – no engine power. Some airplanes have dedicated EICAS/ECAM messages to alert the flight crew to an engine rolling back below idle speed in flight; generally, an ENG FAIL or ENG THRUST message.

A flameout at take-off power is unusual – only about 10% of flameouts are at takeoff power. Flameouts occur most frequently from intermediate or low power settings such as cruise and descent. During these flight regimes, it is likely that the autopilot is in use. The autopilot will compensate for the asymmetrical thrust up to its limits and may then disconnect. Autopilot disconnect must then be accompanied by prompt, appropriate control inputs from the flight crew if the airplane is to maintain a normal attitude. If no external visual references are available, such as when flying over the ocean at night or in IMC, the likelihood of an upset increases. This condition of low-power engine loss with the autopilot on has caused several aircraft upsets, some of which were not recoverable. Flight control displacement may be the only obvious indication. Vigilance is required to detect these stealthy engine failures and to maintain a safe flight attitude while the situation is still recoverable. Once the fuel supply has been restored to the engine, the engine may be restarted in the manner prescribed by the applicable Airplane Flight or Operating Manual. Satisfactory engine restart should be confirmed by reference to all primary parameters – using only N1, for instance, has led to confusion during some inflight restarts. At some flight conditions, N1 may be very similar for a wind milling engine and an engine running at flight idle.

1.2.3 Fire

Engine fire almost always refers to a fire outside the engine but within the nacelle. A fire in the vicinity of the engine should be annunciated to the flight crew by a fire warning in the flight deck. It is unlikely that the flight crew will see, hear, or immediately smell an engine fire. Sometimes flight crews are advised of a fire by communication with the control tower.

It is important to know that, given a fire in the nacelle, there is adequate time to make the first priority "fly the airplane" before attending to the fire. It has been shown that, even in incidents of

fire indication immediately after takeoff, there is adequate time to continue climb to a safe altitude before attending to the engine. There may be economic damage to the nacelle, but the first priority of the flight crew should be to ensure the airplane continues in safe flight.

Flight crews should regard any fire warning as a fire, even if the indication goes away when the thrust lever is retarded to idle. The indication might be the result of pneumatic leaks of hot air into the nacelle. The fire indication could also be from a fire that is small or sheltered from the detector so that the fire is not apparent at low power. Fire indications may also result from faulty detection systems. Some fire detectors allow identification of a false indication (testing the fire loops), which may avoid the need for an IFSD. There have been times when the control tower has mistakenly reported the flames associated with a compressor surge as an engine "fire."

In the event of a fire warning annunciation, the flight crew must refer to the checklists and procedures specific to the airplane being flown. In general, once the decision is made that a fire exists and the aircraft is stabilized, engine shutdown should be immediately accomplished by shutting off fuel to the engine, both at the engine fuel control shutoff and the wing/pylon spar valve. All bleed air, electrical, and hydraulics from the affected engine will be disconnected or isolated from the airplane systems to prevent any fire from spreading to or contaminating associated airplane systems. This is accomplished by one common engine "fire handle." This controls the fire by greatly reducing the fuel available for combustion, by reducing the availability of pressurized air to any sump fire, by temporarily denying air to the fire through the discharge of fire extinguishant and by removing sources of re-ignition such as live electrical wiring and hot casings. It should be noted that some of these control measures may be less effective if the fire is the result of severe damage – the fire may take slightly longer to be extinguished in these circumstances. In the event of a shut down after an in-flight engine fire, there should be no attempt to restart the engine unless it is critical for continued safe flight – as the fire is likely to re-ignite once the engine is restarted.

1.2.4 Tailpipe Fires

One of the most alarming events for passengers, flight attendants, ground personnel and even air traffic control (ATC) to witness is a tailpipe fire. Fuel may puddle in the turbine casings and exhaust during start-up or shutdown, and then ignite. This can result in a highly-visible jet of flame out of the back of the engine, which may be tens of feet long. Passengers have initiated emergency evacuations in these instances, leading to serious injuries.

There may be no indication of an anomaly to the flight crew until the cabin crew or control tower draws attention to the problem. They are likely to describe it as an "Engine Fire," but a tailpipe fire will NOT result in a fire warning on the flight deck.

If notified of an engine fire without any indications in the cockpit, the flight crew should accomplish the tailpipe fire procedure. It will include motoring the engine to help extinguish the flames, while most other engine abnormal procedures will not.

Since the fire is burning within the turbine casing and exhaust nozzle, pulling the fire handle to discharge extinguishant to the space between casings and cowls will be ineffective. Pulling the fire handle may also make it impossible to dry motor the engine, which is the quickest way of extinguishing most tailpipe fires.

1.2.5 Hot Starts

During engine start, the compressor is very inefficient, as already discussed. If the engine experiences more than the usual difficulty accelerating (due to such problems as early starter cut-out, fuel mis-scheduling, or strong tailwinds), the engine may spend a considerable time at very low RPM (sub-idle). Normal engine cooling flows will not be effective during subidle operation, and turbine temperatures may appear relatively high. This is known as a hot start (or, if the engine completely stops accelerating toward idle, a hung start). The AFM indicates acceptable time/temperature limits for EGT during a hot start. More recent, FADEC-controlled engines may incorporate auto-start logic to detect and manage a hot start.

1.2.6 No Thrust Lever Response

A "no Thrust Lever Response" type of malfunction is more subtle than the other malfunctions previously discussed, so subtle that it can be completely overlooked, with potentially serious consequences to the airplane.

If an engine slowly loses power – or if, when the thrust lever is moved, the engine does not respond – the airplane will experience asymmetric thrust. This may be partly concealed by the autopilot's efforts to maintain the required flight condition.

If no external visual references are available, such as when flying over the ocean at night or in IMC, asymmetric thrust may persist for some time without the flight crew recognizing or correcting it. In several cases, this has led to airplane upset, which was not always recoverable. Vigilance is required to detect these stealthy engine failures and to maintain a safe flight attitude while the situation is still recoverable. As stated, this condition is subtle and not easy to detect.

Symptoms may include:

- Multiple system problems such as generators dropping off-line or low engine oil pressure
- Unexplained airplane attitude changes
- Large unexplained flight control surface deflections (autopilot on) or the need for large flight control inputs without apparent cause (autopilot off)
- Significant differences between primary parameters from one engine to the next.

If asymmetric thrust is suspected, the first response must be to make the appropriate trim or rudder input. Disconnecting the autopilot without first performing the appropriate control input or trim may result in a rapid roll maneuver.

1.3 Turbofan Simulation Using C-MAPSS Tool

Duo to the rare failures of the turbofan engines, C-MAPSS tool will be used to model operation and failures of the engine. C-MAPSS stands for 'Commercial Modular Aero-Propulsion System Simulation' and it is a tool for the simulation of a realistic large commercial turbofan engine. The code is a combination of Matlab (The MathWorks, Inc.) and Simulink (The MathWorks, Inc.) with a number of graphical user interface (GUI) screens that allow point-and-click operation and with editable fields that allow the user to enter specific values of his/her own choice. In addition to the engine model (called the 90K because it is produces about 90,000 lb of thrust), the package includes an atmospheric model capable of operation at:

(i) altitudes from sea level to 40,000 ft.

(ii) mach numbers from 0 to 0.90.

(iii) sea-level temperatures from –60 to 103 °F.

The package also includes a power-management system that allows the engine to be operated over a wide range of thrust levels throughout the full range of flight conditions.

A comprehensive control system is included that consists of:

(i) A fan-speed controller for which the user specifies the throttle-resolver angle (TRA).

(ii) Three high-limit regulators that prevent the engine from exceeding its design limits for core speed, engine-pressure ratio, and HPT exit temperature.

(iii) A fourth limit regulator that prevents the static pressure at the HPC exit from going too low.

(iv) Acceleration and deceleration limiters for the core speed.

(v) A comprehensive logic structure that integrates these control-system components in a manner similar to that used in real engine controllers such that integrator-windup problems are avoided.

Furthermore, all of the gains for the fan-speed controller and the four limit regulators are scheduled such that the controller and regulators perform as intended over the full range of flight conditions and power levels. The engine diagram in Figure 11 shows the main elements of the engine model and the flow chart in Figure 12 shows how the various subroutines are assembled in the simulation.

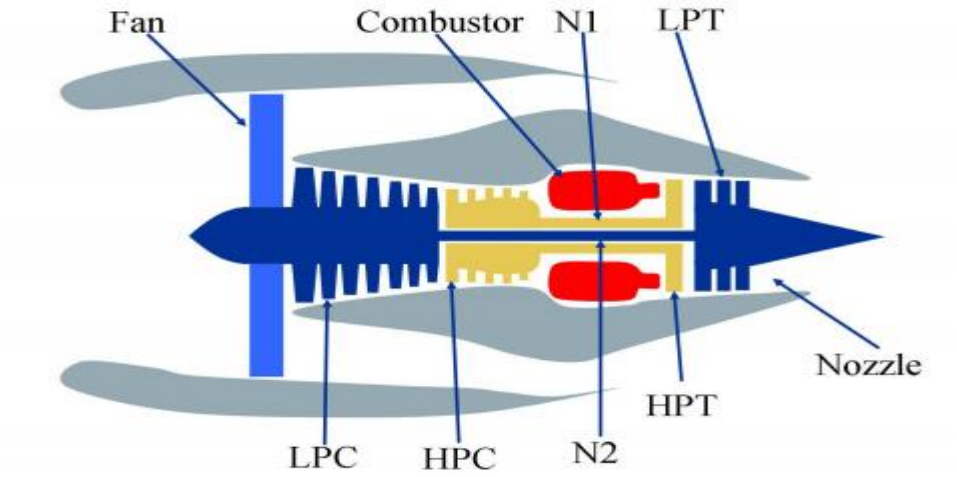

Figure 1-11 Simplified diagram of the 90K engine.

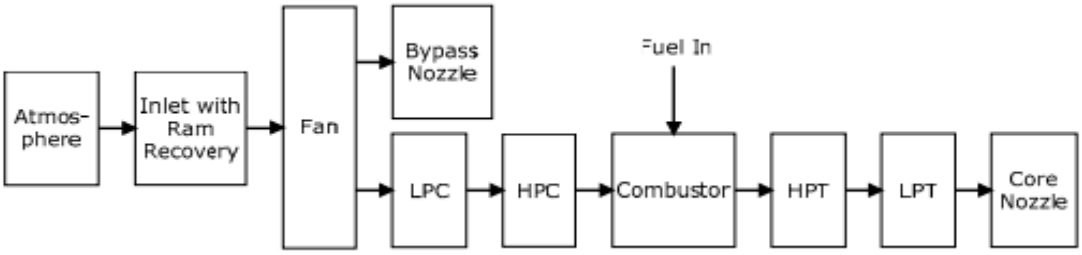

Figure 1-12 Subroutines of the 90K engine simulation with ducts and bleed omitted.

A number of GUI screens have been developed that make it easy for the user to work with either the open-loop engine (without any controller) or with the engine and its control system (closed loop). For the open-loop engine, transient simulations to doublet inputs can be run and linear engine models (LEMs) can be developed that have 14 inputs (Table 1) and 27 outputs (Table 2). C-MAPSS variables that are currently available internally but are not among the output variables are listed in Table 1.3. The inputs are fuel flow and a set of 13 health-parameter inputs that allow the user to simulate the effects of faults and deterioration in any of the engine's five rotating components (fan, LPC, HPC, HPT, and LPT). Using the GUIs provided for open-loop analysis, it is a simple matter for the user to save the LEM for later use and to compare its response with that of the nonlinear engine.

Table 1-1 Index, name, and symbol of 14 inputs to the 90K engine model

Index	Name	Symbol
1	Fuel flow	Wf (pps)
2	Fan efficiency modifier	fan_eff_mod
3	Fan flow modifier	Fan_flow_mod
4	Fan pressure-ratio modifier	Fan_PR_mod
5	LPC efficiency modifier	LPC_eff_mod
6	LPC flow modifier	LPC_flow_mod
7	LPC pressure-ratio modifier	LPC_PR_mod
8	HPC efficiency modifier	HPC_eff_mod
9	HPC flow modifier	HPC_flow_mod
10	HPC pressure-ratio modifier	HPC_PR_mod
11	HPT efficiency modifier	HPT_eff_mod
12	HPT flow modifier	HPT_flow_mod
13	LPT efficiency modifier	LPT_eff_mod
14	HPT flow modifier	HPT_flow_mod

Table 1-2 List of 27 output variables, with their indices in the output vector y and their units.

Index	Symbol	Description	Units
1	Nf	Physical fan speed	rpm
2	Ne	Physical core speed	rpm
3	epr	Engine pressure ratio (P50/P2)	..
4	P21	Total pressure at fan outlet	psia
5	T21	Total temperature at fan outlet	^0R
6	P24	Total pressure at LPC outlet	psia
7	T24	Total temperature at LPC outlet	^0R
8	P30	Total pressure at HPC outlet	psia
9	T30	Total temperature at HPC outlet	^0R
10	P40	Total temperature at burner outlet	psia
11	T40	Total temperature at burner outlet	^0R
12	P45	Total temperature at HPT outlet	psia
13	T48	Total temperature at HPT outlet	^0R
14	P50	Total temperature at LPT outlet	psia
15	T50	Total temperature at LPT outlet	^0R
16	W21	Fan flow	pps
17	Fn	Net thrust	lpf
18	Fg	Gross thrust	lpf
19	SmFan	Fan stall margin	..
20	SmLPC	LPC stall margin	..
21	SmHPC	HPC stall margin	..
22	NRf	Corrected fan speed	rpm
23	NRe	Corrected core speed	rpm
24	P15	Total pressure in bypass-duct	psia
25	PCNfR	Percent corrected fan speed	pct
26	Ps30	Static pressure at HPC outlet	psia
27	phi	Ratio of fuel flow to Ps30	pps/psi

A controller-design GUI guides the user in the design of fan-speed controllers and limit regulators, using a LEM to represent the engine. The design GUI implements the model-matching algorithm of John Edmunds (ref. 1). However, it can be adapted for use with other design methods, should the user wish to do so. In order to avoid model complexity that is not required unless controllers are being designed that are capable of running a real engine, and to be able to attain fast execution speeds, the sensors and actuators are assumed to be ideal.

Chapter 2

Dimensionality Reduction Techniques

2.1 Principal Component Analysis

2.1.1 Introduction to PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called *principal components* (or sometimes, principal modes of variation). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA was invented in 1901 by Karl Pearson [11] as an analogue of the principal axis theorem in mechanics; it was later independently developed and named by Harold Hotelling in the 1930s [12]. Depending on the field of application, it is also named the discrete Karhunen–Loève transform (KLT) in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition(POD) in mechanical engineering, singular value decomposition (SVD) of X (Golub and Van Loan, 1983), eigenvalue decomposition (EVD) of $\boldsymbol{X^T X}$ in linear algebra, factor analysis (for a discussion of the differences between PCA and factor analysis see Ch. 7 of [13]), Eckart–Young theorem (Harman, 1960), or Schmidt–Mirsky theorem in psychometrics, empirical orthogonal functions (EOF) in meteorological science, empirical eigenfunction decomposition (Sirovich, 1987), empirical component analysis (Lorenz, 1956), quasiharmonic modes (Brooks et al., 1988),spectral decomposition in noise and vibration, and empirical modal analysis in structural dynamics.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It's often used to visualize genetic distance and relatedness between populations. PCA can be done by

eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, usually after mean centering (and normalizing or using Z-scores) the data matrix for each attribute [14]. The results of a PCA are usually discussed in terms of *component scores*, sometimes called *factor scores* (the transformed variable values corresponding to a particular data point), and *loadings* (the weight by which each standardized original variable should be multiplied to get the component score) [15].

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualized as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.

PCA is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset [16] [17].

2.1.2 Derivation of PCA

2.1.2.1 Intuition

PCA can be thought of as fitting an *n*-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a commensurately small amount of information.

To find the axes of the ellipsoid, we must first subtract the mean of each variable from the dataset to center the data around the origin. Then, we compute the covariance matrix of the data, and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix. Then we must normalize each of the orthogonal eigenvectors to become unit vectors. Once this is done, each of the mutually orthogonal, unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data. The proportion of the variance that each eigenvector represents can be

calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. This procedure is sensitive to the scaling of the data, and there is no consensus as to how to best scale the data to obtain optimal results.

2.1.2.2 Notation and Assumption

Before we go to the mathematical procedures for deriving PCA we'll define some notations and make some assumptions:

- $\boldsymbol{x}$ is a vector of $\boldsymbol{p}$ random variables
- α_k is a vector of $\boldsymbol{p}$ constants
- $\boldsymbol{\alpha_k' x} = \sum_{j=1}^{\boldsymbol{p}} \boldsymbol{\alpha_{kj} x_j}$
- $\boldsymbol{\Sigma}$ is the *known* covariance matrix for the random variable **x**
- Foreshadowing : $\boldsymbol{\Sigma}$ will be replaced with **S**, the sample covariance matrix, when $\boldsymbol{\Sigma}$ is unknown.

As a first step we have to find $\boldsymbol{\alpha_k' x}$ that maximizes:

$$\text{Var}(\boldsymbol{\alpha_k' x}) = \boldsymbol{\alpha_k' \Sigma \alpha_k} \tag{2 .1}$$

Without any constrain we would pick a very big $\boldsymbol{\alpha_k}$ so we'll choose a normalization constrain (unit length vector):

$$\boldsymbol{\alpha_k' \alpha_k = 1} \tag{2.2}$$

Then to maximize $\boldsymbol{\alpha_k' \Sigma \alpha_k}$ subject to $\boldsymbol{\alpha_k' \alpha_k = 1}$, *Lagrange multipliers* will be used. We maximize the function:

$$\boldsymbol{\alpha_k' \Sigma \alpha_k} - \lambda(\boldsymbol{\alpha_k' \alpha_k - 1}) \tag{2.3}$$

w.r.t. to $\boldsymbol{\alpha_k}$ by differentiating w.r.t. to $\boldsymbol{\alpha_k}$.

And this results in:

$$\frac{d}{d\boldsymbol{\alpha_k}}(\boldsymbol{\alpha_k' \Sigma \alpha_k} - \lambda(\boldsymbol{\alpha_k' \alpha_k - 1})) = 0 \tag{2.4}$$

$$\boldsymbol{\Sigma\alpha_k} - \lambda\boldsymbol{\alpha_k} = 0$$
$$\boldsymbol{\Sigma\alpha_k} = \lambda\boldsymbol{\alpha_k}$$

This should be recognizable as an eigenvector equation where $\boldsymbol{\alpha_k}$ is an eigenvector of $\boldsymbol{\Sigma_b f}$ and $\boldsymbol{\lambda}_k$ is the associated eigenvalue. The question now is which eigenvector to choose? If we recognize that the quantity to be maximized

$$\boldsymbol{\alpha_k} = \boldsymbol{\alpha_k'}\lambda_k\boldsymbol{\alpha_k} = \lambda_k\boldsymbol{\alpha_k'}\boldsymbol{\alpha_k} = \lambda_k \quad (2.5)$$

then we should choose λ_k to be as big as possible. So, calling λ_1 the largest eigenvalue of Σ and $\boldsymbol{\alpha_1}$ the corresponding eigenvector then the solution to:

$$\boldsymbol{\Sigma\alpha_1} = \lambda_1\boldsymbol{\alpha_1} \quad (2.6)$$

is the first principal component of $\boldsymbol{x}$. In general $\boldsymbol{\alpha_k}$ will be the k^{th} PC of **x** and Var($\boldsymbol{\alpha}'\mathbf{x}$) = λ_k We will demonstrate this for $k = 2$, $k > 2$ is more involved but similar.

The second PC, $\boldsymbol{\alpha_2 x}$ maximizes $\boldsymbol{\alpha_2 \Sigma \alpha_2}$ subject to being uncorrelated with $\boldsymbol{\alpha_1}\mathbf{x}$. The uncorrelation constraint can be expressed using equation 2.7:

$$cov(\boldsymbol{\alpha_1'x}, \boldsymbol{\alpha_2'x}) = \boldsymbol{\alpha_1'\Sigma\alpha_2} = \boldsymbol{\alpha_2'\Sigma\alpha_1} = \boldsymbol{\alpha_2'}\lambda_1\boldsymbol{\alpha_1'}$$
$$= \lambda_1\boldsymbol{\alpha_2'\alpha} = \lambda_1\boldsymbol{\alpha_1'\alpha_2} = \mathbf{0} \quad (2.7)$$

if we choose the last we can write a Lagrangian to maximize $\boldsymbol{\alpha_2}$ we get equation 2.8 :

$$\boldsymbol{\alpha_2'\Sigma\alpha_2} - \lambda(\boldsymbol{\alpha_2'\alpha_2} - \mathbf{1}) - \phi\boldsymbol{\alpha_2'\alpha_1} \quad (2.8)$$

Differentiation of this quantity w.r.t. $\boldsymbol{\alpha_2}$ (and setting the result equal to zero) yields:

$$\frac{\boldsymbol{d}}{\boldsymbol{d\alpha_2}}(\boldsymbol{\alpha_2'\Sigma\alpha_2} - \lambda(\boldsymbol{\alpha_2'\alpha_2} - \mathbf{1}) - \phi\boldsymbol{\alpha_2'\alpha_1}) = \mathbf{0}$$
$$\boldsymbol{\Sigma\alpha_2} - \lambda_2\boldsymbol{\alpha_2} - \phi\boldsymbol{\alpha_1} = \mathbf{0} \quad (2.9)$$

If we left multiply $\boldsymbol{\alpha_1}$ into this expression:

$$\boldsymbol{\alpha_1'\Sigma\alpha_2} - \lambda\boldsymbol{\alpha_1'\alpha_2} - \phi\boldsymbol{\alpha_1'\alpha_1} = \mathbf{0} \quad (2.10)$$

$$\mathbf{0} - 0 - \phi\mathbf{1} = \mathbf{0}$$

then we can see that ϕ must be zero and that when this is true that we are left with:

$$\boldsymbol{\Sigma\alpha_2} - \lambda\boldsymbol{\alpha_2} = \mathbf{0} \tag{2.11}$$

Clearly the last equation is another eigenvalue equation and the same strategy of choosing α_2 to be the eigenvector associated with the second largest eigenvalue yields the second PC of **x**, namely $\boldsymbol{\alpha}_2'\mathbf{x}$.

This process can be repeated for $k = 1 \ldots p$ yielding up to p different eigenvectors of $\boldsymbol{\Sigma}$ along with the corresponding eigenvalues $\lambda_1, \ldots \lambda_p$.

Furthermore, the variance of each of the PC's are given by equation 2.12:

$$\text{Var}[\boldsymbol{\alpha}_k'\mathbf{x}] = \lambda_k, \quad k = 1, 2, \ldots, p \tag{2.12}$$

2.1.3 Properties of PCA

In this section three of the mathematical and statistical properties of PCs are discussed, based on a known population covariance (or correlation) matrix $\boldsymbol{\Sigma}$.

2.1.3.1 Property 1

For any integer q, $1 \leq q \leq p$, consider the orthogonal linear transformation

$$\boldsymbol{y} = \boldsymbol{B}'\boldsymbol{x} \tag{2.13}$$

where $\boldsymbol{y}$ is a *q-element* vector and $\boldsymbol{B}'$ is a $(q \times p)$ matrix, and let $\boldsymbol{\Sigma_y} = \boldsymbol{B}'\boldsymbol{\Sigma}\boldsymbol{B}$ be the variance-covariance matrix for $\boldsymbol{y}$. Then the trace of $\boldsymbol{\Sigma_y}$, denoted $\boldsymbol{tr(\Sigma_y)}$, is maximized by taking$\boldsymbol{B} = \boldsymbol{A_q}$, where $\boldsymbol{A_q}$ consists of the first q columns of is the transposition of $\boldsymbol{A}$.

2.1.3.2 Property 2

Consider again the orthonormal transformation

$$\boldsymbol{y} = \boldsymbol{B}'\boldsymbol{x} \tag{2.14}$$

with $\boldsymbol{x}$, $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{\Sigma_y}$ defined as before. Then $\boldsymbol{tr(\Sigma_y)}$ is minimized by taking $\boldsymbol{B = A_q^*}$ where $\boldsymbol{A_q^*}$ consists of the last q columns of $\boldsymbol{A}$.

The statistical implication of this property is that the last few PCs are not simply unstructured left-overs after removing the important PCs. Because these last PCs have variances as small as possible they are useful in their own right. They can help to detect unsuspected near-constant linear relationships between the elements of x, and they may also be useful in regression, in selecting a subset of variables from x, and in outlier detection.

2.1.3.3 Property 3

Spectral decomposition of $\boldsymbol{\Sigma}$

$$\boldsymbol{\Sigma} = \lambda_1 \boldsymbol{\alpha_1' \alpha_1} + \cdots + \lambda_p \boldsymbol{\alpha_p' \alpha_p} \tag{2.15}$$

Before we look at its usage, we first look at diagonal elements,

$$Var(x_j) = \sum_{k=1}^{P} \lambda_k \alpha_{kj}^2 \tag{2.16}$$

Then, perhaps the main statistical implication of the result is that not only can we decompose the combined variances of all the elements of x into decreasing contributions due to each PC, but we can also decompose the whole covariance matrix into contributions $\lambda_k \boldsymbol{\alpha_k \alpha_k'}$ from each PC. Although not strictly decreasing, the elements of $\lambda_k \boldsymbol{\alpha_k \alpha_k'}$ will tend to become smaller as $\mathbf{k}$ increases, as $\lambda_k \boldsymbol{\alpha_k \alpha_k'}$ is non-increasing for $\boldsymbol{k}$ increasing, whereas the elements of $\boldsymbol{\alpha_k}$ tend to stay about the same size because of the normalization constraints: $\boldsymbol{\alpha_k \alpha_k' = 1, k = 1, \ldots, p.}$

2.1.4 PCA Using the Sample Covariance Matrix

If we recall that the sample covariance matrix (an unbiased estimator for the covariance matrix of $\mathbf{x}$) is given by equation 2.17

$$\boldsymbol{S = \frac{1}{n-1} X'X} \tag{2.17}$$

where $\boldsymbol{X}$ is a ($n \times p$) matrix with (i, j)*th* element ($x_{ij} - x_j$) (in other words, **X** is a zero mean design matrix). We construct the matrix **A** by combining the p eigenvectors of **S** (or eigenvectors of $\boldsymbol{X'X}$ – they're the same) then we can define a matrix of PC scores

$$\boldsymbol{Z} = \boldsymbol{XA} \tag{2.18}$$

Of course, if we instead form **Z** by selecting the q eigenvectors corresponding to the q largest eigenvalues of **S** when forming **A** then we can achieve an "optimal" (in some senses) q-dimensional projection of **x**.

2.1.4.1 Computing the PCA Loading Matrix

Given the sample covariance matrix

$$\boldsymbol{S} = \frac{\mathbf{1}}{\boldsymbol{n-1}}\boldsymbol{X'X} \tag{2.19}$$

the most straightforward way of computing the PCA loading matrix is to utilize the singular value decomposition of $\mathbf{S} = \boldsymbol{A'\Lambda A}$ where **A** is a matrix consisting of the eigenvectors of **S** and **Λ** is a diagonal matrix whose diagonal elements are the eigenvalues corresponding to each eigenvector. Creating a reduced dimensionality projection of **X** is accomplished by selecting the q largest eigenvalues in **Λ** and retaining the q corresponding eigenvectors from **A.**

2.2 Independent Component Analysis (ICA)

Independent component analysis (ICA) is a method for finding underlying factors or components from multivariate (multidimensional) statistical data. What distinguishes ICA from other methods is that it looks for components that are both *statistically independent*, and *nongaussian*.

2.2.1 Motivation

Imagine that you are in a room where two people are speaking simultaneously. You have two microphones, which you hold in different locations. The microphones give you two recorded time signals, which we could denote by $x_1(t)$ and $x_2(t)$, with x_1 and x_2 the amplitudes, and t the time index. Each of these recorded signals is a weighted sum of the speech signals emitted by the two speakers, which we denote by s_1 (t) and s_2 (t). We could express this as a linear equation:

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$
$$x_2(t) = a_{21}s_1 + a_{22}s_2 \quad (2.20)$$

where a_{11}, a_{12}, a_{21}, and a_{22} are some parameters that depend on the distances of the microphones from the speakers. It would be very useful if you could now estimate the two original speech signals s_1 (*t*) and s_2 (*t*), using only the recorded signals x_1 (*t*) and x_2 (*t*). This is called the *cocktail-party problem*. For the time being, we omit any time delays or other extra factors from our simplified mixing model.

As an illustration, consider the waveforms in Figure 2.1 and Figure 2.2. These are, of course, not realistic speech signals, but suffice for this illustration. The original speech signals could look something like those in Figure 2.1 and the mixed signals could look like those in Figure 2.2. The problem is to recover the data in Figure 2.1 using only the data in Figure 2.2.

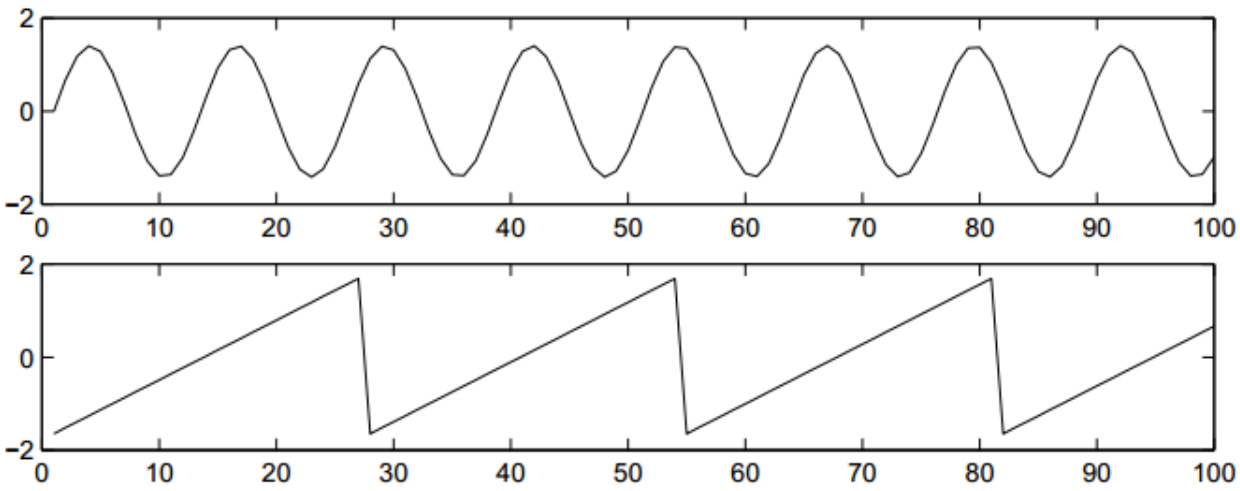

Figure 2-1 The original signals

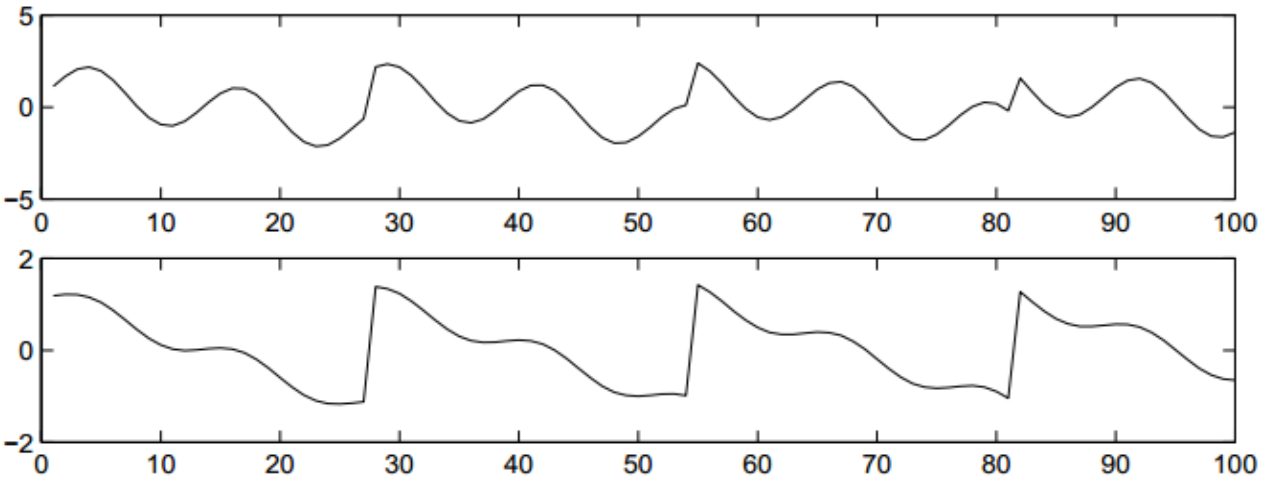

Figure 2-2 The observed mixtures of the source signals in Figure 2.2

Actually, if we knew the parameters a_{ij}, we could solve the linear equation (2.20) by classical methods. The point is, however, that if you don't know the a_{ij}, the problem is considerably more difficult.

One approach to solving this problem would be to use some information on the statistical properties of the signals $s_i(t)$ to estimate the a_{ii}. Actually, and perhaps surprisingly, it turns out that it is enough to assume that s_1 (t) and s_2 (t), at each time instant t, are *statistically independent*. This is not an unrealistic assumption in many cases, and it need not be exactly true in practice. The recently developed technique of Independent Component Analysis, or ICA, can be used to estimate the a_{ij} based on the information of their independence, which allows us to separate the two original source signals $s_1(t)$ and $s_2(t)$ from their mixtures $x_1(t)$ and $x_2(t)$. Figure 2.3 gives the two signals estimated by the ICA method. As can be seen, these are very close to the original source signals (their signs are reversed, but this has no significance.)

2.2.2 Definition of ICA

To rigorously define ICA (Jutten and Hérault, 1991; Comon, 1994), we can use a statistical "latent variables" model. Assume that we observe n linear mixtures $x_1,...,x_n$ of n independent components

$$x_j = a_{j1}s_1 + a_{j2}s_2 + \ldots + a\, a_{jn}s_n, \text{for all } j. \quad (2.21)$$

We have now dropped the time index *t*; in the ICA model, we assume that each mixture x_j as well as each independent component s_k is a random variable, instead of a proper time signal. The observed values x_j (*t*), e.g., the microphone signals in the cocktail party problem, are then a sample of this random variable. Without loss of generality, we can assume that both the mixture variables and the independent components have zero mean: If this is not true, then the observable variables x_i can always be centered by subtracting the sample mean, which makes the model zero-mean.

It is convenient to use vector-matrix notation instead of the sums like in the previous equation. Let us denote by $\boldsymbol{x}$ the random vector whose elements are the mixtures x_1,..., x_n, and likewise by **s** the random vector with elements s_1,..., s_n. Let us denote by **A** the matrix with elements a_{ij}. Generally, bold lower case letters indicate vectors and bold upper-case letters denote matrices. All vectors are understood as column vectors; thus $\boldsymbol{X^T}$, or the transpose of **X**, is a row vector. Using this vector-matrix notation, the above mixing model is written as

$$\mathbf{x} = \mathbf{As}. \quad (2.22)$$

Sometimes we need the columns of matrix **A**; denoting them by $\boldsymbol{a_j}$ the model can also be written as

$$\boldsymbol{x} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i \quad (1.23)$$

The statistical model in equation (2.22) is called independent component analysis, or ICA model. The ICA model is a generative model, which means that it describes how the observed data are generated by a process of mixing the components s_i. The independent components are latent variables, meaning that they cannot be directly observed. Also the mixing matrix is assumed to be unknown. All we observe is the random vector **x**, and we must estimate both **A** and **s** using it. This must be done under as general assumptions as possible.

The starting point for ICA is the very simple assumption that the components s_i are *statistically independent*. It will be seen below that we must also assume that the independent component must have *nongaussian* distributions. However, in the basic model we do not assume these distributions

known (if they are known, the problem is considerably simplified.) For simplicity, we are also assuming that the unknown mixing matrix is square, but this assumption can be sometimes relaxed. Then, after estimating the matrix **A**, we can compute its inverse, say **W**, and obtain the independent component simply by:

$$\mathbf{s} = \mathbf{W}\mathbf{x}. \tag{2.24}$$

ICA is very closely related to the method called *blind source separation* (BSS) or blind signal separation. A "source" means here an original signal, i.e. independent component, like the speaker in a cocktail party problem. "Blind" means that we know very little, if anything, on the mixing matrix, and make little assumptions on the source signals. ICA is one method, perhaps the most widely used, for performing blind source separation. In many applications, it would be more realistic to assume that there is some noise in the measurements (see e.g. (Hyvärinen, 1998a; Hyvärinen, 1999c)), which would mean adding a noise term in the model. For simplicity, we omit any noise terms, since the estimation of the noise-free model is difficult enough in itself, and seems to be sufficient for many applications.

2.2.3 Preprocessing

Before examining specific ICA algorithms, it is instructive to discuss preprocessing steps that are generally carried out before ICA.

2.2.3.1 Centering

A simple preprocessing step that is commonly performed is to "center" the observation vector x by subtracting its mean vector $m = E\{x\}$. That is then we obtain the centered observation vector, x_c, as follows:

$$x_c = x - m \tag{2.25}$$

This step simplifies ICA algorithms by allowing us to assume a zero mean. Once the unmixing matrix has been estimated using the centered data, we can obtain the actual estimates of the independent components as follows:

(2.26)

$$\hat{s}(t) = A^{-1}(x_c + m)$$

From this point on, all observation vectors will be assumed centered. The mixing matrix, on the other hand, remains the same after this preprocessing, so we can always do this without affecting the estimation of the mixing matrix.

2.2.3.2 Whitening

Another step which is very useful in practice is to prewhiten the observation vector *x*. Whitening involves linearly transforming the observation vector such that its components are uncorrelated and have unit variance [18]. Let x_w denote the whitened vector, then it satisfies the following equation:

$$E\{x_w x_w^T\} = I \tag{2.27}$$

where $E\{x_w x_w^T\}$ is the covariance matrix of x_w. Also, since the ICA framework is insensitive to the variances of the independent components, we can assume without loss of generality that the source vector, *s*, is white, i.e. $E\{ss^T\} = I$

A simple method to perform the whitening transformation is to use the eigenvalue decomposition (EVD) of *x* [18]. That is, we decompose the covariance matrix of *x* as follows:

$$E\{xx^T\} = VDV^T \tag{2.28}$$

where *V* is the matrix of eigenvectors of $E\{xx^T\}$,and *D* is the diagonal matrix of eigenvalues, i.e. $D = diag\{\lambda_1,\lambda_2,...,\lambda_n\}$ The observation vector can be whitened by the following transformation:

$$x_w = VD^{-1/2}V^T x \tag{2.29}$$

where the matrix $D^{-1/2}$ is obtained by a simple component wise operation as $D^{-1/2} = diag\{\lambda_1^{-1/2}, \lambda_2^{-1/2},..., \lambda_n^{-1/2}\}$. Whitening transforms the mixing matrix into a new one, which is orthogonal.

$$x_w = \mathrm{V}D^{-1/2}V^T\mathrm{As} = A_w\mathrm{s} \tag{2.30}$$

Hence,

$$\begin{aligned} E\{x_w x_w^T\} &= A_w E\{ss^T\}A_w^T \\ &= A_w A_w^T \\ &= I \end{aligned} \tag{2.31}$$

Whitening thus reduces the number of parameters to be estimated. Instead of having to estimate the n^2 elements of the original matrix *A*, we only need to estimate the new orthogonal mixing matrix, where An orthogonal matrix has $n(n-1)/2$ degrees of freedom. One can say that whitening solves half of the ICA problem. This is a very useful step as whitening is a simple and efficient process that significantly reduces the computational complexity of ICA. An illustration of the whitening process with simple ICA source separation process is explained in the later section.

2.2.4 ICA Algorithms

There are several ICA algorithms available in literature. However, the following three algorithms are widely used in numerous signal processing applications. These includes *FastICA*, *JADE*, and *Infomax*. Each algorithm used a different approach to solve equation.

2.2.4.1 FastICA

FastICA is a fixed point ICA algorithm that employs higher order statistics for the recovery of independent sources. FastICA can estimate ICs one by one (deflation approach) or simultaneously (symmetric approach). FastICA uses simple estimates of Negentropy based on the maximum entropy principle, which requires the use of appropriate nonlinearities for the learning rule of the neural network.

Fixed point algorithm is based on the mutual information. Which can be written as:

$$I(s) = \int f_s(s) log \frac{f_s(s)}{\prod f_{s_i}(s_i)} ds \tag{2.32}$$

This measure is kind of distance of independence. Minimizing mutual information leads to ICA solution. For the fast ICA algorithm, the above equation is rewritten as:

$$I(s) = J(s) - \sum_i Js_i + \frac{1}{2} log \frac{\prod C_{ii}}{detC_{ss}} \tag{2.33}$$

where $\hat{s} = Wx$, C_{ss} is the correlation matrix, and c_{ii} is the *i*th diagonal element of the correlation matrix. The last term is zero because s_i are supposed to be uncorrelated. The first term is constant for a problem, because of the invariance in Negentropy. The problem is now reduced to separately maximising the Negentropy of each component.

Estimation of Negentropy is a delicate problem. The papers [19] [20] and [21] [22] have addressed this problem. For the general version of fixed point algorithm, the approximation was based on a maximum entropy principle. The algorithm works with whitened data, although aversion of non-whitened data exists.

– Criteria

The maximisation is preferred over the following index

$$J_G(w) = [E\{G(w^T v)\} - E\{G(v)\}^2 \tag{2.34}$$

to find one independent component, with ν standard gaussian variable, and *G*, the one-unit contrast function.

– Update rule

Update rule for the generic algorithm is

$$w^* = E\{vg(w^T v)\} - E\{\acute{g}(w^T v)\}w$$
$$w = w^*/\|w^*\| \tag{2.35}$$

to extract one component. There is symmetric version of the FP algorithm, whose update rule is

$$W^* = E\{g(Wv)v^T\} - Diag(E\{\acute{g}(Wv)\})W$$
$$W = (W^* W^{*T})^{-\frac{1}{2}} W^* \tag{2.36}$$

where *Diag*(*v*) is a diagonal matrix with $Diag_{ii}(v) = v_i$.

– Parameters

FastICA uses the following nonlinear parameters for convergence.

$$g(y) = \begin{cases} y^3 \\ \tanh(y) \end{cases} \tag{2.37}$$

The choice is free except that the symmetric algorithm with *tanh* non linearity does not separate super Gaussian signals. Otherwise the choice can be devoted to the other criteria, for instance the cubic non linearity is faster, whereas the *tanh* linearity is more stable. These questions are addressed in [23].

In practice, the expectations in FastICA must be replaced by their estimates. The natural estimates are of course the corresponding sample means. Ideally, all the data available should be used, but this is often not a good idea because the computations may become too demanding. Then the averages can be estimated using a smaller sample, whose size may have a considerable effect on the accuracy of the final estimates. The sample points should be chosen separately at every iteration. If the convergence is not satisfactory, one may then increase the sample size. This thesis uses FastICA algorithm for all applications.

Chapter 3

Ensemble Machine Learning

3.1 Introduction

Ensemble is a Machine Learning concept in which the idea is to train multiple models using the same learning algorithm. The ensembles take part in a bigger group of methods, called multiclassifiers, where a set of hundreds or thousands of learners with a common objective are used together to solve the problem.

3.1.1 Bagging

Bagging (*Bootstrap Aggregating*) is an ensemble method that creates separate samples of the training dataset and creates a classifier for each sample. The results of these multiple classifiers are then combined (such as averaged or majority voting). The trick is that each sample of the training dataset is different, giving each classifier that is trained, a subtly different focus and perspective on the problem [24].

3.1.2 Boosting

Boosting is an ensemble method that starts out with a base classifier that is prepared on the training data. A second classifier is then created behind it to focus on the instances in the training data that the first classifier got wrong. The process continues to add classifiers until a limit is reached in the number of models or accuracy [24].

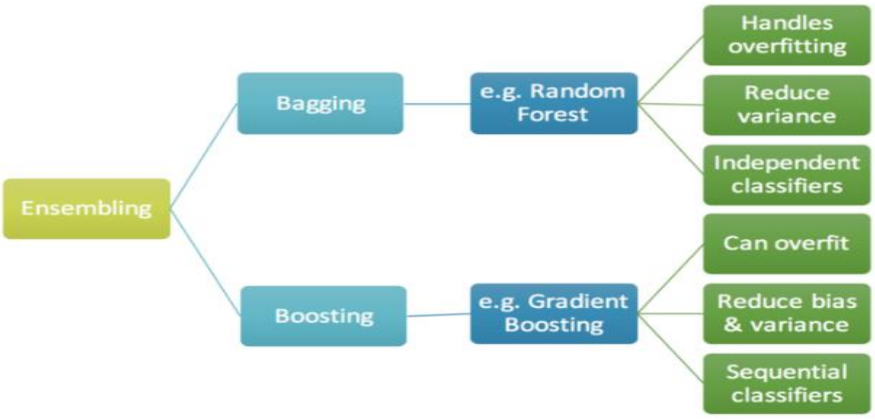

Figure 3-1 Ensembling Diagram

3.2 Decision Tree

3.2.1 Introduction

A decision tree is a predictive model which can be used to approximate discrete-valued target functions. Decision tree are usually represented graphically as hierarchical structure. The topmost node, which does not have any incoming edge, is called root node. A node with outgoing edges are called internal node. Each internal node denotes a test on an attribute. Each edge represents an outcome of the test. All other nodes are leaf nodes. Each leaf holds a class label. When classifying a new instance, the instance is navigated from the root node down to the leaf, according to the outcome of the tests along the path. The class label in the leaf node indicates the class to which the instance should belong.

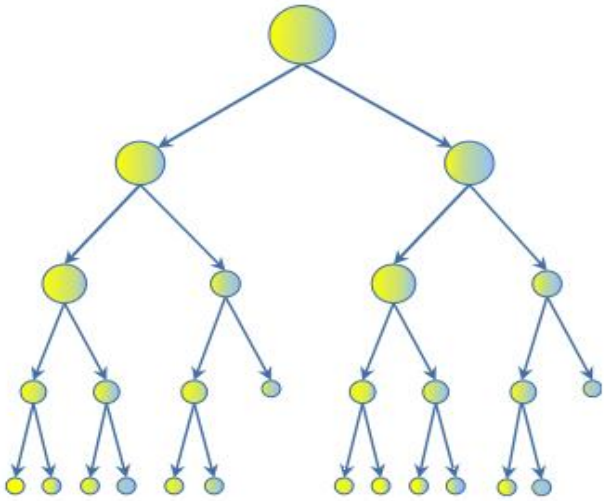

Figure 0-2 Decision Tree

3.2.2 Decision Tree Algorithm

Denote *D* as a data partition, attribute list is a list of candidate attributes describing the data set and attribute selection method is a heuristic method for selecting the splitting criterion that best separates a given data partition, *D*. A basic decision tree algorithm, called buildTree (*D*, attribute list) is summarized as follows [32].

• The tree first starts as a single node *N*.

• If the instances in *D* are all of the same class, *N* becomes a leaf and labeled with that class.

• Otherwise, *Attribute selection method* is called to decide the splitting criterion. The splitting criterion indicates the splitting attribute and may also indicate a split point. If the splitting attribute is nominal, it will be removed from the attribute list.

• The node *N* is labeled with the splitting criterion, which serves as a test at the node. A branch is grown from node N for each of the outcomes of the splitting criterion. The training instances in D are partitioned accordingly into, for example, *D*1, *D*2, ...*Dm*.

• Let *Di* be the set of instances in *D* satisfying outcome *i*. If *Di* is empty, *N* is attached a leaf labeled with the majority class in *D*. Otherwise, it is attached the node returned by buildTree (*D*, attribute list). The recursive partitioning stops when any one of the following terminating conditions is reached.

– All instances in the training set belong to a single class.

– There are no remaining attributes which can be used for further partition.

– There are no instances for a given branch.

Besides three stopping criteria presented above, in some algorithms, there are some other conditions, such as the maximum tree depth has been reached, the number of cases in the terminal node is less than the minimum number of cases for parent nodes or the gained information at the best splitting criterion is not greater than a certain threshold.

• The resulting decision tree is returned.

Decision tree learning is one of the most popular methods and has been successfully applied in many fields, such as finance, marketing, engineering and medicine. The reason for its popularity, according to many researchers, is that it is simple and transparent. The construction of a decision tree is fast and does not require any domain knowledge or parameter setting. Its representation in tree form is intuitive and easy to interpret for humans. However, successful use may depend on the data set at hand.

Many decision tree algorithms have been developed, including ID3 [25], C4.5(a successor of ID3) [26] and CART (Classification and Regression Trees) [27]. Most of them adopt a greedy approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Although those algorithms differ in many aspects, the main differences are their attribute selection measures and pruning tree methods.

The next sections will present some attribute selection measures and pruning tree methods that are commonly used.

3.2.3 Attribute Selection Measures

3.2.3.1 Information Gain

The ID3 algorithm [25] uses information gain as its attribute selection measure, which is a measure for selecting the splitting criterion that best separates a given data partition. The idea behind the method is to find which attribute would cause the biggest decrease in entropy if being chosen as a split point. The information gain is defined as the entropy of the whole set minus the entropy when a particular attribute is chosen. The entropy of a data set is given by

$$Entropy(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{3.1}$$

where p_i is the probability that an instance in set *D* belongs to class *Ci*. It is calculated by $|C_i|/|D|$. Suppose the attribute A is now considered to be the split point and A has *v* distinct values $\{a_1, a_2, ..., a_v\}$. Attribute A can be used to split *D* into *v* subsets $\{D_1, D_2, ... D_v\}$ where D_i consists of instances in D that have outcome a_j. The new entropy is defined by the following equation.

$$Entropy_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} * Entropy(\mathrm{D}_j) \tag{3.2}$$

The information gain when using attribute A as a split point is as follows.

$$Gain(A) = Entropy(D) - Entropy_A(D) \tag{3.3}$$

Gain(*A*) presents how much would be gained by branching on A. Therefore, the attribute A with the highest Gain(A) should be chosen to use.

3.2.3.2 Gain Ratio

The information gain measure presented in section **3.2.3.1** is bias toward attributes having a large number of values, thus leading to a bias toward tests with many outcomes. C4.5 [26], a successor of ID3, uses an extension to information gain called *Gain ratio*, which attempts to overcome this shortcoming. The method normalizes information gain by using a *split information* factor, defined as follows.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log_2(\frac{|D_j|}{|D|}) \quad (3.4)$$

Gain ratio is then given by the following equation.

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \quad (3.5)$$

The attribute with the highest gain ratio is selected as the splitting point.

3.2.3.3 Gini Index

The CART algorithm [27] uses the *gini index* as its attribute selection measure. The Gini index measures the impurity of set *D*. Therefore, it is also called *Gini impurity*. The Gini index only consider a binary split for each attribute. Gini index point of *D* is defined as follows.

$$Gini(D) = 1 - \sum_{i=1}^{m} {p_i}^2 \quad (3.6)$$

Suppose the attribute A is now considered to be the split point and A has 2 distinct values a_1, a_2. Attribute A can then be used to split D into D_1 *and* D_2 where D_i consists of instances in D that have outcome a_j. The gini index of D given that partitioning is given by the following equation.

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \quad (3.7)$$

The reduction in impurity is defined as:

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \quad (3.8)$$

The attribute with the highest reduction in impurity is selected for the next classification step.

3.2.3.4 ReliefF

Unlike the algorithms presented above, ReliefF [28] is not impurity based. It selects splitting points according to how well their values distinguish between similar instances. A good attribute is the one that can separate similar instances with different classes and leave similar instances with the same classes together.

Let D be the training set with n instances of p attributes. Each attribute is scaled to the interval [0, 1]. Let W be a p-long weight vector of zero. The algorithm will be repeated m times, and at each iteration, it chooses a random instance X. The closest same-class instance is called *near-hit*, and the closest different-class instance is called *near-miss*. The weight vector W is updated as follows.

$$W_i = W_{i-1} - (x_i - nearHit_i)^2 + (x_i - nearMiss_i)^2 \quad (3.9)$$

After m iterations, each element of the weight vector is divided by m. This vector is called *relevance vector*. Attributes are selected if their relevance is greater than a specified threshold.

3.3 Random Forest

3.3.1 Introduction

Random Forests were introduced by Leo Breiman in 2001. they are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x [29].

The idea behind the random forests (for both classification and regression) is as follows [30]:

1. Draw n_{tree} bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an *unpruned* classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among

those variables. (Bagging can be thought of as the special case of random forests obtained when $m_{\text{try}} = p$, the number of predictors.)

3. Predict new data by aggregating the predictions of the n_{tree} trees (i.e., majority votes for classification, average for regression).

An estimate of the error rate can be obtained, based on the training data, by the following:

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls "out-of-bag", or OOB, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions. (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calculate the error rate, and call it the OOB estimate of error rate.

3.3.2 The Random Forest Algorithm

As the name suggests, a Random Forest is a tree-based ensemble with each tree depending on a collection of random variables. More formally, for a p-dimensional random vector X = $(X_1, \ldots, X_P)^T$ representing the real-valued input or predictor variables and a random variable Y representing the real-valued response, we assume an unknown joint distribution E_{XY} (X,Y) [31]. The goal is to find a prediction function f(X) for predicting Y. The prediction function is determined by a loss function L(Y, f(X)) and defined to minimize the expected value of the loss

$$E_{XY}\,(L(Y, f(X))\,) \tag{3.2}$$

where the subscripts denote expectation with respect to the joint distribution of X and Y. Intuitively, L(Y, f(X)) is a measure of how close f(X) is to Y; it penalizes values of f(X) that are a long way from Y. Typical choices of L are squared error loss $L(Y, f(X)) = (Y - f(X))^2$ for regression and zero-one loss for classification:

$$L\big(Y, f(X)\big) = I\big(Y \neq f(X)\big) = \begin{cases} 0 \text{ if } Y = f(X) \\ 1 \text{ otherwise.} \end{cases} \tag{3.3}$$

It turns out that minimizing E_{XY} (L(Y, f(X))) for squared error loss gives the conditional expectation

$$f(x) = E(Y|X = x) \tag{3.4}$$

otherwise known as the regression function. In the classification situation, if the set of possible values of Y is denoted by Y , minimizing E_{XY} (L(Y, f(X))) for zero-one loss gives

$$f(x) = \arg\max_{y \in Y} P(Y = y|X = x) \tag{3.5}$$

otherwise known as the Bayes rule.

Ensembles construct f in terms of a collection of so-called "base learners" h_1 (x),..., h_J(x) and these base learners are combined to give the "ensemble predictor" f(x). In regression, the base learners are averaged

$$f(x) = \frac{1}{J}\sum_{j=1}^{J} h_j(x) \tag{3.6}$$

while in classification, f(x) is the most frequently predicted class ("voting")

$$f(x) = \arg\max_{y \in Y} \sum_{\boldsymbol{j=1}}^{\boldsymbol{J}} I(y = h_j(x)) \tag{3 .7}$$

In Random Forests the jth base learner is a tree denoted $h_j(X, \Theta_j)$, where Θ_j is a collection of random variables and the Θ_j's are independent for j = 1,...,J.

3.3.3 The Out-of-Bag (OOB) Data

In the forest building process, when bootstrap sample set is drawn by sampling with replacement for each tree, about one-third of the cases are left out and not used in the construction of that tree.

This set of cases is called Out-of-bag data. Each tree has its own OOB data set which is used for calculating the error rate for an individual tree [30].

3.3.4 Out-of-Bag (OOB) Error

Out-of-bag (OOB) error, also called out-of-bag estimate, is a method of measuring the prediction error of random forests. To get the OOB error rate of a whole random forest, put each case left out in the construction of the kth tree down the kth tree to get a classification. Take j to be the class that gets most of the votes every time case n is OOB. The proportion of times that j is not equal to the true class of n averaged over all cases is the OOB error estimate [30].

For regression with squared error loss, generalization error is typically estimated using the out-of-bag mean squared error (MSE):

$$\mathrm{MSE}_{\mathrm{oob}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}_{oob}(x_i))^2 \tag{3.8}$$

where $\hat{f}_{oob}(x_i)$ is the out-of-bag prediction for observation i.

For classification with zero one loss, generalization error rate is estimated using the out-of-bag error rate:

$$E_{oob} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq \hat{f}_{oob}(x_i)) \tag{3.9}$$

3.3.5 Variable Importance

Random forests can be used to rank the importance of variables (features) in a regression or classification problem. The following steps were described in [12].

• In every tree grown in the forest, put down the OOB cases and count the number of votes cast for the correct class

• To measure the importance of variable m, randomly permute the values of variable m in the OOB cases and put these cases down the tree.

• Subtract the number of votes for the correct class in the perturbed data from the number of votes for the correct class in the original data. The average of this number

over all trees in the forest is the raw importance score of variable m.

Variable which produce large values for this score are ranked as more important than variables which produce small values.

3.3.6 Proximity Matrix

Let N be the number of cases in the training set. A proximity matrix is an NxN matrix, which gives an intrinsic measure of similarities between cases. At each tree, put all cases (both training and OOB) down the tree. If case i and case j both land in the same terminal node, increase the proximity between i and j by one. At the end of the run, the proximities are divided by the number of trees in the run. The proximity between a case and itself is set equal to one [32].

Each cell in the proximity matrix shows the proportion of trees over which each pair of observations falls in the same terminal node. The higher the proportion is, the more alike those observations are, and the more proximate they have.

Proximity matrix can be used to replace missing values for training and test set. It can also be employed to detect outliers. The following sections will illustrate how missing values are replaced and outliers are detected using the proximity matrix.

3.3.7 Missing Value Replacement

There are two ways which can be used to replace missing values in random forest. The first way is fast, simple and easy to implement. To be specific, if the m'th variable of case n is missing and it is numeric, it is replaced with the median of all values of this variable in the same class, say j, with case n. On the other hand, if the mth variable is categorical, it is replaced with the most frequent non-missing value in class j [32].

A more advanced algorithm capitalizes on the proximity matrix. This algorithm is computationally more expensive but more powerful. It starts by imputing missing values using the first algorithm, then it builds a random forest with the completed data. The proximity matrix from the random forests is used to update the imputations of the missing values. For numerical variable, the imputed value is the weighted average of the nonmissing cases, where the weights are the proximities. For categorical variable, the imputed value is the category with the largest proximity. So, by following this algorithm, cases more similar to the case with the missing data are given greater weight.

3.4 Gradient Boosted Machine

3.4.1 Introduction

Gradient boosting machines are a family of powerful machine-learning techniques that have shown considerable success in a wide range of practical applications. They are highly customizable to the particular needs of the application, like being learned with respect to different loss functions.

3.4.2 Function Estimation

Consider the problem of function estimation in the classical supervised learning setting. The fact that the learning is supervised leaves a strong restriction on the researcher, as the data has to be provided with the sufficient set of proper target labels (which can be very costly to extract, e.g., come from an expensive experiment). We arrive with the dataset $(x, y)_{i=1}^{N}$, where $x = (x_1, \dots, x_d)$ refers to the explanatory input variables and *y*

to the corresponding labels of the response variable. The goal is to reconstruct the unknown functional dependence $x \xrightarrow{f} y$ with our estimate $\hat{f}(x)$, such that some specified loss function $\boldsymbol{\Psi}(y, f)$ is minimized:

$$\hat{f}(x) = y$$
$$\hat{f}(x) = \arg min_{f(x)} \ \boldsymbol{\Psi}(y, f(x)) \tag{3.10}$$

Please note that at this stage, we don't make any assumptions about the form of neither the true functional dependence *f(x)*, nor the form of the function estimate $\hat{f}(x)$. If we rewrite the estimation problem in terms of expectations, the equivalent formulation would be to minimize the expected loss function over the response variable $E_y(\boldsymbol{\Psi}[y, f(x)])$, conditioned on the observed explanatory data *x*:

$$\hat{f}(x) = \arg min_{f(x)} \ E_x[E_y(\boldsymbol{\Psi}[y, f(x)])|x] \tag{3.11}$$

Where $E_y(\boldsymbol{\Psi}[y, f(x)])$ is the expected y loss and $E_x\left[E_y(\boldsymbol{\Psi}[y, f(x)])\middle|x\right]$ is the expectation over the whole dataset.

The response variable *y* can come from different distributions. This naturally leads to specification of different loss functions. In particular, if the response variable is binary, i.e., $y \in \{0, 1\}$, one can

consider the binomial loss function. If the response variable is continuous, i.e., $y \in R$, one can use classical $L2$ squared loss function or the robust regression Huber loss. For other response distribution families like the Poisson-counts, specific loss functions have to be designed.

To make the problem of function estimating tractable, we can restrict the function search space to a parametric family of functions $f(x, \theta)$. This would change the function optimization problem into the parameter estimation one:

$$\hat{f}(x) = f(\mathrm{x},\ \hat{\theta}\ ) \tag{3.12}$$

$$\hat{\theta} = \ \arg min_{\theta}\, E_x[E_y(\boldsymbol{\Psi}[y, f(x, \theta)])|x] \tag{3.13}$$

Typically, the closed-form solutions for the parameter estimates are not available. To perform the estimation, iterative numerical procedures are considered.

3.4.3 Numerical Optimization

Given M iteration steps, the parameter estimates can be written in the incremental form:

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i \tag{3.14}$$

The simplest and the most frequently used parameter estimation procedure is the steepest gradient descent. Given N data points $(x, y)_{i=1}^{N}$ we want to decrease the empirical loss function $J(\theta)$ over this observed data:

$$J(\theta) = \sum_{i=1}^{N} \Psi(y_i, f(x_i, \hat{\theta})) \tag{3.15}$$

The classical steepest descent optimization procedure is based on consecutive improvements along the direction of the gradient of the loss function $\nabla J(\theta)$. As the parameter estimates $\hat{\theta}$ are presented in an incremental way, we would distinguish the estimate notation. By the subscript index of the estimates $\hat{\theta}_t$ we would consider the t-th incremental step of the estimate $\hat{\theta}$. The superscript $\hat{\theta}^t$

corresponds to the collapsed estimate of the whole ensemble, i.e., sum of all the estimate increments from step 1 up till step *t*. The steepest descent optimization procedure is organized as follows:

1. Initialize the parameter estimates $\hat{\theta}_0$ for each iteration *t*, repeat:
2. Obtain a compiled parameter estimate $\hat{\theta}^t$ from all of the previous iterations:

$$\hat{\theta}^t = \sum_{i=0}^{t-1} \hat{\theta}_i \tag{3.16}$$

3. Evaluate the gradient of the loss function $\nabla J(\theta)$, given the obtained parameter estimates of the ensemble:

$$\nabla \mathrm{J}(\theta) = \{\nabla \mathrm{J}(\theta_i)\} = [\frac{\partial J(\theta)}{\partial J(\theta_i)}]_{\theta=\hat{\theta}^t} \tag{3.17}$$

4. Calculate the new incremental parameter estimate $\hat{\theta}_t$:

$$\hat{\theta}_\mathrm{t} \longleftarrow -\nabla \mathrm{J}(\theta) \tag{3.18}$$

5. Add the new estimate $\hat{\theta}_t$ to the ensemble.

3.4.4 Optimization in Function Space

The principle difference between boosting methods and conventional machine-learning techniques is that optimization is held out in the function space. That is, we parameterize the function estimate *f* in the additive functional form:

$$\hat{f}(x) = \hat{f}^M(x) = \sum_{i=0}^{M} \hat{f}_i(x) \tag{3.19}$$

In this representation, *M* is the number of iterations, $\hat{f}_0$ is the initial guess and $\{\hat{f}_i\}_{i=1}^{M}$ are the function increments, also called as "boosts."

To make the functional approach feasible in practice, one can follow a similar strategy of parameterizing the family of functions. Here we introduce the parameterized "base-learner"

functions *h(x, θ)* to distinguish them from the overall ensemble function estimates $\hat{f}(x)$. One can choose different families of baselearners such as decision trees or splines.

We can now formulate the "greedy stagewise" approach of function incrementing with the base-learners. For this purpose, the optimal step-size ρ should be specified at each iteration.

For the function estimate at the *t*-th iteration, the optimization rule is therefore defined as:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x,\theta_t) \tag{3.20}$$

$$(\rho_t,\theta_t) = \arg min_{\rho,\theta} \sum_{i=1}^{N} \Psi(y_i, \hat{f}_{t-1}) + \rho\, h(x_i,\theta) \tag{3.21}$$

3.4.5 Gradient Boost Algorithm

One can arbitrarily specify both the loss function and the baselearner models on demand. In practice, given some specific loss function *Ψ(y, f)* and/or a custom base-learner *h(x, θ)*, the solution to the parameter estimates can be difficult to obtain. To deal with this, it was proposed to choose a new function $h(x,\theta_t)$ to be the most parallel to the negative gradient $\{g_t(x_i)\}_{i=1}^{N}$ along the observed data:

$$g_t(x) = E_y\left[\frac{\partial \Psi(y,f(x))}{\partial f(x)} | x\right]_{f(x)=\hat{f}^{t-1}(x)} \tag{3.22}$$

Instead of looking for the general solution for the boost increment in the function space, one can simply choose the new function increment to be the most correlated with $-g_t(x)$. This permits the replacement of a potentially very hard optimization task with the classic least-squares minimization one:

$$(p_t,\theta_t) \arg min_{\rho,\theta} \sum_{i=1}^{N} [-g_t(x_i) + \rho\, h(x_i,\theta)\,]^2 \tag{3.23}$$

To summarize, we can formulate the complete form of the gradient boosting algorithm, as originally proposed by Friedman (2001). The exact form of the derived algorithm with all the corresponding formulas will heavily depend on the design choices of Ψ(y, f) and h(x, θ) [33].

Chapter 4

Experiment and Results

4.1 Introduction

This chapter presents the methodology that has been applied to solve the problem of predicting failures in turbofan engines. We will describe the dataset used, the method used for prediction and the preparations that had to be done. R programming environment was used to construct the GBM and RF models and test the results. The data sets were read from files and the algorithms were applied on it using R language. The advantage of R environment is that it is easy to port data to R, process and visualize the results. The results were stored in variables. These variables were plotted using the visualization methods available in R.

Figure 4.1 summarizes the workflow of our work:

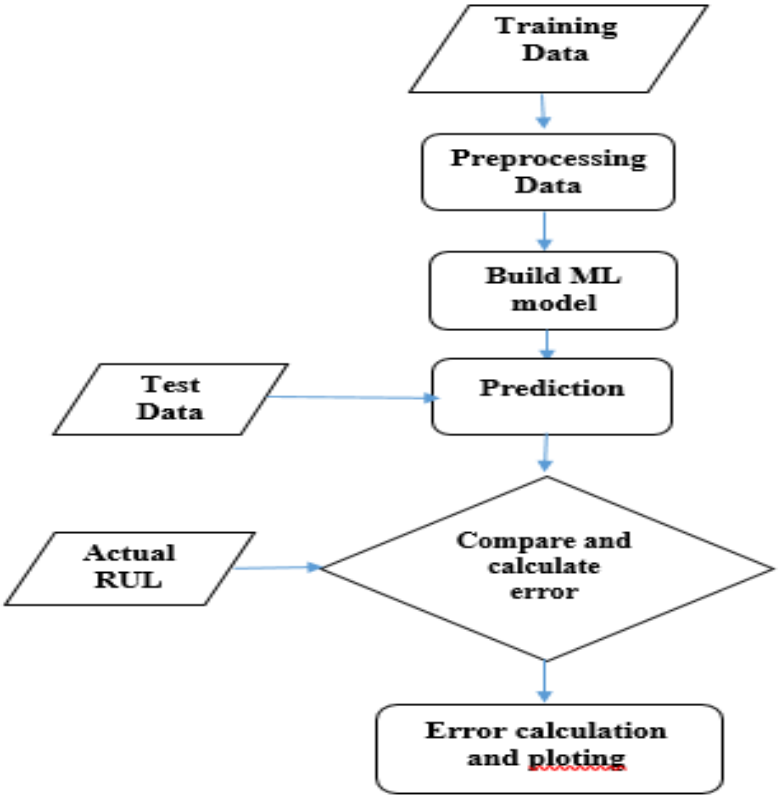

Figure 4-1 Flowchart for the Machine Learning model

4.2 Data Description

The dataset is taken from NASA data repository. The dataset selected includes the run-to-failure sensor measurements from degrading turbofan engines. Although the turbo fan engines are of same type, each engine starts with different degree of initial conditions and there are variations in the manufacturing process of the engines, which are not known to the user.

For the turbo fan engines under consideration, the performance of each engine can be changed by adjusting three operational settings. Each engine has 21 sensors collecting different measurements related to the engine state at runtime. The main characteristic of the dataset is that it is a time series data, the schema of which is included in Table 4.1.

Table 4-1 Dataset schema

Index	Data Fields	Types	Descriptions
1	Id	Integer	Aircraft Engine Identifier
2	Cycle	Integer	Time, in cycles
3	Setting 1	Double	Operational Setting 1
4	Setting 2	Double	Operational Setting 2
5	Setting 3	Double	Operational Setting 3
6	S1	Double	Sensor Measurement 1
7	S2	Double	Sensor Measurement 2
8	..	..	..
9	S21	Double	Sensor Measurement 21

At the beginning of the time series, the engine's operation is normal. After many cycles, a fault is developed in the engine and gradually the engine fails.

Three data sets were provided as text files for training, testing and measurement of accuracy as part of our approach. The dataset has been classified as:

- Training data: It is the aircraft engine’s run-to-failure data.
- Testing data: It is the aircraft engine’s operating data without failure events recorded.
- Ground truth data: It contains the information of true remaining cycles for each engine in the testing data. In the training set, the amount of fault grows in magnitude until the system fails. In the test set, the time series ends some time before the failure of the system.

4.3 Data Preparation

Data preparation is the first and crucial step in developing a predictive model. Data preparation is an indispensable step in order to convert various data forms and types into proper format that is meaningful to machine learning predictive model. Large amounts of data are generated using C-MPASS tool. The generated data comprise all the variables including the predictor variables that can be used for establishment of prediction models. Data available are "horizontal": too many different variables available (to be reduced) and few observations available in the same operating conditions. With variable selection, chi-square, correlation analysis, ICA and PCA we want to reduce the number of regressors while with data clustering we aim to increment observations usable for modeling.

Data Preparation includes:

- RUL target function
- Cleansing data
- Feature scaling

4.3.1 RUL Target Function

According to the original PHM '08 Prognostic Data Challenge, the objective of the competition was to predict the number of operational cycles *after* the last cycle for the partial time series, also known as the RUL. Based on this and the fact that our cycle increases linearly with data point order for each unit, we should expect our RUL to decrease linearly in relation to the cycle for each data point in each unit.

And here is the RUL where *x* is the cycle number of a given data point and *c* is the maximum cycle number of a given unit:

$$RUL(x) = \begin{cases} c - x & if\ x \leq c \\ 0 & O.W. \end{cases} \quad (4.1)$$

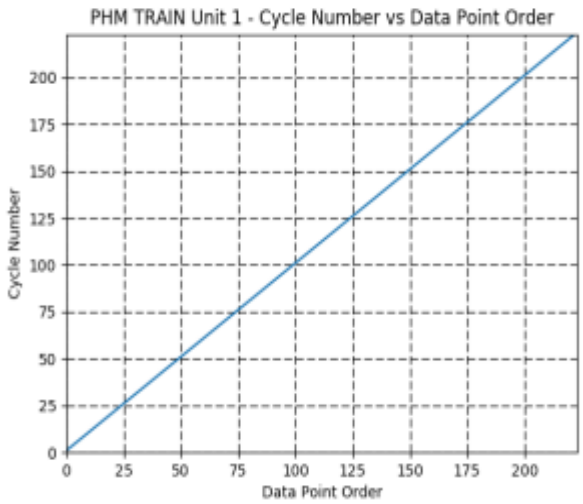

Figure 4-3 Cycle Number of Data points in PHM TRAIN

Figure 4-2 RUL target of Data points in PHM TRAIN

If we understand that each unit has a last cycle, which is also the maximum cycle number, and that our initial cycle number, which is also our minimum cycle number, is 1, then we can simply take the maximum cycle number and subtract it by the cycle number of the given dataset. This can be further simplified by simply reversing the cycle numbers of the given unit and subtract by one, since the cycle number for each unit in the PHM training set increases by 1 for each data point. This true RUL function is shown in Figure 1.1 and generalized in equation 4.1.

4.3.2 Cleansing Data

Data cleansing phase is an important phase in predictive modeling. The dataset needs to be cleansed for anomalies and also the data should be normalized for all the range of values of raw data varies widely. A data cleansing procedure discards 452 instances with null and missing values.

4.3.3 Feature Scaling

"Feature Scaling" is the technique that followed to normalize the data set. The goal of the data cleansing is to obtain a complete cleansed data set that can be modeled with outliers removed and solutions for handling of missing data applied. To normalize the input data set, the continuous variables were transformed on a linear scale to a value with a range of 0 to 1 or -1 to 1. Ordinal data were spaced equally over the same range. Missing values were substituted with the class mean. Data with different scales can induce instability in neural networks (Weigend and

Gershenfeld, 1994). In order to normalize the raw data of input and output the following normalization equation is used:

$$x_{norm} = 2 \times \frac{(x - x_{min})}{(x_{max} - x_{min})} - 1 \quad (4.2)$$

where *x* is the data to be normalized, i.e., and *x* min and *x* max are minimum and maximum values of the raw data. In such a way, all the inputs and the desired outputs are normalized within the range of ±1.

4.4 Data Pre-processing

In data preprocessing several actions against the data we'll be performed in order to make it ready for model building and to address some problems like:

- Over-fitting
- More Computational Power
- Less Prediction Accuracy

4.4.1 Near Zero Variance Analysis

One interesting aspect of this dataset is that it contains some variables that have extremely low variances. This means that there is very little information in these variables because they mostly consist of a single value (e.g. one). Near zero variance algorithm takes in data x, then looks at the ratio of the most common value to the second most common value, freqCut. And the percentage of distinct values out of the number of total samples, uniqueCut.

Table 4-2 Near zero variance values

	Frequency Ratio	Percent Unique	Zero Variance	Near zero variance
Setting 1	1.0376	0.7658	FALSE	FALSE
Setting 2	1.0033	0.0630	FALSE	FALSE
Setting 3	0.0000	0.0048	TRUE	TRUE
Sensor 1	0.0000	0.0048	TRUE	TRUE
Sensor 2	1.0052	1.5025	FALSE	FALSE
Sensor 3	1.0384	14.5993	FALSE	FALSE
Sensor 4	1.1111	19.6354	FALSE	FALSE
Sensor 5	0.0000	0.0048	TRUE	TRUE
Sensor 6	49.8152	0.0096	FALSE	TRUE
Sensor 7	1.0086	2.4865	FALSE	FALSE
Sensor 8	1.0189	0.2568	FALSE	FLASE
Sensor 9	1.0666	31.0358	FALSE	FALSE
Sensor 10	0.0000	0.0048	TRUE	TRUE
Sensor 11	1.0088	0.7706	FALSE	FALSE
Sensor 12	1.0514	2.0697	FALSE	FALSE
Sensor 13	1.0174	0.2714	FALSE	FALSE
Sensor 14	1.0000	29.4604	FALSE	FALSE
Sensor 15	1.0270	9.2966	FALSE	FALSE
Sensor 16	0.0000	0.0048	TRUE	TRUE
Sensor 17	1.1893	0.0630	FALSE	FALSE
Sensor 18	0.0000	0.0048	TRUE	TRUE
Sensor 19	0.0000	0.0048	TREU	TRUE
Sensor 20	1.0189	0.5816	FALSE	FALSE
Sensor 21	1.3529	22.9993	FALSE	FALSE

4.4.2 Correlation Analysis (CA)

Next step to reduce the set of parameters was done with correlation analysis. Correlation Analysis is useful for determining the direction and strength of the association (linear relationship) between two variables. This technique is performed to omit parameters bringing little information to the dataset. The strength of the linear association between two variables is quantified by the correlation coefficient. For every couple of measurement parameters (x_1, x_2) we compute the correlation coefficient. The most familiar measure of dependence between two quantities is the Pearson product-moment correlation coefficient or "Pearson's correlation coefficient", commonly called simply the "correlation coefficient". The population correlation coefficient $\rho_{x,y}$ between two random variables *X* and *Y* with expected values μx and μy and standard deviations σ_x and σ_y is defined as

$$\begin{aligned} \rho_{x,y} &= cor(X,Y) \\ &= (cov(X,Y))/(\sigma_x\sigma_y) \\ &= \frac{E[(X-\mu x)(Y-\mu y)]}{\sigma_x\sigma_y} \end{aligned} \tag{4.3}$$

where μx , μy and σ_x , σ_y are respectively the mean and the standard deviations of *x*, *y*. *E* is the expected value operator, cov means covariance. Variables/Predictors showing a pairwise correlation of +/-0.8 or higher are removed, Coefficient $\rho_{x,y}$ between two random variables *X* and *Y*. Figures 4.4, 4.5 show the correlation matrix. The reduced parameter set contains *p*= 10 predictors.

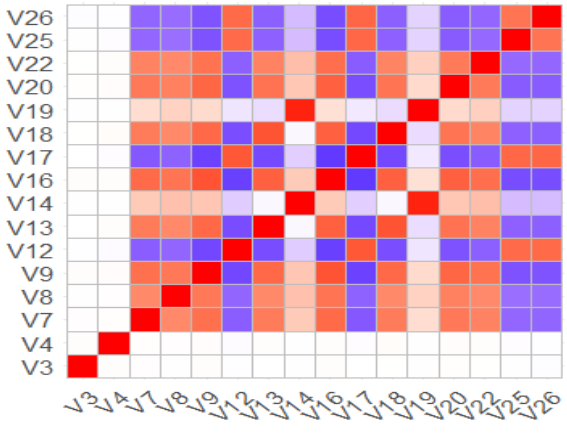

Figure 4-5 Heat map of variables correlation before CA

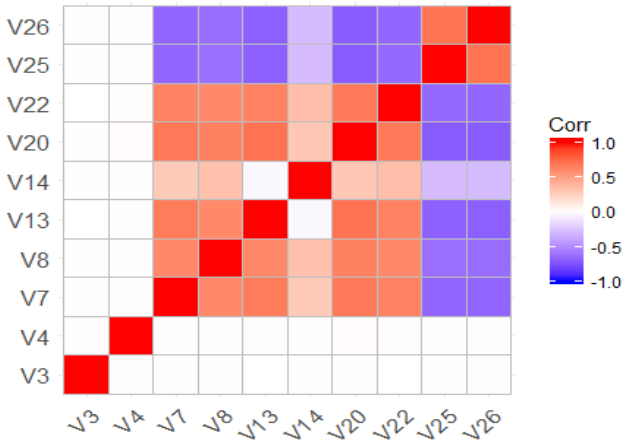

Figure 4-4 Heat map of variables correlation after CA

The algorithm is as follows:

1. Calculate the correlation matrix of the predictors
2. Determine the two predictors associated with the largest absolute pair-wise correlation (call them predictors A and B)
3. Determine the average correlation between A and other variables. Do the same for predictor B.
4. If A has a larger average correlation, remove it; otherwise remove predictor B.

Repeat Step 1–Step 4 until no absolute correlations are above the threshold. The idea is to first remove the predictors that have most correlated relationships.

After performing the correlation analysis algorithm 6 variables has been removed and it remains 10 variables.

4.4.3 Chi Square Test of Importance

To reduce the number of features even more we'll use chi-square independence test when having independent variables, and is requested to compare an observed frequency-distribution to a theoretical expected frequency-distribution to test the null hypothesis of independence. Using the formula:

$$x^2 = \sum \frac{(\mathrm{O} - \mathrm{E})^2}{\mathrm{E}} \tag{4.4}$$

we obtained the following results in table 4.3 for each variable and comparing to certain threshold we've been able to omit *2* variables that are independent of the outcome RUL.

Table 4-3 Chi-square values

Attribute	Importance
Setting 1	0.0000
Setting 2	0.0000
Sensor 2	0.3789
Sensor 3	0.3603
Sensor 8	0.3447
Sensor 9	0.3218
Sensor 15	0.4069
Sensor 17	0.3773
Sensor 20	0.3911
Sensor 21	0.3993

4.4.4 Feature aggregation

The added features are lagged covariates of the sensor readings to add previous information of variables and moving averages and standard deviations as simple and common type of smoothing used in time series analysis and time series forecasting and to the hope of smoothing and to remove noise and better expose the signals of the underlying causal processes. Therefore, the aggregated features are:

- for any sensor reading, X(t), we included $X(t-k)$, for all nonzero k, where $k \geq$ min lag and $k \leq$ max lag.
- for any sensor reading, X(t), we included Xa(t) =avg(X(t-1) + …+X(t-w)) and Xs(t) =sd (X(t-1) + …+X(t-w)) where w =5 is the chosen window for averaging.

4.4.5 PCA Data Reduction

At this point we've been left with *24* variables, *8* original scaled features and *16* aggregated features. And those variables represent the full variance of the data. So as a dimensionality reduction step and to bring a new set of variables (PCs) to the model, removing the redundancy in the information that can be gathered by the dataset, we applied PCA algorithm with 0.85 variance retained and the number of components was chosen with a trivial algorithm that computes the

biggest eigenvalues and use the associated eigenvectors. Figure 4.6 shows the percent variance represented per each principal component. With only 10 components we were able to explain +85% of data variance.

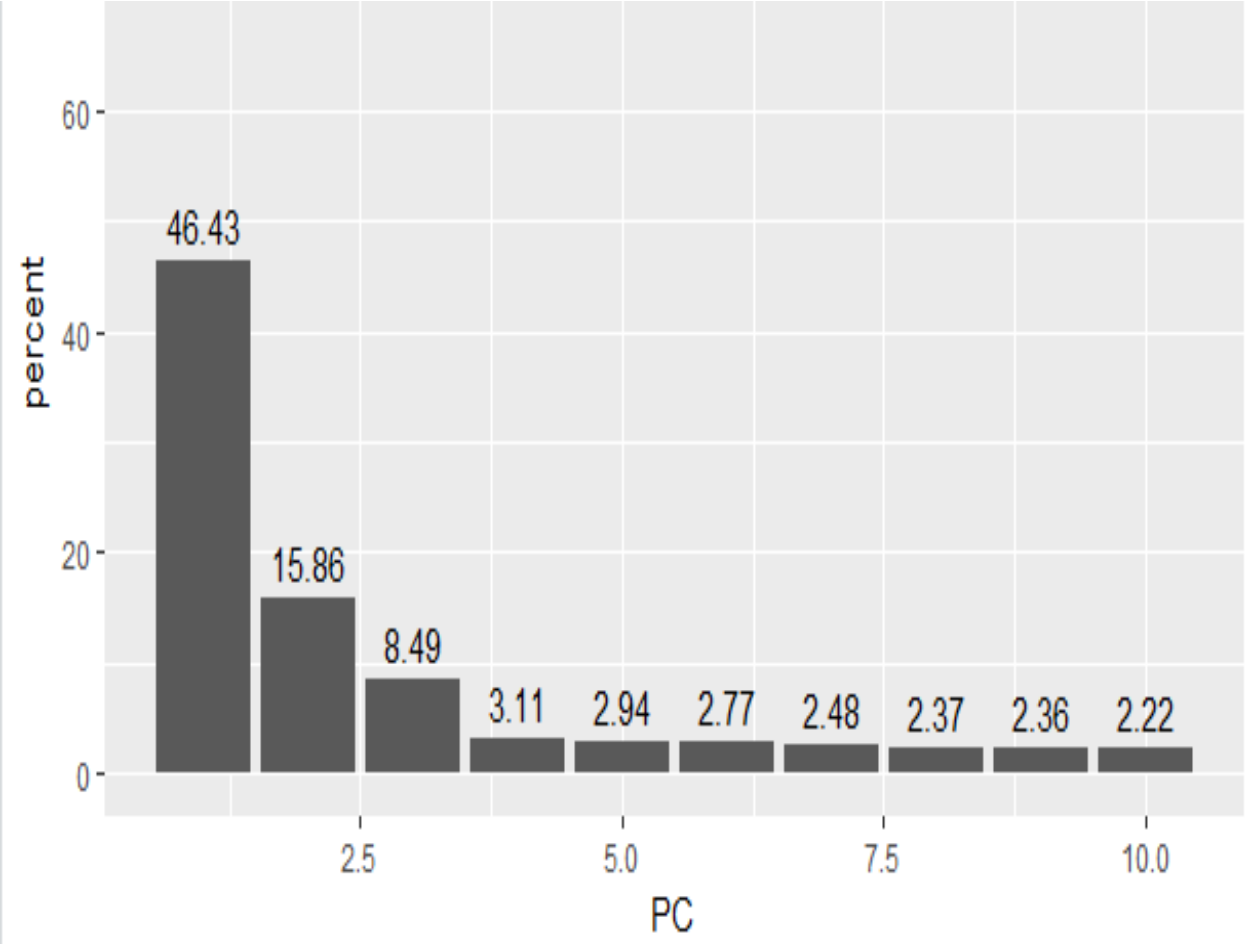

Figure 4-6 Variance represented by each PC

The first component explains 46.43% of the variance, the second component explains 15.86%, the third explains 8.49% and the rest are below 4%.

Figure 4.7 shows the relationship between components and the possibility of failure in next n=20 steps.

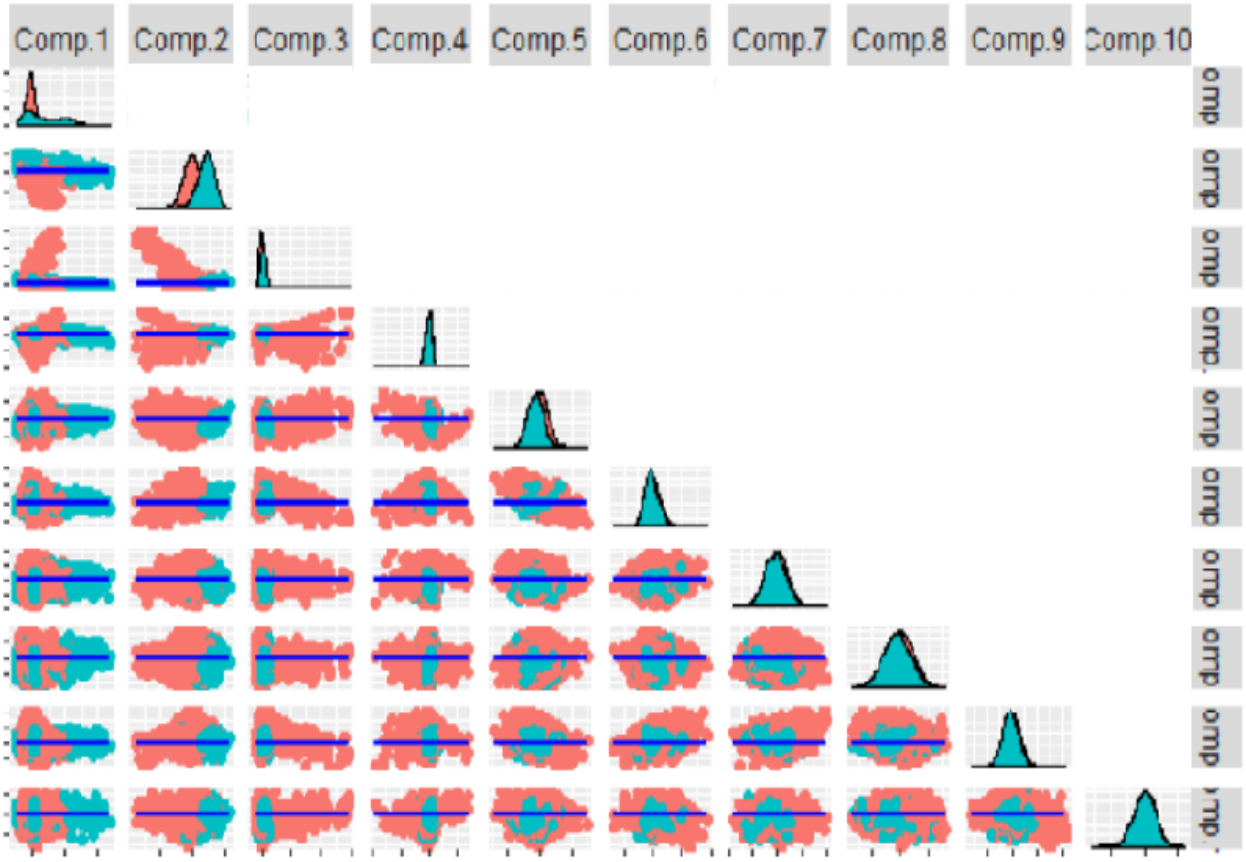

Figure 4-7 Scatter plot of principal component

As we can see, there is a better separation between data points that represent the happening of failure and other points in the first column than in the second column and the rest columns, and also the same thing for the densities of the components. This results to the possibility that PCA data has a relationship with RUL target function and truly represent the information in the original data.

4.4.6 ICA Data Reduction

ICA is the blind source separation problem where the goal is to recover mutually independent but unknown source signals from their linear mixtures without knowing the mixing coefficients. Two differences between PCA and ICA are that the components here are statistically independent and not uncorrelated. And second, the un-mixing matrix is not orthogonal like PCA. The algorithm works on the principle of minimizing mutual information between the variables, minimizing mutual information is the correct criteria for judging independence. Also minimizing mutual

information is same as maximizing entropy. There are several algorithms for doing ICA and the one used is FastICA.

Figure 4.8 shows the relationship between the independent components.

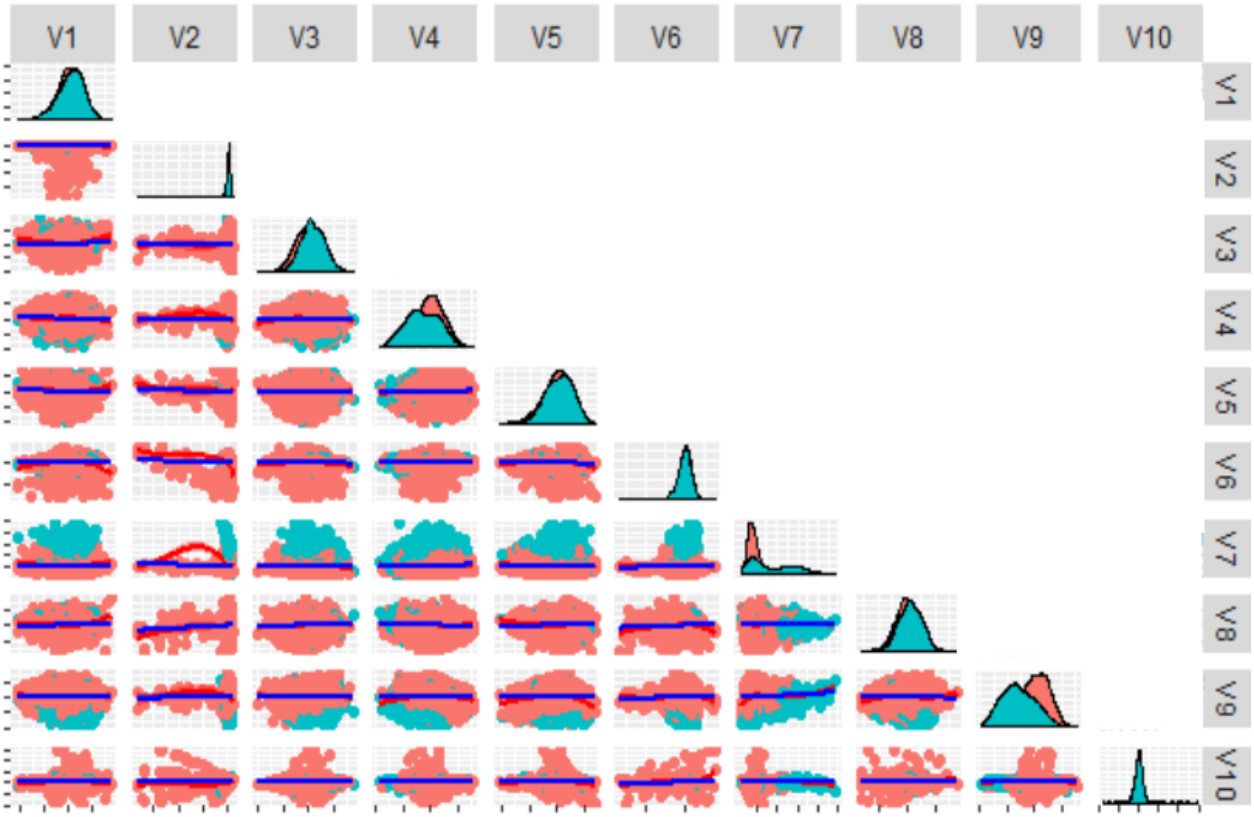

Figure 4-8 Scatter plot of independent components

4.5 Model Building Phase

The data set is divided into two datasets, train data and test data. For datasets from PCA and ICA dimensionality reduction steps, evaluate model accuracy with 10-fold cross validation technique. The validation of our work was evaluated with the test set. Build a prediction model with the following algorithms:

1-Random Forest: Random forest is an ensemble learning method, it operates by constructing a set of decision trees at training time and outputting the mean prediction of the individual trees.

2-Gradient Boosting Method(GBM): Gradient Boosting Method is a forward learning ensemble method. It is based on the idea that good predictive results can be obtained through increasingly refined approximations. GBM sequentially builds regression trees on all the features of the dataset in a fully distributed way - each tree is built in parallel.

Caret R package was used to make finding optimal parameters for an algorithm very easy. It provides a grid search method for searching parameters, combined with various methods for estimating the performance of a given model. The table below shows the best parameters for our models.

Table 4-4 Optimal hyperparameters for the models

	n_estemators	**min_samples_leaf**	**n.trees**	**shrinkage**	**interaction.depth**
RF	4	50	x	x	x
GBM	x	x	150	0.1	3

4.6 Results

Combining the prediction results from gradient boosted machine and random forest over the two reduced datasets PCA_data and ICA_data gives rise to the table below:

Table 4-5 Results obtainded by each model.

	RMSE	**Rsquared**	**MAE**
RF_PCA	47.11034	0.5544315	35.87538
RF_ICA	48.72274	0.5254168	37.32867
GBM_PCA	43.24703	0.6259616	31.15214
GBM_ICA	45.13191	0.5940377	32.95328

The following set of graphs includes the cycles against RUL plots for the 2 machine learning algorithms under consideration, covering both predicted and actual RUL values. The variation of predicted RULs and actual RULs of PCA and ICA based reduced dataset predicted using GBM and RF models are depicted in these graphs (Figures 4.9, 4.10, 4.11, 4,12). If actual and predicted

lines coincide, it indicates maximum accuracy. For the most accurate algorithm, maximum points will be overlapped on the regression lines. It can be noticed in the graphs that the actual RUL exceeds the predicted RUL in maximum and minimum points, and we can take this as errors in the regressions. Also it can be noticed ripples in the predicted RUL and that could be taken-off if some smoothing filter applied at the end of model prediction step.

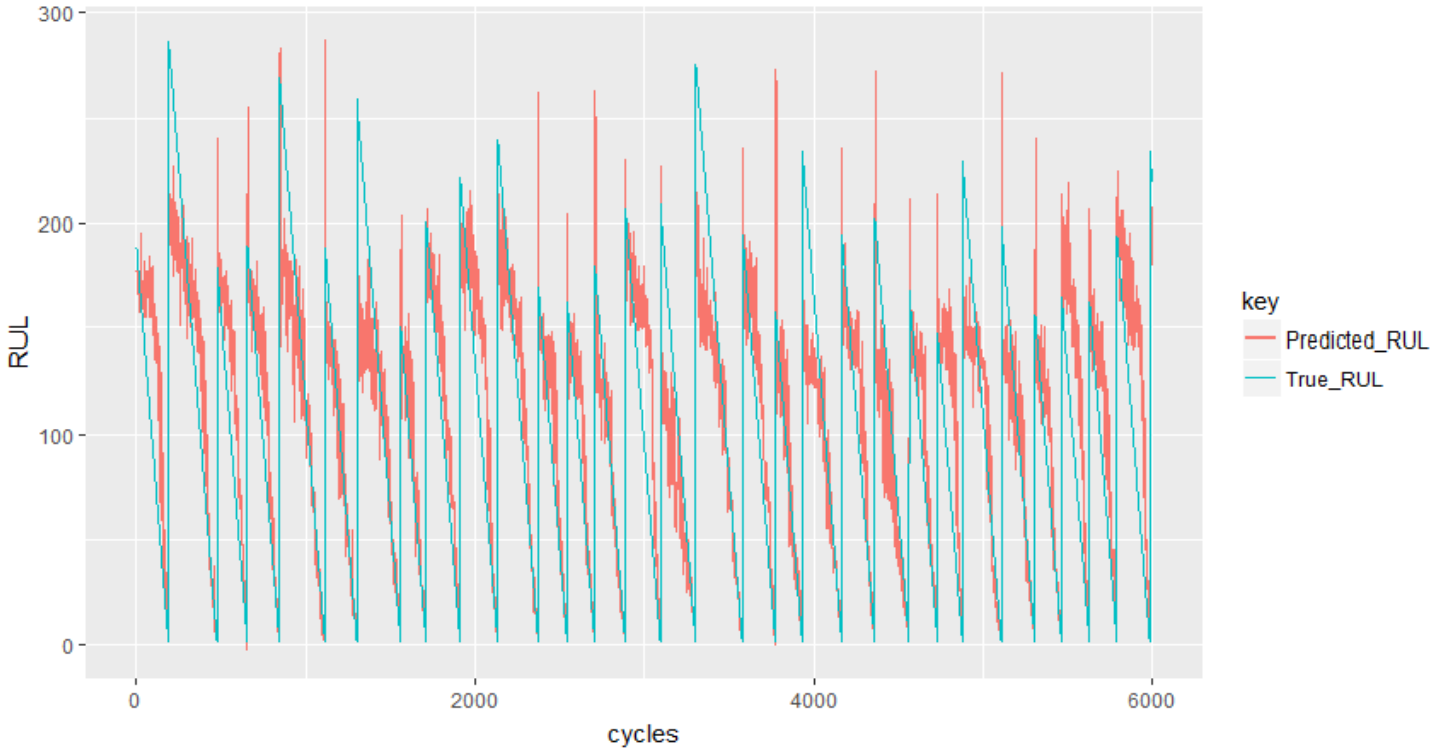

Figure 4-9 Actual RUL vs Predicted RUL for GBM model over PCA data

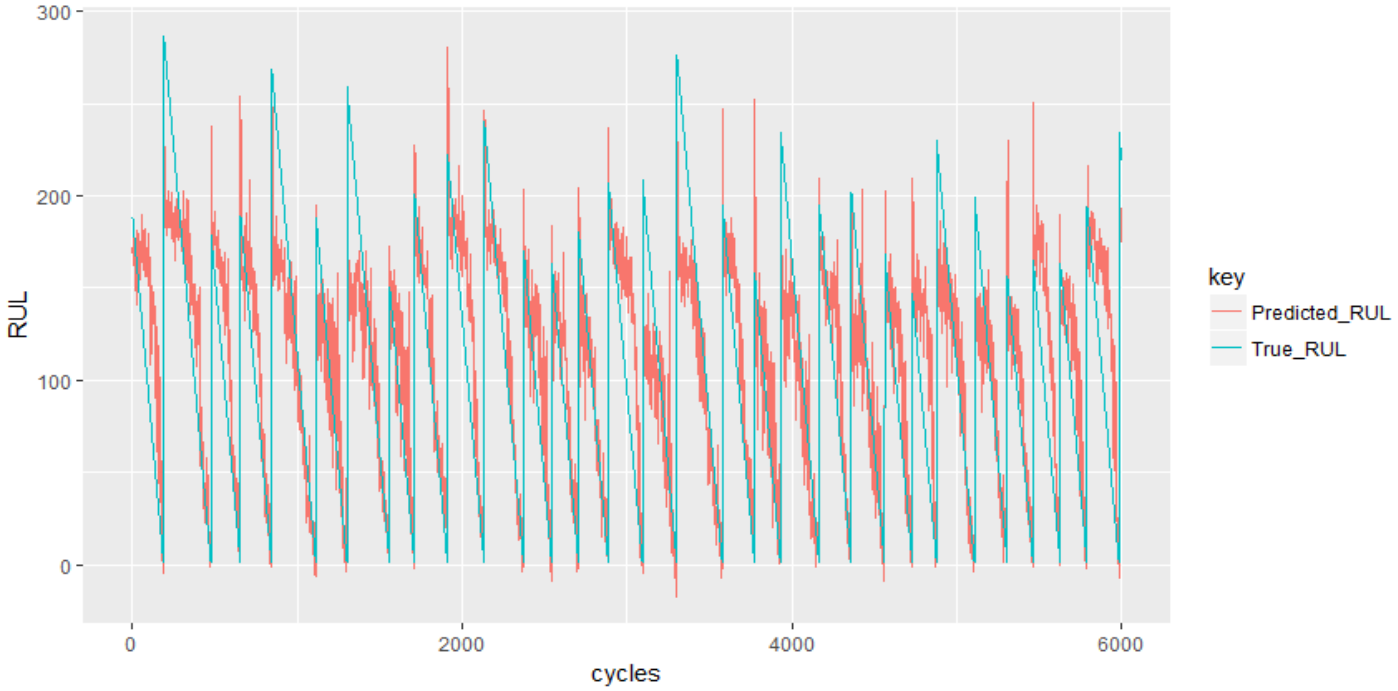

Figure 4.10 Actual RUL vs Predicted RUL for GBM model over ICA data

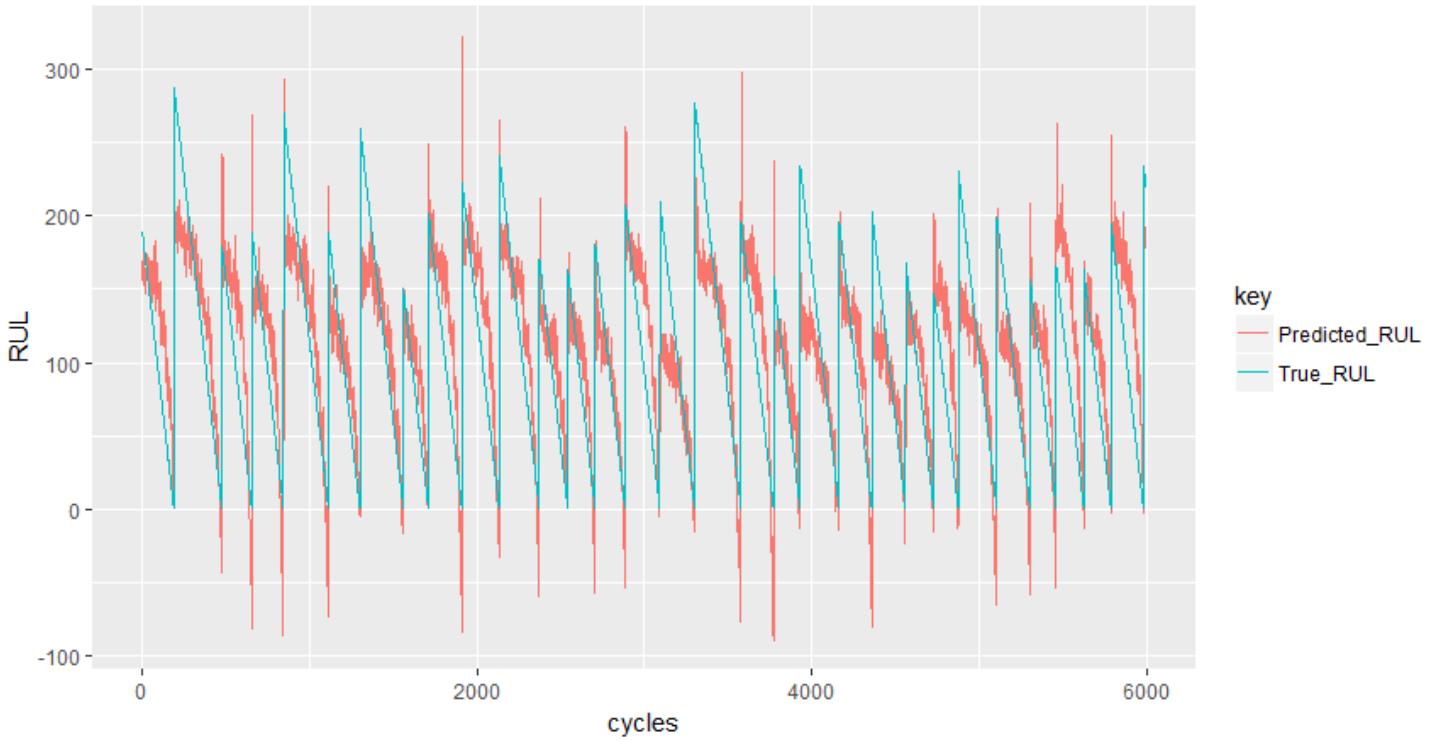

Figure 4.11 Actual RUL vs Predicted RUL for RF model over PCA data

Figure 4.12 Actual RUL vs Predicted RUL for RF model over ICA data

4.7 Discussion and Conclusion

The two different algorithms were evaluated under similar conditions on the same two datasets for consistent comparison of results. While predicting RUL, the main objective is to reduce the error between the actual RUL and the predicted RUL. For each dataset, the test results were compared with the actual values of the RUL available in the dataset. The root mean squares of the errors were sited and it is observed that the best results were obtained by gradient boosted machine on the PCA dataset. Gradient boosted machine captures the variance of several input variables at the same time and enables high number of observations to take part in the prediction. It was observed that the performance of all two algorithms were consistent in the two different datasets, generating proportional accuracy for the different algorithms tested.

With more resources of machine maintenance data becoming available in the future, we can try to build more individual learning models and add them to the GBM and RF models for ensemble modeling. Cross-validation can also be done as an improvement instead of simply dividing the data into training and testing set.

General Conclusion

The main objective of predictive maintenance is to predict the equipment failure. The Remaining Useful Lifetime prediction has been carried out so as to plan the maintenance requirements of the turbo fan engine. By doing predictive maintenance, failures can be predicted and maintenance can be scheduled in advance. This reduces the cost and effort for doing maintenance. It increases safety of employees and reduces lost production time.

This report dealt with data driven approach using machine learning instead of model based approach, where generally more complex to derive the model of failure and predict it. In addition, using the aggregated features was more helpful where we generated new features in building the model so we gained more dynamic information which helped us to find more accurate results.

In our work, we pre-processed the data using chi-square and correlation analysis to find and use only relevant attributes in our model. After that we generated new features using moving average and moving standard deviation to be included in the model building phase, and the reason was to bring previous information to current data points. PCA and ICA were used to reduce dimensionality of our data. Then we have studied the performance of two machine learning algorithms, random forest and gradient boosted machine. Another approach we take in consideration was the comparison between the dimensionality reduction techniques, where we found that in PCA the data was better explained. In other hands, the comparison between the two algorithms stated that the gradient boosted machine GBM was slightly better than random forest in predicting the RUL over the tested dataset.

As future work we suggest to extend this approach on other industrial processes where we can predict the RUL that can save money and cost. Moreover, the algorithms can be tested for more real time data and always be one step ahead in predicting the maintenance requirements.

Appendix A

Random Forest and GBM Algorithms

Definition

A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input **x**.

Random Forest Algorithm for Regression or Classification

1. **For** b = 1 **to** B:

   (a) Draw a bootstrap sample Z^* of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

   - Select m variables at random from the p variables.
   - Pick the best variable/split-point among the m.
   - Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

   To make a prediction at a new point x:

   Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$

   Classification: Let $\hat{C}_b$(x) be the class prediction of the bth random-forest tree. Then $\hat{C}_{rf}^B$(x) = majority vote $\{\hat{C}_b(\text{x})\}_1^B$.

Gradient Boosted Machine algorithm

Inputs:

• input data $(x, y)_{i=1}^N$

• number of iterations *M*

• choice of the loss-function $\Psi(y, f)$

• choice of the base-learner model *h(x, θ)*

Appendix A

Algorithm:

1: initialize $\hat{f}_0$ with a constant

2: **for** t = 1 to M **do**

3: compute the negative gradient $gt(x)$

4: fit a new base-learner function $h(x, \theta t)$

5: find the best gradient descent step-size ρt:

$$\rho_t = argmin_\rho \sum_{i=1}^{N} \Psi[y_i, \hat{f}_{t-1}(x_i) + \rho\, h(x_i, \theta_t)]$$

6: update the function estimate:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x, \theta_t)$$

7: **end for**

Appendix B

FastICA Algorithm

Input: C Number of desired components

Input: $X \in \mathbb{R}^{N \times M}$ Prewhitened matrix, where each column represents an

N-dimensional sample, where C <= N

Output: $W \in \mathbb{R}^{N \times C}$ Un-mixing matrix where each column projects X onto independent component.

Output: $S \in \mathbb{R}^{C \times M}$ Independent components matrix, with M columns representing a sample with C dimensions.

For p in 1 to c:

$w_p \longleftarrow$ Random vector of length N

While w_p changes

$$w_p \longleftarrow \frac{1}{M} X g\left(w_p{}^T X\right)^T - \frac{1}{M} g'(w_p{}^T X) 1 w_p$$

$$w_p \longleftarrow w_p - \left(\sum_{j=1}^{p-1} w_p{}^T w_j w_j{}^T\right)^T$$

$$w_p \longleftarrow \frac{w_p}{\|w_p\|}$$

Output: $W = [w_1, \ldots, w_C]$

Output: $S = W^T X$

List of References

[1] Vimala Mathew, Tom Toby, Vikram Singh, B Maheswara Rao, M Goutham Kumar "Prediction of Remaining Useful Lifetime (RUL) of Turbofan Engine using Machine Learning" Information Technology Group, National Institute of Electronics and Information Technology, NIT Campus PO, Calicut, Kerala, India, 2017.

[2] Khanna Le Son, Mitra Fouladirad, Anne Barros, Eric Levrat, Benoıt Iung "Remaining useful life estimation based on stochastic deterioration models: A comparative study" Reliability Engineering and System Safety 112 (2013) 165–175.

[3] John Scott Bucknam "Data analysis and processing techniques for remaining useful life estimations", *Rowan University*, 6-5-2017.

[4] Halligan, Gary R., "Fault detection and prediction with application to rotating machinery" (2009). *Masters Theses*. 4722. http://scholarsmine.mst.edu/masters_theses/4722.

[5] Amit Kumar Jain, Pradeep Kundu, Bhupesh Kumar Lad "Prediction of Remaining Useful Life of an Aircraft Engine under Unknown Initial Wear" Discipline of Mechanical Engineering, Indian Institute of Technology Indore, M.P. India, 453441, 2014.

[6] Soumik Sarkar, Xin Jin, Asok Ray "Data-Driven Fault Detection in Aircraft Engines With Noisy Sensor Measurements" Article in Journal of Engineering for Gas Turbines and Power · August 2011. DOI: 10.1115/1.4002877.

[7] Narendhar Gugulothu, Vishnu TV, Pankaj Malhotra, Lovekesh Vig, Puneet Agarwal, Gautam Shroff "Predicting Remaining Useful Life using Time Series Embeddings based on Recurrent Neural Networks" TCS Research, New Delhi, India, arXiv:1709.01073v2 [cs.LG] 6 Oct 2017.

[8] Xiaopeng Xi, Maoyin Chen, and Donghua Zhou "Remaining Useful Life Prediction for Degradation Processes with Memory Effects", 2016.

[9] "Turbofan Engine Malfunction Recognition and Response" FAA Engine & Propeller Directorate ANE-110.12 New England Executive Park Burlington, MA 01803, 07/17/2009.

[10] AIA/AECMA "Project Report on Propulsion System Malfunction Plus Inappropriate Crew Response". 11/1/98.

[11] Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space" (http://stat.smmu.edu.cn/history/pearson1901.pdf)(PDF). *Philosophical Magazine*. 2 (11): 559–572.doi:10.1080/14786440109462720(https://doi.org/10.1080%2F14786440109462720).

[12] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24, 417–441, and 498–520. Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28, 321–377.

[13] Jolliffe I.T. *Principal Component Analysis*, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4.

[14] Abdi. H., & Williams, L.J. (2010)."Principal component analysis" (http://arxiv.org/pdf/1108.4372.pdf)(PDF).*Wiley Interdisciplinary Reviews: Computational Statistics.* 2 (4): 433– 459.doi:10.1002/wics.101(https://doi.org/10.1002%2 Fwics.101).

[15] Shaw P.J.A. (2003) *Multivariate statistics for the Environmental Sciences,* Hodder-Arnold.ISBN 0-340-80763-6.

[16] Barnett, T. P. & R. Preisendorfer. (1987). "Origins and levels of monthly and seasonal forecast skill for United States surface air temperatures determined by canonical correlation analysis". *Monthly Weather Review 115*.

[17] Hsu, Daniel, Sham M. Kakade, and T ong Zhang (2008). "A spectral algorithm for learning hidden markov models". arXiv:0811.4413 (https://arxiv.org/abs/0811.4413).

[18] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. Cambridge, UK, 2000.

[19] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287–314, April 1994.

[20] L. Tong, Liu, V. C. Soon, and Y. F. Huang, "Indeterminacy and identifiability of blind identification," *Circuits and Systems, IEEE Transactions on*, vol. 38, no. 5, pp. 499–509, 1991.

[21] C. Jutten and J. Karhunen, "Advances in blind source separation (bss) and independent component analysis (ica) for nonlinear mixtures." *Int J Neural Syst*, vol. 14, no. 5, pp. 267–292, October 2004.

[22] A. Hyvrinen, "New approximations of differential entropy for independent component analysis and projection pursuit," in *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*. MIT Press, 1998, pp. 273–279.

[23] A. Hyvarinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. Wiley-Interscience, May 2001.

[24] Nikunj C.Oza and Stuart Russell "Online Bagging and Boosting" Computer Science Division. University of California Berkeley, CA 94720-1776.

[25] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[26] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[27] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. new edition [**?**].

[28] Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning*, ML92, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[29] Leo Breiman "RANDOM FORESTS" Statistics Department University of California Berkeley, CA 94720 January 2001.

[30] Andy Liaw and Matthew Wiener "Classification and Regression by Random Forest" Article · November 2001 https://www.researchgate.net/publication/228451484.

[31] Adele Cutler, D. Richard Cutler and John R. Stevens "Random Forests" Chapter in Machine Learning · January 2011 DOI: 10.1007/978-1-4419-9326-7_5 · Source: DBLP.

[32] Que Tran "Improving Random Forest Algorithm through Automatic Programming" May 15, 2015 Halden, Norway.

[33] Friedman, J. (2001). Greedy boosting approximation: a gradient boosting machine. *Ann. Stat.* 29, 1189–1232. doi: 10.1214/aos/1013203451.