INDEX

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards our project guide Prof. Mankar Sir for their constant encouragement and valuable guidance during the completion of this min project work. We would also like to thank our project teacher Prof. N.W. Dangare for continuous valuable guidance/Support, suggestions and this precious time in every possible way in spite of his busy schedule throughout our mini project activity.

We take this opportunity to express our sincere thanks to all, our staff members of Computer department for their support whenever required. Finally we express our sincere thanks to all those who helped a directly or indirectly in many ways in completing this project.

ABOUT

NAME : OPTIMAL STOPPING ALGORITHM- WHEN TO QUIT AND WHEN TO SETTLE

DOMAIN : PYTHON WITH DATA SCIENCE

LIBRARIES : MATPLOTLIB, MATH, CSV, SKLEARN, AND OTHER BUILT IN DIRECTORIES.

SUBJECT : SKILLS DEVELOPMENT LAB

TEACHER : PROF. N.W. DANGARE

PROJECT GUIDE : PROF. V. MANKAR.

PROJECT DEVELOPERS : **HUSAIN SHAIKH**
PREMRAJ ZAMBRE
ROHAN RASAL
MOHAN RAUT

LOC : 250 APPROX. including comments.

INPUT : AT RUNTIME BY ADMIN/ INTERVIEWER
TRAINING OF CLASSIFIER

OUTPUT : I. STATISTICS AFTER COMPLETION,
II.SUGGESTION OF ML USING DECISION TREE CLASSIFIER
III. STORAGE OF RELEVANT DATA IN A CSV FILE,
IV. VISUAL REPRESENTATION USING MATPLOTLIB.

Chapter 1

NEED

In Modern Interview Process, It Is Very Important To Record Data Accurately And Efficiently. If You Mess Up Even A Dot, Someone Else Might Pay A Very High Price. On Top Of That It Takes A Lot Of Time To Know Someone And Trust Them In Your Company. But The Most Important And Complex Problem Is Of The Balance Of Time And Quality.

This Mini Project Records Data Of The Candidates In A Systematic Manner And Makes It Easy For The User (Interviewer) To See Visual Output Along With Other Statistics. We Are Taking It One Step Ahead By Implementing Optimal Stopping Algorithm Allowing For The Balance Of Time And Quality Mentioned Above.

On An Average, It Takes About An Hour For A Single Interview To Finish. Even Then, It Seems Very Small In Comparison To "5-6" Hours Interviews Taken By Some Of The Mnc's. What If There Are 100 Candidates? Will You Give Your 100 Hours For Choosing One Candidate?

The Problem We Are Trying To Solve Is Of Time And Quality. If You Select The Very First Candidate Thinking He/She Seems Good Enough To You, You Might Lose The Opportunity Of Selecting Someone Better In A Little Bit More Time. On The Other Hand, If You Interview Everyone And Consider The Top Rankers For Selection, You've Already Lost All Of Your Precious Time. If You Are Hiring For A Well Established Company, You Might Not Bother About Time And Look For Only The Best.

Now The Problem Comes For The Small Companies, Say A Start-Up. If You Are Hiring For A Start-Up, And You Got A Project And Deadline Is Just Next Month. You Do The Math And It Occurs To You That There Is A Chance That You Might Not Meet The Deadline With Number Of Employees You Have. So You Decide To Hire Some Freshmen To Share The Burden. You Contact Nearby College Looking For Good Enough Candidates For Your Task.
The Problem With The Word "Good Enough" Is It's Definition. In Certain Situations A Barely Passed Candidate Might Be "Good Enough", And Sometimes Even 80 Percentile Seems A Bit Low For The "Good Enough". Fortunately, It Depends On The Overall Average Of The Pool You Are Searching In. Again Here, I Said "Overall", Which Means Complete Average Of The Pool, For Which You'll Have To Take Everyone's Interview, Which Seems Impractical When You Can Select The Best And Not Only The Average After Everyone's Interview.

The Solution To This Problem Is Same As Many Other Problems Like The Secretary Problem, The Stable Marriage Problem, House Searching, Toilet Hunting,Etc. And It Is Called Optimal Stopping. It Utilises The Runtime Pool Average And Balances The Time And The Quality.
In Depth Explanation Of Optimal Stopping Is Given In The Introduction Part.

Chapter 2

Introduction

Optimal stopping problem is also known as the secretary problem, stable marriage problem, 37 percent rule, etc. The solution to all these problem is the same.

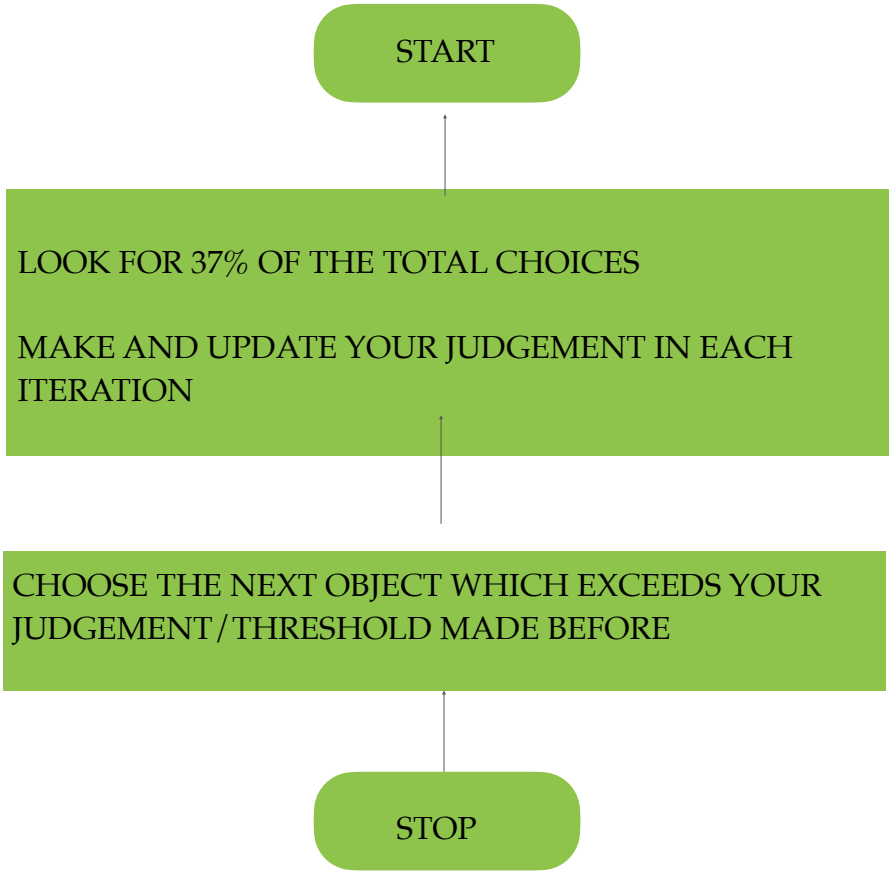
OPTIMAL STOPPING ALGORITHM

There are different entities in different situations and for the sake of our project we will consider **Time** and **Quality**. The balance of Time and Quality will be our goal.

This project is a demonstration of Optimal Stopping Algorithm, specifically the 37% rule. The 37% rule can be summarised in four words as - “Look before you Leap”.

Why 37%? Well it has many lines of theorem behind it. The theorem produces the end result 37% approx. so we will use the end result directly. Although it changes for numbers less than 100, but the impact is immaterial.

Flowchart -



The flowchart seems a little easy to solve that kind of complex problem, and keeping it simple is the policy of many artists. The job is done efficiently and accurately with the above flowchart.

We look at the other options because we should have something to compare it to. If you select early candidates, you have nothing or less data to compare them with. If you search through the end, you've already blown your time.

Optimal stopping algorithm uses **37 percent rule** to make a judgement which we will call as lookup and then it selects a candidate based on that judgement which we will call as selection phase.

Phases-

- **Lookup phase -** In this phase we are looking to make a judgement. Though the concept suggests that you should reject every candidate in 37 percent, we will make an exception because the HR has previous experience and memories which a program cannot have, unless it is Artificially Intelligent.

That being said, we will trust the interviewer/user of this program to enter legitimate data and select a candidate if he finds them to be suitable for the job despite their low score.

- **Selection Phase -** In this phase we will consider the candidates for selection by the program. First I should make it clear that as we gave the command of rejection and selection to the Interviewer, we will try to give them the command of that as well in the selection phase. The problem is, if I give full command to the Interviewer, then this software will only become "**Auto Suggest**" and not "**Auto Select**" which we intended to do.

So, selection phase will automatically select a candidate if they are above the runtime threshold or prompt the interviewer for rejection if they are not. This will ensure that if the interviewer sees some qualities which are not on paper, he can still select that candidate.

Chapter 3

In-Depth Explanation by the Author of the book "Algorithms to live by" Brian Christian :

"It's such a common phenomenon that college guidance counselors even have a slang term for it: the "turkey drop". High-school sweethearts come home for Thanksgiving of their freshman year of college and, four days later, return to campus single.

An angst-ridden Brian went to his own college guidance counselor his freshman year. His high-school girlfriend had gone to a different college several states away, and they struggled with the distance. They also struggled with a stranger and more philosophical question: how good a relationship did they have? They had no real benchmark of other relationships by which to judge it. Brian's counselor recognised theirs as a classic freshman-year dilemma, and was surprisingly nonchalant in her advice: "Gather data."

The nature of serial monogamy, writ large, is that its practitioners are confronted with a fundamental, unavoidable problem. When have you met enough people to know who your best match is? And what if acquiring the data costs you that very match? It seems the ultimate Catch-22 of the heart.

As we have seen, this Catch-22, this angsty freshman cri de coeur, is what mathematicians call an "optimal stopping" problem, and it may actually have an answer: 37%.

Of course, it all depends on the assumptions you're willing to make about love.

The Secretary Problem

In any optimal stopping problem, the crucial dilemma is not which option to pick, but how many options to even consider. These problems turn out to have implications not only for lovers and renters, but also for drivers, homeowners, burglars, and beyond.

The 37% Rule derives from optimal stopping's most famous puzzle, which has come to be known as the "secretary problem." Its setup is much like the apartment hunter's dilemma that we considered earlier. Imagine you're interviewing a set of applicants for a position as a secretary, and your goal is to maximise the chance of hiring the single best applicant in the pool. While you have no idea how to assign scores to individual applicants, you can easily judge which one you prefer. (A mathematician might say you have access only to the ordinal numbers—the relative ranks of the applicants compared to each other—but not to the cardinal numbers, their ratings on some kind of general scale.) You interview*

the applicants in random order, one at a time. You can decide to offer the job to an applicant at any point and they are guaranteed to accept, terminating the search. But if you pass over an applicant, deciding not to hire them, they are gone forever.

"The secretary problem is widely considered to have made its first appearance in print—sans explicit mention of secretaries—in the February 1960 issue of Scientific American, as one of several puzzles posed in Martin Gardner's beloved column on recreational mathematics. But the origins of the problem are surprisingly mysterious. Our own initial search yielded little but speculation, before turning into unexpectedly physical detective work: a road trip down to the archive of Gardner's papers at Stanford, to haul out boxes of his midcentury correspondence. Reading paper correspondence is a bit like eavesdropping on someone who's on the phone: you're only hearing one side of the exchange, and must infer the other. In our case, we only had the replies to what was apparently Gardner's own search for the problem's origins fifty-some years ago. The more we read, the more tangled and unclear the story became.

Harvard mathematician Frederick Mosteller recalled hearing about the problem in 1955 from his colleague Andrew Gleason, who had heard about it from somebody else. Leo Moser wrote from the University of Alberta to say that he read about the problem in "some notes" by R. E. Gaskell of Boeing, who himself credited a colleague. Roger Pinkham of Rutgers wrote that he first heard of the problem in 1955 from Duke University mathematician J. Shoenfield, and I believe he said that he had heard the problem from someone at Michigan.

"Someone at Michigan" was almost certainly someone named Merrill Flood. Though he is largely unheard of outside mathematics, Flood's influence on computer science is almost impossible to avoid. He's credited with popularizing the traveling salesman problem (which we discuss in more detail in chapter 8), devising the prisoner's dilemma (which we discuss in chapter 11), and even with possibly coining the term "software." It's Flood who made the first known discovery of the 37% Rule, in 1958, and he claims to have been considering the problem since 1949—but he himself points back to several other mathematicians.

Suffice it to say that wherever it came from, the secretary problem proved to be a near-perfect mathematical puzzle: simple to explain, devilish to solve, succinct in its answer, and intriguing in its implications. As a result, it moved like wildfire through the "mathematical circles of the 1950s, spreading by word of mouth, and thanks to Gardner's column in 1960 came to grip the imagination of the public at large. By the 1980s the

problem and its variations had produced so much analysis that it had come to be discussed in papers as a subfield unto itself.

As for secretaries—it's charming to watch each culture put its own anthropological spin on formal systems. We think of chess, for instance, as medieval European in its imagery, but in fact its origins are in eighth-century India; it was heavy-handedly "Europeanized" in the fifteenth century, as its shahs became kings, its viziers turned to queens, and its elephants became bishops. Likewise, optimal stopping problems have had a number of incarnations, each reflecting the predominating concerns of its time. In the nineteenth century such problems were typified by baroque lotteries and by women choosing male suitors; in the early twentieth century by holidaying motorists searching for hotels and by male suitors choosing women; and in the paper-pushing, male-dominated mid-twentieth century, by male bosses choosing female assistants. The first explicit mention of it by name as the "secretary problem" appears to be in a 1964 paper, and somewhere along the way the name stuck."

Excerpt From: Brian Christian. "Algorithms to live by"
"1627790365 (PSY)". Apple Books.

Chapter 4
"Whence 37%?

In your search for a secretary, there are two ways you can fail: stopping early and stopping late. When you stop too early, you leave the best applicant undiscovered. When you stop too late, you hold out for a better applicant who doesn't exist. The optimal strategy will clearly require finding the right balance between the two, walking the tightrope between looking too much and not enough.

If your aim is finding the very best applicant, settling for nothing less, it's clear that as you go through the interview process you shouldn't even consider hiring somebody who isn't the best you've seen so far. However, simply being the best yet isn't enough for an offer; the very first applicant, for example, will of course be the best yet by definition. More generally, it stands to reason that the rate at which we encounter "best yet" applicants will go down as we proceed in our interviews. For instance, the second applicant has a 50/50 chance of being the best we've yet seen, but the fifth applicant only has a 1-in-5 chance of being the best so far, the sixth has a 1-in-6 chance, and so on. As a result, best-yet applicants will become steadily more impressive as the search continues (by definition, again, they're better than all those who came before)—but they will also become more and more infrequent.

Okay, so we know that taking the first best-yet applicant we encounter (a.k.a. the first applicant, period) is rash. If there are a hundred applicants, it also seems hasty to make an offer to the next one who's best-yet, just because she was better than the first. So how do we proceed?

Intuitively, there are a few potential strategies. For instance, making an offer the third time an applicant trumps everyone seen so far—or maybe the fourth time. Or perhaps taking the next best-yet applicant to come along after a long "drought"—a long streak of poor ones.

But as it happens, neither of these relatively sensible strategies comes out on top. Instead, the optimal solution takes the form of what we'll call the Look-Then-Leap "Rule: You set a predetermined amount of time for "looking"—that is, exploring your options, gathering data—in which you categorically don't choose anyone, no matter how impressive. After that point, you enter the "leap" phase, prepared to instantly commit to anyone who outshines the best applicant you saw in the look phase.

We can see how the Look-Then-Leap Rule emerges by considering how the secretary problem plays out in the smallest applicant pools. With just one applicant the problem is easy to solve—hire her! With two applicants, you have a 50/50 chance of success no matter what you do. You can hire the first applicant (who'll turn out to be the best half the time), or dismiss the first and by default hire the second (who is also best half the time).

*Add a third applicant, and all of a sudden things get interesting. The odds if we hire at random are one-third, or 33%. With two applicants we could do no better than chance; with three, can we? It turns out we can, and it all comes down to what we do with the second interviewee. When we see the first applicant, we have no information—she'll always appear to be the best yet. When we see the third applicant, we have no agency—we have to make an offer to the final applicant, since we've dismissed the others. But when we see the second applicant, we have a little bit of both: we know whether she's better or worse than the first, and we have the freedom to either hire or dismiss her. What happens when we just hire her if she's better than the first applicant, and dismiss her if she's not? This turns out to be the best possible strategy when facing three applicants; using this approach it's possible, surprisingly, to do just as well in the three-applicant problem as with two, "choosing the best applicant exactly half the time.**

Enumerating these scenarios for four applicants tells us that we should still begin to leap as soon as the second applicant; with five applicants in the pool, we shouldn't leap before the third.

As the applicant pool grows, the exact place to draw the line between looking and leaping settles to 37% of the pool, yielding the 37% Rule: look at the first 37% of the applicants, choosing none, then be ready to leap for anyone better than all those you've seen so far.*

"As it turns out, following this optimal strategy ultimately gives us a 37% chance of hiring the best applicant; it's one of the problem's curious mathematical symmetries that the strategy itself and its chance of success work out to the very same number. The table above shows the optimal strategy for the secretary problem with different numbers of applicants, demonstrating how the chance of success—like the point to switch from looking to leaping—converges on 37% as the number of applicants increases.

A 63% failure rate, when following the best possible strategy, is a sobering fact. Even when we act optimally in the secretary problem, we will still fail most of the time—that is, we won't end up with the single best applicant in the pool. This is bad news for those of us who would frame romance as a search for "the one." But here's the silver lining. Intuition would suggest that our chances of picking the single best applicant should steadily decrease as the applicant pool grows. If we were hiring at random, for instance, then in a pool of a

hundred applicants we'd have a 1% chance of success, and in a pool of a million applicants we'd have a 0.0001% chance. Yet remarkably, the math of the secretary problem doesn't change. If you're stopping optimally, your chance of finding the single best applicant in a pool of a hundred is 37%. And in a pool of a million, believe it or not, your chance is still 37%. Thus the bigger the applicant pool gets, the more valuable knowing the optimal algorithm becomes. It's true that you're unlikely to find the needle the majority of the time, but optimal stopping is your best defence against the haystack, no matter how large.

"

Excerpt From: Brian Christian. "Algorithms to live by"
"1627790365 (PSY)". Apple Books.

Chapter 5

Functions:

Take_interview(prnlist)

This function will be called for each candidate and collect the data about their name, PRN, aptitude, General Knowledge,Personality, email address, etc. prnlist is passes to this function to uniquely identify the candidates and not repeat the same PRN's for two different candidates.

It will return multiple values.

find_37(number)

This function return the 37% of the number passed to it. Although this function is called only once so it was not mandatory to make it as a separate function, but as we wanted keep our program easy to debug and modular, we decided to make it as a function.

check(candidates, lookup)

This function checks if the lookup phase is over, it will return boolean value, 0 or 1. It will be called multiple times for each candidate.

Chapter 6

MACHINE LEARNING FOR SUGGESTIONS USING A CLASSIFIER

Classification and regression comes under Supervised learning of Machine Learning.

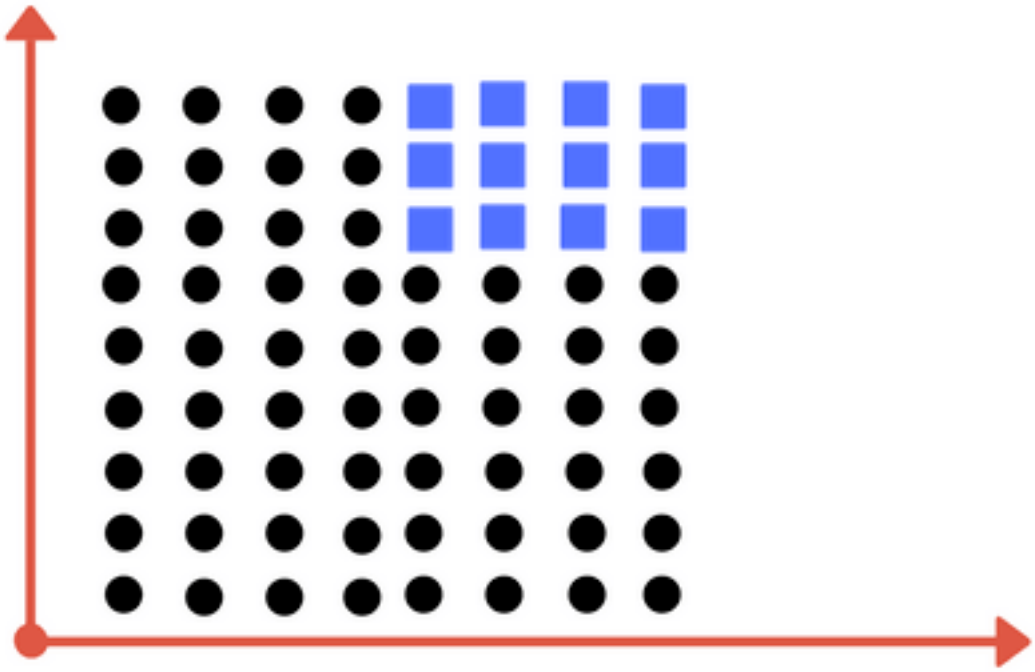
CLASSIFICATION : Identifying to which category an object belongs to, IN OUR CASE SELECTION OR REJECTION.

Applications: Spam detection, Image recognition.

Extracted from the Documentation of Decision tree Classifier :

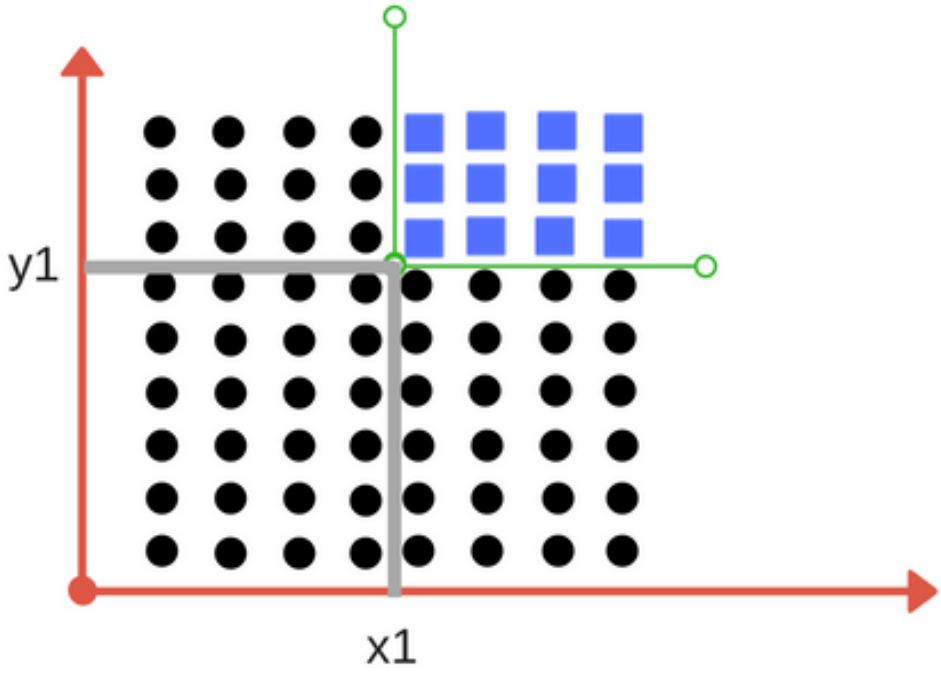
0. Motivation

Suppose we have following plot for two classes represented by black circle and blue squares. Is it possible to draw a single separation line ? Perhaps **no.**



Can you draw single division line for these classes?

We will need more than one line, to divide into classes. Something similar to following image:

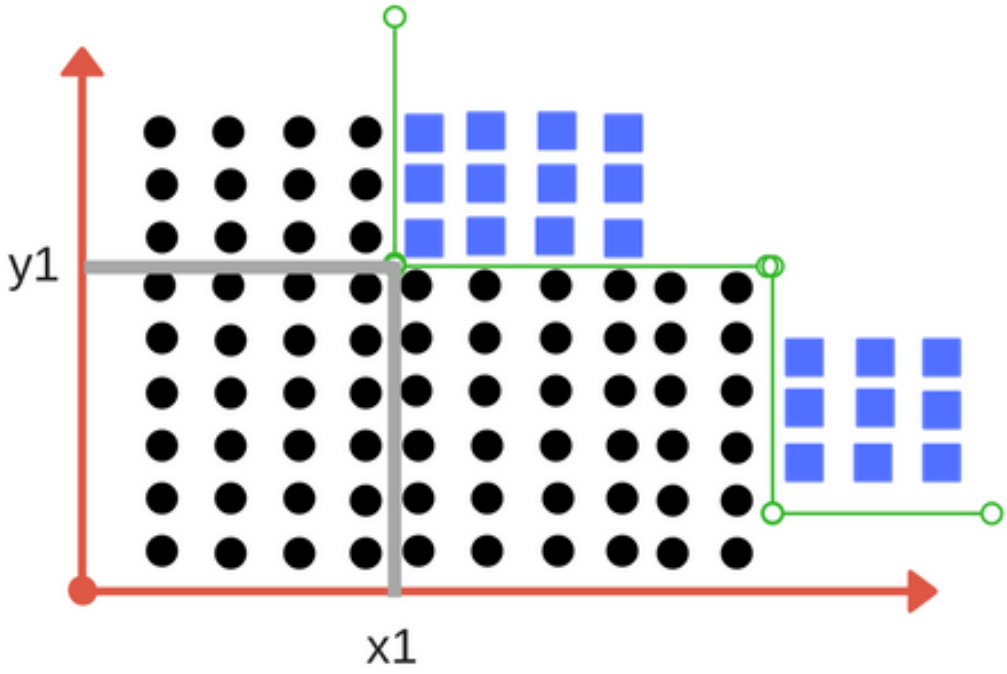


We need two lines one for threshold of x and threshold for y.

We need two lines here one separating according to threshold value of ***x*** and other for threshold value of ***y*.**

As you have guessed now what decision tree tries to do.

Decision Tree Classifier, repetitively divides the working area(plot) into sub part by identifying lines. (repetitively because there may be two distant regions of same class divided by other as shown in image below).



So when does it terminate?

. Either it has divided into classes that are pure (only containing members of single class)

. Some criteria of classifier attributes are met.

We shall see these two points shortly.

1. Impurity

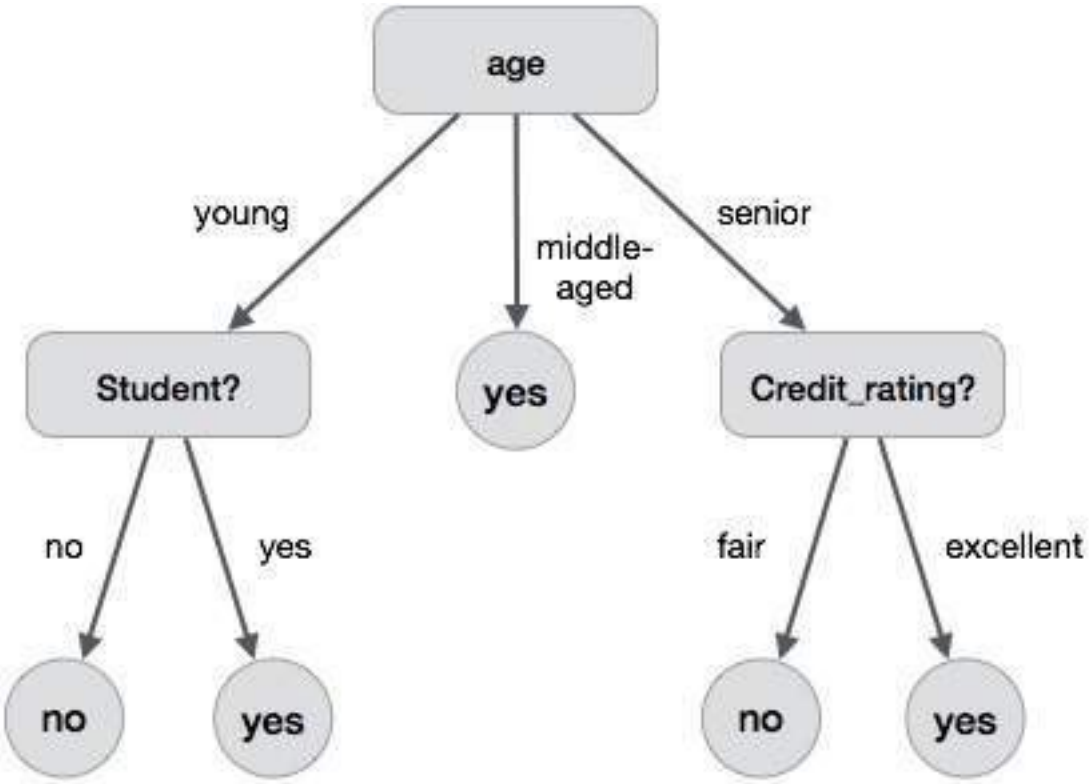
In above division, we had clear separation of classes. But what if we had following case?

Impurity is when we have a traces of one class division into other. This can arise due to following reason

* We run out of available features to divide the class upon.

* We tolerate some percentage of impurity (we stop further division) for faster performance. (There is always trade off between accuracy and performance).

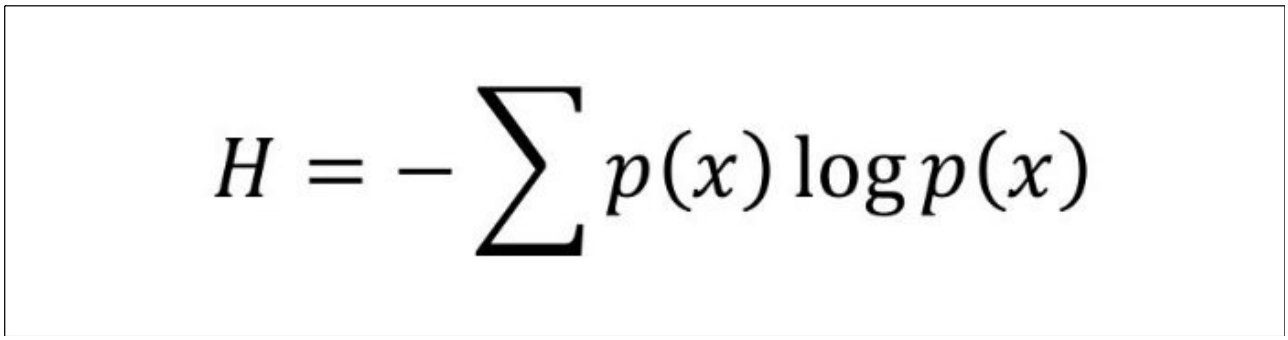
For example in second case we may stop our division when we have x number of fewer number of elements left. This is also known as ***gini impurity.***



Division based on some features.

2. Entropy

Entropy is degree of randomness of elements or in other words it is ***measure of impurity.*** *Mathematically, it can be calculated with the help of probability of the items as:*



$$H = -\sum p(x) \log p(x)$$

p(x) is probability of item x.

It is negative summation of probability times the log of probability of item x.

For example,

```
if we have items as number of dice face occurrence in a throw event as 1123,
the entropy is
   p(1) = 0.5
   p(2) = 0.25
   p(3) = 0.25
entropy = - (0.5 * log(0.5)) - (0.25 * log(0.25)) -(0.25 * log(0.25)
        = 0.45
```

3. Information Gain

Suppose we have multiple features to divide the current working set. What feature should we select for division? Perhaps one that gives us less impurity.

Suppose we divide the classes into multiple branches as follows, the information gain at any node is defined as

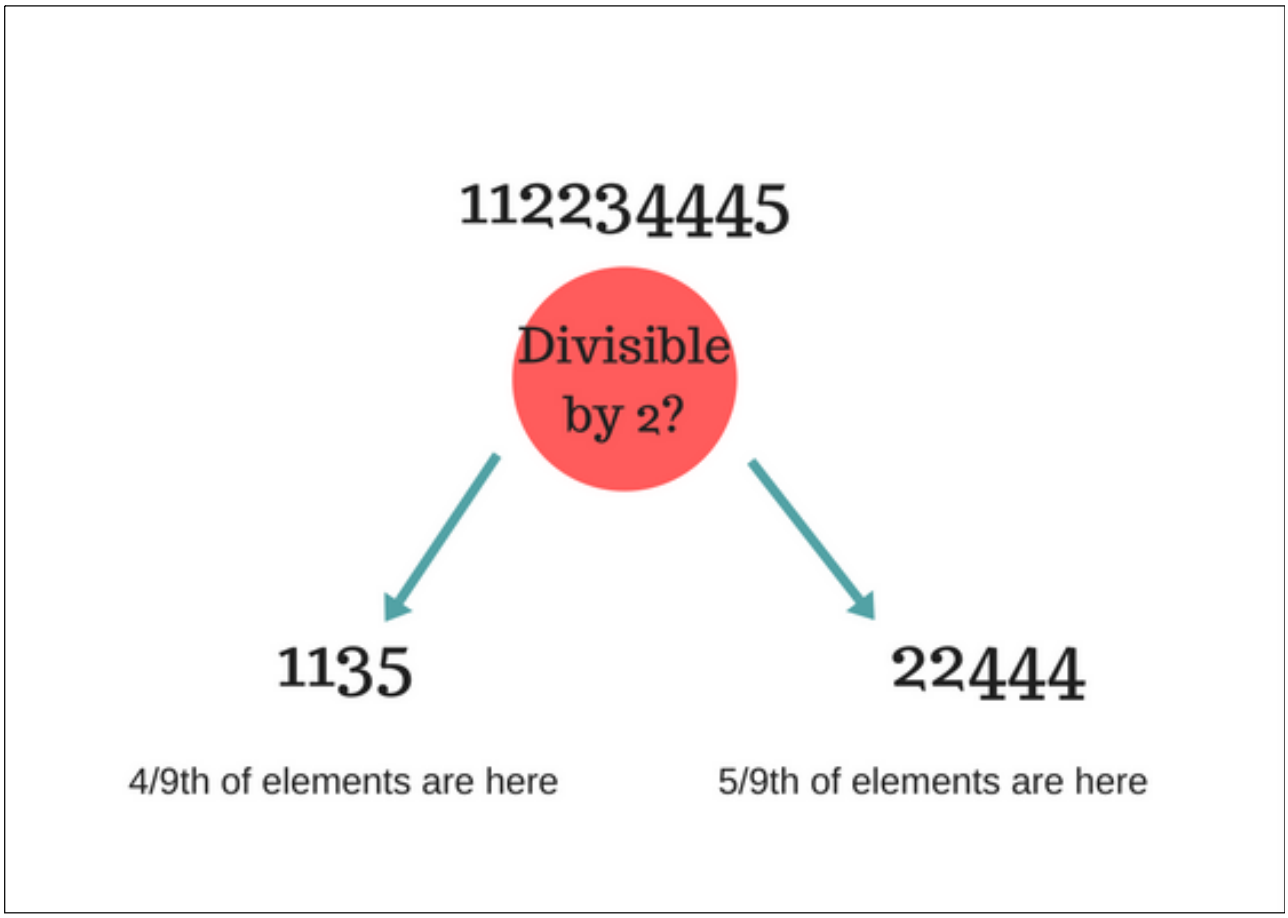
```
Information Gain (n) =
    Entropy(x) - ([weighted average] * entropy(children for feature))
```

This need a bit explanation!

Suppose we have following class to work with intially

112234445

Suppose we divide them based on property: divisible by 2




```
Entropy at root level : 0.66
Entropy of left child : 0.45 , weighted value = (4/9) * 0.45 = 0.2
Entropy of right child: 0.29 , weighted value = (5/9) * 0.29 = 0.16
Information Gain = 0.66 - [0.2 + 0.16] = 0.3
```

Check what information gain we get if we take decision as **prime number instead of divide by 2.** Which one is better for this case?

Decision tree at every stage selects the one that gives best information gain. ***When information gain is 0 means the feature does not divide the working set at all.***

Chapter 7

Runtime Flow of the Program

Let us run a sample test which will show us what will happen during a real life scenario.

Given data :

Number of candidates : 130

Candidates to be selected : 10

The program will prompt for above values and take them from the user.

It will now enter into **Lookup phase** if the values are valid.

Threshold=0

Lookup candidates= 130 *37/100 =48.1~ 48

We will approximate the value using abs() function.

Some early runs :

Candidate name : John Bates

PRN : 1

Aptitude Score : 87

General Knowledge : 67

Technical Knowledge : 87

Personality Score : 79

Overall Score : 80

We will also give the data for training to a **Decision Tree Classifier** and a label will be assigned to it based on selection or rejection.

The Program will prompt the interviewer to select or reject this candidate. The Interviewer decides based on his experience that this candidate should be selected. Program will record this candidate's data and prepare a threshold which is now 80. Because we have nothing to compare it with.

We are using PRN to uniquely identify candidates. So it must be distinct for each.

Just like John Bates the interviewer selects 4 more candidates in the lookup phase-

Betsy White : 76

Allan Crowbar : 69

Samual Rogers : 87

Now the Threshold will be the average of those 48 candidates from the look up phase. Suppose Threshold is now 70.

Entering the Selection phase, program will give the same prompts as before, but after taking scores, program will automatically decide whether to select this candidate or reject. If the program decides to select this candidate it will do so automatically based on threshold score. If it decides to reject them, it will first prompt the interviewer about this rejection. Now it will be in the interviewer's hand whether to select or reject this candidate. *While doing that program will just print the prediction of Machine Learning. Since ML requires lots of training data we can't trust it with auto-selection at this level.*

After the Required number of candidates are selected it will terminate the interview process and now show the Output in different forms.

It will first show the statistics :

Number of candidates interviewed : 60

Number of candidates selected : 10

Number of candidates Rejected : 50

Max score of the Interview : Samual Rogers : 87

Min score of the Interview : Zoey Stalin : 55

Now it will save the data in a CSV file.

After that it will plot a bar graph where X-axis have candidates and Y-axis have scores. The selected candidates will be shown in Green bar, Eligible candidates (Having score more than threshold but were not selected because they were in lookup phase and interviewer decided to reject them.) in yellow and rejected/ non_eligible candidates in Red.

This will Conclude the program.

Chapter 8

Advantages & Disadvantages

Advantages :

- Facilitates the interviewer to add the data of the candidates at run time hassle free.
- Gives in-depth statistics in three different output formats- Statistics, by plotting a bar chart and storage of data in a comma separated value file.
- Visual statistics give a better idea of performance to the interviewer.
- Different colours indicating different type of candidates- Selected, Eligible but rejected and Rejected.
- PRN verification satisfies the condition of no repetition and allows to uniquely identify candidates with same names.
- Demonstration of Optimal Stopping Algorithm is done in an efficient way.
- It finds the perfect balance of Time and Quality.

Disadvantages :

- If we try to give full command to the interviewer this program will become auto suggest and not auto select.
- Once a candidate is Selected or Rejected we cannot change that.

Conclusion :

Thus, we have Demonstrated the Optimal Stopping Algorithm Successfully using HR interview Process.