## 



Application   
Development

Cloud   
Solutions

Business   
Intelligence

Learning   
Solutions

IT   
Staffing



**Compunnel Software Group**  
**Machine Learning for Recruitment** |Process Flow

**PFD Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 09-02-2018 | 1.1 | Process Flow Diagram – Machine Learning for Recruitment (Initial Version) | Rananjay Singh  Shikhar Priya Srivastava  Alabhya Singh |

Table of Contents

[1](#_Toc503358485)

[Introduction](#_Toc503358486) 4

[Scenario 4](#_Toc503358487)

[Problem Statement 4](#_Toc503358487)

[Solution 5](#_Toc503358487)

[Use Case 5](#_Toc503358488)

[Process Flow Diagram 6](#_Toc503358488)

Sub Process: [Parse the resume 7](#_Toc503358489)

Sub Process: [Apply Machine Learning 8](#_Toc503358489)

[Flowchart Summary 9](#_Toc503358489)

[Tools and Technologies 9](#_Toc503358492)

[Landmarks and Accuracy 10](#_Toc503358500)

[Landmark 1 10](#_Toc503358489)

[Landmark 2 10](#_Toc503358489)

[Accuracy Matrix 10](#_Toc503358489)

[Detailed Process View – Step Wise 11](#_Toc503358501)

[Step 1: Basic Categories of a resume 11](#_Toc503358489)

[Step 2: Parse the resume 11](#_Toc503358489)

[Step 3: Apply Machine Learning 14](#_Toc503358489)

[Step 4: Is data accurate? 17](#_Toc503358489)

[Step 5: Database Storage 18](#_Toc503358489)

[Frequently Asked Questions 19](#_Toc503358501)

Machine Learning for Recruitment



**Source:** www.google.com

Introduction

**Why you should read this document?**

Have you ever wondered how a machine can perform the task of 100 persons in a fraction of the time taken by one person? And how can it perform these tasks in a more accurate manner as compared to humans?

If you are a novice or even have some experience in this field , this document will surprise you about how easy it is to use include Machine Learning in your day to day tasks such as recruitment.

Let us take a scenario of an organization which performs its recruitment process.

## Scenario

A company hires a person for recruitment purposes. This person takes a resume as an input, segregates the important contents into his/her desired format and then pushes this data to a data store.. Below table data will give a brief about the productivity and time involved.

|  |  |  |  |
| --- | --- | --- | --- |
| No. of persons | Time taken per resume | Total office hours | Productivity |
| 1 | 30 mins | 8 | 16 resumes per day |
| 2 | 30 mins | 8 | 32 resumes per day |

## Problem Statement

The above chart gives us a unique picture which is summed up in below points:-

1. The number of resumes that can be finished per day is limited.
2. The accuracy level is also limited since the process will become error prone if the management increases its expectations in terms of productivity.
3. There is huge fraction of repetitive processes involved in the above task.
4. There is no predefined or fixed pattern in terms of a resume received as input.
5. There is a considerable amount of time taken per resume.
6. In case the number of resumes increase to thousands or millions, the time and cost will also increase to a considerable extent.

Is there a more efficient way for performing the above tasks? How do you think a machine can help an organization improve the productivity and save time and cost in an exponential manner?

## Solution

The solution to the above problem is **Machine Learning**.

***What is machine learning?***

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. Many researchers also think it is the best way to make progress towards human-level AI.

**In simple words**, if we take an example of the use case of recruitment, Machine Learning can take a set of resumes as an input, classify them in different categories and in any desired format and push it to data store for further reference. The algorithms involved will take care of the uncertainty in pattern by keeping a track and adding improvements in a file known as “training data”. This training data will help the algorithm in providing results with desired accuracy. The algorithm can also make a predictive model for decision making in terms of which candidate to hire and who can stay for the longest period.

Use Case

* Given a resume as input, define a mechanism to identify the headings, classify the contents against these headings and store them in a data store. This has been detailed as below:-

|  |  |  |
| --- | --- | --- |
| Input | Content Type | Formats |
| Resume | Unstructured | All(doc, pdf ,Scanned pdf etc.) |

|  |  |  |
| --- | --- | --- |
| Output | Content Type | Formats |
| Table or document | Structured | Text or JSON |

Process Flow Diagram

**Input :** Resume of desired file format(e.g, PDF,docx)

This operation is currently being done for PDFs using **Tesseract OCR. Tesseract** is an open source [Optical Character Recognition](https://en.wikipedia.org/wiki/Optical_character_recognition) (OCR) Engine, available under the [Apache 2.0 license.](http://www.apache.org/licenses/LICENSE-2.0) For all the formats, the operation done using Apache Tika by extraction of headings.

**Step 1: Identify the basic categories in a resume**

Add new Categories if identified

**Step 2: Parse the resume and extract txt and headings**

**B**

If NO, then improve dataset

New Categories found?

**Step 3: Apply ML to categorize the headings and convert it into JSON**

**C**

Using **sci-kit** to perform this operation.

**Feature extraction** is being done using **Count Vectorizor.**

Trained data is created and updated manually**.**

**k- NearestNeighbors** algorithmis used for **classification .**

**Step 4:** **Is data accurate?**??

**Step 5: Store in database**

## Sub-Process: Parse the resume

START

**B**

No

Yes

**Output:** resumeContents.txt, parsedHTML.txt

**Input to ML**

If PDF

Perform data segmentation in the form of rectangular sub areas.

Apply OCR to extract data in a textual format.

Parse documents to HTML format

Extract headings from the HTML format

END

## Sub-Process: Apply Machine Learning

END

**C**

**Output:** Txt file with data classified in different categories in JSON format

**Input:** Use only resumeContent.txt

START

If categories are less than 3

Extract features from the text file

Apply Machine Learning to classify the data using above features

## Flowchart Summary

1. Categorizing the raw data
   1. Selecting the different fields which need to be extracted like name, address, skills, etc.
2. Parsing the resumes
   1. Standards of reading the file formats.
   2. Extracting text based on the headings.
   3. Using Apache Tika to extract headings from the documents
3. Categorizing the data using ML
   1. Categorizing the headings from the parsedhtml.txt
   2. Checking their implementation
4. Checking the accuracy
   1. Selecting the optimal ways to check the accuracy
   2. Either manual or automated
5. Storing and fetching the data
   1. Storing this processed data in a more suitable database based on the categories.
   2. Selecting a database for fast retrieval and storing.
6. Providing feedback
   1. Calculating error
   2. Checking at which step we have to provide the feedback
   3. Optimizing the current method

Tools and Technologies

|  |  |
| --- | --- |
| Tools and Technologies | Purpose |
| Apache Tika | For resume parsing |
| Tesseract OCR | To scan PDF |
| Apache Jsoup | To generate HTML |
| Sci-kit learn | To apply Machine Learning |
| Mongo DB | For data storage |
| jdk 1.8 | For java implementation |
| python 3.6 | For python implementation |

Landmarks and Accuracy

We tested our process against a set of inputs with the following configuration details and marked the results as “Landmark”. Each landmark allows us to identify the gaps and areas of improvement. These gaps are filled and pushed into next landmark as described below:-

## Landmark 1

* Major task was to extract features for text headings since the concept of Natural Language processing fails in such kind of sample inputs.
* We used combination of words as features and compared them among one another.
* We built a model which considered every line as a heading and classified it into a category. We used outlier detection and removal approach to remove such kind of misclassification.
* Some headings were being left out, so we separated each word in a heading and then extracted features out of each word , further classifying them into a category.
* OCR applicable in case of PDF files.
* Multithreading is not implemented.
* Basic training data was provided.
* Self-learning is not implemented.

## Landmark 2

* Parse the document into HTML format to facilitate proper identification of headings based on patterns derived from the HTML.
* Then we applied classification on these headings rather than on the whole document which provided better and faster results.
* OCR applicable in case of PDF files.
* Mutlithreading was implemented.
* Basic training data was provided.
* Self-learning also implemented.

## Accuracy Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Landmark | No. of inputs | Accuracy | Time taken |  |  |
| Landmark 1 | **30** | 50% | 11 mins |  |  |
| Landmark 2 | **35** | 80% | 6 mins |  |  |
| Landmark 2 | **48** | 80% | 7 mins |  |  |

Detailed Process View – Step Wise

## Step 1: Basic categories in a resume

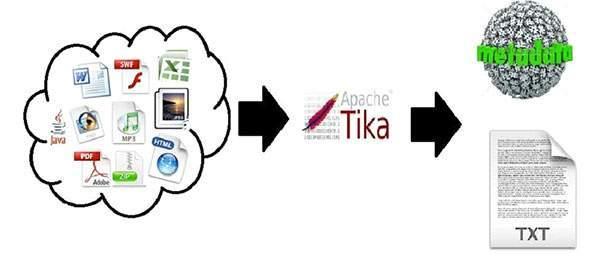
Data of the following 13 attributes for approximately 3000 candidates who were screened for approximately 100 openings over the last 6 months.

1. Skill Sets
2. Educational Details
3. Professional Experience
4. Interests
5. Hobbies
6. Career Highlights and Objectives
7. Strengths
8. Co-Curricular Activities
9. Personal Detail
10. Achievements
11. Projects Done
12. Career Summary
13. Certifications

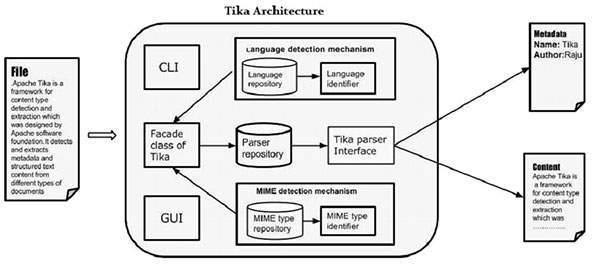
## Step 2: Parsing the resume

Apache Tika libraries can be used to achieve this purpose. A simple and powerful mechanism for client applications to extract structured text content and metadata from all sorts of documents.

[Supported Document Formats](https://tika.apache.org/0.7/formats.html#Supported_Document_Formats)

* [HyperText Markup Language](https://tika.apache.org/0.7/formats.html#HyperText_Markup_Language)
* [XML and derived formats](https://tika.apache.org/0.7/formats.html#XML_and_derived_formats)
* [Microsoft Office document formats](https://tika.apache.org/0.7/formats.html#Microsoft_Office_document_formats)
* [OpenDocument Format](https://tika.apache.org/0.7/formats.html#OpenDocument_Format)
* [Portable Document Format](https://tika.apache.org/0.7/formats.html#Portable_Document_Format)
* [Electronic Publication Format](https://tika.apache.org/0.7/formats.html#Electronic_Publication_Format)
* [Rich Text Format](https://tika.apache.org/0.7/formats.html#Rich_Text_Format)
* [Compression and packaging formats](https://tika.apache.org/0.7/formats.html#Compression_and_packaging_formats)
* [Text formats](https://tika.apache.org/0.7/formats.html#Text_formats)
* [Audio formats](https://tika.apache.org/0.7/formats.html#Audio_formats)
* [Image formats](https://tika.apache.org/0.7/formats.html#Image_formats)
* [Video formats](https://tika.apache.org/0.7/formats.html#Video_formats)
* [Java class files and archives](https://tika.apache.org/0.7/formats.html#Java_class_files_and_archives)
* [The mbox format](https://tika.apache.org/0.7/formats.html#The_mbox_format)

### TIKA Architecture



### Criteria for Tika parsing design

|  |  |
| --- | --- |
| **Streamed parsing** | The interface should require neither the client application nor the parser implementation to keep the full document content in memory or spooled to disk. This allows even huge documents to be parsed without excessive resource requirements. |
| **Structured content** | A parser implementation should be able to include structural information (headings, links, etc.) in the extracted content. A client application can use this information for example to better judge the relevance of different parts of the parsed document. |
| **Input metadata** | A client application should be able to include metadata like the file name or declared content type with the document to be parsed. The parser implementation can use this information to better guide the parsing process. |
| **Output metadata** | A parser implementation should be able to return document metadata in addition to document content. Many document formats contain metadata like the name of the author that may be useful to client applications. |
| **Context sensitivity** | While the default settings and behaviour of Tika parsers should work well for most use cases, there are still situations where more fine-grained control over the parsing process is desirable. It should be easy to inject such context-specific information to the parsing process without breaking the layers of abstraction. |

**By default , Apache TIKA is set to “NO\_OCR” strategy. For reading pdf with scanned images, we need Tesseract OCR.**

**Optical character recognition** (also **optical character reader**, **OCR**) is the [mechanical](https://en.wikipedia.org/wiki/Machine) or [electronic](https://en.wikipedia.org/wiki/Electronics) conversion of [images](https://en.wikipedia.org/wiki/Image) of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast).[[1]](https://en.wikipedia.org/wiki/Optical_character_recognition#cite_note-1) It is widely used as a form of information entry from printed paper data records, whether passport documents, invoices, bank statements, computerised receipts, business cards, mail, printouts of static-data, or any suitable documentation.

**OCR Implementation using Tesseract** is an open source [Optical Character Recognition](https://en.wikipedia.org/wiki/Optical_character_recognition) (OCR) Engine, available under the [Apache 2.0 license.](http://www.apache.org/licenses/LICENSE-2.0) It can be used directly, or (for programmers) using an [API](https://github.com/tesseract-ocr/tesseract/blob/master/api/baseapi.h) to extract typed, handwritten or printed text from images. It supports a wide variety of languages.

Document will be segmented based on the Orientation and script detection[OSD] and Ocr Engine Mode[OEM] value.

### Process implemented:

* Parse the document to HTML format.
* Headings are extracted from the HTML format based on the tags of the content.
* Create ParsedHTMl.txt for the headings and resumeContent.txt for the content.
* If the document is in PDF format, then OCR is applied to extract the text from it.

## Step -3: Apply Machine Learning

### ML tools/ Libraries:

1. ***Sci-Kit learn (Python)***

It is an open source library implemented in Python to execute machine learning algorithms.

It is widely accepted and various support communities are actively present for support.

Easy to implement

Neural networks can also be implemented through this library.

Has vast algorithms for different types of datasets.

Sample datasets are also available for model training.

Methods for feature extraction and feature evaluation are also present.

Methods to check the fitness of the model are also present

**Algorithms/ Methods available:**

*Supervised learning methods*

·         LDA for dimension reduction

·         Regression methods

·         Support vector machines

·         Naïve Bayes

·         Nearest neighbor classification

·         Decision trees

·         Ensemble Methods

·         Neural network models

*Unsupervised Learning*

·         Gaussian Mixture models

·         Clustering

·         Biclustering

·         Outlier detection

·         Neural network models (unsupervised)

*Model selection and evaluation*

·         Cross- validation

·         Tuning of parameters

·         Validation curves

·         Model evaluation

*Dataset Transformation*

·         Preprocessing data

·         Feature extraction

·         Random projection

*Computational Performance*

·         Prediction Latency

·         Prediction Throughput

**2. WEKA (JAVA)**

Implemented in Java.

Rich library, started as Local library now available widely for implementation.

Targeted on Data Mining

Has API to implement or use them in projects.

Algorithms available:(Listed only few)

Classification:

Naive Bayes classifier

Clustering:

Gaussian Mixture

Regression:

Multiple regression

Outlier Detection

Ensemble learning

Preprocessing methods

No method for Fitness of the model.

No method for checking accuracy.

**3. TensorFlow (Python)**

Very useful for high dimension data

Mostly used

Richest of all

Complex to implement

Can be applied to all the machine learning use cases, includes image processing.

Must be required necessarily for deep learning problems.

Widely accepted.

***4. Apache Mahout (Java)***

It is a java implemented library to implement machine learning algorithms. It was originally built to be used as a wrapper for Apache Hadoop.

Some methods of the Mahout can be used for Apache Hadoop while some can also be implemented independently also.

Implementation is easy if used over the Apache Hadoop library, while it becomes a little complex when implemented independently since it will require data pre-processing for which methods are not defined in the Mahout library.

Algorithms available

Classification:- Naïve Bayes Classifier

Clustering :- k means

Linear regression implementation

Preprocessor:- Least mean Square

No scope for ensembling or increasing the accuracy of the model

No method for checking the fitness of the training.

No method to check the importance value of the model

No specific datasets for training the model

**It will be more efficient to use Sci-Kit since it has rich methods and we can extend the usage of this library in further projects also. More online help and content of Sci-Kit is available and it is easy to implement.**

### 

### Deep Learning

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.

It uses Neural networks, CNN, Hidden Markov models to apply the concepts. We will require high level libraries like TensorFlow to implement such methods.

**We had also implemented this using Tensorflow, but due to lack of dataset it was not able to achieve much accuracy, so we switched to our original Sci-kit implemented method.**

## Step -4: Is data accurate?

Currently, supervised learning is being done with the ML algorithm. After checking accuracy, either of the below two steps will be performed:-

1. If any new categories are found , they will be added into the data sets defined in **Step 2.**
2. If no new categories are found, then either the algorithm or its implementation need to be changed.

## Step -5: Database Storage

**Why NoSQL?**

* We need a large, scalable database.
* We don’t require major relations in particular
* Required to implement through different languages.

**Why document oriented database?**

* Data which we are using before is a plain text, later this only database can be used after categorizing the data.
* High scalability
* Low complexity

Stores data in XML, JSON, etc.

Some Examples

DynamoDB

* Only available at AWS
* Integrated with Various other services of AWS
* Supports both document and key-value model

MongoDB

* Available both locally and on AWS
* Widest models in which data can be stored
* Rich language library
* More established and easy to deploy

Couchbase

* N1QL queries, similar syntax to SQL
* Full – text search
* JSON supported along with key-value structure
* Available both at AWS and local server
* Works well with mobile, web and also provide lite version to cater less load on applications

OrientDB

* Both document and graph based database
* Has a layer of SQL and very similar queries

Clusterpoint DB

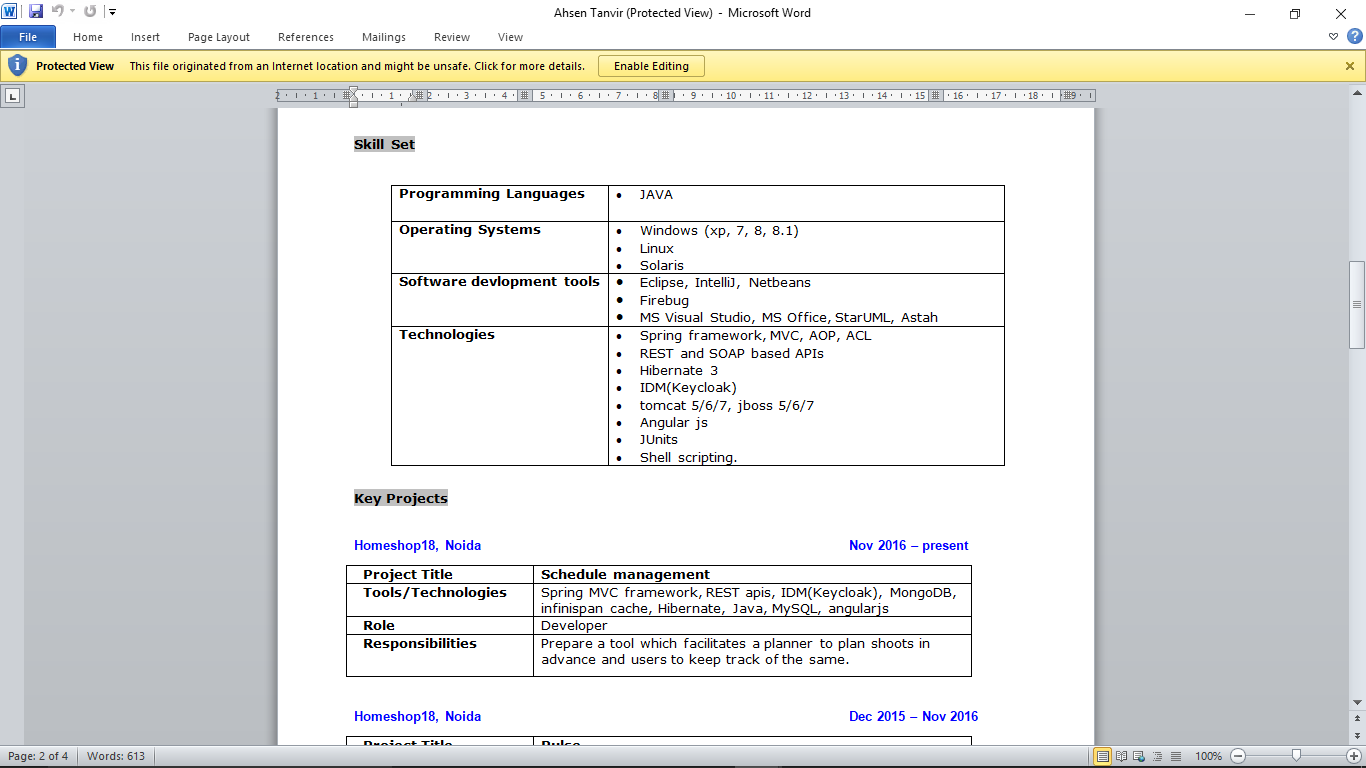
* Available locally only
* Full text search engine

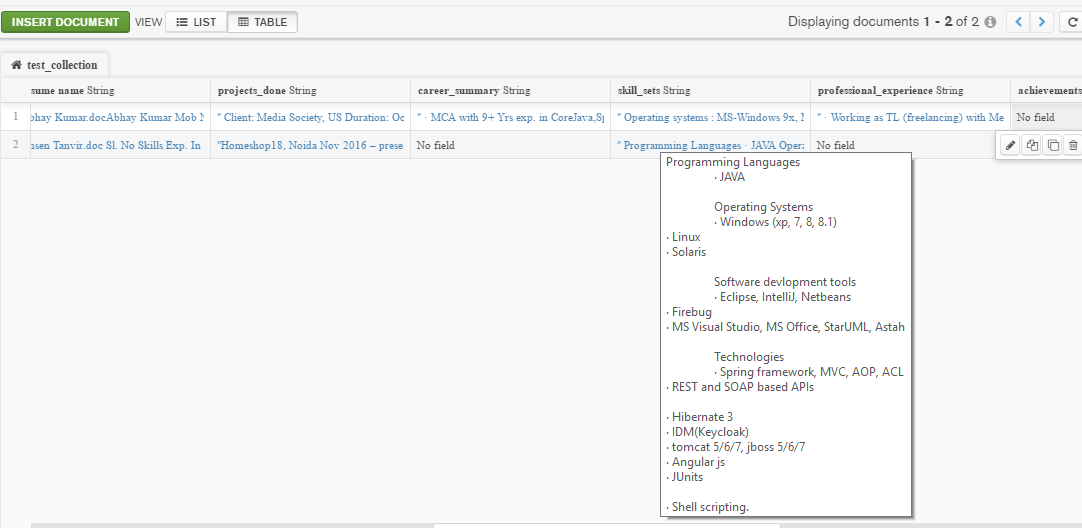
DynamoBD will be good if we are looking to implement on AWS due to integration with other services but other than that MongoDB will provide a good solution to our problem.

Frequently Asked Questions

**Q.** How do we check accuracy after machine learning has done its job?

**A.** We currently have a manual mechanism to check that headings are categorized accurately and the data is mapped accurately against the corresponding heading.

**INPUT:-**

**OUTPUT:-** The part of the image marked in oval in both input and output contains the same data for “Skill Set”, which confirms us that the machine learning process has given accurate output. **Out of 10 categories present in the input resume, if 8 are found and mapped correctly we conclude the accuracy as 80%.**

**Q.** **What are the limitations in the traditional approach for resume parsing?**

**A.** Currently the job sites, use a traditional approach of asking an applicant to fill their details in a pre-defined format . This format is used to classify and extract details further. This approach would not work in case of an undefined and random format of a resume.

**Q.** **Why Machine Learning?**

**A.** As discussed in the previous question, the task can be performed by writing an automated process in a programming language using traditional methods such as regular expressions or string matching. But there is a limitation to this approach. It can be used only in case of an input with a predefined pattern.

For the current use case, there is no predefined and fixed pattern in the input. Machine Learning is a technique that can be used as an alternative approach to tackle this problem. In this case, Machine Learning uses a set of algorithms and training data to perform classification of headings in a resume.

**Q.** **What are the algorithms available in Machine Learning for this purpose?**

**A.** Available methods are :- naive bayes classifier , support vector machines and k- Nearest Neighbors.

* **Naive bayes classifier** is probabilistic based method which is useful when a sample can be put in a class based on the classes of other samples but is chosen based on the maximum probability associated with the class.
* In our use case, a sample can exist in a class based on the features of itself only and not on the features of any other sample.
* **Support vector machines** are useful when we have to build a general model and it is complicated to implement in a case like ours which has more than 2 classes in which data can be classified.
* **k-nearest neighbors** selects the nearest k number of neighbors and classifies the sample in a class with a dominant neighbourhood.

**Q.** **What are the advantages of k- Nearest Neighbors over the other alternatives?**

**A.** Below points provide an answer to this :-

1. Outlier detection and removal as per the requirement.
2. Easy implementation and customization.

**Q.** **Why can’t we use neural networks in this use case?**

**A.** Below are the limitations to neural networks:-

1. It requires a large and more consistent data to give better results.
2. It can't remove outliers.