Data Categorization of Resumes using Machine Learning

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

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

Apply OCR to extract data in a textual format.

Perform data segmentation in the form of rectangular sub areas.

**Step 2:** **Parse the resume**

Add new Categories if identified

This operation is currently being done 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)

------------------

**Output:** resumeContents.txt

**Input to ML**

YES

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 .**

If NO, then improve algorithm

**Step 3: Apply ML**

New Categories found?

Apply Machine Learning to classify the data using above features

Extract features from the text file

-----------------

**B**

**Output:** Txt file with data classified as:- 1.Heading

2. Content

3. Outliers

**B**

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

NO, then give FEEDBACK

YES

**Step 5: Store in database**

**Below is the flowchart summed up in bullet points:-**

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. Categorizing the data using ML
   1. Classifying the extracted headings as the categories in which we want to store.
   2. Identifying best ML algorithms and tools for the task.
   3. 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

**Step-1: Basic categories in a resume**

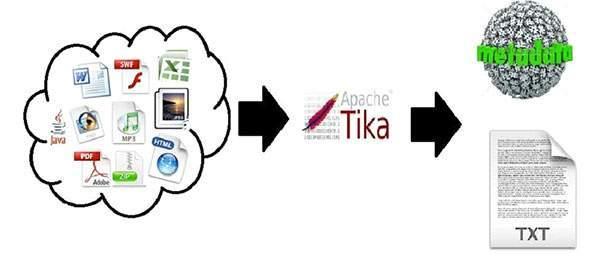
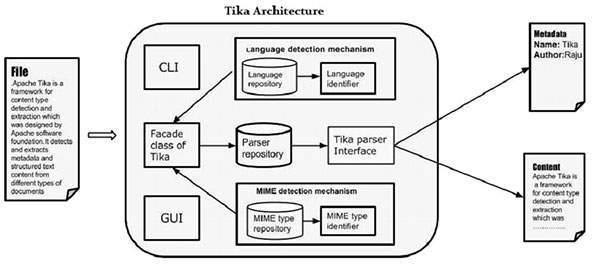
Data of the following 13 attributes for approximately 3000 candidates who were screened for approximately 100 openings over the last 6 months.

1. Current Compensation
2. Expected Compensation
3. Education
4. Specialization
5. Location
6. Earliest Start Date
7. Total Experience
8. Relevant Experience
9. Communication
10. Current Employer
11. Stability
12. Education Gap
13. Work Gap.

**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:-**

**Criterias 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.

**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.

**Can be easily integrated with MongoDB** as per below link:-

<https://github.com/selvinsource/mongodb-datamining-shell/wiki/Weka-MongoDB>

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.

**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