Cognitive Approach in Document Indexing

Savo TOMOVIĆ[1], Kosta PAVLOVIĆ[2]

Abstract

Even though the digital processing of documents is increasingly widespread in industry, printed documents are still largely in use. Datum Solutions Cognitive Capture implements the automatic processing of administrative documents that need to be treated in a close to real time manner. The software can handle complex documents, in which the contents of different regions and fields can be highly heterogeneous with respect to layout, printing quality and the utilization of fonts and typing standards.

Keywords: cognitive capture, text classification, document indexing

Introduction

With the growing use of information technology, web systems and recent advances in database management systems used for creating, retrieving, updating and managing data, as well as fast and secure access to data repositories through computer networks and grids, the amount of data available to numerous companies, agencies and scientific laboratories has increased exponentially (Bernus and Noran, 2017).

Naturally, all users want to utilize these huge repositories to improve their business activities. As a result, cognitive systems have become a very popular topic, technique and method for solving the "data rich and information poor" syndrome (Gliozzo, 2017).

Cognitive systems are about exploring the data - the process of semiautomatic, reliable and intelligent analysis of large sets of mainly unstructured data and discovery of useful knowledge, information, instructions, answers, correlations and new context in that data to provide innovative solutions (Bernus et all 2017).

Cognitive techniques are getting increasingly integrated in day-to-day business operations making them more efficient and cheaper. At the same time, cognitive computing is getting increasingly interesting for research workers and scientists from different areas.

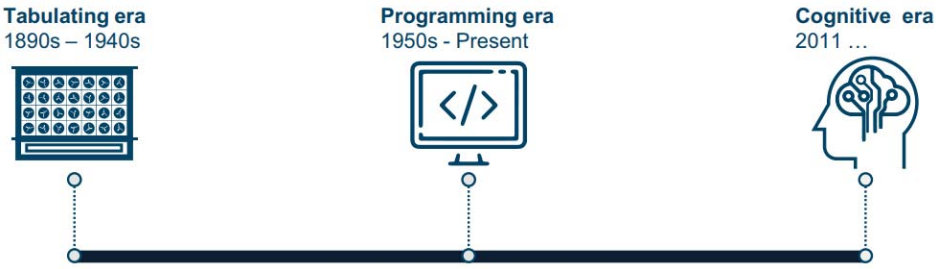
Consequently, real-world problems and new cognitive topics are constantly growing.

There are two distinct eras of computing have occurred: the tabulating era and the programming era. We are entering the third and most transformational era in computing's evolution, the cognitive computing era (Gliozzo, 2017).

[1] Savo TOMOVIC is associate professor at the University of Montenegro, Montenegro, E-mail: savot@ac.me

[2] Kosta PAVLOVIC is teaching assistant at the University of Montenegro, Montenegro, E-mail: kosta@ac.me

Figure 1. The three eras of computing (Gliozzo, 2017)



Dr. John E. Kelly defines cognitive computing (Kelly J. E, 2015): "Cognitive computing refers to systems that learn at scale, reason with purpose and interact with humans naturally. Rather than being explicitly programmed, they learn and reason from their interactions with us and from their experiences with their environment."

Cognitive computing systems has wider and more advanced characteristics from those generally attributed to artificial intelligence. Cognitive techniques do not understand just replacing and/or replicating the way that the human brain works. They are meant to extend the capabilities of the human brain (Bernus et all 2017).

People are good at fast reasoning, deep thinking, and solving complex problems. But the human ability to read, analyse, and process huge volumes of data, both structured and unstructured, is quite poor. That is the strength of the computer system (Bernus et all 2017).

Cognitive computing system is to combine strengths of human and machine into a collaborative situation. Cognitive computing uses machine strengths to "simulate" the human thought processes in a computerized model (Bernus et all 2017).

One more key element of cognitive systems is a more natural interaction between human and machine, combined with the capability to learn and adapt over time (Bernus et all 2017).

Cognitive systems use techniques, such as machine learning, data mining, natural language processing, and pattern matching to mimic how a human brain works.

Cognitive Capture

Core Cognitive Capture functionalities cover automatic processing of administrative documents that need to be treated in a close to real time manner. By the automatic processing we consider *document classification* and *indexing*.

Many commercial products offering such services are based on the definition of fixed spatial templates that basically map the locations in which the OCR must read the fields to extract. Such basic strategies might perfectly work in simple scenarios but start to cause problems as soon as the number of documents to deal with become massive and if the document layout varies along time.

The Cognitive Capture can handle complex documents, in which the contents of different regions and fields can be highly heterogeneous with respect to layout, printing quality and the utilization of fonts and typing standards.

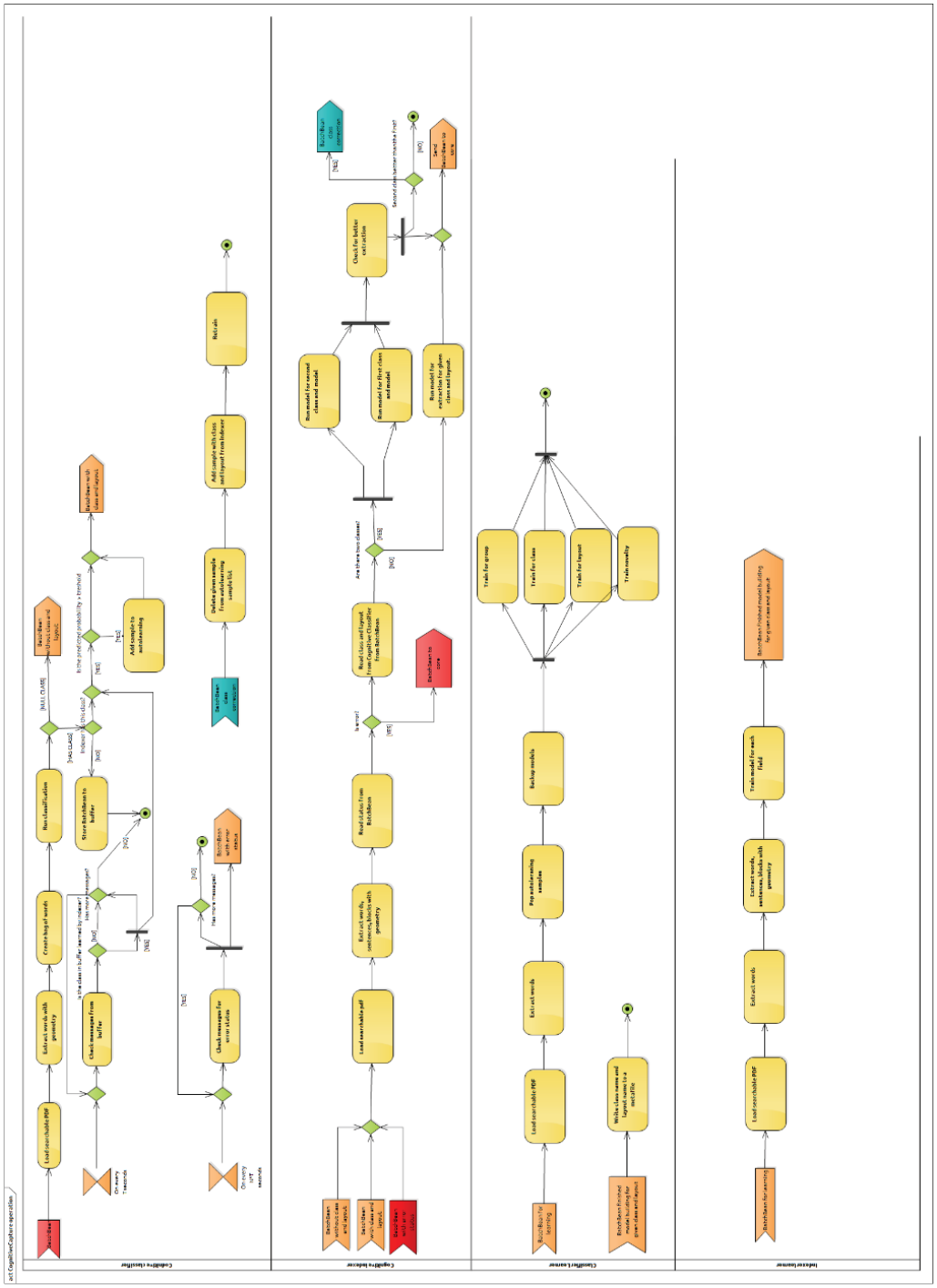
The Cognitive Capture employs a machine learning approach, whereby the system is first provided with a set of training documents in which the target fields are manually tagged. The system automatically learns how to classify future documents and extract desired fields in them.

Additionally, the Cognitive Capture can build a document model given a single training sample. In that sample user has marked fields that must be extracted. The model is representing structural relationships among target fields. The model is incrementally refined as the system processes more and more documents. To additionally improve model accuracy, Cognitive Capture builds different model for each document class.

Special attention has been given in our solution to ensure high usability of the resulting system. The semi-automatic construction of a model for a document has been designed to simplify and speed-up the required human intervention, and the matching between the model and new instances of the document is fully automated and based on computationally efficient techniques.

The Cognitive Capture contains several connected modules. The architecture is presented on the following picture.

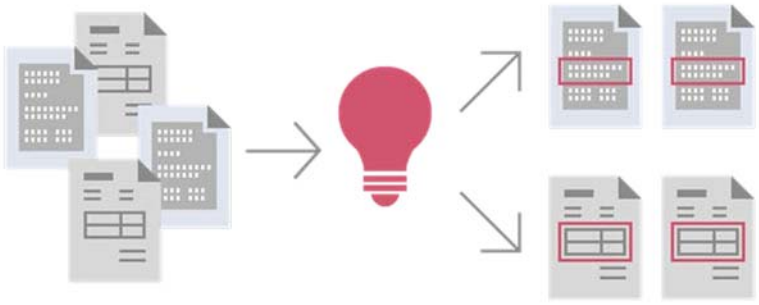
Figure 2. Cognitive Capture activity diagram



Document Classification

By document classification we consider classifying or grouping various documents into predefined set of classes or groups or categories.

Figure 3. Document classification step



Cognitive Classifier can achieve that documents in the same category are more similar in some sense (i.e. contextually) to each other than to those in other classes.

The library scikit-learn for Machine Learning in Python (Smola and Schölkopf, 2004) is used for implementation of the Cognitive Classifier.

Document classification is implemented as linear classifier with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

The system can start with just one training sample as well as very large set of documents for which classes they belong to are known in advance (supervised learning approach). Based on this training set the system creates initial classification model.

Classifier is built in two steps: training and testing. In the training step we apply SGD machine learning algorithm on the given set of documents and create model. It is common to build several models during this step. In testing phase, we want to estimate model performances. There are several measures. The most common one is model accuracy that is percentage of correctly classified instances. To achieve realistic estimation, it is highly recommended to have independent sets of samples for training and testing. At the end we usually choose model with the highest accuracy.

For document representation we use a bag-of-words approach – the text content is turned into numerical feature vectors. Text pre-processing, tokenizing and filtering of stop words are included in an elevated level scikit-learn component named CountVectorizer. It can build a dictionary of features and transform documents to feature vectors. The CountVectorizer assigns a fixed integer id to each word occurring in any document of the training set (for instance by building a dictionary from words to integer indices). After that, for each document *#i*, count the number of occurrences of each word *w* and store it in *X[i, j]* as the value of feature *#j* where *j* is the index of word *w* in the dictionary.

As we mentioned earlier, the best accuracy (above 90%) we achieved with linear support vector machine (SVM) algorithm, which is widely regarded as one of the best

text classification algorithms. Of course, we can change the learner by just plugging a different classifier object into our pipeline.

The Cognitive Classifier implements auto-learning feature for document classification. It means that the system can dynamically be extended with new classes.

For new class detection we used OneClassSVM classifier (Rusiñol, Benkhelfallah and dAndecy, 2013. It is an unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the current set.

When the Cognitive Classifier made wrong prediction, procedure named Manual Assembly is activated. Manual Assembly is primarily web-based component.

Manual Assembly provides user interface for assigning the current document to one of the existing classes. Also, it is possible to create new class and assign the current document to it. To achieve the best accuracy, user can specify key words for each new class.

In the backend, there is a batch application that creates JSON messages for the Cognitive Classifier as consequence of the user actions through Manual Assembly interface. These messages are referred as learning messages.

The learning messages are core of auto-learning process for the Cognitive Classifier. All learning messages are prepared in JSON format and written in the appropriate queue from where the Cognitive Classifier can consume them. The messages contain information about correct class when classifier made wrong prediction as well as description of the new class if it must be added to the system.

After receiving learning messages, the Cognitive Classifier maintains extending and tuning procedures of the existing model to achieve correct classification for subsequent documents.

The Cognitive Classifier has option for learning from data that doesn't fit into main memory - out-of-core approach.

Output of the Cognitive Classifier is a JSON message that is written to the appropriate queue. The message contains classification result: group label, class label and layout label. This means that this module processes hierarchical classification because document can belong to one of many layouts into one of many classes into one of many groups.

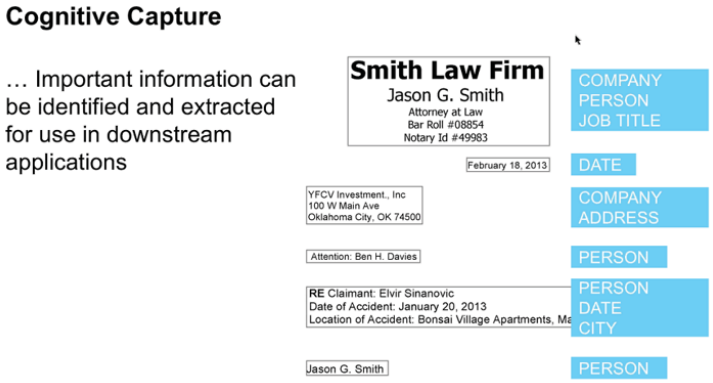
The Cognitive Classifier can be used as standalone module. But, in the Cognitive Capture document classification is used as the first step in extracting/indexing target fields from a given document. The Cognitive Capture builds different indexing model for each document layout and in that way improve indexing accuracy.

Document Indexing

By document indexing we consider extracting desired fields from a given document.

Document can be in any format, but the most efforts are directed on scanned document images. Potentially, documents could be from any context, but we were concentrated to administrative documents, more precisely on invoices.

Figure 4. Document indexing step



Our solution is tested on experimental set of documents – scanned invoices downloaded from the web. We can achieve accuracy greater than 90% for the following target fields: invoice number, invoice date, vendor, address and total amount.

We employ a machine learning approach, whereby the system is first provided with a set of training documents in which the target fields are manually tagged, and automatically learns how to extract these fields in future documents. We implemented two approaches.

The first one is hybrid solution based on implementing rules and/or classifier for each desired field. We have separate rules/classifier for each target field.

Rules are not static and rigid and are not based on absolute position of the target field in the document. We implemented dynamically adaptable rules. Our rules allocate some points to each filed in the document. At the end, the desired filed should be the winner – the filed to which our algorithm gave the most points.

Rules allocate points based on filed content and format, surrounding fields' content and format, relative position in the document, etc. We can modify, add and delete rules and their points allocation algorithms dynamically, with respect to new documents being processed (auto-learning procedure).

Alternative to rules is classification model that we build for desired filed. Model should be trained and tested on realistic set of documents. As fields' attributes we consider relative positions, size, font size and use NLP techniques to extract contextual information. The best accuracy we achieved with Decision trees. Of course, we can change between rules and classifiers by just plugging a different object into our pipeline.

The second approach builds a document model representing structural relationships among target fields given a single training sample in which the user has marked which fields must be extracted from a document group, class and layout. The main idea is to make the model learns document layout.

Layout is document geometry. The model stores positions of each filed with respect to other in polar coordinates. Target fields is found by managing voting process where each filed give or does not give vote to the target. The voting is based on geometry relations between voter and target encoded in polar coordinates. The model is

incrementally refined as the system processes more and more documents from the same class.

The Cognitive Indexer is batch application. It accepts messages from the Cognitive Classifier message queue. It uses classification labels assigned to the document being processed to extract target fields.

Document indexing maintains separate models for each target filed and for each document group, class and layout. If the field from the current document is recognized with high confidence the corresponding model is tuned to achieve better model accuracy for subsequent documents from the same group, class and layout.

All fields that are extracted with sufficiently high confidence are packed in JSON message that is written to the appropriate queue. This is the end of cognitive flow. The separate module with appropriate web interface listens the cognitive result queue and highlights extracted fields in the user window.

When the Cognitive Indexer is not able to extract target fields with sufficiently high confidence or when wrong content is extracted the process terminates with failure. The current document is sent to Manual Indexing module to initiate auto-learning procedure of the system (incremental learning and models tuning).

Manual Indexing is primarily web-based component. It provides user interface for definition of target fields. User can specify filed type, format, labels and other attributes to achieve the best accuracy.

In the backend, there is a batch application that creates JSON messages for the Cognitive Indexer as consequence of the user actions through Manual Indexing interface.

The messages are core of auto-learning process. All messages are prepared in JSON format and written in the appropriate queue from where the Cognitive Indexer can consume them. The messages contain information about correct content that should be extracted when indexer made wrong decisions.

After receiving learning messages, the Cognitive Indexer recognizes which models caused wrong extraction and should be updated. It maintains extending and tuning procedures of the models to achieve better results for subsequent documents.

References

Bernus P., Noran O. (2017) *Data Rich – But Information Poor*. Collaboration in a Data-Rich World. PRO-VE 2017. IFIP Advances in Information and Communication Technology, vol 506. Springer, Cham

Gliozzo A. (2017) *Building Cognitive Applications with IBM Watson Services: Volume 1 Getting Started*, ISBN-10: 073844264X, 2017.

Kelly J. E.*: Computing, cognition and the future of knowing: How humans and machines are forging a new age of understanding*, 2015.

Rusiñol M., Benkhelfallah T. and dAndecy V. P. (2013) *Field Extraction from Administrative Documents by Incremental Structural Templates*, Document Analysis and Recognition (ICDAR), 2013. DOI: 10.1109/ICDAR.2013.223

Smola A. J. and Schölkopf B. (2004) *A Tutorial on Support Vector Regression*, - Statistics and Computing archive, Volume 14 Issue 3, 2004, pp. 199-222

scikit-learn for Machine Learning in Python, scikit-learn.org