+Overall AI NLP Project Design Document

Date of initial Document: December 2020

Document Owner: Jonathan Brun

Affected Software: Nemo, NimonikApp, Leborg

Affected Cells: CO Cell, Updates Cell,

Design Principles:

Is this change making the service more Granular, Duplication and Faster

Strategy

Is this change helping a strategic pillar : Completeness, Meta-Data or Managing Internal Obligations (Select one or multiple)

Nimonik 2020-2023 Product Vision Document

CO Cell Strategic Objectives

This document brings together information, plans and objectives for the automation of all of our data processing efforts. It will highlight what has been done and what the next stage is.

Progress Dashboard (KPI)

- # of outstanding parse requests
- # of outstanding classification requests
- # of parse requests completed per week
- # of outstanding classification requests completed per week
- # of jurisdictions with classified clauses for all L1 Topics

Dependant Design Documents:

Parser Improvements

Topics Overhaul Project

Definitions

Complete Thought

Complete Thought is the combination of a clause with its parent clause(s).

Class

In AI class refer to:

1. It is the category or set where the data is *“labelled” or “tagged” or “classified”* to belong to a specific class based on their common property or attribute.

2. Class label is the discrete attribute having finite values (dependent variable) whose value you want to predict based on the values of other attributes(features).

In Nimonik's Project refer to:

Class is either an **Obligation (for a company) or Non-Obligation** for all other types of data.

Classification

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

For example in Nimonik's project **class is either Obligation or Non-Obligation** and **classification would be assignin obligation or non-obligation to clauses**

Clause

Clause is a CLCO within Leborg, it can be a sentence or multiple sentences, but it should not contain multiple items that are different levels of hierarchy.

True Positive

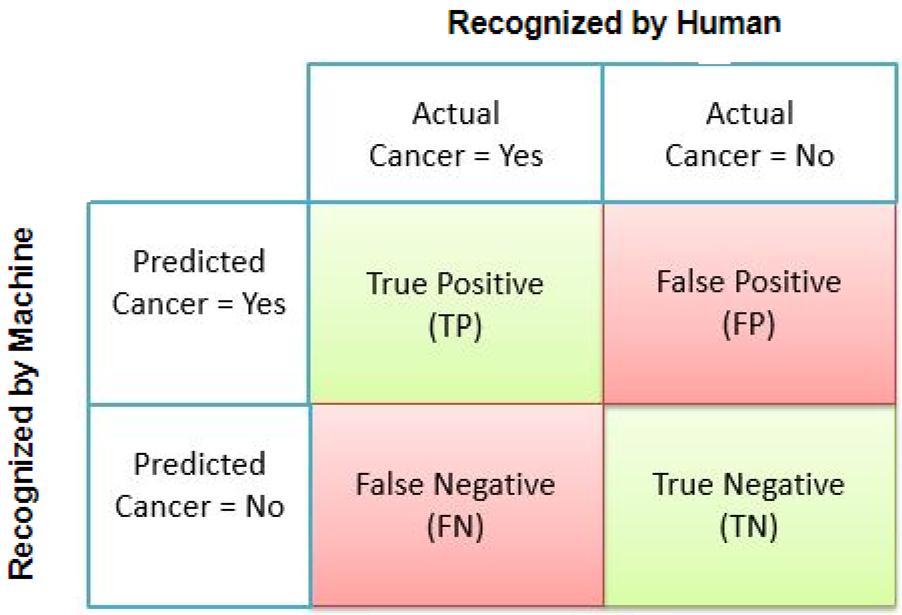
It means the AI correctly classified an obligation clause as an obligation.

False Positive

It means the AI wrongly classified a non-obligation clause as an obligation.

False Negative

It means the AI wrongly classified an obligation clause as a non-obligation.



Recognized by Machine \ Recognized by Human	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive (TP)	False Positive (FP)
Predicted Cancer = No	False Negative (FN)	True Negative (TN)

Accuracy

It means how often the AI assins the correct class to the predicted ones.

If the samples in the data set include 50%/50% real Obligation and real Non-Obligation, then calculating Accuracy is enough, and there is no need for calculating Recall and precision.

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total Samples}} = \frac{\text{Total Correct Asigned Class}}{\text{Total Samples}}$$

Recall

It means how often the AI assigned obligation to a real obligation clause.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Precision

It means if the AI predicted a clause as an obligation, how often it is really an obligation.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Common Regulation(CR)
Some regulations in Training and Testing set are the same , but their used clauses are totally different. It improves the Accuracy as the legal language in the tested jurisdiction is similar; but the achieved Accuracy is more sensitive towards the unseen jurisdictions.

Separated Regulation(SR)
There is no the same regulation in Training and Testing set to evaluate the behavior of applied algorithm on the unseen legal clause structures in new jurisdictions. It reduces the achieved Accuracy but leads to having a more robust/stable result.

Background Information

1. SR & D Claims
2. IRAP Claims
3. All AI Meeting Minutes

Workflows

Document Processing Workflow Spreadsheet Format

KPI Tracking

LeBorg Stats
Weekly Dashboard
Jurisdiction Status Sheet - Beta

Incremental Improvements to Leborg UI/UX

AI/NLP Small Changes, Big Impact - Small Incremental Improvements

Key Required Data for Further Decision Making

Ohno Circle Videos

- Ohno Circle of Parsing - Video from Danni and Ashok
- Ohno Circle of Classification - Video from Moh, Sara V., Greg, Benjamin

Anatomy of a design doc

Context and scope

This section gives the reader a very rough overview of the landscape in which the new system is being built and what is actually being built. This isn't a requirements doc. Keep it succinct! The goal is that readers are brought up to speed but some previous knowledge can be assumed and detailed info can be linked to. This section should be entirely focused on objective background facts.

Goals and non-goals

Goals

As we are working on a live commercial product, we must simultaneously improve our efficiency AND improve the value we deliver to the customer. The prioritization of design decisions should be made on the value for the customer first, then the internal benefits.

Short-Term Goals (6 months)

- Parse all outstanding parse requests
- Classify (Obligations and Topics) all outstanding requests
- Identify all documents in Audit Protocols. All of these documents need to be Parsed and Classified

Long-Term Goals

- Machine Parse 80% of Documents without Human Intervention
- Identify 50% of Obligations automatically
- Process Internal documents from customers for Obligations

Non-Goals

-

The actual design

This section should start with an overview and then go into details.

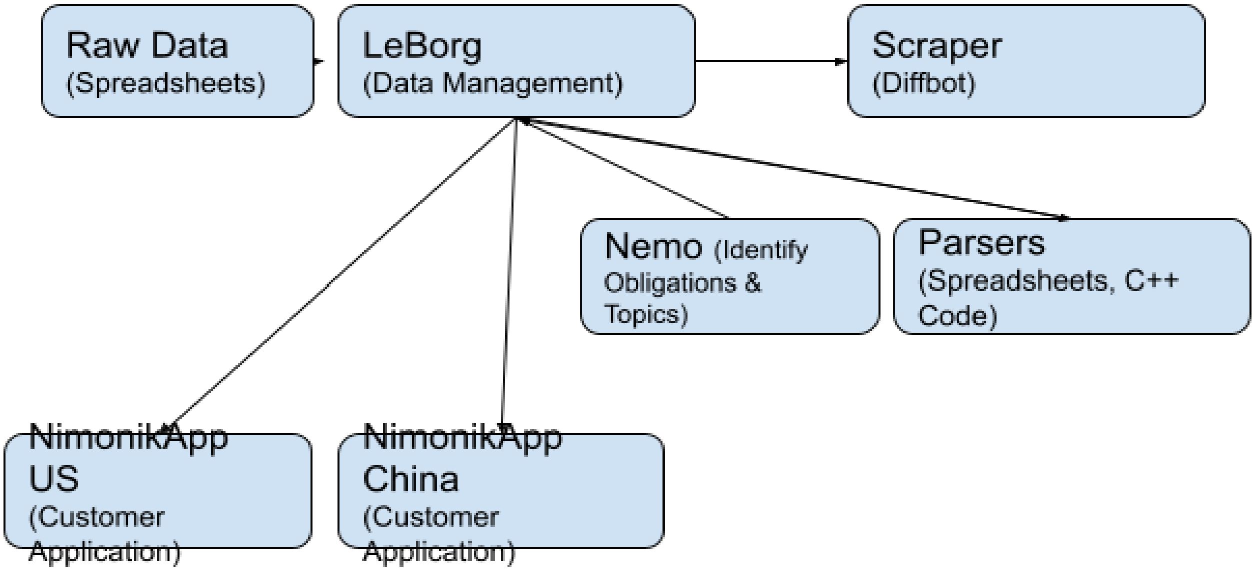
The design doc is *the place to write down the trade-offs* you made in designing your software. Focus on those trade-offs to produce a useful document with long-term value. That is, given the context (facts), goals and non-goals (requirements), the design doc is the place to suggest solutions and show why a particular solution best satisfies those goals.

The point of writing a document over a more formal medium is to provide the flexibility to express the problem set at hand in an appropriate manner. Because of this, there is no explicit guidance for how to actually describe the design.

Having said that, a few best practices and repeating topics have emerged that make sense for a large percentage of design docs:

System-context-diagram #

In many docs a *system-context-diagram* can be very useful. Such a diagram shows the system as part of the larger technical landscape and allows readers to contextualize the new design given its environment that they are already familiar with.



System-context-diagram of parsing and classification of data.

Granularity & Transparency

We must create transparency and granularity at all times. We must avoid abstractions. Examples of changes that would improve the these two design elements are

Improvements to LeBorg Clause Classification Dashboard - Done
Star Feature for Documents - Done
Star Feature for Clauses - Pending completion of CLCO copy to NimonikApp
Add Number of Uses of Document on NimonikApp Page - Not Urgent

Objective: Faster

We must make the identification and management of obligations much faster. It needs to both be faster for our team, who are processing the data and faster for the customer who wants to know the obligations. Changes that would help make things faster:

Trello Cards

Create copy of CLCOs on NimonikApp - Critical
Add CLCOs to Legislation search on NimonikApp
Better workflow for selecting CLCO to add

Topics Equivalent Terms

Document Processing Workflow

Upload Parsed Documents to LeBorg

I'm writing with a question about uploading newly parsed documents to Leborg. Based on your timesheets, it seems to take several minutes per document to upload a document to Leborg. I'd like to see if we can improve that.

1. Is this the procedure that you follow to upload a parsed document?
https://wiki.nimonik.com/index.php?title=How_to_parse_a_document#Put_in_use_in_Leborg

2. What are the steps (or substeps) in the procedure that take the longest, and why?

3. Do you have any suggestions for how we could make this faster or easier?

Excel to CSV

1. All excel files are uploaded on Google Drive.
2. Open each excel file in Google Drive and download it as CSV in local drive
3. Rename CSV file with Leg. Ref - That makes it easier and faster to locate the correct csv file while uploading it to the server. If a jurisdiction has more than 20 documents, I create a batch file (.bat) to rename files. Example: "51_AL-AAC-335-3-4.xlsx - Sheet1.csv" renamed as "AL-AAC-335-3-4.csv"

Note: Excel to CSV conversion in local system does not handle UTF-8 characters properly so it is necessary to convert csv file from Google Drive.

CSV File Upload

1. Search a Leg Ref in LeBorg to locate a document.
2. Upload CSV file to Server (Scraped versions -> Import Clauses)
3. Add raw text (Scraped version -> Add reference version for updates)
4. Edit (Update in force date, Protected draft = Yes, Retrieved good content? = Yes)
5. Clauses (Change state as "In Use")
6. Request Clause Classification (assign document for classification)

If your concern is regarding "***Arizona: 10 docs uploaded on the Server 22-Dec-20 2:00***", definitely it is much more time than expected. All these 10 documents are large in size that take a little bit more time to process. Second and main cause of much time is document AZ-AAC-20-5. This document contains a huge appendix at the end. It is difficult and time consuming to process it in excel/csv file. Danni had suggested a special treatment to handle it directly in LeBorg. This document was the root cause of spending much time in the timesheet.

APIs #

The existing LeBorg API Documentation can be found here.

The existing Nemo API Documentation can be found here.
The existing NimonikApp API documentation can be found here.

Issues and gaps in current APIs:

Data storage #

Systems that store data should likely discuss how and in what rough form this happens. Similar to the advice on APIs, and for the same reasons, copy-pasting complete schema definitions should be avoided. Instead focus on the parts that are relevant to the design and its trade-offs.

Code and pseudo-code #

Design docs should rarely contain code, or pseudo-code except in situations where novel algorithms are described. As appropriate, link to prototypes that show the implementability of the design.

Degree of constraint #

Dataset Size

Current Nemo Process System (March 2021)

Ali Abbaszadeh Please insert overview of current AI Algorithms into this section.

The whole NLP algorithm is divided into two main sections, predicting unknown clause's class and training the model as is shown in figure below. Each of these sections includes pre-processing, and feature extraction steps. The model is created in the training section and then will be used for predicting the class of the input clause.

Once document is parsed and put in use, goes to Nemo for processing

Topics (Equipment only at this point)

On Topics list (Equipment only) and on actual content (parsed)

- Stemming
- Singularization

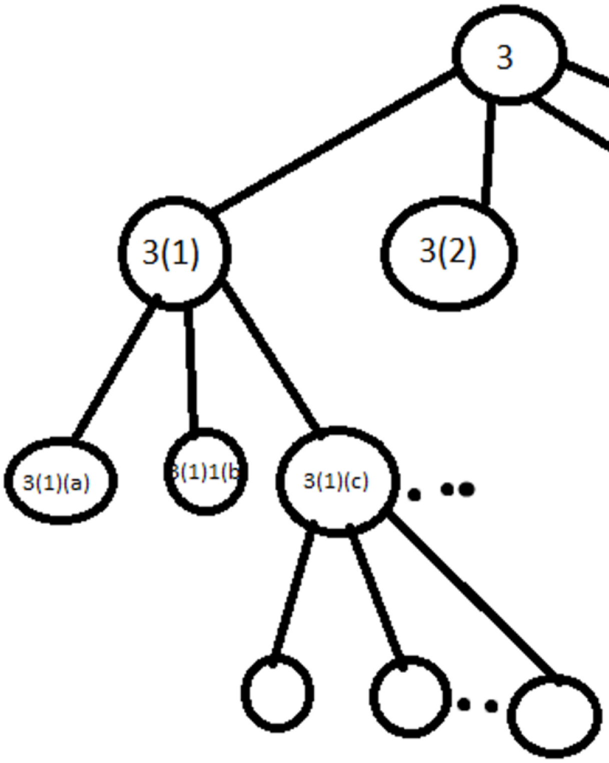
- Convert to lowercase
- Simple tokenization

We have hard coded certain Equipment terms (such as refuse) the confidence as 1 or 100. Where 1 = an equipment with high chance it is not accurate and 100 is high confidence it is the actual equipment. This is determined in advance by the CO cell. If no Equipment (Topic) is associated to a clause, Leborg assigns a confidence of 0.

Complete thoughts (CT)

Create CT using the section Ids and build them up using a hierarchy structure. The algorithm for creating CT is based on Tree Data Structure, to create CT a tree should be created first by all the clauses on the document then traverse through the nodes and edges to find the hierarchy between parents and children

.



The following examples illustrate this process:

Example 1

Clauses before applying CT:

“(1) Every employer must provide each employee with”

“(a) A respirator;”

“(b) Safety boots; and”

“(c) A hard hat.”

After applying CT:

“Every employer must provide each employee with A respirator;”

“Every employer must provide each employee with Safety boots; and”

“Every employer must provide each employee with A hard hat.”

Example 2

Clauses before applying CT:

> “(1) This section shall be as follows:”
>
> > “(i) One conductor of a single-phase, two-wire system shall be grounded;”
> >
> > “(ii) The neutral conductor of a single-phase, three-wire system shall be grounded;”
> >
> > “(iii) The neutral conductor of a multiphase system in which one phase is used as a neutral conductor shall be grounded.”

After applying CT:

> “This section shall be as follows: One conductor of a single-phase, two-wire system shall be grounded”
>
> “This section shall be as follows: The neutral conductor of a single-phase, three-wire system shall be grounded;”
>
> “This section shall be as follows: The neutral conductor of a multiphase system in which one phase is used as a neutral conductor shall be grounded.”

Clause Class

We are setting to Obligation or Not Set.

We start with pre-processing.

- Stemming
- Convert to lowercase
- Simple tokenization

Feature extraction

- More advanced tokenization

  - Remove stop words (things “at”, “the”, “on”,...) - words are not important
  - Cleaning lemmatizer (similar to stemming, but different)
- Feature extraction Word2Vec (set the space 100,000 - 100 columns, and 1000 rows)

Classification

- Current dataset - what is in it?
  - Only documents that were QCed by humans about two months ago
  - English and across all jurisdictions
- We use multi-class SVM (Support Vector Machine) due to the large because vectors are very large
  - We have modified this kernel function to obtain higher accuracy.

  One of the major advantages of SVMs over other methods is the ability of SVMs to use kernel functions to map data points to different feature spaces where separation may be easier to achieve. The general form of an SVM, once the optimization problem has been solved through the following equation

$$f(\bar{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\bar{x}_i, \bar{x}) + b$$

  For a linear SVM, the kernel is

$$K(\bar{x}_i, \bar{x}_j) = \bar{x}_i^T \bar{x}_j$$

  But here the following kernel has been used:

$$K(\bar{x}_i, \bar{x}_j) = e^{\frac{- \| \bar{x}_i - \bar{x}_j \|^2}{2\sigma^2}}$$

- This gives us the confidence and clause class
  - How does it calculate confidence, it will send obligation (60%) and non-obligation (40%).

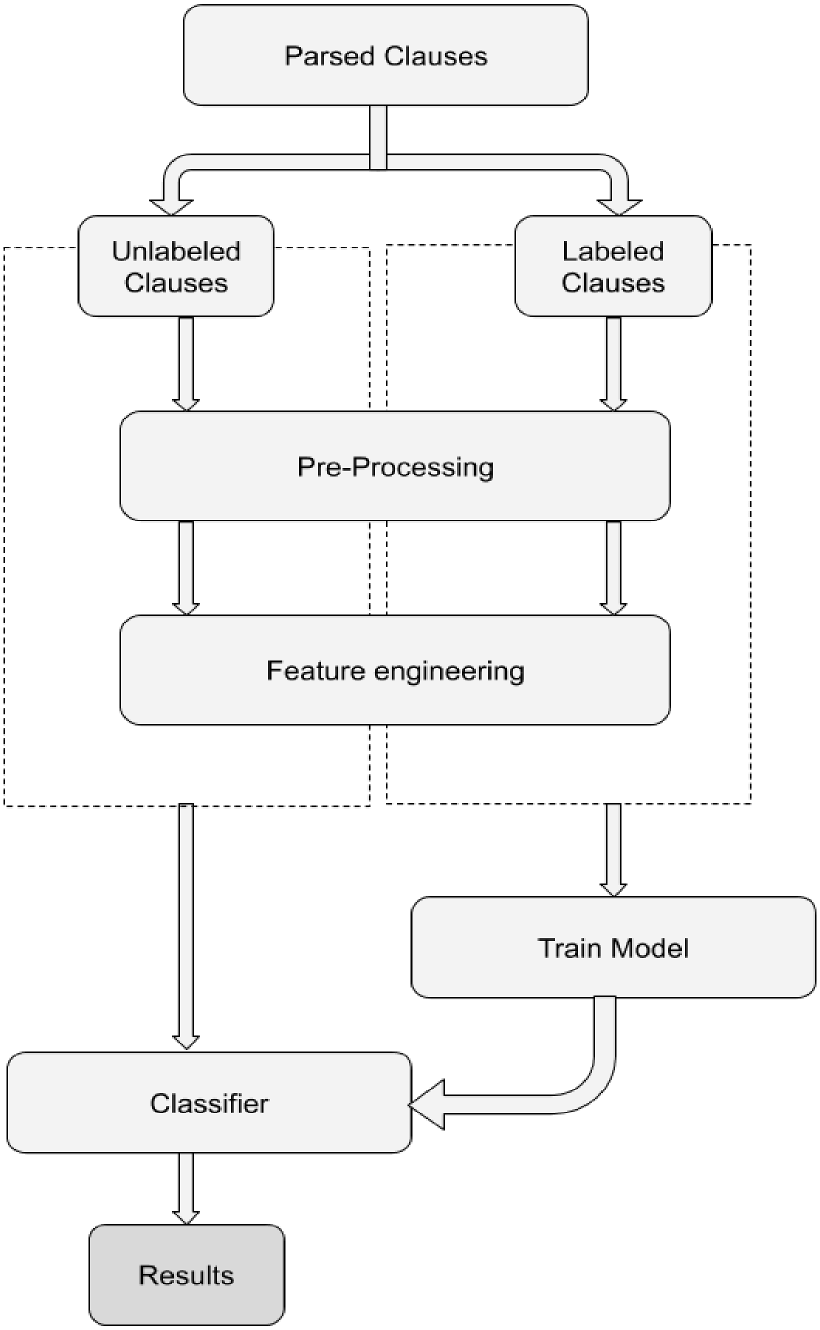
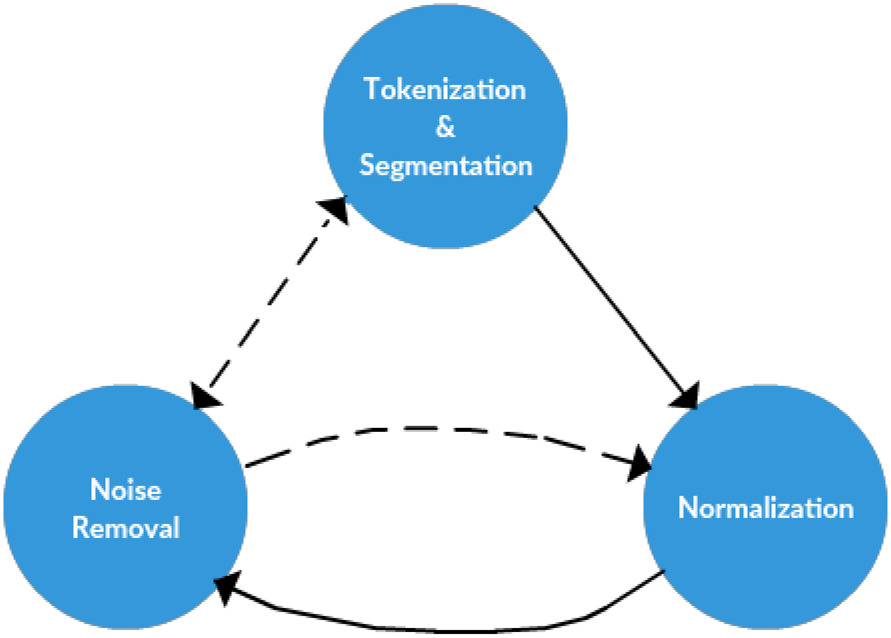

Parsed Clauses
Unlabeled Clauses
Labeled Clauses
Pre-Processing
Feature engineering
Train Model
Classifier
Results

Diagram of the system - Potential AI Algorithms

The following is a brief description

1. Text Preprocessing

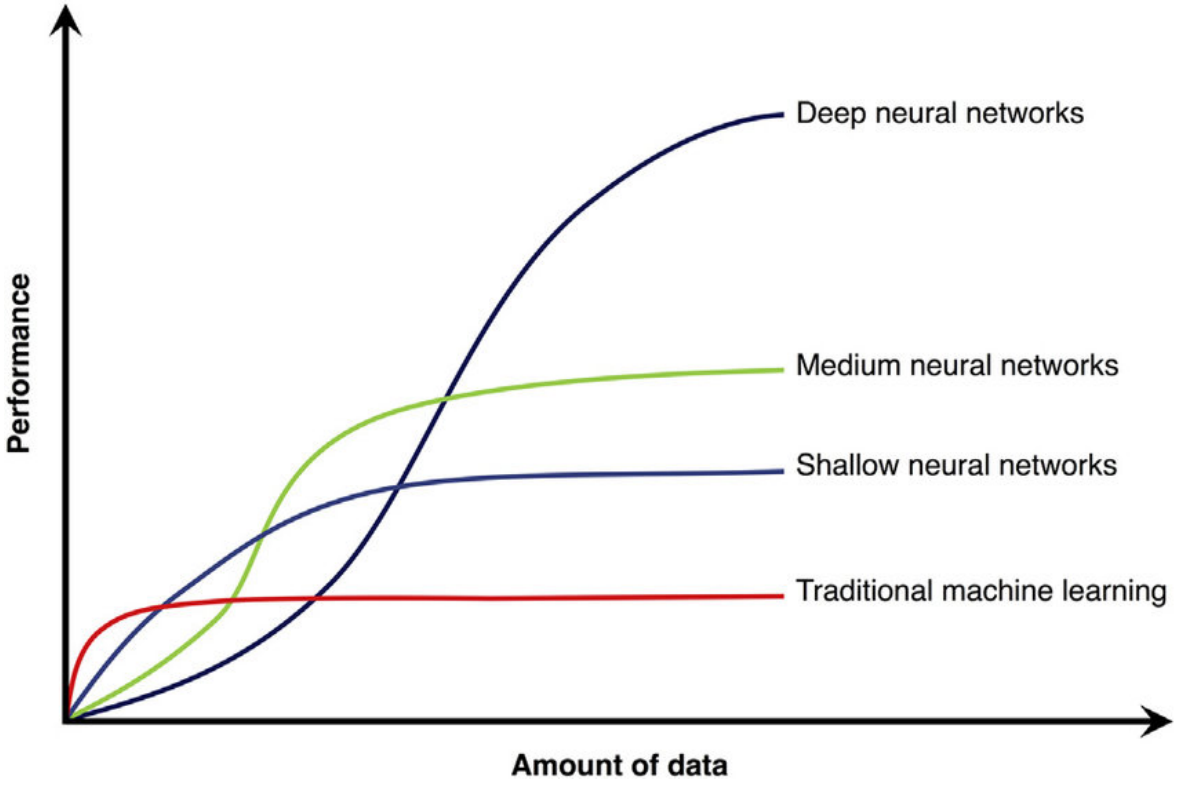
In the preprocess step to normalize the input text and get the text ready for further process the following framework is used.



2. Feature Extraction

For the feature extraction Word2Vec algorithm is used as this algorithm is a pre-training algorithm and can understand meaning of the words from the previous texts, it’s so fast and the feature vector has enough data for classification.

3. Classification
   The selected classifier is SVM as it is the best traditional machine learning based classifier for high-dimensional feature vectors and able to fairly handle imbalanced data.



Solutions to get better class confidence

Solutions

To get better class confidence or portion for classes with more than 65% confidence, there are 3 main solutions as follows:

1. Deep Learning(DL) based approaches
2. Ensemble classification
3. Post processing

Below is a brief comparison of these solutions. As of March 2021, we have decided to pursue Ensemble Classification (using SVM) and also Post-Processing.

Ensemble classification

Plan to implement Ensemble Classification

Ali - please insert your plan here.

Advantages

1. Can benefit of large amount of data with traditional machine learning
2. Somehow can handle imbalanced data
3. No need a high end infrastructure like deep learning
4. Will not increase the accuracy but can increase confidence

Downsides

1. Takes a lot of time to cleaning data
2. Need a time consuming and complex feature extraction process
3. Efficiency will decrease in compare with traditional machine learning

Minimum development time

6 months

Requirements

Dataset size: We need at least 300k classified (Qced) samples

Quality of data: Samples must be cleaned, part of that can be done with current automatically

Infrastructure: It can be done on current AI server

Post processing

Requirements for Post Processing Rules

Ali Abbaszadehplease add an outline of the structure of rules that the CO Cell should come back to you with.

Advantages

1. Independent from size of the data-set
2. Will not affect efficiency that much
3. We can design it to increase one of metrics (Recall/precision or confidence)
4. The required tools can be useful in other projects like topic assignment

Downsides

1. Takes a lot of time for building initial data
2. Need someone from CoC to work on building initial data
3. Can decrease accuracy

Development time

6-12 months

Requirements

Dataset size: Independent from size of the data-set

Quality of data: Samples have to be cleaned, with current automatic normalization we can not clean our data as we need. So cleaning is an issue in this approach and has to be solved at the beginning.

Infrastructure: It can be done on current AI server

Creating post-processing dataset: We need someone in COC to help us in this part

Alternatives considered

For the time being, (as of March 2021), we will not pursue Deep Learning due to the costs, data set and available resources.

Deep Learning

Advantages

1. DL can fairly manage domain diversity (Jurisdiction in our issue)
2. Deep Learning outperform other techniques if the data size is large (a few millions in our case)
3. Somehow can manage uncleaned data.
4. No need for feature engineering (Save a lot of time)

Downsides

1. High end infrastructure to train
2. Need large data set
3. Imbalanced data-set can decrease the performance

Minimum development time

12 months

Requirements

Dataset size:

- We need at least 500k classified (Qced) samples for small-size models and a few millions for bigger models!
- For initial training we need the above mentioned dataset + at least 10 millions parsed clauses (I need more investigation but maybe I can take care of this one).

Quality of data: Samples must be cleaned, part of that can be done with current automatically

Infrastructure:

- Instance type: c2-standard-30, GPU dies: 4 NVIDIA TESLA K80, Total Estimated Cost: USD 2,025.92 per 1 month
- For initial training: Instance type: c2-standard-30, GPU dies: 8 NVIDIA TESLA V100, Total Estimated Cost: USD 14,589.83 per 1 month

Bibliography

Archived Documents

Current and Future Workflow Diagram - Archived