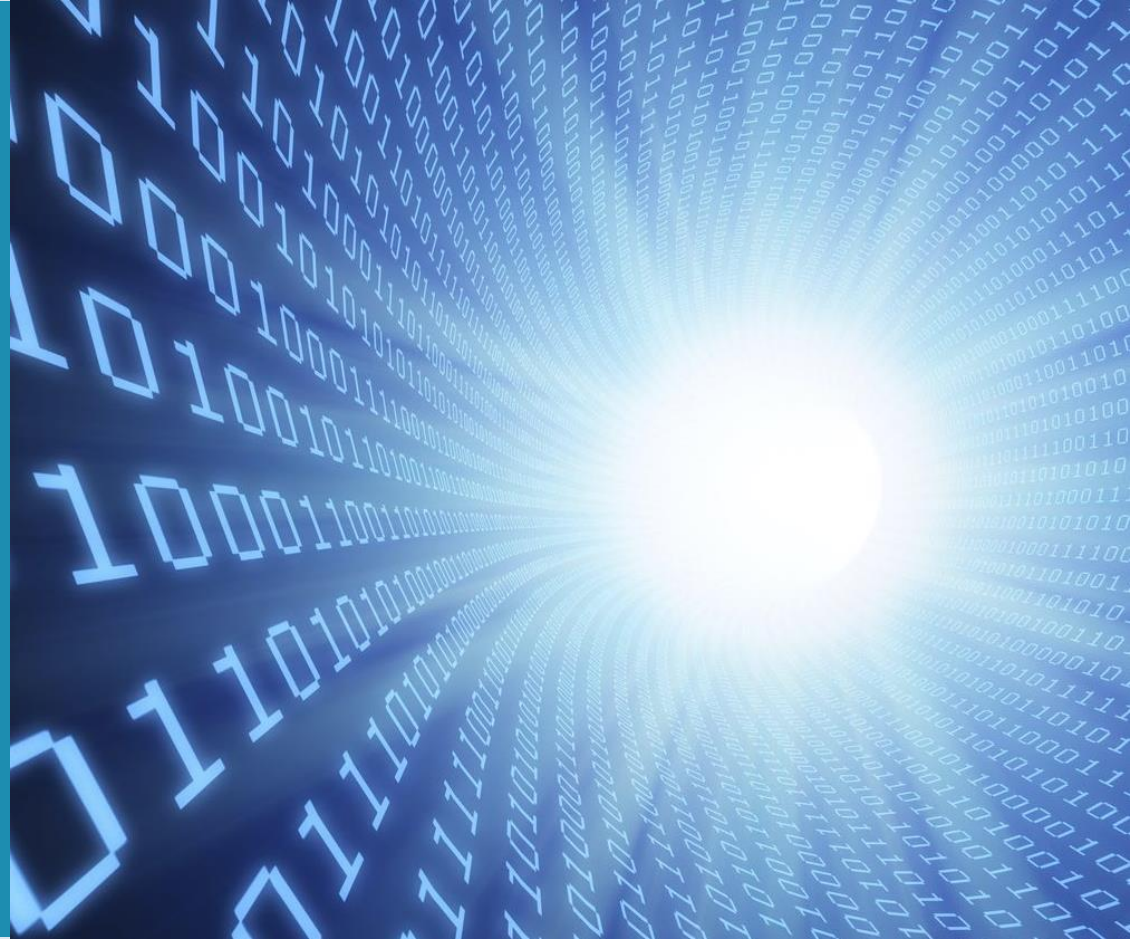


# Web scraping company information for official statistics:

Automatically finding company URLs and generating NACE codes



Ipek Senay, Hazem Eldabaa, Luis Miguel Lara Zanozain

# Official Statistics and Web Scraping



# Official Statistics and Web Scraping

- Survey non-response
  - Administrative data
- Digital data
  - Answering survey questions
  - Reduce the burden of response
- Survey data vs digital data



*Web scraping: automatic collection of data on the Internet*

# Official Statistics and Web Scraping

- Research questions
  - How the accurate URLs of companies active in Flanders can be automatically found using web scraping?
  - How can BERT/transformers model accurately classify "about us" webpages and acquire the accurate NACE codes of companies active in Flanders?

# Automated URL identification



# Automated URL identification

- Identifying company URLs using **Search engines** (Arora et al. 2021)
  - Bing search returns candidate URLs: set of 195 companies
    - 154 labelled: company-URL match 963 non-match
    - 5 ML models trained -> high accuracy scores
      - Linear SVM, RBF SVM, Gaussian processes, Decision tree, AdaBoost

# Automated URL identification

- Identifying company URLs using **Search engines** (Arora et al. 2021)

Table 4. Firm URL match prediction results.

Predictions received	Number of firms in prediction class	Retained matches	Precision	Average search result rank of proposed match
0	165	30	18.2%	1.55
1	106	71	67.0%	2.14
2	81	57	70.4%	1.30
3	65	65	100.0%	1.29
4	84	81	96.4%	1.06
5	986	984	99.8%	1.00
Total	1,487	1,288	86.6%	1.18

- Result: Researcher can significantly reduce the time spent (by more than 75%) on manually finding the correct URL
  - simply by checking URL matches identified by 2 or less models

# Classification of Webpages Using Automated Web-Scraping Methods





# Website Crawling and Classifying Webpages

- Crawling company websites through Python expressions
  - Crawl all pages with a depth of 1
  - Limited to homepage and about us pages
- Comparing the performance of 5 supervised ML methods (Linear SVM, RBF SVM, Decision Tree, AdaBoost, Multilayer Neural network)

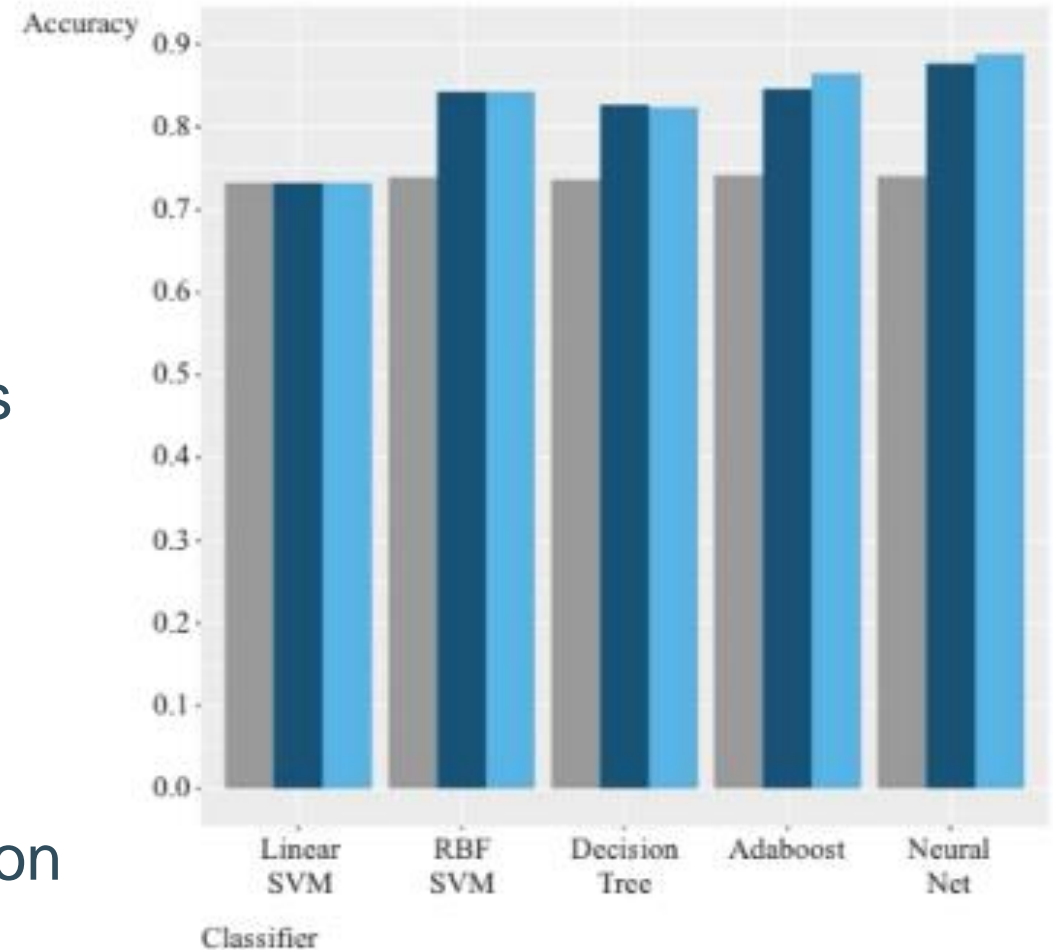


Fig. 3. Webpage classification model results for 'about us' pages.

# BERT

- Based on Transformers model developed by Google and the University of Toronto
  - Uses an encoder to transform input vector sequences and a decoder to generate desired output sequence
- Pre-Training
  - Model is trained on large unlabeled corpus
- Fine-tuning
  - Parameters of the trained model fine-tuned using labelled data
- BERT outperformed traditional machine learning models such as voting classifier, logistic regression, linear SVC, multinomial NB, ridge classifier, passive aggressive classifier

# NACE Codes

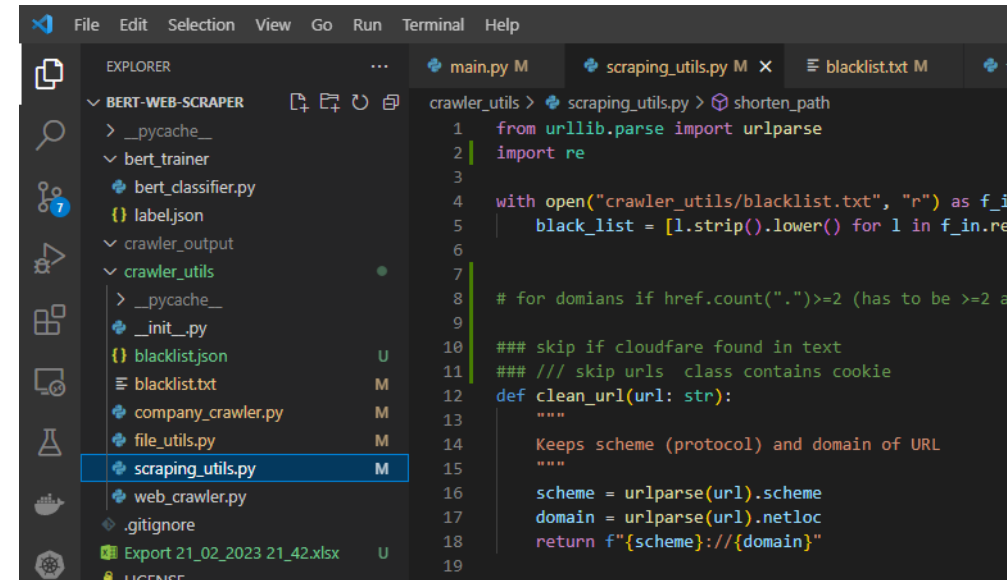
- When Belgian companies start their businesses they must register with Crossroads Bank for Enterprises (CBE, *Kruispuntbank van Ondernemingen* or *KBO in Dutch*) with their NACEBEL codes for their activities.
- It is the official European list of activity descriptions and is used for drawing up various statistics and overviews.

# Methodology



# Building the web scraper (data collection)

- We build our web scraper with the following requirements
  - Easy to develop our different modules and to deliver in a short period of time.
  - A programming language with best open-source libraries for all our needs (scraping, file management, ML, etc.)
- The obvious choice was **Python**.
- GitHub repository: <https://tinyurl.com/mtx9fbur>



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'BERT-WEB-SCRAPER' with folders like 'bert\_trainer', 'crawler\_output', and 'crawler\_utils'. The 'crawler\_utils' folder is expanded, showing files like 'scraping\_utils.py'. The code editor shows the following Python code:

```
1 from urllib.parse import urlparse
2 import re
3
4 with open("crawler_utils/blacklist.txt", "r") as f_in:
5     blacklist = [l.strip().lower() for l in f_in.readlines()]
6
7
8 # for domains if href.count(".")>2 (has to be >=2 at least)
9
10 ### skip if cloudflare found in text
11 ### /// skip urls class contains cookie
12 def clean_url(url: str):
13     """
14     Keeps scheme (protocol) and domain of URL
15     """
16     scheme = urlparse(url).scheme
17     domain = urlparse(url).netloc
18     return f"{scheme}://{domain}"
19
```

# Data gathering, labelling and preprocessing

- We started our initial scraping process with a sample of private companies
  - 1,000 companies with the status "Active" and based in the Flemish Region.
  - Source: Bureau van Dijk (Moody's Analytics) Orbis
- First results:
  - Over 7,000 URL candidates found for our initial sample of 1,000 companies.
  - Over 70 GB of HTML files.
  - We used this initial data lake to build our blacklist and used it to reduce it's size 4,952 URL candidates and 50 GB of HTML files.

	Company name Latin alphabet	Website	About us filename	About us content
1.	PFIZER SERVICE COMPANY			
2.	JANSSEN PHARMACEUTICA			
3.	EXXONMOBIL PETROLEUM & CHEMICAL			
4.	ETABLISSEMENTEN FRANZ COLRUYT NV			
5.	CARGILL NV			
6.	BASF ANTWERPEN			
7.	DELHAIZE LE LION / DE LEEUW			
8.	VOLVO CAR BELGIUM			

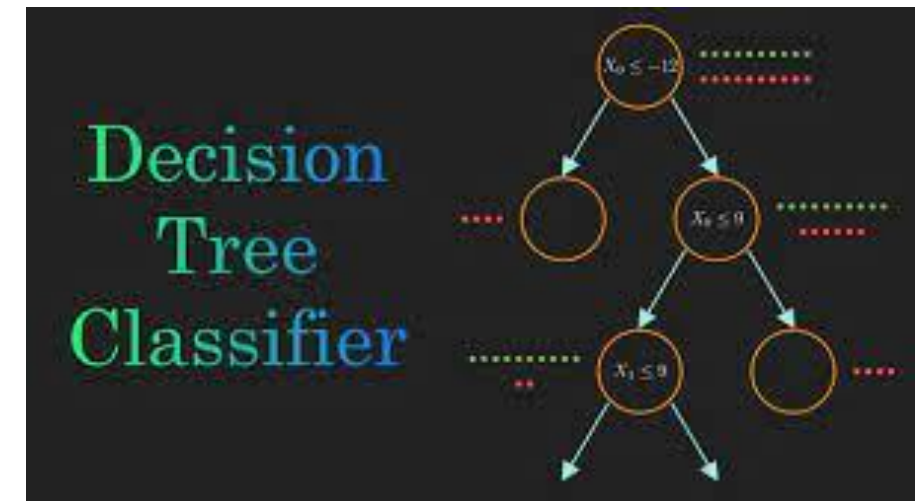
# Data gathering, labelling and preprocessing

- We manually labeled each result (URL) with the following values:
  - True URL (dependent variable): 1 if True, 0 if False.
  - URL and Company name similarity: Float between 0 a 1.
  - Domain region: 1 if .be or .eu, 0 for others.
  - URL containing the "about us" section.
- We developed processes with Python and re (regular expressions) library to extract the following data:
  - Company name in index text: 1 if True, 0 if False.
  - Belgian phone number in "about us" text: 1 if True, 0 if False.
  - Belgian VAT number in "about us" text: 1 if True, 0 if False.
  - Belgian physical address in "about us" text: 1 if True, 0 if False.



# Finding the right URL

- We used the Sci-kit learn library and our labeled data to train our Random Forests model (classifier).
- The goal of our model is to classify (True or False) a URL as the company.





# Obtaining the NACE code

- We used the PyTorch framework to fine tune our chosen pre-trained BERT model (bert-base-uncased)
- Our dataset was built with the information found in the "About us" section of the company's site.

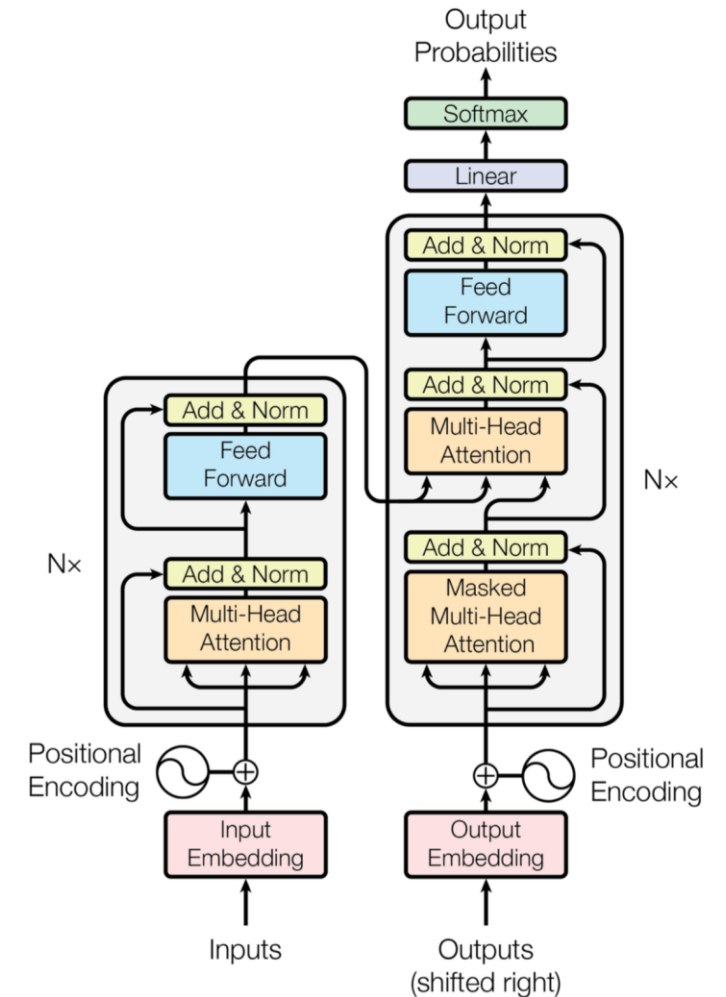
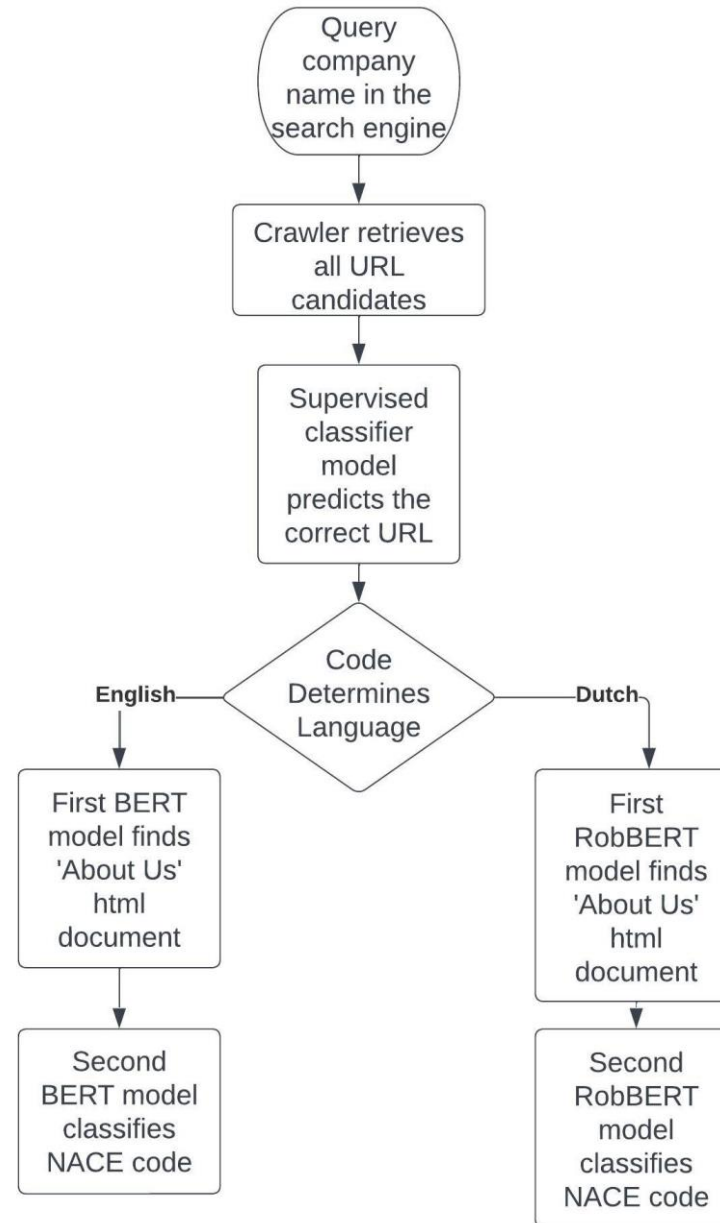


Figure 1: The Transformer - model architecture.

# Designing our solution



# Our solution's process

- Our scraper follows these steps:
  1. Goes to DuckDuckGo.com (our search engine)
  2. Changes the region to BE-NL
  3. Searches the company name
  4. Gathers all URL's (links and text)
  5. Applies a blacklist (removing URL's with Wikipedia, Facebook and other domains)
  6. Scrapes the index of each URL for contact details.
  7. Applies our Random Forests model to choose the right URL.
  8. Applies our algorithm to find the "About us" URL.
  9. Uses our BERT model to obtain the NACE code.



# Building our Solution

- We used the following technologies:
  - Python programming language.
  - Selenium 4 (for web automation)
  - Sci-kit learn (Decision Tree model training and testing)
  - PyTorch (BERT and RobBERT models fine-tuning)
  - Docker (containerizing our solution)
  - AWS Lambda (deploying our solution in parallel)



# Results



# NACE code generator

- Our experiment was conducted to compare the performance of transformers models with a more traditional machine learning model.

Model	Accuracy	Precision	Recall	F1-score
TF-IDF and Logistic Regression	0.11	0.05	0.11	0.05
BERT	0.62	0.62	0.6	0.61
RobBERT	0.61	0.61	0.57	0.58

# URL Finder

- Our experiment was conducted to compare the results after training a Decision Tree, Random Forests and SVM models.

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	0.82	0.82	0.65	0.43
Random Forests	0.82	0.82	0.65	0.49
Support Vector Machines	0.82	0.81	0.63	0.41

# Demo

