Classification challenge

Team: Please add your team name

Classification challenge
Team: TeamTITLE

Classification Approach Description

Methodology *(Replicability; Scalability; Interpretability)*

Our approach for this competition consisted of the following five main phases:

1. Extending and processing the ISCO taxonomy
2. Preprocessing the competition data
3. Embedding the processed documents
4. Nearest neighbor search
5. LLM re-ranking

Our code fully implementing all of the above steps is available on GitHub and under a permissive MIT license: https://github.com/acatovic/isco-job-classification. The full job classification pipeline can be run end-to-end by simply executing **python pipeline/run.py**. We also provide references to specific source files in the sections below.

1. Extending and processing the ISCO taxonomy

While exploring the ISCO taxonomy during EDA, we discovered the ESCO taxonomy, which extends the initial ISCO taxonomy, providing up to four additional levels of occupation nodes. Each ESCO occupation node is more specific than its ISCO ancestor node and also provides descriptions, representative skills, and common job titles. Here is an illustration showing how these two taxonomies relate to each other:

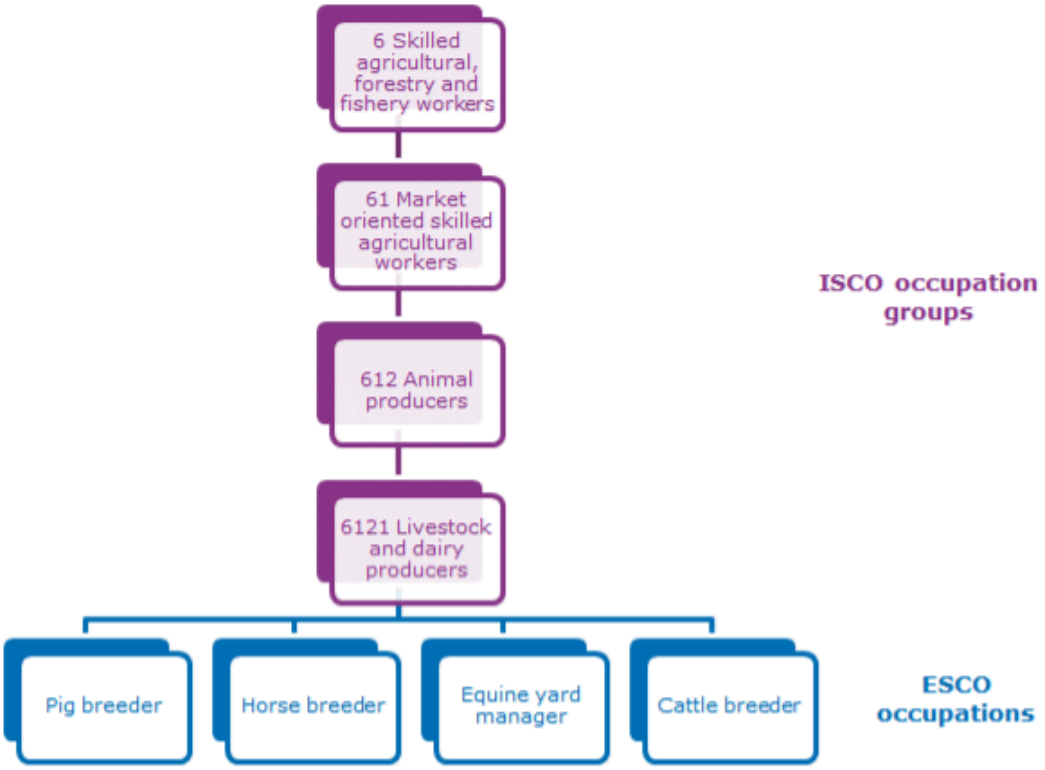


Figure 3: The structure of the occupations pillar

Because of the increased specificity and additional information available, we decided to use the leaf nodes (nodes without any children) of the ESCO taxonomy as targets for our nearest neighbor search. The leaf nodes were chosen because they represented the most specific form of each occupation. Data for both the ISCO and ESCO taxonomies were scraped from the ESCO website and saved as a JSON object using a separate script. See **data/get_data.py** in our repo.

During inspection, we noticed that the ESCO descriptions often included examples of occupations and skillsets that were in fact **not** appropriate to be classified to a given node. We worried the embedding model we used might have been unable to accurately handle this type of negation, so we removed this information. An example of this can be found below, and the removed part is highlighted in blue.

> Street and related services workers provide a variety of services on streets and in other public places, including cleaning shoes, washing car windows, running errands, handing out leaflets, looking after property, and providing other on-the-spot street services.
> Tasks include -
> (a) obtaining the materials necessary to perform services;
> (b) approaching people on the street to offer services;
> …
> (h) receiving immediate payment.
> Examples of the occupations classified here:
> - Car window washer
> - Car guard

- Errand boy
- Free newspaper distributor
- Leaflet distributor
- Shoe-polisher

Some related occupations classified elsewhere:
- Service station attendant - 5245
- Leaflet and newspaper deliverer - 9621

2. Preprocessing the competition data

Data preprocessing of the competition data consisted of the following steps:

1. Detect non-English ads and translate them to English. See **pipeline/translation.py** in our repo. This step was run overnight on the laptop described in the hardware section below.
2. Extract the key elements from the noisy, "raw" titles and descriptions, ignoring mentions of company culture, benefits, application process, etc. We instructed an LLM to provide a simplified job title, a distilled job description, and the skills necessary to perform the job. This step was run overnight on the laptop described in the hardware section below. See **pipeline/skills_extraction.py** in our repo

We undertook these steps with the assumption that a) any similarity-based methods would perform best if the text being embedded was more focused and less noisy and b) downstream processing using LLMs would be much faster using shorter texts. In fact, these extractions reduced the amount of text to process by an average of 80%!

Extracting the required skills from each job posting aligns us with available information from the ISCO/ESCO nodes and is in line with research indicating that skills are often more useful when classifying job families than titles alone.

Here is an example of the same job ad before and after processing. Note that this ad is synthetic; it does not appear in the competition dataset.

Before:

Location: Brussels, Belgium

About Us:
At EuroLegal Consulting, we are dedicated to shaping the future of legal and regulatory frameworks within the European Union. Our team of experts collaborates with governments, institutions, and private sector stakeholders to develop innovative policies that enhance the legal sector.

Position Overview:
EuroLegal Consulting is seeking a part-time Legal Policy Officer to join our dynamic team in Brussels. This role is crucial for researching, analyzing, and developing policies that aim to improve existing regulations within the legal sector. The successful candidate will collaborate closely with partners, external organizations, and stakeholders to provide them with regular updates and insights.

Key Responsibilities:
- Conduct in-depth research and analysis of legal policies.
- Develop and implement strategies for government policy implementation.
- Advise on legislative acts and provide expert legal advice.
- Analyze legal evidence and manage legal case documentation.

- Compile comprehensive legal documents and reports.

Required Skills and Qualifications:
- A degree in Legal Studies or a related field.
- Proven experience in government policy implementation.
- Strong analytical skills with the ability to analyze legal evidence.
- Proficiency in managing legal cases and compiling legal documents.
- Expertise in advising on legislative acts and providing legal advice.
- Familiarity with scientific research methodologies.
- Knowledge of competition law and European Structural and Investment Funds regulations.

What We Offer:
- A collaborative and inclusive work environment.
- Opportunities for professional development and growth.
- Competitive salary and benefits package.
- Flexible working hours to accommodate your schedule.

How to Apply:
Interested candidates are invited to send their resume and a cover letter outlining their qualifications and experience to careers@eurolegalconsulting.eu by [closing date]. Please include ""Legal Policy Officer Application"" in the subject line.

After:

```
{
   "job_title": "Legal Policy Officer",
   "job_description": "Conduct in-depth research and analysis of legal
       policies to develop and implement strategies for government policy
       implementation.",
   "job_skills": "legal policy analysis, government policy implementation,
       legislative act advice, legal case management, legal document
       compilation, scientific research methodology, competition law
       expertise, EU regulations knowledge, legal evidence analysis, policy
       development, legal advice provision, case documentation management,
       report compilation, research methodology application, policy
       implementation strategy, legal sector expertise, EU law knowledge,
       policy analysis, legal document management, research support, policy
       development support."
}
```

3. Embedding the processed documents

Executing the previous steps gave us two aligned datasets to work with. We now had job title, description, and required skills for both the ISCO/ESCO nodes and the job postings. We embedded both of these, treating the job postings as "queries", and the ISCO/ESCO node data as our search/reference set. We ran the embedding model on the laptop as described in the hardware section, which took approximately 15 minutes. See **pipeline/nn.py** in our repo.

4. Nearest Neighbor Search

Using the embeddings obtained above, we found the nearest 15 ISCO/ESCO leaf nodes for each competition job posting using a simple cosine similarity calculation. We took the ISCO “ancestor” code (ie, 1234.12.2 → 1234) of each nearest leaf node and added it to the candidate set for each job posting. This search was almost instantaneous. See **pipeline/nn.py** in our repo.

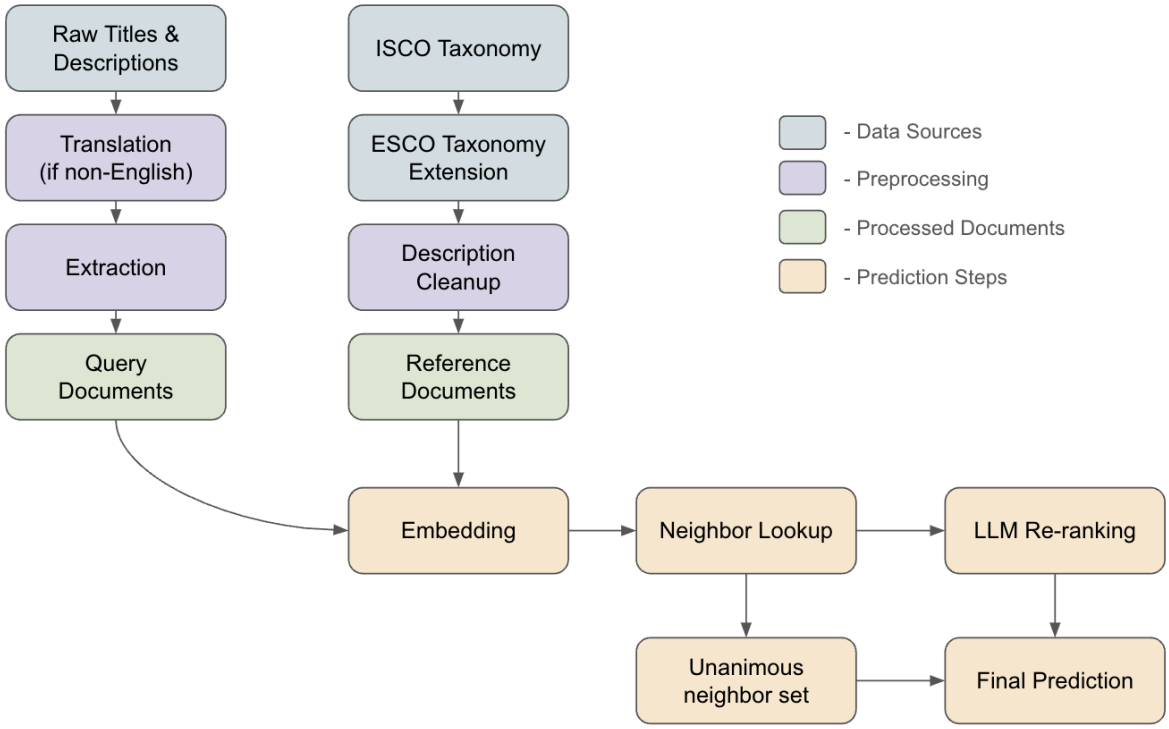
4. Final Prediction / LLM reranking

For candidate sets with unanimous agreement, the single code constituted our prediction for a given job posting. Where there was disagreement, we provided an LLM with the extracted job information

and the descriptions of the candidate leaf nodes and asked it to decide the closest one. This step was run overnight on the laptop described in the hardware section below. See **pipeline/reranking.py** in our repo.

Architecture

The architecture is described in the diagram below. This methodology can easily scale to larger datasets, see the "Contributions to scientific field" section for further discussion.



Hardware Specifications *(Replicability; Scalability; Interpretability)*

All the hardware used in this project is a mid-to-high end off-the-shelf hardware. We conducted our work on commercial grade Apple Macbook devices consisting of fully-integrated Apple Silicon (ARM-based M3 chipsets). Therefore no special data centre or server is needed. The exact specifications for the hardware used are listed below.

Machine 1 - Apple M3 Pro

CPUs	Apple M3 Pro Chip (12 CPU cores / Apple Silicon)
GPUs	Apple M3 Pro Chip - integrated GPU (18 GPU cores / Apple Silicon)
TPUs	N/A
Disk space	32 GiB - sufficient for loading all the models and data

Libraries *(Maintainability)*

During our project, we only adhered to open source permissive libraries. These packages are widely used royalty-free across numerous academic and commercial products. The full list of all libraries used and their purpose, is provided below.

- Numpy for array slicing and manipulation
- Sklearn's cosine_similarity function for nearest neighbor search
- Pandas dataframes for EDA and manipulation of intermediate results
- Langdetect open-source library for detecting non-English texts
- Sentence Transformers open-source library for loading sentence/document embedding models (e.g. Stella)
- Hugging Face's open models for
    - **translation, description extraction, reranking**: Llama 3.1 8B Instruct 8-bit (mlx-community/Meta-Llama-3.1-8B-Instruct-8bit)
    - **document embedding**: Stella English-only 400M Sentence Transformer (dunzhang/stella_en_400M_v5)
- NetworkX for taxonomy leaf node detection
- MLX-LM open source library from Apple, for executing LLMs (e.g. LLama) efficiently on Apple based silicon

Open license *(Maintainability)*

All the libraries used during our project were under liberal/permissive open source licenses, i.e. Apache, MIT and BSD. The libraries with their corresponding licenses are listed below.

Sentence Transformers (Apache 2.0)
MLX-LM (MIT)
Llama 3.1 (Permissive Llama 3.1 License)
Stella (MIT)
Langdetect (MIT)
Pandas (BSD-3)
Numpy (BSD)
Sklearn (BSD-3)
NetworkX (BSD-3)

Similarities/differences to State-of-the-Art techniques *(Originality)*

Document embedding and nearest neighbor search is a well-known methodology for this type of problem. The novelty of our approach comes from using LLMs to rerank ambiguous cases. More information about the utility of reranking as part of a nearest-neighbor search can be found here.

Contribution to scientific field *(Future orientation)*

Our approach is highly scalable and re-usable. We prioritized a modular solution and as such, each step can be iterated and improved upon in isolation. This is especially true if one has access to the classification label because the impact of any change can be directly measured. For example, smaller models can be evaluated for each step and decisions can be made around the tradeoff between accuracy vs cost of inference.

Additionally, our unsupervised embedding/nearest-neighbor/re-ranking approach has the advantage over approaches involving supervised models or rules-based heuristics in that retraining or rules updates are not necessarily required if the taxonomy changes.

In terms of modelling/algorithmic improvements, please see “Lessons Learned” in the next section where we have outlined those in some detail.

In terms of engineering / scaling improvements, we see several potential pathways. Since we did this work on Macbooks, and used Apple silicon (ARM) optimized libraries, i.e. MLX, the LLM based pipelines (translation, extraction and re-ranking) will not work on non Apple silicon devices. Therefore one improvement would be to check for hardware type, and choose either MLX for Apple or the Transformers module from HuggingFace for Intel based devices. The second potential improvement comes from massive parallel scaling. Here we were once again restricted to working on our laptops. Although we have modularized our pipelines, they are essentially sequential. If our approach is ported to a larger compute cluster with many CPUs or GPUs, it could benefit greatly from applying a map-reduce approach across translation, extraction, and re-ranking tasks.

Lessons Learned *(Future orientation)*

Our learnings and suggestions for future work are as follows:

Extraction Improvements:
Our extraction step focused on job title, job description, and required skills. However, since many similar occupations in the ISCO/ESCO taxonomy are ultimately differentiated by a) the seniority (ie manager vs individual contributor) or b) the industry they operate in, we feel there is opportunity to enrich these extractions further by including these items. This would theoretically make the neighbor search more accurate.

Leveraging Synthetic Data:
We toyed with having an LLM create synthetic job ads based on the ESCO/ISCO node information. The goal was to run the extraction and embedding steps on them and use the results for nearest neighbor search. These ads were highly convincing (in fact an example of can be seen in Step 2 above) and had the benefit that we knew precisely their labels.

Unfortunately, we struggled to match the distribution of tone, structure, and quality of the competition ads and after some disappointing submission scores, this method was abandoned. However, we maintain that this is a highly attractive avenue of future work. Truly representative synthetic data attached to known labels could vastly reduce and/or supplement manual labeling efforts while supporting a nearest neighbor workflow involving job postings to job postings (as opposed to ours involving job postings to taxonomy nodes).

Choosing the appropriate K for the Neighbor Search:
In retrospect, choosing k=15 for our node neighbor search may have been incorrect. This is because relatively few ISCO nodes have 15 leaf nodes beneath them. The result is that fewer unanimous neighbor node sets were predicted and more postings were passed to the re-ranker. Future work would be to select K based on evaluations against known labels.

Input data quality monitoring / human in the loop:
We observed a fair number of job postings that were raw HTML, loading error/redirect pages, or simply text from company recruitment landing pages. In these cases, no actual information about a job was available and as such, no accurate prediction could be made. We suggest two solutions for this issue (note these suggestions have less to do with our method and more with the classification task in general):

- An LLM-based preprocessing step that decides whether a given text actually contains a job to classify
- The ability to "decline to predict" if a given ad is too ambiguous. This task could be added directly to the re-ranking step. These cases could be surfaced to a human expert (human in the loop) to make a final judgement.

Short description of the Team – area of expertise

Andrew McCornack and Armin Catovic are Senior ML/AI Engineers working at a private equity firm in Stockholm, Sweden.

In fact, we worked on an extremely similar project at work in 2022… this was one of the reasons we were initially attracted to this competition!