Contents

Quick Start Guide - Observatorio de Demanda Laboral

Guía rápida para entender, configurar y ejecutar el sistema completo Autor: Nicolás Camacho & Alejandro Pinzón **Fecha:** Noviembre 2025

Tabla de Contenidos

Overview del Sistema

¿Qué es este proyecto?

Sistema completo de **observatorio de demanda laboral** para América Latina que: - **Scrapea** 30,660+ ofertas de 11 portales (Computrabajo, Bumeran, Magneto, etc.) - **Extrae** habilidades técnicas usando 3 pipelines (NER+Regex, TF-IDF, LLM) - **Normaliza** con taxonomía ESCO (14,174 skills) - **Clusteriza** semánticamente (UMAP+HDBSCAN) - **Analiza** tendencias temporales

Resultados Clave

Métrica	Pipeline A (NER)	Pipeline B (LLM)
F1-Score (post-ESCO)	72.15%	**84.26%**
Precision	66.28%	**89.25%**
Recall	79.17%	79.81%
Skills emergentes	-	**59.5%**
Velocidad	13s/job	42s/job

Conclusión: Pipeline B (LLM Gemma 3 4B) es superior en todas las métricas.

Stack Tecnológico

```
Frontend: CLI (orchestrator.py)

Scraping: Scrapy + BeautifulSoup + Selenium
NLP: spaCy (es_core_news_lg), Transformers
LLM: Gemma 3 4B Instruct (4.3GB)
Embeddings: multilingual-e5-base (768d)
Clustering: UMAP + HDBSCAN

Base de Datos: PostgreSQL 14 (labor_observatory)
```

```
Cache: Redis (opcional)

Lenguaje: Python 3.11
Paquetes: 47 dependencias (requirements.txt)
```

Arquitectura Técnica

Pipeline Completo (10 Componentes)

```
 SCRAPING          LIMPIEZA          EXTRACCIÓN
11 spiders           ETL/Junk            Pipeline A/B

           ESCO MATCHING             EMBEDDINGS
           3 layers (E/F/S)            768d E5

 CLUSTERING           ANÁLISIS
UMAP+HDBSCAN             TEMPORAL
```

Base de Datos (Tablas Principales)

```
-- 1. Ofertas crudas (scraping)
raw_jobs (43 columnas)
   job_id (UUID, PK)
   portal, country, url, title, company
   description, requirements
   content_hash (SHA256, deduplicación)
   tracking fields (is_processed, extraction_status, ...)

-- 2. Ofertas limpias (ETL)
cleaned_jobs
```

```
   job_id (FK → raw_jobs)
   title_cleaned, description_cleaned, requirements_cleaned
   combined_text (concatenado para extracción)
   combined_word_count, combined_char_count

-- 3. Skills extraídas (Pipeline A: NER+Regex)
extracted_skills
   skill_id (serial, PK)
   job_id (FK → raw_jobs)
   skill_text (ej: "Python")
   skill_type (hard/soft)
   extraction_method (ner/regex/tfidf)
   confidence_score
   esco_skill_uri (FK → esco_skills)

-- 4. Skills LLM (Pipeline B)
enhanced_skills
   skill_id (serial, PK)
   job_id (FK → raw_jobs)
   skill_text
   skill_type
   llm_model (ej: "gemma-3-4b-instruct")
   is_explicit (true/false)
   esco_skill_uri (FK → esco_skills)

-- 5. Taxonomía ESCO
esco_skills (14,174 skills)
   skill_uri (PK)
   preferred_label_es, preferred_label_en
   skill_type, skill_group
   is_active

-- 6. Gold Standard (evaluación)
gold_standard_annotations (7,848 anotaciones)
   annotation_id (serial, PK)
   job_id (FK → raw_jobs)
   skill_text
   skill_type (hard/soft)
   is_explicit (true/false - inferido o mencionado)
   context_snippet

-- 7. Embeddings (clustering)
skill_embeddings
   embedding_id (serial, PK)
   skill_text
   embedding (vector 768d)
   model_name ("multilingual-e5-base")
```

```
   created_at

-- 8. Clusters
skill_clusters
   cluster_id (serial, PK)
   cluster_label (ej: "Frontend Development")
   algorithm ("umap_hdbscan")
   n_skills (cantidad de skills en cluster)
   silhouette_score, davies_bouldin_score
```

Setup Inicial

1. Prerrequisitos

```
# Sistema operativo
- macOS 12+ (M1/M2) o Ubuntu 20.04+
- 32GB RAM recomendado (mínimo 16GB)
- 50GB espacio disponible

# Software
- Python 3.11+
- PostgreSQL 14+
- Git
```

2. Clonar Repositorio

```
cd ~/Documents/Tesis
git clone https://github.com/alejandro09pf/observatorio-demanda-laboral.git
cd observatorio-demanda-laboral
```

3. Activar Virtual Environment

```
# Crear venv (primera vez solamente)
python3.11 -m venv venv

# Activar venv (cada vez que abras terminal nueva)
source venv/bin/activate  # macOS/Linux
# o
.\venv\Scripts\activate   # Windows
```

```
# Verificar activación (debe mostrar "(venv)" en prompt)
which python
# Output: /Users/nicocamacho/Documents/Tesis/observatorio-demanda-laboral/venv/bin/python
```

4. Instalar Dependencias

```
# Actualizar pip
pip install --upgrade pip

# Instalar dependencias del proyecto (47 paquetes)
pip install -r requirements.txt

# Instalar modelo spaCy español (560MB)
python -m spacy download es_core_news_lg

# Verificar instalación
python -c "import spacy; nlp=spacy.load('es_core_news_lg'); print('  spaCy OK')"
```

5. Configurar Base de Datos

```
# Iniciar PostgreSQL
brew services start postgresql@14  # macOS
# o
sudo systemctl start postgresql    # Linux

# Crear base de datos
createdb labor_observatory

# Crear usuario
psql -d labor_observatory -c "CREATE USER labor_user WITH PASSWORD '123456';"
psql -d labor_observatory -c "GRANT ALL PRIVILEGES ON DATABASE labor_observatory TO labor_user
```

6. Configurar Variables de Entorno

```
# Crear archivo .env en raíz del proyecto
cat > .env << EOF
# Database
DATABASE_URL=postgresql://labor_user:123456@localhost:5432/labor_observatory

# LLM Models
HF_HOME=/Users/nicocamacho/.cache/huggingface
TRANSFORMERS_CACHE=/Users/nicocamacho/.cache/huggingface
```

```
# Scraping
SCRAPY_SETTINGS_MODULE=src.scraper.settings
USER_AGENT_LIST=Mozilla/5.0,Chrome/91.0,Safari/14.0

# Embeddings
EMBEDDING_MODEL=intfloat/multilingual-e5-base
EMBEDDING_DIM=768

# Clustering
UMAP_N_NEIGHBORS=15
UMAP_MIN_DIST=0.1
HDBSCAN_MIN_CLUSTER_SIZE=12
EOF

# Cargar variables
export $(cat .env | xargs)
```

7. Ejecutar Migraciones

```
# Crear esquema de base de datos
python scripts/setup_database.py

# Importar taxonomía ESCO (14,174 skills)
python scripts/import_real_esco.py

# Verificar
psql -d labor_observatory -c "SELECT COUNT(*) FROM esco_skills;"
# Output: 14174
```

8. Verificar Instalación

```
# Test de conexión a base de datos
python scripts/test_database_connection.py

# Test de orquestador
python -m src.orchestrator status

# Output esperado:
#   Database: Connected (labor_observatory)
#   ESCO Skills: 14,174 loaded
#   Raw Jobs: 30,660
#   Cleaned Jobs: 30,660
#   Gold Standard: 300 jobs, 7,848 annotations
```

Comandos del Orquestador

El orquestador (`src/orchestrator.py`) es la **CLI unificada** para ejecutar TODO el sistema.

Sintaxis General

```
# Activar venv primero
source venv/bin/activate

# Sintaxis
python -m src.orchestrator <comando> [opciones]

# Ver ayuda
python -m src.orchestrator --help
```

1. Estado del Sistema

```
# Ver estado general
python -m src.orchestrator status

# Output:
#
#    OBSERVATORIO DE DEMANDA LABORAL - ESTADO DEL SISTEMA
#
#
#   BASE DE DATOS
#    Conexión:   labor_observatory
#    Raw Jobs: 30,660 (usable: 30,660)
#    Cleaned Jobs: 30,660
#    ESCO Skills: 14,174
#    Gold Standard: 300 jobs, 7,848 annotations
#
#   PIPELINES
#    Pipeline A (NER+Regex): 300 jobs procesados
#    Pipeline B (LLM): 298 jobs procesados (2 errores)
#    Extracted Skills: 8,301 únicas
#
#   EMBEDDINGS
#    ESCO Skills: 14,174 embeddings (768d)
#    Extracted Skills: 8,301 embeddings
#    Modelo: multilingual-e5-base
#
```

```
#   CLUSTERING
#    Configuraciones: 8 finales
#    Clusters: 53 (Pipeline A 30k post)
#    Visualizaciones: 36 PNG generadas
```

2. Scraping

```
# Listar spiders disponibles
python -m src.orchestrator list-spiders

# Output:
# Available spiders:
#    computrabajo (CO, MX, AR)
#    bumeran (CO, MX, AR, CL, PE, EC, PA, UY)
#    elempleo (CO, MX, AR)
#    magneto (CO, MX, AR)
#    zonajobs (AR, CO, MX)
#    occmundial (MX, CO, AR)
#    clarin (AR, CO, MX)
#    infojobs (CO, MX, AR)
#    hiring_cafe (CO, MX, CL, AR)
#    lego (CO, MX, AR)
#    indeed (MX)

# Scraping de un spider (1 página)
python -m src.orchestrator run-once computrabajo --country CO

# Scraping de un spider (10 páginas)
python -m src.orchestrator run-once computrabajo --country CO --max-pages 10

# Scraping múltiple (3 spiders en secuencia)
python -m src.orchestrator run computrabajo,bumeran,elempleo --country MX --max-pages 5

# Output esperado:
#    Starting spider: computrabajo
#    Country: CO
#    Max pages: 10
#    Mode: programmatic
#
# [2025-11-11 10:30:00] Spider started
# [2025-11-11 10:32:15] Scraped 243 jobs
# [2025-11-11 10:32:15] Inserted: 231 | Duplicates: 12 (4.9%)
#   Spider completed in 2m 15s
```

3. Limpieza de Datos (ETL)

```
# Limpiar todos los jobs pendientes
python -m src.orchestrator clean

# Limpiar con batch específico
python -m src.orchestrator clean --batch-size 2000

# Limpiar portal específico
python -m src.orchestrator clean --portal computrabajo --country CO

# Output:
#  CLEANING RAW JOBS
#   Batch size: 1000
#   Portal: all
#   Country: all
#
# [Batch 1/31] Processing 1000 jobs...
#     Junk detected: 23 (2.3%)
#     Cleaned: 977
#     Time: 45s
#
# [Batch 2/31] Processing 1000 jobs...
#     Junk detected: 18 (1.8%)
#     Cleaned: 982
#     Time: 43s
#
# ...
#
#  CLEANING COMPLETE
#   Total jobs: 30,660
#   Cleaned: 30,203 (98.5%)
#   Junk: 457 (1.5%)
#   Total time: 23m 12s
```

4. Extracción de Skills (Pipeline A)

```
# Procesar jobs con Pipeline A (NER+Regex)
python -m src.orchestrator process-pipeline-a

# Procesar gold standard específicamente
python -m src.orchestrator process-pipeline-a --gold-standard-only

# Procesar lista de job IDs
python -m src.orchestrator process-pipeline-a --job-ids job1,job2,job3
```

```
# Output:
#  PIPELINE A: NER + REGEX + ESCO MATCHING
#   Mode: gold_standard
#   Jobs to process: 300
#   Model: es_core_news_lg
#
# [1/300] Processing job abc123...
#     Title: "Backend Engineer Senior"
#     NER entities: 12
#     Regex matches: 8
#     TF-IDF phrases: 5
#     Total extracted: 18 skills
#     ESCO matched: 14 (77.8%)
#     Time: 13s
#
# [2/300] Processing job def456...
#     Title: "Full Stack Developer"
#     NER entities: 15
#     Regex matches: 11
#     TF-IDF phrases: 6
#     Total extracted: 23 skills
#     ESCO matched: 18 (78.3%)
#     Time: 15s
#
# ...
#
#  PIPELINE A COMPLETE
#   Jobs processed: 300/300 (100%)
#   Skills extracted: 2,633
#   Unique skills: 487
#   ESCO coverage: 78.1%
#   Total time: 1h 5m
```

5. Extracción de Skills (Pipeline B - LLM)

```
# Listar modelos LLM disponibles
python -m src.orchestrator llm-list-models

# Output:
# Available LLM Models:
#   gemma-3-1b-instruct (1.2GB)   downloaded
#   gemma-3-4b-instruct (4.3GB)   downloaded
#   llama-3.2-3b-instruct (3.4GB)   downloaded
#   qwen2.5-3b-instruct (3.3GB)   downloaded
#   qwen2.5-7b-instruct (6.5GB)   not downloaded
```

```
#   phi-3.5-mini (3.8GB)   downloaded
#   mistral-7b (7.2GB)   not downloaded

# Descargar modelos faltantes
python -m src.orchestrator llm-download-models --all

# Procesar con Pipeline B (gold standard)
python -m src.orchestrator process-gold-standard --model gemma-3-4b-instruct

# Procesar batch de 10 jobs
python -m src.orchestrator llm-process-jobs --batch-size 10 --model gemma-3-4b-instruct

# Output:
#  PIPELINE B: LLM EXTRACTION
#   Model: gemma-3-4b-instruct (4.3GB)
#   Mode: gold_standard
#   Jobs to process: 300
#   Temperature: 0.3
#
# [1/300] Processing job abc123...
#     Title: "Backend Engineer Senior"
#     Combined text: 1,247 chars
#     LLM inference time: 38s
#     Hard skills extracted: 15
#     Soft skills extracted: 5
#     ESCO matched: 8 (40.0%)
#     Emergent skills: 12 (60.0%)
#     Total time: 42s
#
# [2/300] Processing job def456...
#     Title: "Full Stack Developer"
#     Combined text: 892 chars
#     LLM inference time: 35s
#     Hard skills extracted: 18
#     Soft skills extracted: 4
#     ESCO matched: 7 (31.8%)
#     Emergent skills: 15 (68.2%)
#     Total time: 39s
#
# ...
#
#  PIPELINE B COMPLETE
#   Jobs processed: 298/300 (99.3%)
#   Jobs failed: 2 (0.7%) - mode collapse
#   Skills extracted: 8,301
#   Unique skills: 2,847
#   ESCO coverage: 40.5%
```

```
#   Emergent skills: 59.5%
#   Total time: 3h 28m
```

6. Comparación de Modelos LLM

```
# Comparar 4 modelos en 10 jobs
python -m src.orchestrator llm-compare-models --sample-size 10

# Output:
#  LLM MODEL COMPARISON
#   Models: gemma-3-4b, llama-3.2-3b, qwen2.5-3b, phi-3.5-mini
#   Sample: 10 jobs (randomly selected)
#   Metrics: hard_skills, soft_skills, emergent_skills, hallucinations
#
# [Model 1/4] gemma-3-4b-instruct
#     Jobs processed: 10/10 (100%)
#     Avg hard skills: 23.4
#     Avg soft skills: 8.1
#     Emergent: 80.6%
#     Hallucinations: 0
#     Avg time: 42s/job
#
# [Model 2/4] llama-3.2-3b-instruct
#     Jobs processed: 10/10 (100%)
#     Avg hard skills: 34.2
#     Avg soft skills: 0.0
#     Emergent: 26.5%
#     Hallucinations: 7
#     Avg time: 15s/job
#
# ...
#
#  WINNER: gemma-3-4b-instruct
#   Reason: Zero hallucinations, balanced hard/soft, high emergent detection
#   Recommendation: Use for production
```

7. Generación de Embeddings

```
# Generar embeddings de ESCO skills
python -m src.orchestrator generate-embeddings

# Generar embeddings de skills extraídas
python -m src.orchestrator generate-extracted-embeddings
```

```
# Construir índice FAISS (para búsqueda rápida)
python -m src.orchestrator build-faiss-index

# Testear embeddings
python -m src.orchestrator test-embeddings --verbose

# Output:
#  GENERATING EMBEDDINGS
#   Model: intfloat/multilingual-e5-base
#   Dimension: 768
#   Skills to embed: 14,174 (ESCO)
#   Batch size: 32
#
# [Batch 1/443] Embedding skills 1-32...
#     Time: 2.3s
#
# [Batch 2/443] Embedding skills 33-64...
#     Time: 2.1s
#
# ...
#
#  EMBEDDINGS COMPLETE
#   Total embeddings: 14,174
#   Dimension: 768
#   Storage: 85.2MB
#   Total time: 16m 42s
#
#  BUILDING FAISS INDEX
#   Index type: IndexFlatIP (inner product)
#   Embeddings: 14,174
#   Time: 12s
#
#  FAISS INDEX BUILT
#   File: data/embeddings/esco.faiss
#   Size: 43.7MB
#   Search speed: ~0.5ms/query
```

8. Clustering

```
# Listar configuraciones de clustering disponibles
ls configs/clustering/final/

# Output:
# manual_300_post.json
# manual_300_pre.json
```

```
# pipeline_a_300_post.json
# pipeline_a_300_pre.json
# pipeline_a_30k_post.json
# pipeline_a_30k_pre.json
# pipeline_b_300_post.json
# pipeline_b_300_pre.json

# Ejecutar clustering con configuración específica
python -m src.orchestrator cluster pipeline_b_300_post

# Ejecutar todas las configuraciones finales
python -m src.orchestrator cluster-all-final

# Output (ejemplo):
#   CLUSTERING: pipeline_b_300_post
#    Skills: 1,937
#    Embeddings: 768d
#    Algorithm: UMAP + HDBSCAN
#    Config: configs/clustering/final/pipeline_b_300_post.json
#
# [1/4] Generating embeddings...
#      Time: 3m 12s
#
# [2/4] UMAP dimensionality reduction...
#      n_neighbors: 15
#      min_dist: 0.1
#      metric: cosine
#      Time: 45s
#
# [3/4] HDBSCAN clustering...
#      min_cluster_size: 12
#      min_samples: 3
#      Time: 23s
#
# [4/4] Calculating metrics...
#      Clusters: 50
#      Noise: 319 skills (16.5%)
#      Silhouette: 0.348
#      Davies-Bouldin: 0.687
#      Time: 8s
#
#   CLUSTERING COMPLETE
#    Output: outputs/clustering/final/pipeline_b_300_post/
#    Visualizations: 3 PNG files
#    Metrics: metrics_summary.json
#    Total time: 4m 28s
```

9. Análisis Temporal

```
# Análisis temporal de skills por trimestre
python scripts/temporal_clustering_analysis.py

# Output:
#   TEMPORAL ANALYSIS
#    Period: 2015 Q1 - 2025 Q4
#    Quarters: 44
#    Skills tracked: 8,301
#    Output: outputs/temporal/
#
# [Q1 2015] Analyzing skills...
#      Jobs: 87
#      Skills: 234
#      Top 5: Python, Java, SQL, JavaScript, HTML
#
# [Q2 2015] Analyzing skills...
#      Jobs: 103
#      Skills: 287
#      Top 5: Python, Java, JavaScript, SQL, React
#
# ...
#
# [Q4 2025] Analyzing skills...
#      Jobs: 1,456
#      Skills: 2,847
#      Top 5: Python, React, AWS, Docker, Kubernetes
#
#   TRENDS DETECTED
#    Emerging (growth >20%):
#      Docker: +127% (Q1 2020 → Q4 2025)
#      Kubernetes: +215% (Q1 2020 → Q4 2025)
#      AWS: +89% (Q1 2020 → Q4 2025)
#      React: +64% (Q1 2020 → Q4 2025)
#
#    Declining (drop >20%):
#      AngularJS: -78% (Q1 2020 → Q4 2025)
#      jQuery: -45% (Q1 2020 → Q4 2025)
#      Flash: -92% (Q1 2020 → Q4 2025)
#
#    Stable (change <20%):
#      Python: +12%
#      Java: -8%
#      SQL: +5%
#
#   TEMPORAL ANALYSIS COMPLETE
```

```
#   Heatmaps: 6 PNG files
#   CSV exports: 44 files (1 per quarter)
#   Total time: 12m 34s
```

10. Evaluación de Pipelines

```
# Evaluar Pipeline A vs Pipeline B en gold standard
python scripts/evaluate_pipelines_dual.py

# Output:
#  DUAL EVALUATION: Pipeline A vs Pipeline B
#   Gold Standard: 300 jobs, 7,848 annotations
#   Pipelines: 2 (Pipeline A, Pipeline B)
#   Evaluation Levels: Pre-ESCO, Post-ESCO
#
#
# LEVEL 1: PRE-ESCO (Pure Extraction)
#
#
# Pipeline A (NER+Regex):
#   Precision: 20.66%
#   Recall: 25.20%
#   F1-Score: 22.70%
#   Predicted: 2,633 skills
#   Support: 2,159 skills (gold standard)
#
# Pipeline B (LLM Gemma):
#   Precision: 38.94%
#   Recall: 55.82%
#   F1-Score: 46.23%
#   Predicted: 8,301 skills
#   Support: 7,848 skills (gold standard)
#
#  Winner (Pre-ESCO): Pipeline B (+23.53pp F1)
#
#
# LEVEL 2: POST-ESCO (Normalized)
#
#
# Pipeline A (NER+Regex):
#   Precision: 66.28%
#   Recall: 79.17%
#   F1-Score: 72.15%
#   ESCO coverage: 10.52%
#   Skills lost in mapping: 2,356 (89.5%)
```

```
#
# Pipeline B (LLM Gemma):
#   Precision: 89.25%
#   Recall: 79.81%
#   F1-Score: 84.26%
#   ESCO coverage: 11.30%
#   Skills lost in mapping: 4,945 (59.5%)
#
#  Winner (Post-ESCO): Pipeline B (+12.11pp F1)
#
#
# EMERGENT SKILLS ANALYSIS
#
#
# Pipeline B detected 4,945 emergent skills (59.5%):
#
# Top 20 Emergent Skills:
# 1. SAM (AWS Serverless Application Model)
# 2. CDK (Cloud Development Kit)
# 3. SST (Serverless Stack)
# 4. React Hooks
# 5. Kubernetes CRDs
# 6. Terraform CDK
# 7. Pulumi
# 8. Deno
# 9. Bun
# 10. Next.js App Router
# ...
#
#  EVALUATION COMPLETE
#   Report: data/reports/EVALUATION_REPORT_20251111_103045.md
#   CSV: data/reports/comparison_20251111_103045.csv
#   Total time: 8m 12s
```

Flujos Completos

Flujo 1: Nuevo Scraping → Evaluación

```
# 1. Activar venv
source venv/bin/activate

# 2. Scrapear nuevas ofertas
python -m src.orchestrator run-once computrabajo --country CO --max-pages 10
```

```
# 3. Limpiar datos
python -m src.orchestrator clean

# 4. Procesar con Pipeline B
python -m src.orchestrator llm-process-jobs --batch-size 243 --model gemma-3-4b-instruct

# 5. Generar embeddings
python -m src.orchestrator generate-extracted-embeddings

# 6. Ejecutar clustering
python -m src.orchestrator cluster pipeline_b_300_post

# Total tiempo estimado: ~3 horas
```

Flujo 2: Comparar Nuevo Modelo LLM

```
# 1. Descargar modelo nuevo
python -m src.orchestrator llm-download-models --model llama-3.3-70b-instruct

# 2. Comparar con modelos existentes
python -m src.orchestrator llm-compare-models --sample-size 50

# 3. Si es superior, procesar gold standard completo
python -m src.orchestrator process-gold-standard --model llama-3.3-70b-instruct

# 4. Evaluar contra Pipeline B actual
python scripts/evaluate_pipelines_dual.py --models gemma-3-4b,llama-3.3-70b

# Total tiempo estimado: ~6 horas
```

Flujo 3: Análisis Completo para Paper

```
# 1. Estado del sistema
python -m src.orchestrator status > report_status.txt

# 2. Evaluación dual
python scripts/evaluate_pipelines_dual.py

# 3. Clustering todas las configs
python -m src.orchestrator cluster-all-final

# 4. Análisis temporal
python scripts/temporal_clustering_analysis.py
```

```
# 5. Generar visualizaciones
python scripts/regenerate_visualizations.py

# Total tiempo estimado: ~2 horas
```

Componentes Técnicos Detallados

Pipeline A: NER + Regex + TF-IDF

```
# src/extractor/ner_extractor.py

import spacy

# Cargar modelo español grande (560MB)
nlp = spacy.load("es_core_news_lg")

def extract_ner_entities(text: str) -> List[str]:
    """Extrae entidades nombradas tecnológicas."""
    doc = nlp(text)

    skills = []
    for ent in doc.ents:
        # Filtrar solo entidades tecnológicas
        if ent.label_ in ['MISC', 'ORG', 'PRODUCT']:
            if is_technology_entity(ent.text):
                skills.append(ent.text)

    return skills

def is_technology_entity(text: str) -> bool:
    """Valida si entidad es tecnológica."""
    # Diccionario de 847 tecnologías conocidas
    TECH_KEYWORDS = {
        'Python', 'Java', 'JavaScript', 'TypeScript', 'Go', 'Rust',
        'React', 'Angular', 'Vue', 'Django', 'Flask', 'Spring',
        'AWS', 'Azure', 'GCP', 'Docker', 'Kubernetes',
        'PostgreSQL', 'MySQL', 'MongoDB', 'Redis',
        # ... 847 términos totales
    }

    # Búsqueda case-insensitive
    return text.lower() in {k.lower() for k in TECH_KEYWORDS}
```

1. Modelo NER (spaCy) Diccionario Tecnológico: 847 términos

Categorías: - **Lenguajes** (89): Python, Java, JavaScript, TypeScript, Go, Rust, C, C++, C#, Swift, Kotlin, Scala, R, MATLAB, Perl, Ruby, PHP, Elixir, Erlang, Haskell, OCaml, F#, Clojure, Groovy, Dart, Julia, Lua, Shell, Bash, PowerShell, SQL, PL/SQL, T-SQL, NoSQL, GraphQL, SPARQL, XQuery, Prolog, Lisp, Scheme, Racket, APL, J, K, Q, COBOL, Fortran, Ada, Pascal, Delphi, Visual Basic, VB.NET, Assembly, WebAssembly, Solidity, Move, Cairo, Vyper, Yul, WASM, Rust, Zig, Nim, Crystal, Pony, V, Odin, Carbon, Mojo, Jai, ReScript, Gleam, Roc, Unison, Elm, PureScript, Idris, Agda, Coq, Lean

- **Frameworks Web** (127): React, Angular, Vue.js, Svelte, Next.js, Nuxt.js, Gatsby, Remix, SvelteKit, Qwik, Astro, Solid.js, Preact, Lit, Stencil, Alpine.js, htmx, Ember.js, Backbone.js, Knockout.js, Meteor, Aurelia, Polymer, Marko, Mithril, Riot.js, Hyperapp, Inferno, Dojo, Express.js, Koa, Hapi, Fastify, NestJS, Adonis, Sails.js, LoopBack, Restify, Feathers, Moleculer, Strapi, Keystone, Django, Flask, FastAPI, Tornado, Pyramid, Bottle, CherryPy, web2py, Falcon, Sanic, Quart, Starlette, BlackSheep, Rails, Sinatra, Hanami, Padrino, Grape, Spring Boot, Spring MVC, Spring WebFlux, Micronaut, Quarkus, Vert.x, Play Framework, Akka HTTP, Http4s, Finch, Scalatra, Laravel, Symfony, CodeIgniter, Slim, Lumen, Yii, Phalcon, CakePHP, FuelPHP, ASP.NET, ASP.NET Core, Nancy, ServiceStack, Carter, Phoenix, Plug, Cowboy, Maru, Express, Oak, Deno, Bun, Hono
- **Frameworks Mobile** (31): React Native, Flutter, Ionic, Xamarin, Cordova, Capacitor, NativeScript, Expo, SwiftUI, Jetpack Compose, UIKit, Android SDK, Kotlin Multiplatform, .NET MAUI, Tauri, Electron, NW.js, Neutralino, Wails, React Native for Windows, React Native Web, Kotlin Native, Swift for TensorFlow, Flutter Web, Blazor Hybrid, Uno Platform, Avalonia, MAUI, Qt, GTK
- **Bases de Datos** (64): PostgreSQL, MySQL, MariaDB, SQLite, SQL Server, Oracle, DB2, MongoDB, Cassandra, Redis, Elasticsearch, CouchDB, Neo4j, ArangoDB, OrientDB, InfluxDB, TimescaleDB, QuestDB, CrateDB, ClickHouse, DuckDB, Snowflake, BigQuery, Redshift, Athena, Presto, Trino, Druid, Pinot, Vertica, Teradata, Greenplum, Exasol, VoltDB, MemSQL, SingleStore, TiDB, CockroachDB, YugabyteDB, FaunaDB, DynamoDB, CosmosDB, DocumentDB, Firebase Realtime Database, Firestore, Supabase, PocketBase, RethinkDB, HarperDB, SurrealDB, EdgeDB, Prisma, Hasura, Dgraph, Dolt, LanceDB, Milvus, Weaviate, Qdrant, Pinecone, Chroma
- **Cloud & DevOps** (98): AWS, Azure, GCP, DigitalOcean, Linode, Vultr, Hetzner, OVH, Heroku, Vercel, Netlify, Cloudflare, Railway, Render, Fly.io, Deta, Supabase, Convex, PlanetScale, Neon, Xata, Turso, Docker, Kubernetes, Podman, containerd, CRI-O, Nomad, OpenShift, Rancher, k3s, k0s, MicroK8s, KinD, Minikube, Docker Compose, Docker Swarm, ECS, EKS, AKS, GKE, Fargate, Lambda, Cloud Functions, Cloud Run, Azure Functions, Terraform, Pulumi, CloudFormation, ARM, Bicep, CDK, Crossplane, Ansible, Chef, Puppet, Salt, Jenkins, GitLab CI, GitHub Actions, CircleCI, Travis CI, Drone, Tekton, Argo CD, Flux, Spinnaker, Harness, Codefresh, Buildkite, TeamCity, Bamboo, GoCD, Concourse, Screwdriver, Woodpecker, Prometheus, Grafana, Datadog, New Relic, Dynatrace, AppDynamics, Elastic APM, Jaeger, Zipkin, OpenTelemetry, Fluentd, Logstash, Vector
- **Data Science & ML** (87): TensorFlow, PyTorch, Keras, Scikit-learn, XGBoost, LightGBM, CatBoost, NumPy, Pandas, SciPy, Matplotlib, Seaborn, Plotly, Bokeh, Altair, Streamlit, Dash, Gradio, Jupyter, JupyterLab, Google Colab, Kaggle, Databricks, MLflow, Kubeflow,

TFX, ZenML, Metaflow, Kedro, DVC, Weights & Biases, Neptune.ai, Comet, ClearML, Sacred, Guild AI, Polyaxon, Ray, Dask, Spark, PySpark, Hadoop, Hive, Pig, Mahout, Flink, Storm, Samza, Beam, Airflow, Prefect, Dagster, Luigi, Oozie, Azkaban, Hugging Face, LangChain, LlamaIndex, Haystack, txtai, Semantic Kernel, AutoGPT, BabyAGI, NLTK, spaCy, Gensim, FastText, Word2Vec, GloVe, BERT, GPT, T5, BART, RoBERTa, ALBERT, DistilBERT, ELECTRA, DeBERTa, Longformer, BigBird, Reformer, Linformer, Performer, Synthesizer, FNet, CANINE, ByT5, mT5, XLM-RoBERTa

- **Testing & QA** (45): Jest, Mocha, Chai, Jasmine, Karma, Cypress, Selenium, Puppeteer, Playwright, TestCafe, WebDriverIO, Nightwatch, Protractor, Cucumber, Behave, SpecFlow, JUnit, TestNG, Mockito, JMock, EasyMock, PowerMock, WireMock, REST Assured, Karate, Postman, Insomnia, Paw, HTTPie, curl, Pytest, unittest, nose, Robot Framework, Locust, JMeter, Gatling, k6, Artillery, wrk, ab, hey, Siege, Vegeta, autocannon, Bombardier

- **Herramientas** (306): Git, SVN, Mercurial, Perforce, npm, Yarn, pnpm, Bun, pip, Poetry, Pipenv, uv, Conda, Maven, Gradle, Ant, sbt, leiningen, Mix, Cargo, Composer, Bundler, RubyGems, NuGet, vcpkg, Conan, Homebrew, apt, yum, dnf, pacman, zypper, Chocolatey, Scoop, winget, WebStorm, IntelliJ IDEA, PyCharm, Visual Studio, VS Code, Atom, Sublime Text, Vim, Neovim, Emacs, Eclipse, NetBeans, Xcode, Android Studio, AppCode, CLion, DataGrip, GoLand, PhpStorm, Rider, RubyMine, Zed, Fleet, Lapce, Helix, Kakoune, micro, nano, gedit, Kate, Geany, Brackets, Light Table, Code::Blocks, Qt Creator, KDevelop, MonoDevelop, SharpDevelop, BlueJ, DrJava, JCreator, jGRASP, Processing, Arduino IDE, PlatformIO, Thonny, Spyder, JupyterLab, RStudio, Rodeo, Beaker, nteract, Apache Zeppelin, Databricks Notebooks, Google Colab, Kaggle Notebooks, Observable, Deepnote, Hex, Mode, Redash, Metabase, Superset, Tableau, Power BI, Qlik, Looker, Sisense, Domo, MicroStrategy, SAP BusinessObjects, Oracle Analytics, IBM Cognos, SAS, SPSS, Stata, EViews, MATLAB, Octave, R, Julia, Mathematica, Maple, Maxima, wxMaxima, GeoGebra, Desmos, WolframAlpha, SymPy, SageMath, SciPy, NumPy, Pandas, Polars, Modin, Vaex, Dask, Ray, Spark, Hadoop, Hive, Presto, Trino, ClickHouse, DuckDB, Parquet, Arrow, Feather, Avro, ORC, Protobuf, FlatBuffers, Cap'n Proto, MessagePack, BSON, CBOR, Smile, Ion, Thrift, gRPC, REST, GraphQL, SOAP, XML-RPC, JSON-RPC, WebSockets, Server-Sent Events, WebRTC, MQTT, AMQP, Kafka, RabbitMQ, ActiveMQ, ZeroMQ, NanoMsg, NATS, Redis Pub/Sub, AWS SQS, AWS SNS, Azure Service Bus, Google Pub/Sub, Apache Pulsar, Apache Camel, Mulesoft, WSO2, Talend, Informatica, Pentaho, Kettle, Apache NiFi, StreamSets, Airbyte, Fivetran, Stitch, Segment, RudderStack, PostHog, Mixpanel, Amplitude, Heap, Pendo, FullStory, Hotjar, Crazy Egg, Optimizely, VWO, Google Optimize, AB Tasty, Split, LaunchDarkly, Unleash, FlagSmith, ConfigCat, Bullet Train, DevCycle, GrowthBook, Statsig, Eppo, Molasses, Flipt, Flagsmith, Tggl, CloudBees Feature Flags, Harness Feature Flags, Split.io

Total: 847 términos tecnológicos

```
# src/extractor/regex_patterns.py

REGEX_PATTERNS = {
```

```
    # Lenguajes de programación
    'languages': [
        r'\b(Python|Java|JavaScript|TypeScript|Go|Rust|C\+\+|C#|Swift|Kotlin|Scala|Ruby|PHP|Per
        r'\b(NodeJS|Node\.js)\b',
    ],

    # Frameworks web
    'web_frameworks': [
        r'\b(React|Angular|Vue(?:\.js)?|Svelte|Next\.js|Nuxt\.js|Gatsby|Remix)\b',
        r'\b(Express(?:\.js)?|Koa|Hapi|Fastify|NestJS)\b',
        r'\b(Django|Flask|FastAPI|Tornado|Pyramid)\b',
        r'\b(Spring(?:\s+Boot)?|Micronaut|Quarkus)\b',
        r'\b(Laravel|Symfony|CodeIgniter|Slim)\b',
        r'\b(ASP\.NET(?:\s+Core)?|Nancy|ServiceStack)\b',
    ],

    # Frameworks mobile
    'mobile_frameworks': [
        r'\b(React\s+Native|Flutter|Ionic|Xamarin|Cordova|Capacitor|NativeScript|Expo)\b',
        r'\b(SwiftUI|Jetpack\s+Compose|UIKit|Android\s+SDK)\b',
    ],

    # Bases de datos
    'databases': [
        r'\b(PostgreSQL|MySQL|MariaDB|SQLite|SQL\s+Server|Oracle|DB2)\b',
        r'\b(MongoDB|Cassandra|Redis|Elasticsearch|CouchDB|Neo4j|ArangoDB)\b',
        r'\b(InfluxDB|TimescaleDB|ClickHouse|DuckDB)\b',
        r'\b(DynamoDB|CosmosDB|Firestore|Supabase)\b',
    ],

    # Cloud & DevOps
    'cloud_devops': [
        r'\b(AWS|Azure|GCP|Google\s+Cloud(?:\s+Platform)?|DigitalOcean|Heroku|Vercel|Netlify)\b
        r'\b(Docker|Kubernetes|K8s|Podman|containerd|Nomad|OpenShift)\b',
        r'\b(Terraform|Pulumi|CloudFormation|ARM|CDK|Ansible|Chef|Puppet|Salt)\b',
        r'\b(Jenkins|GitLab\s+CI|GitHub\s+Actions|CircleCI|Travis\s+CI|Drone|Argo\s+CD)\b',
        r'\b(Prometheus|Grafana|Datadog|New\s+Relic|Dynatrace|Elastic\s+APM)\b',
    ],

    # Data Science & ML
    'data_ml': [
        r'\b(TensorFlow|PyTorch|Keras|Scikit-learn|XGBoost|LightGBM|CatBoost)\b',
        r'\b(NumPy|Pandas|SciPy|Matplotlib|Seaborn|Plotly|Bokeh)\b',
        r'\b(Jupyter|JupyterLab|Google\s+Colab|Kaggle|Databricks)\b',
        r'\b(MLflow|Kubeflow|TFX|Ray|Dask|Spark|PySpark|Airflow|Prefect)\b',
        r'\b(Hugging\s+Face|LangChain|LlamaIndex|NLTK|spaCy|Gensim)\b',
        r'\b(BERT|GPT|T5|BART|RoBERTa|DistilBERT|LLaMA|Gemma)\b',
```

```
    ],

    # Testing & QA
    'testing': [
        r'\b(Jest|Mocha|Chai|Jasmine|Karma|Cypress|Selenium|Puppeteer|Playwright)\b',
        r'\b(JUnit|TestNG|Mockito|Pytest|unittest|Robot\s+Framework)\b',
        r'\b(Postman|Insomnia|JMeter|Gatling|k6|Locust)\b',
    ],

    # Herramientas
    'tools': [
        r'\b(Git|GitHub|GitLab|Bitbucket|SVN|Mercurial)\b',
        r'\b(npm|Yarn|pnpm|pip|Poetry|Maven|Gradle|Cargo|Composer)\b',
        r'\b(VS\s+Code|Visual\s+Studio|IntelliJ|PyCharm|WebStorm|Xcode|Android\s+Studio)\b',
        r'\b(Webpack|Vite|Rollup|Parcel|esbuild|Babel|ESLint|Prettier)\b',
        r'\b(REST|GraphQL|gRPC|SOAP|WebSockets|Kafka|RabbitMQ|Redis\s+Pub/Sub)\b',
    ],

    # Siglas y abreviaciones
    'acronyms': [
        r'\b(API|SDK|CLI|CI/CD|CD|ML|AI|NLP|CV|RL|DL|CNN|RNN|LSTM|GRU|GAN|VAE|RL|DQN)\b',
        r'\b(ETL|ELT|OLAP|OLTP|ACID|BASE|CAP|CRUD|REST|SOAP|JSON|XML|YAML|TOML|CSV|TSV)\b',
        r'\b(HTTP|HTTPS|TCP|UDP|IP|DNS|TLS|SSL|SSH|FTP|SFTP|SMTP|IMAP|POP3|WebRTC|gRPC)\b',
        r'\b(SQL|NoSQL|ORM|ODM|RDBMS|DBMS|DBA|DWH|BI|ETL|CDC|CQRS|DDD|TDD|BDD|ATDD)\b',
        r'\b(SaaS|PaaS|IaaS|FaaS|CaaS|BaaS|DBaaS|MLaaS|AIOps|DevOps|GitOps|SecOps|DataOps)\b',
        r'\b(JWT|OAuth|SAML|OIDC|SSO|MFA|2FA|RBAC|ABAC|ACL|IAM|PAM|PKI|KMS|HSM|WAF|IDS|IPS)\b'
    ],
}

def extract_regex_matches(text: str) -> List[Tuple[str, str]]:
    """Extrae matches de regex con categoría."""
    matches = []

    for category, patterns in REGEX_PATTERNS.items():
        for pattern in patterns:
            for match in re.finditer(pattern, text, re.IGNORECASE):
                skill = match.group(0)
                matches.append((skill, category))

    return matches
```

2. Regex Patterns

```
# src/extractor/tfidf_extractor.py
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import spacy

nlp = spacy.load("es_core_news_lg")

def extract_tfidf_phrases(text: str, corpus: List[str], max_features: int = 50) -> List[str]:
    """Extrae frases nominales relevantes con TF-IDF."""

    # Extraer noun chunks del texto
    doc = nlp(text)
    noun_chunks = [chunk.text.lower() for chunk in doc.noun_chunks]

    # TF-IDF scoring
    vectorizer = TfidfVectorizer(
        max_features=max_features,
        ngram_range=(1, 3),  # Unigrams, bigrams, trigrams
        stop_words=STOPWORDS_SPANISH + STOPWORDS_ENGLISH,
        min_df=2,  # Mínimo 2 documentos
        max_df=0.8,  # Máximo 80% de documentos
    )

    # Fit en corpus completo (300 jobs gold standard)
    tfidf_matrix = vectorizer.fit_transform(corpus)

    # Transform texto actual
    text_vector = vectorizer.transform([text])

    # Obtener features ordenadas por score
    feature_names = vectorizer.get_feature_names_out()
    scores = text_vector.toarray()[0]

    # Top features
    top_indices = scores.argsort()[-max_features:][::-1]
    top_features = [feature_names[i] for i in top_indices if scores[i] > 0]

    # Filtrar solo noun chunks
    skills = [f for f in top_features if f in noun_chunks]

    return skills
```

3. TF-IDF Noun Phrases

Pipeline A.1: TF-IDF + N-grams (Experimento Fallido)

¿Qué era Pipeline A.1? Pipeline A.1 fue un **experimento académico** para evaluar si métodos estadísticos puros (sin deep learning) podían competir con NER+Regex y LLMs.

Hipótesis: TF-IDF con n-gramas y noun phrases puede extraer skills técnicas sin necesidad de modelos entrenados.

```python
# src/extractor/ngram_extractor.py

from sklearn.feature_extraction.text import TfidfVectorizer
import spacy

class NGramExtractor:
    """Extractor estadístico basado en TF-IDF + N-grams."""

    # Stopwords específicas para ofertas laborales (bilingual ES+EN)
    STOPWORDS_DOMAIN = [
        # Español
        'años', 'experiencia', 'conocimiento', 'requisitos', 'responsabilidades',
        'funciones', 'oferta', 'perfil', 'candidato', 'puesto', 'trabajo',
        'empresa', 'equipo', 'cliente', 'proyecto', 'deseable', 'necesario',

        # Inglés
        'years', 'experience', 'knowledge', 'requirements', 'responsibilities',
        'functions', 'offer', 'profile', 'candidate', 'position', 'job',
        'company', 'team', 'client', 'project', 'desirable', 'required',
    ]

    # Patrones de ruido (NO son skills)
    NOISE_PATTERNS = [
        r'^\d+$',                          # Números puros: "2", "5"
        r'^\d+[a-z]$',                     # Patterns: "2Innovate", "3D"
        r'^[a-z]$',                        # Letras solas (excepto R, C)
        r'^\d+\s*(años?|years?)$',         # "3 años", "5 years"
        r'^(enero|february|lunes|monday)$', # Meses/días
        r'^\d{3,}',                        # 3+ dígitos: "000", "220"
    ]

    def __init__(self):
        self.nlp = spacy.load("es_core_news_lg")
        self.vectorizer = None

    def fit(self, corpus: List[str]):
        """Entrena TF-IDF en corpus de 300 jobs gold standard."""

        self.vectorizer = TfidfVectorizer(
            ngram_range=(1, 3),           # Unigrams, bigrams, trigrams
            max_features=500,             # Top 500 n-grams
            min_df=2,                     # Mínimo 2 documentos
```

```
            max_df=0.8,                    # Máximo 80% documentos
            stop_words=self.STOPWORDS_DOMAIN,
        )

        self.vectorizer.fit(corpus)
        print(f"  TF-IDF fitted on {len(corpus)} documents")

    def extract_skills(self, text: str, top_n: int = 20) -> List[str]:
        """Extrae top N skills por TF-IDF score."""

        # Paso 1: TF-IDF scoring
        tfidf_vector = self.vectorizer.transform([text])
        feature_names = self.vectorizer.get_feature_names_out()
        scores = tfidf_vector.toarray()[0]

        # Paso 2: Ordenar por score descendente
        top_indices = scores.argsort()[-top_n:][::-1]
        candidates = [feature_names[i] for i in top_indices if scores[i] > 0]

        # Paso 3: Filtrar noun phrases
        doc = self.nlp(text)
        noun_chunks = {chunk.text.lower() for chunk in doc.noun_chunks}

        # Paso 4: Filtrar ruido
        skills = []
        for candidate in candidates:
            # Skip si es ruido
            if any(re.match(pattern, candidate) for pattern in self.NOISE_PATTERNS):
                continue

            # Preferir noun phrases
            if candidate in noun_chunks:
                skills.append(candidate)
            # O si es término técnico conocido
            elif self._is_technical_term(candidate):
                skills.append(candidate)

        return skills

    def _is_technical_term(self, term: str) -> bool:
        """Valida si es término técnico."""
        # Heurísticas simples
        if term.lower() in ['python', 'java', 'sql', 'react', 'docker']:
            return True
        if term.isupper() and len(term) >= 2:  # Acrónimos: AWS, API
            return True
        return False
```

Implementación Técnica

Proceso de Iteración (4 experimentos)

Experimento	Cambio	F1 Pre-ESCO	Problema
Iter 1	Baseline TF-IDF puro	8.3%	Extraía stopwords ("años experiencia")
Iter 2	+ Stopwords dominio (193)	11.2%	Extraía ruido ("000 confidencial")
Iter 3	+ Filtros ruido (9 patrones)	14.8%	Extraía genéricos ("gestión", "desarrollo")
Iter 4	+ Noun phrases filter	**11.69%**	Bajo recall, skills muy genéricas

Resultado Final: - Pre-ESCO: F1=**11.69%** (Precision=8.75%, Recall=17.62%) - Post-ESCO: F1=**48.00%** (ESCO normaliza mucho, pero sigue bajo)

¿Por Qué Falló Pipeline A.1?

Problema 1: TF-IDF Detecta Frecuencia, No Semántica

```
Ejemplo Real (Job #42):

Gold Standard: ["Python", "Django", "PostgreSQL", "Docker", "AWS"]

Pipeline A.1 extrajo:
- "desarrollo backend"    ← Genérico (TF-IDF alto pero no es skill)
- "experiencia sólida"    ← No es skill
- "equipo desarrollo"     ← No es skill
- "python"                ←  Correcto (TF-IDF bajo porque es común)
- "base datos"            ← Muy genérico, no específico

Problema: TF-IDF prioriza términos raros, no necesariamente skills.
"PostgreSQL" tiene TF-IDF BAJO (aparece en muchas ofertas) → no se extrae.
"desarrollo backend" tiene TF-IDF ALTO (frase distintiva) → se extrae incorrectamente.
```

Problema 2: N-grams Capturan Contexto, No Skills

```
Trigrams extraídos:
- "años experiencia python"   ← Contexto, no skill atómica
- "desarrollo aplicaciones web" ← Demasiado genérico
- "trabajo equipo desarrollo"   ← Soft skill implícita
```

```
Skills reales perdidas:
- "Python" (perdida en trigram)
- "Django" (no detectada, TF-IDF bajo)
- "Docker" (no detectada, TF-IDF bajo)
```

Problema 3: Sin Contexto Sintáctico Pipeline A (NER) entiende:

```
"Experiencia en Python y Django"
     ↓ NER identifica entidades
["Python", "Django"]
```

Pipeline A.1 (TF-IDF) ve:

```
"Experiencia en Python y Django"
     ↓ TF-IDF scoring
["experiencia python django"] (trigram)
```

Problema 4: Stopwords Insuficientes Agregamos 193 stopwords de dominio, pero aún extraía ruido:

```
Ruido extraído:
- "000 confidencial"
- "220 talentosos"
- "15 liderando"
- "frontend backend fullstack" (concatenado)
- "años mínimo experiencia"

Razón: TF-IDF scoring alto por ser términos distintivos de ese documento.
```

¿Qué Aprendimos?

Lección 1: TF-IDF No Es Suficiente para Named Entity Recognition **TF-IDF es excelente para:** - Recuperación de documentos (search engines) - Detección de tópicos generales - Identificar documentos similares

TF-IDF NO funciona para: - Extracción de entidades nombradas técnicas - Skills que aparecen frecuentemente (ej: Python, SQL) - Distinguir skills de contexto descriptivo

Conclusión: Necesitas entender **semántica y sintaxis**, no solo frecuencia.

```python
# TF-IDF corpus-level (lo que hicimos)
vectorizer.fit(corpus_300_jobs)  # Aprende de 300 docs
tfidf_vector = vectorizer.transform([job_text])  # Transforma doc individual
```

```
Problema:
- Skills comunes (Python, SQL, Java) tienen IDF BAJO
- Son penalizadas por aparecer en muchos documentos
- Pero son EXACTAMENTE lo que queremos extraer!

Paradoja: TF-IDF penaliza lo que queremos detectar.
```

Lección 2: Corpus-Level vs Document-Level

Lección 3: Noun Phrases Skills

```
Noun phrases extraídos por spaCy:
- "desarrollo de software"     ← Actividad, no skill
- "equipo de desarrollo"       ← Contexto organizacional
- "experiencia comprobable"    ← Requisito, no skill
- "python django"              ←  Podría ser skill, pero irregular

Skills reales (nombres propios):
- "Python"                     ← Nombre propio → NER lo detecta
- "Django"                     ← Nombre propio → NER lo detecta
- "PostgreSQL"                 ← Nombre propio → NER lo detecta
```

Conclusión: Noun phrases capturan frases, pero skills técnicas son **nombres propios** (NER domain).

Lección 4: Baseline Académico Válido Aunque Pipeline A.1 falló (F1=11.69% pre-ESCO), cumplió su propósito:

Propósito académico: - Demostrar que métodos estadísticos puros NO son suficientes - Establecer baseline inferior para comparación - Justificar necesidad de NER (Pipeline A) o LLM (Pipeline B)

Comparación final: - Pipeline A.1 (TF-IDF): F1=11.69% ← Baseline estadístico - Pipeline A (NER+Regex): F1=22.70% → **+11pp mejora** - Pipeline B (LLM Gemma): F1=46.23% → **+34pp mejora**

Conclusión científica: NER y LLMs superan ampliamente métodos estadísticos puros.

Lección 5: Iteración Rápida Es Valiosa Proceso completo Pipeline A.1: - Implementación: 2 días - 4 iteraciones experimentales: 3 días - Evaluación: 1 día - **Total: 6 días**

Valor: Descartamos rápido un approach no viable, ahorrando semanas de optimización innecesaria.

Código Final Documentado El código completo está en `src/extractor/ngram_extractor.py` (200 líneas).

Archivos relacionados: - `docs/PIPELINE_A1_IMPLEMENTATION_LOG.md` (820 líneas) - Log completo de iteraciones - `data/reports/EVALUATION_REPORT_*_A1.md` - Resultados de evaluación - `scripts/evaluate_pipeline_a1.py` - Script de evaluación

Para reproducir:

```
# Activar venv
source venv/bin/activate

# Procesar con Pipeline A.1
python scripts/run_pipeline_a1.py --gold-standard-only

# Evaluar
python scripts/evaluate_pipelines_dual.py --pipelines A,A.1,B

# Ver reporte
cat data/reports/EVALUATION_REPORT_*_A1.md
```

Comparación Final: A vs A.1 vs B

Métrica	A.1 (TF-IDF)	A (NER+Regex)	B (LLM)
F1 Pre-ESCO	11.69%	22.70%	**46.23%**
F1 Post-ESCO	48.00%	72.15%	**84.26%**
Precision	8.75%	20.66%	**38.94%**
Recall	17.62%	25.20%	**55.82%**
Skills extraídas	4,103	2,633	8,301
Skills únicas	892	487	2,847
Velocidad	8s/job	13s/job	42s/job
Complejidad implementación	Baja	Media	Alta
Dependencias	scikit-learn, spaCy	spaCy	Transformers (4.3GB)

Conclusión académica: La complejidad adicional de NER y especialmente LLMs está **justificada** por mejoras sustanciales en F1-Score.

Pipeline B: LLM (Gemma 3 4B)

```
# src/extractor/llm_extractor.py

from transformers import AutoTokenizer, AutoModelForCausalLM
```

```python
import torch
import json

class GemmaExtractor:
    """Extractor de skills con Gemma 3 4B Instruct."""

    def __init__(self, model_name: str = "google/gemma-2-2b-it"):
        self.model_name = model_name
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(
            model_name,
            torch_dtype=torch.float16,  # Half precision para reducir memoria
            device_map="auto"  # Auto-asignar a GPU si disponible
        )

    def extract_skills(self, title: str, description: str, requirements: str = "") -> dict:
        """Extrae skills de una oferta laboral."""

        # Construir prompt estructurado
        prompt = self._build_prompt(title, description, requirements)

        # Tokenizar
        inputs = self.tokenizer(prompt, return_tensors="pt", truncation=True, max_length=2048)

        # Generar
        outputs = self.model.generate(
            inputs.input_ids,
            max_new_tokens=512,
            temperature=0.3,  # Baja temperatura = menos creatividad, menos alucinaciones
            top_p=0.9,
            top_k=50,
            do_sample=True,
            pad_token_id=self.tokenizer.eos_token_id,
            repetition_penalty=1.2,  # Penalizar repeticiones
        )

        # Decodificar
        output_text = self.tokenizer.decode(outputs[0], skip_special_tokens=True)

        # Extraer JSON de la respuesta
        skills_json = self._extract_json(output_text)

        return skills_json

    def _build_prompt(self, title: str, description: str, requirements: str) -> str:
        """Construye prompt estructurado en español."""
```

```
        combined_text = f"{description}\n{requirements}".strip()

        prompt = f"""Eres un experto extractor de habilidades técnicas de ofertas laborales.

OFERTA LABORAL:
Título: {title}
Descripción: {combined_text}

INSTRUCCIONES:
1. Extrae TODAS las habilidades técnicas (lenguajes, frameworks, herramientas, cloud, bases de
2. Extrae habilidades blandas (liderazgo, comunicación, trabajo en equipo, resolución de proble
3. Infiere habilidades implícitas de las responsabilidades mencionadas
4. NO inventes habilidades que no están mencionadas o implícitas
5. Retorna SOLO JSON válido, sin texto adicional

FORMATO SALIDA:
{{
  "hard_skills": ["Python", "Django", "PostgreSQL", "Docker", "AWS"],
  "soft_skills": ["Liderazgo", "Comunicación", "Trabajo en equipo"]
}}

JSON:"""

        return prompt

    def _extract_json(self, text: str) -> dict:
        """Extrae JSON de respuesta del modelo."""

        # Buscar primer { y último }
        start = text.find('{')
        end = text.rfind('}') + 1

        if start == -1 or end == 0:
            return {"hard_skills": [], "soft_skills": []}

        json_str = text[start:end]

        try:
            return json.loads(json_str)
        except json.JSONDecodeError:
            # Intentar limpiar JSON malformado
            json_str = json_str.replace("'", '"')  # Comillas simples → dobles
            json_str = json_str.replace(",]", "]")  # Trailing commas
            json_str = json_str.replace(",}", "}")

            try:
                return json.loads(json_str)
```

```
            except:
                return {"hard_skills": [], "soft_skills": []}
```

ESCO Matcher: 3 Capas

```
# src/extractor/esco_matcher_3layers.py

from fuzzywuzzy import fuzz
import psycopg2

class ESCOMatcher3Layers:
    """Matcher ESCO con 3 capas: Exact, Fuzzy, Semantic."""

    FUZZY_THRESHOLD = 0.92  # Optimizado tras experimentación
    FUZZY_THRESHOLD_SHORT = 0.95  # Para strings 4 chars
    LAYER3_ENABLED = False  # FAISS deshabilitado

    def __init__(self):
        self.db_url = os.getenv('DATABASE_URL')
        self._load_esco_skills()

    def _load_esco_skills(self):
        """Carga todas las skills ESCO en memoria."""
        conn = psycopg2.connect(self.db_url)
        cursor = conn.cursor()

        cursor.execute("""
            SELECT skill_uri, preferred_label_es, preferred_label_en,
                   skill_type, skill_group
            FROM esco_skills
            WHERE is_active = TRUE
        """)

        self.esco_skills = []
        for row in cursor.fetchall():
            self.esco_skills.append({
                'uri': row[0],
                'label_es': row[1],
                'label_en': row[2],
                'type': row[3],
                'group': row[4],
            })

        cursor.close()
        conn.close()
```

```
        print(f"  Loaded {len(self.esco_skills):,} ESCO skills")

    def match_skill(self, skill_text: str) -> Optional[dict]:
        """Match skill con estrategia de 3 capas."""

        if not skill_text or len(skill_text.strip()) < 2:
            return None

        skill_text = skill_text.strip()

        # Layer 1: Exact Match
        match = self._layer1_exact(skill_text)
        if match:
            return match

        # Layer 2: Fuzzy Match
        match = self._layer2_fuzzy(skill_text)
        if match:
            return match

        # Layer 3: Semantic Match (DESHABILITADO)
        # if self.LAYER3_ENABLED:
        #     match = self._layer3_semantic(skill_text)
        #     if match:
        #         return match

        # No match → emergent skill
        return None

    def _layer1_exact(self, skill_text: str) -> Optional[dict]:
        """Layer 1: Exact match (case-insensitive)."""

        skill_lower = skill_text.lower()

        for esco in self.esco_skills:
            if esco['label_es'] and skill_lower == esco['label_es'].lower():
                return {
                    'skill_text': skill_text,
                    'matched_skill': esco['label_es'],
                    'esco_uri': esco['uri'],
                    'confidence': 1.00,
                    'method': 'exact',
                    'skill_type': esco['type'],
                    'skill_group': esco['group'],
                }

            if esco['label_en'] and skill_lower == esco['label_en'].lower():
```

```
                return {
                    'skill_text': skill_text,
                    'matched_skill': esco['label_en'],
                    'esco_uri': esco['uri'],
                    'confidence': 1.00,
                    'method': 'exact',
                    'skill_type': esco['type'],
                    'skill_group': esco['group'],
                }

        return None

    def _layer2_fuzzy(self, skill_text: str) -> Optional[dict]:
        """Layer 2: Fuzzy match con fuzzywuzzy."""

        # Threshold adaptativo
        threshold = self.FUZZY_THRESHOLD_SHORT if len(skill_text) <= 4 else self.FUZZY_THRESHOL

        best_match = None
        best_ratio = 0

        skill_lower = skill_text.lower()

        for esco in self.esco_skills:
            # Match contra español
            if esco['label_es']:
                ratio = fuzz.ratio(skill_lower, esco['label_es'].lower()) / 100
                if ratio >= threshold and ratio > best_ratio:
                    best_match = esco
                    best_ratio = ratio

            # Match contra inglés
            if esco['label_en']:
                ratio = fuzz.ratio(skill_lower, esco['label_en'].lower()) / 100
                if ratio >= threshold and ratio > best_ratio:
                    best_match = esco
                    best_ratio = ratio

        if best_match:
            return {
                'skill_text': skill_text,
                'matched_skill': best_match['label_es'] or best_match['label_en'],
                'esco_uri': best_match['uri'],
                'confidence': best_ratio,
                'method': 'fuzzy',
                'skill_type': best_match['type'],
                'skill_group': best_match['group'],
```

```
            }

        return None
```

Normalización

```
# src/evaluation/normalizer.py

class SkillNormalizer:
    """Normaliza variantes de skills a forma canónica."""

    # 193 mapeos canónicos
    CANONICAL_NAMES = {
        # Lenguajes
        'python': 'Python', 'py': 'Python', 'python3': 'Python',
        'javascript': 'JavaScript', 'js': 'JavaScript', 'java script': 'JavaScript',
        'typescript': 'TypeScript', 'ts': 'TypeScript', 'type script': 'TypeScript',
        'java': 'Java',
        'c++': 'C++', 'cpp': 'C++', 'cplusplus': 'C++',
        'c#': 'C#', 'csharp': 'C#', 'c sharp': 'C#',
        'go': 'Go', 'golang': 'Go',
        'rust': 'Rust',
        'ruby': 'Ruby',
        'php': 'PHP',
        'swift': 'Swift',
        'kotlin': 'Kotlin',
        'scala': 'Scala',
        'r': 'R',

        # Frameworks
        'react': 'React', 'reactjs': 'React', 'react.js': 'React', 'react js': 'React',
        'vue': 'Vue.js', 'vuejs': 'Vue.js', 'vue.js': 'Vue.js', 'vue js': 'Vue.js',
        'angular': 'Angular', 'angularjs': 'Angular', 'angular.js': 'Angular',
        'svelte': 'Svelte', 'sveltejs': 'Svelte',
        'next': 'Next.js', 'nextjs': 'Next.js', 'next.js': 'Next.js', 'next js': 'Next.js',
        'django': 'Django',
        'flask': 'Flask',
        'fastapi': 'FastAPI', 'fast api': 'FastAPI',
        'express': 'Express.js', 'expressjs': 'Express.js', 'express.js': 'Express.js',
        'nestjs': 'NestJS', 'nest': 'NestJS', 'nest.js': 'NestJS',
        'spring': 'Spring', 'spring boot': 'Spring Boot', 'springboot': 'Spring Boot',
        'laravel': 'Laravel',

        # Databases
        'postgres': 'PostgreSQL', 'postgresql': 'PostgreSQL', 'pgsql': 'PostgreSQL', 'postgre
        'mysql': 'MySQL', 'my sql': 'MySQL',
```

```
        'mongodb': 'MongoDB', 'mongo': 'MongoDB', 'mongo db': 'MongoDB',
        'redis': 'Redis',
        'elasticsearch': 'Elasticsearch', 'elastic search': 'Elasticsearch',
        'cassandra': 'Cassandra',
        'neo4j': 'Neo4j', 'neo 4j': 'Neo4j',

        # Cloud
        'aws': 'AWS', 'amazon web services': 'AWS',
        'azure': 'Azure', 'microsoft azure': 'Azure',
        'gcp': 'GCP', 'google cloud': 'GCP', 'google cloud platform': 'GCP',

        # DevOps
        'docker': 'Docker',
        'kubernetes': 'Kubernetes', 'k8s': 'Kubernetes', 'k8': 'Kubernetes',
        'jenkins': 'Jenkins',
        'gitlab': 'GitLab', 'git lab': 'GitLab',
        'github': 'GitHub', 'git hub': 'GitHub',
        'terraform': 'Terraform',
        'ansible': 'Ansible',

        # Tools
        'git': 'Git',
        'npm': 'npm',
        'yarn': 'Yarn',
        'webpack': 'Webpack',
        'vite': 'Vite',

        # Data Science
        'pandas': 'Pandas',
        'numpy': 'NumPy', 'num py': 'NumPy',
        'scipy': 'SciPy', 'sci py': 'SciPy',
        'scikit-learn': 'Scikit-learn', 'sklearn': 'Scikit-learn', 'scikit learn': 'Scikit-lea
        'tensorflow': 'TensorFlow', 'tensor flow': 'TensorFlow',
        'pytorch': 'PyTorch', 'py torch': 'PyTorch', 'torch': 'PyTorch',
        'keras': 'Keras',

        # Testing
        'jest': 'Jest',
        'mocha': 'Mocha',
        'cypress': 'Cypress',
        'selenium': 'Selenium',
        'pytest': 'Pytest', 'py test': 'Pytest',
        'junit': 'JUnit', 'j unit': 'JUnit',

        # APIs
        'rest': 'REST', 'restful': 'REST',
        'graphql': 'GraphQL', 'graph ql': 'GraphQL',
```

```
        'grpc': 'gRPC', 'g rpc': 'gRPC',

        # ... (193 mapeos totales)
    }

    def normalize(self, skill_text: str) -> str:
        """Normaliza skill a forma canónica."""

        if not skill_text:
            return skill_text

        # Paso 1: Lowercase + strip
        skill_lower = skill_text.lower().strip()

        # Paso 2: Buscar en diccionario canónico
        if skill_lower in self.CANONICAL_NAMES:
            return self.CANONICAL_NAMES[skill_lower]

        # Paso 3: Remover caracteres especiales
        skill_clean = re.sub(r'[^\w\s]', '', skill_lower)

        if skill_clean in self.CANONICAL_NAMES:
            return self.CANONICAL_NAMES[skill_clean]

        # Paso 4: Capitalizar correctamente
        # "python" → "Python"
        # "react native" → "React Native"
        return skill_text.title()

    def normalize_batch(self, skills: List[str]) -> List[str]:
        """Normaliza batch de skills."""
        return [self.normalize(skill) for skill in skills]
```

Troubleshooting

Problema 1: "ModuleNotFoundError: No module named 'spacy'"

```
# Solución: Activar venv
source venv/bin/activate

# Verificar
which python
# Debe mostrar: .../venv/bin/python
```

```
# Re-instalar si necesario
pip install -r requirements.txt
```

Problema 2: "Can't find model 'es_core_news_lg'"

```
# Descargar modelo spaCy
python -m spacy download es_core_news_lg

# Verificar
python -c "import spacy; nlp=spacy.load('es_core_news_lg'); print('OK')"
```

Problema 3: "Database connection refused"

```
# Verificar PostgreSQL está corriendo
brew services list  # macOS
sudo systemctl status postgresql  # Linux

# Iniciar si está parado
brew services start postgresql@14  # macOS
sudo systemctl start postgresql  # Linux

# Verificar conexión
psql -d labor_observatory -U labor_user -h localhost -p 5432
```

Problema 4: "Out of memory (OOM) con LLM"

```
# Opción 1: Usar modelo más pequeño
python -m src.orchestrator llm-process-jobs --model gemma-3-1b-instruct

# Opción 2: Reducir batch size
python -m src.orchestrator llm-process-jobs --batch-size 5

# Opción 3: Cerrar otras aplicaciones
# Gemma 3 4B requiere ~8GB RAM durante inference
```

Problema 5: "Scraping devuelve 0 jobs"

```
# Verificar selector CSS no cambió
python scripts/test_computrabajo_simple.py
```

```
# Ver logs detallados
python -m src.orchestrator run-once computrabajo --country CO --verbose

# Si selector cambió, actualizar spider
# Editar: src/scraper/spiders/computrabajo_spider.py
```

Problema 6: "ESCO matching muy lento"

```
# Opción 1: Construir índice FAISS (una vez)
python -m src.orchestrator build-faiss-index

# Opción 2: Deshabilitar Layer 3 (ya está deshabilitado)
# Ver: src/extractor/esco_matcher_3layers.py
# LAYER3_ENABLED = False

# Opción 3: Reducir threshold para menos matches
# FUZZY_THRESHOLD = 0.95 (más estricto, menos matches)
```

Problema 7: "Git pull conflicto en main.pdf"

```
# Descartar PDF local (regenerable)
git checkout --theirs docs/latex/main.pdf
git add docs/latex/main.pdf

# Recompilar LaTeX
cd docs/latex
pdflatex -interaction=nonstopmode main.tex
```

Referencias Adicionales

Documentación Técnica

- `docs/architecture.md` - Arquitectura completa (907 líneas)
- `docs/technical-specification.md` - Especificación técnica (801 líneas)
- `docs/SISTEMA_COMPLETO_MANUAL.md` - Manual técnico (10,288 líneas)

Logs de Implementación

- `docs/PIPELINE_A_OPTIMIZATION_LOG.md` - 7 experimentos Pipeline A (3,002 líneas)
- `docs/PIPELINE_B_ITERACION_Y_PRUEBAS.md` - Iteraciones Pipeline B (1,375 líneas)
- `docs/CLUSTERING_IMPLEMENTATION_LOG.md` - Clustering (2,373 líneas)

Resultados

- `docs/EVALUATION_MASTER_RESULTS.md` - Resultados consolidados (813 líneas)
- `docs/EVALUATION_SYSTEM.md` - Sistema de evaluación (880 líneas)
- `data/reports/` - Reportes de evaluación generados

Análisis

- `docs/DATASET_ANALYSIS.md` - Análisis exploratorio (1,608 líneas)
- `docs/ESCO_MATCHING_INVESTIGATION.md` - Calidad ESCO (1,112 líneas)

Para la Defensa

Números Clave a Memorizar

- **30,660** ofertas scrapeadas
- **7,848** anotaciones gold standard
- **14,174** skills en ESCO
- **847** términos diccionario NER
- **193** mapeos normalización
- **300** jobs evaluados
- **4** modelos LLM comparados
- **7** experimentos Pipeline A
- **53** clusters finales
- **F1=84.26%** Pipeline B (post-ESCO)
- **59.5%** skills emergentes
- **42s/job** Pipeline B
- **13s/job** Pipeline A

Preguntas Frecuentes del Jurado

1. **"¿Por qué no usaste BERT fine-tuned?"**
   - Ver sección 11, explicación completa con razones técnicas y económicas
2. **"¿Cómo validaste que no hubo overfitting?"**
   - No hubo fine-tuning, solo inference
   - Gold standard es muestra aleatoria de 30k
   - 2 jobs (0.7%) fallaron → reportado honestamente
3. **"¿Por qué 300 jobs en gold standard?"**
   - Cálculo estadístico: n=294.3 con 95% confianza, 5.7% error
   - 8 min/job × 300 = 40 horas de anotación manual
4. **"¿Cómo implementaste NER?"**

- spaCy es_core_news_lg (560MB)
- Diccionario 847 tecnologías
- Regex patterns complementarios
- TF-IDF noun phrases

5. **"¿Cómo mediste las métricas?"**
   - Operaciones de conjuntos: TP, FP, FN
   - Precision = TP/(TP+FP)
   - Recall = TP/(TP+FN)
   - F1 = 2×(P×R)/(P+R)
   - Evaluación dual: pre-ESCO + post-ESCO

Checklist Pre-Defensa

```
# 1. Verificar sistema funciona
source venv/bin/activate
python -m src.orchestrator status

# 2. Tener números frescos
python scripts/evaluate_pipelines_dual.py

# 3. Verificar visualizaciones generadas
ls outputs/clustering/final/*/umap_*.png

# 4. PDF tesis compilado
cd docs/latex
pdflatex -interaction=nonstopmode main.tex
open main.pdf

# 5. Backup de datos
pg_dump labor_observatory > backup_defensa_$(date +%Y%m%d).dump

# 6. Git status limpio
git status
git log --oneline -10

#  Listo para defender!
```

¡Éxito en tu defensa!