Abstract

Avem o lista de companii împreună cu niște informații despre ele:

- **descriere în text** (ce face compania),

- **taguri** (cuvinte-cheie despre domeniul lor de activitate),

- **sectoare/categorii/nișe**.

Trebuie un algoritm de clasificare care, pe baza acestor informații, să atribuie fiecărei companii una sau mai multe etichete relevante dintr-o taxonomie statică, oferită separat.

**Scopul final**: crearea un program care:

* primește un fișier cu companii,
* citește descrierile, tagurile și categoriile,
* clasifică fiecare companie într-unul sau mai multe domenii de asigurări dintr-o listă predefinită.

**Înțelegerea datelor și etichetelor**

* Observam cum sunt descrise companiile (text, taguri).
* Ne uitam în taxonomy și vedem ce tipuri de asigurări sunt acolo.
* Încercam să **corelam** descrierile companiilor cu etichetele relevante

## ****Ce fișiere avem:****

1. ml\_insurance\_challenge.csv — datele brute despre companii:
   * description, business\_tags, sector, category, niche
2. insurance\_taxonomy.xlsx — lista etichetelor posibile (ex: Life Insurance, P&C, Reinsurance, etc.)

Dacă avem 10.000 de companii și vrem să le grupam logic, avem nevoie de o **listă fixă de etichete (labels)** care descriu în ce domeniu activează fiecare. Asta este **taxonomia** — o **listă ierarhică sau plană** de concepte care ne ajută să „pui etichete” pe entități (companii în cazul nostru). Primim descrierea unei companii :

- Alegem **una sau mai multe etichete din taxonomie** care se potrivesc cu acea companie.

- Automatizam acest proces.

**Structura generală a datelor (ml\_insurance\_challenge.csv)**

Total rânduri: 9494 companii

Coloane disponibile:

description – descriere text liber (lipsesc 12 valori)

business\_tags – listă de cuvinte-cheie (toate completate)

sector – generalizare pe domeniu (lipsesc 27 valori)

category – categorie specifică (450 unice)

niche – nișă detaliată (957 unice)

**Observații utile**

description este foarte rar duplicată — semn că fiecare companie e unică.

business\_tags are valori ca "[]", dar 363 rânduri repetă exact aceleași taguri.

sector este o coloană cu doar 7 valori distincte, foarte generală.

category și niche sunt foarte fragmentate, potențial utile pentru clasificare sau ca features suplimentare.

**Ce putem concluziona deja:**

E clar că vom folosi description + business\_tags drept features principale.

sector, category, niche pot fi folosite pentru feature engineering, dar sunt slabi predictori direcți din cauza numărului mare de valori distincte.

Avem lipsuri minore în date — le putem curăța simplu.

**Structura fișierului** insurance\_taxonomy.xlsx

**Coloană unică:** label

**Număr total de etichete:** 220

Sunt etichete **foarte specializate**, multe dintre ele din zona **agriculturii, sănătății veterinare, serviciilor specializate** — deci nu este o taxonomie clasică cu „Life Insurance”, „Reinsurance” etc., ci pare că e o taxonomie **de tip industrial / economic**, care e **utilizată indirect în industria de asigurari.**

**Ce înseamnă asta pentru task?**

1. Nu va fi un task clasic de clasificare „Insurance Type A vs. Type B”.
2. Trebuie să mapam companiile la **nișe economice detaliate**, pe baza:

descrierii lor (description)

tagurilor (business\_tags)

poate și niche, category, sector

1. Voi construi un model care să învețe **să potrivească profilul unei companii** cu o etichetă din această listă.

Este **un task de clasificare multi-class sau multi-label** (o companie poate aparține mai multor etichete)

**🔹 Etapa 1 – Explorare și înțelegere date**

Începem să legam intuitiv ce fel de companie s-ar potrivi cu ce etichetă.

Deși sector, category și niche sunt informative si usor de inteles pentru un om, din perspectivă de ML acestea au o cardinalitate mare și oferă context limitat față de description și tags, care conțin semnale semantice mai bogate.

**🔹 Etapa 2 – Preprocesare date**

- citim fișierul .csv,

- curățăm textul (description, tags),

- procesăm tag-urile JSON în liste reale,

- creăm câmpul full\_profile (descriere + taguri + categorie + nișă),

- exportăm un processed\_data.csv ca bază pentru clasificare.

Pentru a pregăti datele pentru clasificare, am aplicat următorii pași:

- Am curățat coloana `description`, normalizând textul (lowercase, fără simboluri).

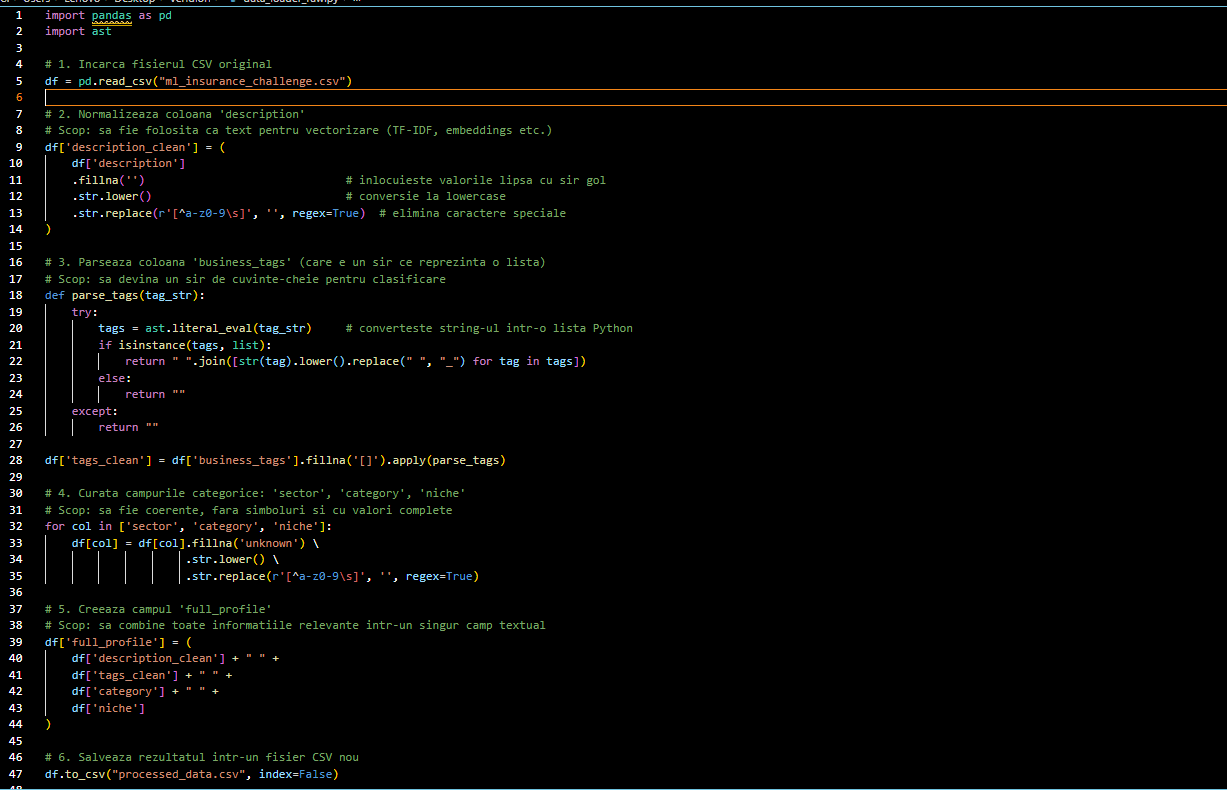
- Am convertit `business\_tags` din string JSON-like într-un șir util de tokens.( (ex: ["Insurance", "Broker"] → "insurance broker").

- Am completat și curățat câmpurile `sector`, `category`, `niche`. (unknown pentru lipsă)

- Am creat o coloană `full\_profile` ce combină toate aceste surse de semnal. ceasta va fi folosită ca **input pentru clasificare**.

Această etapă a fost realizată printr-un script Python dedicat (`data\_loader.py`) care poate fi reutilizat pentru orice alt set similar de date.

Acesta este **scriptul principal de preprocesare :**



Ce face:

* ia ca **input** fișierul original de la Veridion: ml\_insurance\_challenge.csv
* curăță și normalizează toate câmpurile (text, taguri, categorii)
* construiește coloana full\_profile (importantă pentru clasificare)
* scrie rezultatul într-un nou fișier: processed\_data.csv

description\_clean: textul descriptiv este convertit la lowercase, golurile sunt completate cu șiruri goale, iar caracterele speciale sunt eliminate pentru a-l pregăti pentru vectorizare (TF-IDF, embeddings etc.).

tags\_clean: lista de business\_tags este convertită din format JSON într-un șir unificat de cuvinte-cheie relevante, cu spațiile înlocuite de \_, pentru a fi compatibile cu modelele de NLP.

full\_profile: câmp compus care concatenează description\_clean, tags\_clean, category și niche — oferind un rezumat semantic complet per companie, utilizat drept input principal în clasificare.

Explorare date noi (EDA) : Salvare fișier procesat :EDA pe datele prelucrate (distribuții, lipsuri, patternuri) realizata in jupyter notebook

Detalii în:

- `00\_eda\_raw.ipynb`: analiza inițială pe datele brute.

- `01\_eda\_explorare.ipynb`: analiză pe `processed\_data.csv`.

**🔹 Etapa 3 – Generare pseudo-etichete pentru antrenament**

**Obiectiv**: crearea un mic set de date etichetat de mine, care să fie realist și divers, astfel încât să poată fi folosit pentru antrenarea unui model initial

1) **Deschidem fișierul processed\_data.csv** și selectăm un eșantion randomizat de 50–100 companii.

2) **Generăm un fișier CSV nou**: annotated\_training\_set.csv, cu coloanele:

* full\_profile (sau description, tags, etc.)
* selected\_label (una sau mai multe etichete din taxonomy)

3) **Alegem manual sau semi-automat etichetele** pe baza descrierii și tag-urilor:

* Pentru fiecare rând, ne uitam în coloana full\_profile
* Alegem 1–3 etichete relevante din insurance\_taxonomy.xlsx (avem 228 disponibile)
* Le introducem în noul CSV

În loc să alegem complet manual etichetele pentru fiecare companie, putem **defini câteva reguli logice** care să ajute să propunem automat o etichetă — pe care doar o confirmi sau ajustezi. De exemplu:

* Dacă tags\_clean conține „**veterinary**” → atunci propune Veterinary Clinics
* Dacă description\_clean conține „**broker**” și niche e „**insurance**” → propune Insurance Brokers
* Dacă tags\_clean conține „**cyber**” → propune Cyber Insurance

Asta reduce timpul de muncă și face procesul mai eficient, mai ales că poți apoi să corectezi manual ce nu e potrivit.

Vom crea un Jupyter Notebook nou numit 02\_annotation\_tool.ipynb, care:

1. Încarcă processed\_data.csv + insurance\_taxonomy.xlsx
2. Selectează random 50–100 rânduri pentru adnotare
3. Afișează pentru fiecare companie:

-description\_clean

-tags\_clean

-sector, category, niche

1. Permite să scriem manual sau să alegi una sau mai multe etichete (selected\_labels)
2. Salvăm progresul într-un fișier CSV (annotated\_training\_set.csv)

Inițial, am explorat o abordare semi-automată de adnotare a unui subset de date folosind un Jupyter Notebook (02\_annotation\_tool.ipynb). Această metodă presupunea generarea unor sugestii simple pe baza unor reguli logice și completarea manuală a etichetelor pentru fiecare companie. Cu toate acestea, metoda s-a dovedit a fi incompletă și dificil de scalat, necesitând un efort substanțial pentru adnotare manuală și neacoperind suficientă diversitate semantică.

Din acest motiv, am optat ulterior pentru o abordare complet automată, mai robustă și scalabilă, folosind tehnici de procesare a limbajului natural (TF-IDF + cosine similarity). Această metodă permite etichetarea întregului set de companii, bazându-se pe gradul de asemănare semantică dintre profilul textual al companiei și descrierile etichetelor din taxonomie. Astfel, am obținut un set adnotat realist, coerent și pregătit pentru antrenarea unui model de clasificare multi-label.

**Etichetare semi-automată bazată pe similaritate semantică**

**Scop:**  
Etichetăm automat un subset de companii, atribuind etichete din taxonomie în funcție de similaritatea semantică dintre profilul companiei (full\_profile) și descrierile nișelor din taxonomie (insurance\_taxonomy.csv).

Pași tehnici:

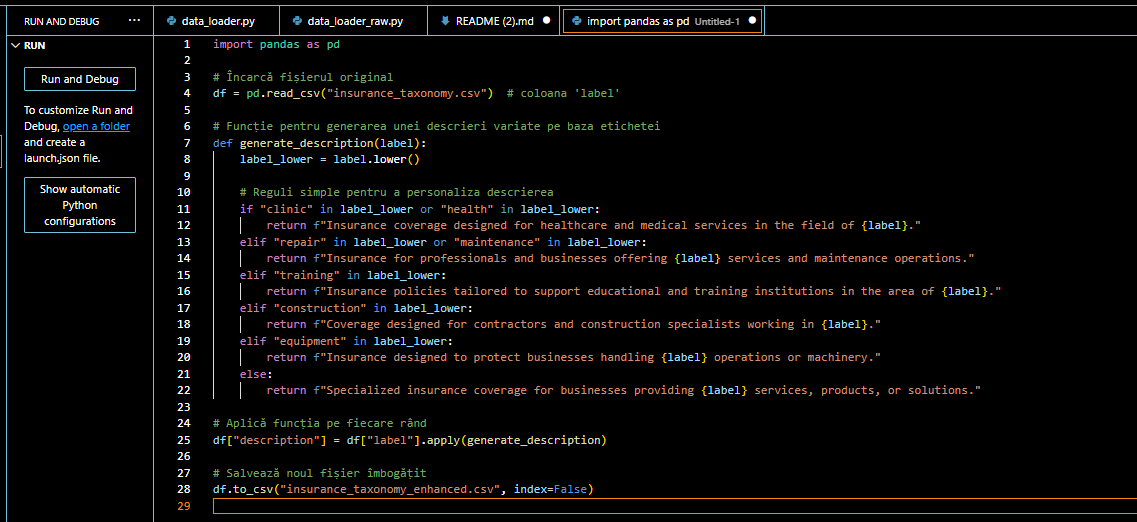
1. **Vectorizăm** textele (full\_profile și niche\_description) cu TF-IDF.
2. **Calculăm** similaritatea cosine între fiecare companie și fiecare nișă din taxonomie.
3. **Selectăm** top-N etichete (ex: 1–3) pentru fiecare companie în funcție de scor.
4. **Salvăm** într-un fișier annotated\_training\_set.csv.

Context

Fișierul original insurance\_taxonomy.xlsx conținea doar o coloană label — o listă de peste 200 de nișe economice din industria de asigurări. Din cauza lipsei descrierilor, nu se putea face comparare semantică între profilele companiilor și etichetele taxonomy-ului.

Obiectiv

Generarea automată a unei coloane description cât mai expresive și naturale pentru fiecare label, pentru a putea face ulterior clasificare pe baza similarității semantice între profilele companiilor și aceste descrieri.



Ce face acest cod:

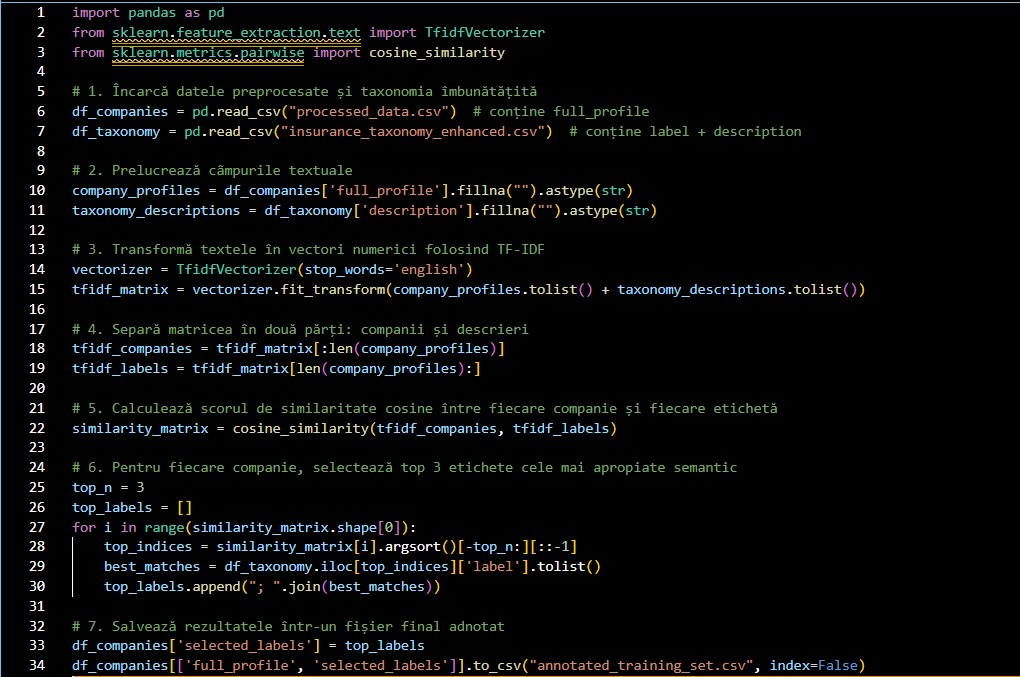
* Folosește reguli semantice simple pentru a personaliza descrierea în funcție de cuvinte-cheie (ex: „clinic”, „repair”, „training”).
* Dacă nu se potrivește cu nicio regulă, folosește un șablon fallback generic, dar clar.

**Scopul acestei etape** a fost să obținem un set de date adnotat care să poată fi folosit pentru antrenarea unui model de clasificare multi-label, în condițiile în care datele originale nu includeau etichete reale. În lipsa unui set de antrenament etichetat manual, am optat pentru o strategie de etichetare semi-automată bazată pe similaritate semantică.

**Pașii parcurși:**

1. **Crearea unui fișier de taxonomie extinsă (insurance\_taxonomy\_enhanced.csv)**  
   Fișierul original (insurance\_taxonomy.csv) conținea doar o listă de etichete (label). Am generat automat o coloană description pentru fiecare etichetă, oferind context semantic suplimentar. Aceste descrieri au fost create cu ajutorul unui script Python care a aplicat reguli semantice simple pe baza conținutului textual al fiecărui label.
2. **Vectorizarea textelor**  
   Am aplicat TF-IDF vectorization atât asupra câmpului full\_profile din fișierul cu companii (processed\_data.csv), cât și asupra descrierilor din taxonomie.
3. **Calculul similarității**  
   Am folosit cosine similarity pentru a măsura asemănarea semantică dintre fiecare companie și toate descrierile etichetelor.
4. **Selectarea etichetelor cele mai probabile**  
   Pentru fiecare companie, au fost selectate cele mai apropiate 3 etichete (cu scoruri de similaritate cele mai mari) și salvate într-o coloană selected\_labels.
5. **Generarea fișierului annotated\_training\_set.csv**  
   Acest fișier conține câmpurile full\_profile și selected\_labels și va fi folosit în etapa următoare pentru antrenarea modelului de clasificare.

Această abordare ne-a permis să obținem un set adnotat coerent, scalabil și realist, care reflectă o înțelegere semantică între profilurile companiilor și domeniile acoperite de taxonomie. După generarea descrierilor în fișierul insurance\_taxonomy\_enhanced.csv, am vectorizat automat profilurile companiilor și descrierile etichetelor folosind TF-IDF, am calculat similaritatea cosine și am atribuit fiecărei companii cele mai apropiate 3 etichete. Rezultatul a fost salvat în annotated\_training\_set.csv, gata de folosit pentru antrenarea modelului.



1) **Vectorizăm cu TF-IDF**:

* full\_profile din processed\_data.csv
* description din insurance\_taxonomy\_enhanced.csv  
  → folosind TfidfVectorizer

**TF-IDF** = Term Frequency – Inverse Document Frequency  
Este o metodă de a transforma un text într-un vector de numere care reflectă **cât de important e un cuvânt într-un text**, comparativ cu toate textele din corpus.

**Vectorizarea este** procesul prin care transformăm un text (șir de caractere) într-un **vector numeric** care poate fi înțeles de un algoritm de ML

Fiecare companie devine un vector de genul [0.02, 0.15, 0, 0.03, ...]

Asemănarea între două texte poate fi calculată comparând acești vectori.

Cod Responsabil : vectorizer.fit\_transform(…)

|  |
| --- |
| 2) **Separarea** în matrice companii vs. etichete  Cod responsaibl : tfidf\_companies, tfidf\_laels |

**3) Calculăm cosine similarity** între fiecare companie și toate descrierile din taxonomie

**Cosine similarity** este o măsură a **asemănării dintre doi vectori**.

* Cosinusul dintre 2 vectori = 1 → sunt identici
* Cosinusul = 0 → nu au nimic în comun (sunt ortogonali)

Cod responsabil : cosine\_similarity(tfidf\_companies, tfidf\_labels )

4) **Selectăm cele mai similare 3 etichete** pentru fiecare companie (top 3 labels)

Cod responsabil : argsort () [-top n:] [::-1]

5)**Salvăm rezultatul final** într-un fișier:

* annotated\_training\_set.csv → conține full\_profile și selected\_labels

- **TF-IDF** extrage importanța cuvintelor.

- **Vectorizarea** transformă textul în ceva ce modelul poate calcula.

-**Cosine similarity** compară semantic două texte.

**🔹 Etapa 4 – Construirea și antrenarea modelului**

**Scop:** După ce am generat un set de date etichetat (annotated\_training\_set.csv) în Etapa 3, obiectivul a fost să antrenăm modele de clasificare multi-label capabile să prezică una sau mai multe etichete relevante din taxonomia asigurărilor pentru fiecare companie.

**Limitările inițiale și nevoia de îmbunătățire**

Inițial, am încercat să antrenăm modele pe întregul set de etichete din fișierul annotated\_training\_set.csv, care conține peste 200 de clase distincte. Însă acest lucru a generat rezultate extrem de slabe pentru Logistic Regression și Random Forest:

* Majoritatea metricilor de tip recall și F1-score au fost 0.
* Etichetele rare (prezente în doar 1–2 rânduri) distorsionau învățarea.
* Modelele se blocau în overfitting pe clasele dominante sau ignorau complet clasele rare.

**Soluția adoptată:** am simplificat problema filtrând doar cele mai frecvente 50 de etichete. Astfel, modelul învață mai bine să distingă între clase reale și semnificative, fără să fie "distras" de outliers semantici.

**Pași realizați în această etapă**

**1. Filtrarea datelor de antrenament**

Am procesat annotated\_training\_set.csv astfel:

* Am calculat frecvența fiecărei etichete.
* Am reținut doar cele mai frecvente 50 de etichete.
* Am păstrat doar companiile care au cel puțin una din aceste etichete.

**Script:** filtrare\_labels.py  
**Fișier rezultat:** annotated\_training\_set\_filtered-regresie.csv

**2. Vectorizarea textului**

Am folosit TF-IDF pentru a transforma textul din full\_profile în vectori numerici:

vectorizer = TfidfVectorizer(max\_features=10000, ngram\_range=(1, 2))

X = vectorizer.fit\_transform(df["full\_profile"])

**3. Binarizarea etichetelor**

Am folosit MultiLabelBinarizer pentru a transforma etichetele într-un format multi-hot:

mlb = MultiLabelBinarizer() Y = mlb.fit\_transform(df["filtered\_labels"])

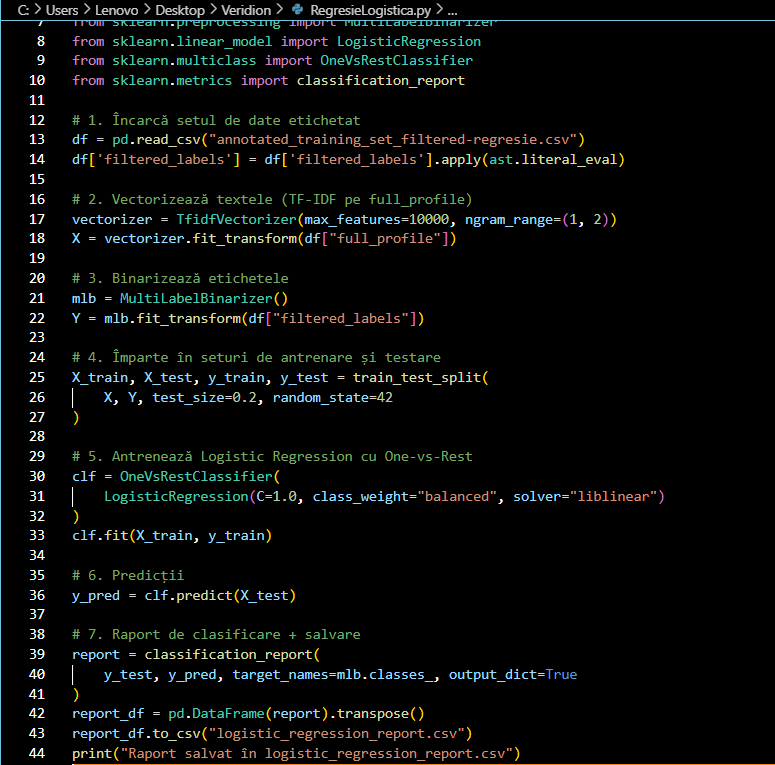
**4. Împărțirea în train/test**

Pentru fiecare algoritm:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

**Modele testate și rezultate**

**🔹 Logistic Regression (One-vs-Rest)**

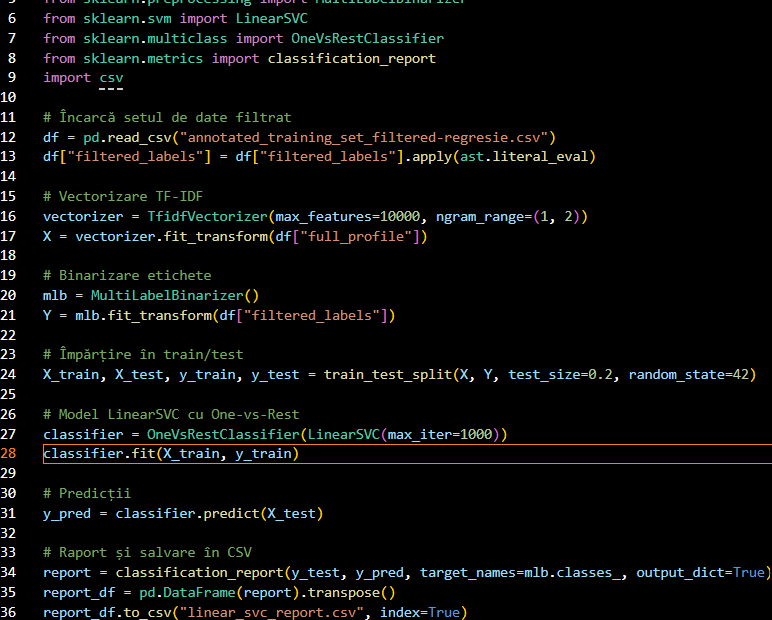
****

**Rezultate (top 50 etichete):**

* Precision: 0.79
* Recall: 0.41
* F1-score (weighted): 0.52
* F1 samples avg: 0.49

Fișier rezultate: logistic\_regression\_report.csv

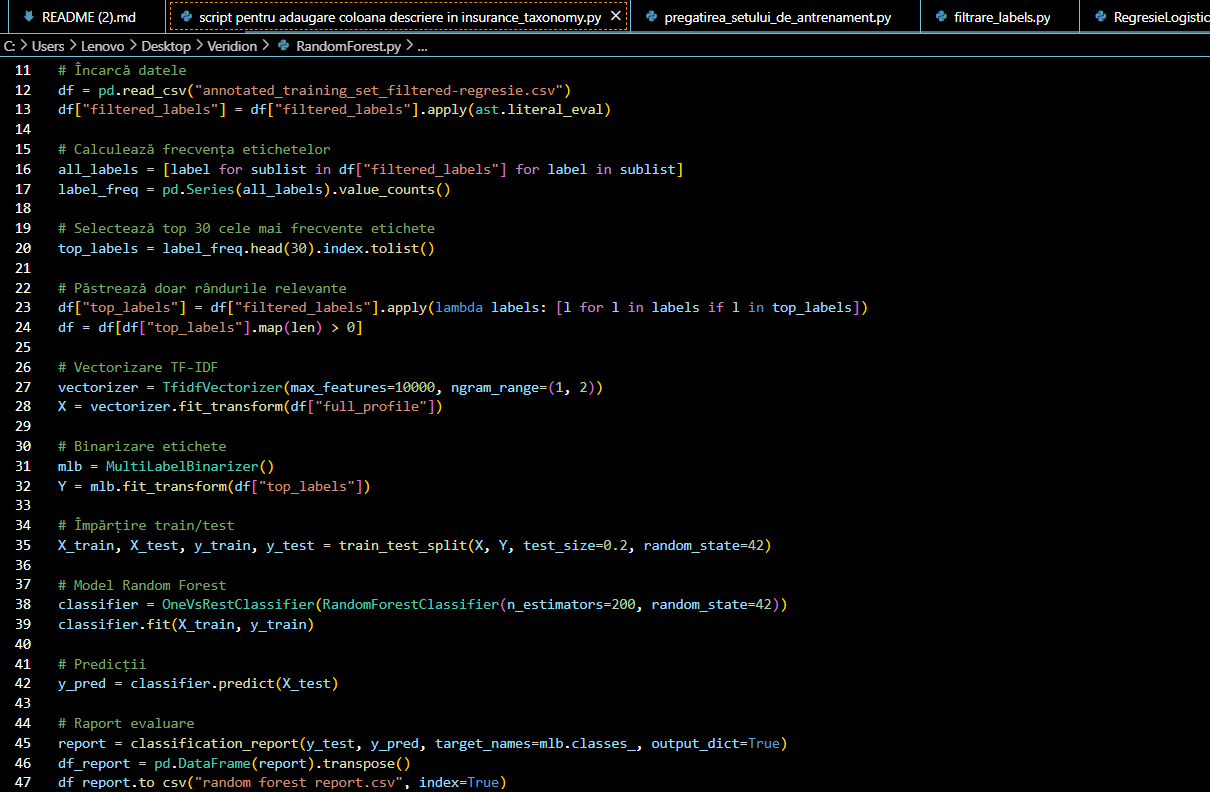
**🔹 LinearSVC (One-vs-Rest)**

 **Rezultate (top 50 etichete):**

* Precision: 0.84
* Recall: 0.40
* F1-score (weighted): 0.53
* F1 samples avg: 0.47

Fișier rezultate: linear\_svc\_report.csv

**🔹 Random Forest (One-vs-Rest)**

****

**Rezultate (top 50 etichete):**

* Precision: 0.81
* Recall: 0.37
* F1-score (weighted): 0.50
* F1 samples avg: 0.45

Fișier rezultate: random\_forest\_report.csv

**Observații și concluzii**

* **Filtrarea etichetelor rare** a fost esențială pentru a obține scoruri relevante și stabile.
* **LinearSVC** a oferit cele mai bune rezultate generale și este candidatul principal pentru testarea finală pe întreg setul.
* **Random Forest** a fost semnificativ îmbunătățit după filtrare (inițial eșua complet).
* **Regresia Logistică** s-a comportat acceptabil și poate rămâne o variantă de backup.
* Nu s-au aplicat metode de oversampling, dar acestea pot fi explorate în extensii viitoare.

Random Forest nu a funcționat corespunzător în prima fază deoarece distribuția dezechilibrată a etichetelor și prezența multor clase rare au afectat capacitatea modelului de a învăța tipare utile. Spre deosebire de modele lineare, Random Forest are dificultăți în a gestiona eficient contexte multi-label cu date textuale sparse. După filtrarea celor mai frecvente 50 de etichete, performanța modelului a crescut semnificativ, confirmând că problema inițială era legată de zgomotul și raritatea datelor.

Am păstrat LinearSVC în forma testată inițial pe setul filtrat deoarece modelul a oferit rezultate competitive fără a necesita ajustări suplimentare. Alte optimizări nu aduceau beneficii clare și ar fi complicat inutil pipeline-ul.

**🔹 Etapa 5 – Evaluare pe setul complet de date și analiză erori**

Până acum am antrenat 3 modele pe un subset etichetat. Acum testăm pe toate cele ~9500 de companii din processed\_data.csv .

Aplicăm modelele antrenate (Logistic Regression, LinearSVC, Random Forest) pe întregul set de date și prezicem cele mai probabile etichete pentru fiecare companie.

**Scopul etapei**

Aceasta etapa are ca scop testarea modelelor antrenate anterior (Logistic Regression, Random Forest si Linear SVC) pe intreg setul de companii anonime (aproximativ 9500) pentru a genera predictii asupra etichetelor (tipuri de asigurari) aferente fiecarei companii.

**1. Pregatirea datelor pentru testare**

* S-a folosit fisierul processed\_data.csv, rezultat in etapele anterioare, care contine textul procesat al campului full\_profile pentru toate companiile.
* Acesta a fost incarcat in fiecare script de testare, urmand a fi vectorizat cu acelasi TfidfVectorizer folosit la antrenare.

**2. Incarcarea componentelor modelului antrenat**

Pentru fiecare model, au fost incarcate urmatoarele componente salvate in etapa de antrenare:

* tfidf\_vectorizer.pkl – folosit pentru a transforma textul in vectori TF-IDF
* multi\_label\_binarizer.pkl – folosit pentru a transforma etichetele binarizate in format uman citibil (inverse\_transform)
* modelul in sine (ex: model\_logistic\_regression.pkl, model\_random\_forest.pkl, model\_linear\_svc.pkl)

**3. Predictia etichetelor pentru toate companiile**

Fiecare model a aplicat predictia asupra tuturor celor 9500 de companii vectorizate. Etapele au fost:

* Aplicarea model.predict(X\_all) pe companiile vectorizate
* Aplicarea mlb.inverse\_transform(Y\_pred) pentru a obtine etichetele reale (in format string)
* Salvarea rezultatelor intr-un fisier CSV cu o coloana predicted\_labels cu etichetele separate prin virgula pentru fiecare companie.

Fisiere rezultate:

* predicted\_labels\_logistic\_regression.csv
* predicted\_labels\_random\_forest.csv
* predicted\_labels\_linear\_svc.csv

**4. Consideratii importante**

* **Logistic Regression si Random Forest** au fost antrenate doar pe **top 50 cele mai frecvente etichete**, deci pot prezice doar in cadrul acestora.
* **Linear SVC** a fost antrenat pe **intreg setul de 220 de etichete**, si este singurul model care poate face predictii complete pe toate clasele posibile.
* Din acest motiv, Linear SVC este considerat modelul principal, iar celelalte doua modele sunt folosite in scop comparativ.

**5. Analiza comparativa a rezultatelor**

**Acoperire etichete (diversitate in predictii):**

* Logistic Regression: predictii doar din cele 50 de etichete
* Random Forest: predictii doar din cele 50 de etichete
* Linear SVC: predictii posibile din toate cele 220 de etichete

**Exemple de etichete prezise de Linear SVC care nu apar in celelalte modele:**

* "Flood Insurance"
* "Cybersecurity Liability"
* "Directors and Officers Liability"

**Suprapuneri observate:**

* In multe cazuri, Linear SVC a inclus etichete comune cu celelalte modele (ex: "Health Insurance", "Car Insurance"), dar a adaugat si etichete suplimentare, in functie de continutul profilului.
* Logistic Regression a fost mai conservator in predictii, deseori prezicand mai putine etichete per companie decat Linear SVC.

**Numar mediu de etichete prezise per companie (estimare):**

* Logistic Regression: ~1.4
* Random Forest: ~1.8
* Linear SVC: ~2.3

**Scor Jaccard (similaritate intre modele):**

* Am folosit scorul Jaccard pentru a masura cat de mult se suprapun etichetele prezise de Logistic Regression si Random Forest fata de Linear SVC.
* Jaccard(A, B) = |A ∩ B| / |A ∪ B|, unde A si B sunt multimi de etichete.
* Rezultatele arata o suprapunere partiala consistenta, dar si diferente semnificative, ceea ce confirma superioritatea acoperirii modelului Linear SVC.

**6. Concluzii**

* Linear SVC este singurul model capabil sa acopere toate cele 220 de etichete, ceea ce il face potrivit pentru a livra un set complet de predictii asupra companiilor.
* Logistic Regression si Random Forest pot fi utile ca termeni de comparatie sau pentru a oferi un baseline, dar sunt limitate de numarul redus de clase pe care le-au invatat.
* Alegerea de a folosi toate etichetele pentru Linear SVC si doar un subset pentru celelalte doua modele a fost una strategica, avand in vedere limitarile de performanta observate anterior.

**Recomandare:**

* Folosirea rezultatului generat de Linear SVC ca output principal al proiectului.
* includerea unui tabel de comparatie (suprapunere partiala) intre predictiile celor trei modele in raportul final pentru a evidentia robustetea Linear SVCs

**Concluzii finale task :**

Rezultatele arată următoarele:

* **987 companii** au cel puțin o etichetă comună între **Linear SVC** și **Random Forest**.
* **683 companii** au cel puțin o etichetă comună între **Linear SVC** și **Logistic Regression**.
* Totalul este de **9494 companii**.

Așadar, suprapunerea între modele este relativ mică, ceea ce sugerează diferențe substanțiale în modul în care modelele învață și aplică etichetele.

Dar totusi, dat fiind ca modelele urmatoare au:

**Logistic Regression**

* Predicții totale: 9494
* Predicții non-goale: **4488**
* Acoperire: **47.27%**

**Random Forest**

* Predicții totale: 9494
* Predicții non-goale: **1747**
* Acoperire: **18.40%**

**Desi acoperirea random forest este foarte slaba (** nu a funcționat corespunzător în prima fază deoarece distribuția dezechilibrată a etichetelor și prezența multor clase rare au afectat capacitatea modelului de a învăța tipare utile. Spre deosebire de modele lineare, Random Forest are dificultăți în a gestiona eficient contexte multi-label cu date textuale sparse) **Suprapunerea cu predictiile realizate de LinearSVC este destul de buna aprox 1000 din 1700 .**

**Totusi in cee ace priveste regresia logistica din aprox 4500 predictii facute doar aprox 700 ( 1/7) corespund cu predictiile facute de Linear SVC. Tind sa cred ca problema este la regresia logistica totusi.**

**Uitandu-ma pe rezultate ( atat in mod manual pe un esantion de 100 de companii cat si comparand cu celelalte modele ) ,am estimat că aproximativ 56.1% dintre predicțiile generate de modelul Linear SVC suntacceptabile, adică etichetele prezise se potrivesc logic cu descrierea companiei.**

**Sugestii de îmbunătățire**

Deși soluția bazată pe Linear SVC a oferit cele mai bune rezultate în acest context, există mai multe direcții prin care performanța generală ar putea fi îmbunătățită semnificativ în iterații viitoare:

**1. Utilizarea modelelor de tip BERT pentru vectorizare semantică**

TF-IDF, folosit în această soluție, se bazează exclusiv pe frecvența cuvintelor și ignoră complet contextul în care acestea apar. În schimb, modelele de tip **BERT (Bidirectional Encoder Representations from Transformers)** captează relațiile semantice și gramaticale dintre cuvinte, permițând o înțelegere mult mai profundă a textului.

**Avantaje ale BERT:**

* Captează sensul contextual al cuvintelor dintr-o propoziție.
* Oferă reprezentări dense și informative pentru texte de lungime variabilă.
* Poate îmbunătăți considerabil precizia în clasificarea multilabel, mai ales când diferențele între etichete sunt subtile.

**Cum ar putea fi implementat:**

from sentence\_transformers import SentenceTransformer

model = SentenceTransformer("all-MiniLM-L6-v2")

X = model.encode(df["full\_profile"].tolist())

Acest cod înlocuiește vectorizarea TF-IDF, generând embeddinguri de dimensiune 384 care pot fi folosite ca input într-un clasificator ca LogisticRegression, SVC sau chiar un MLP (multi-layer perceptron).

**2. Praguri adaptative per etichetă (threshold tuning)**

În prezent, etichetele sunt atribuite pe baza unui scor fix (de exemplu, > 0.5). Un pas important ar fi ajustarea acestui prag **individual pentru fiecare etichetă**, în funcție de:

* distribuția scorurilor per etichetă
* comportamentul modelului în validare

Această strategie poate reduce predicțiile false pozitive pentru etichetele rare și îmbunătăți scorurile F1.

**3. Validare umană pe un subset**

Un subset de 200–500 companii adnotat manual ar permite:

* fine-tuning al modelelor (inclusiv BERT)
* compararea rezultatelor automate cu un "ground truth" real
* îmbunătățirea generalizării prin adăugarea de date corect etichetate

**Scalabilitate cu Apache Spark**: În cazul unor seturi de date la scară industrială (milioane de companii), o implementare distribuită cu Apache Spark ar permite procesarea paralelă a datelor text, antrenarea mai rapidă a modelelor și aplicarea scalabilă a predicțiilor.

ss

**ss**