

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 grupăm 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
3. 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 apartine 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 și usor de înțeles 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 tagurile 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 sir 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 procesare** :

```
1 import pandas as pd
2 import ast
3
4 # 1. Incarca fisierul CSV original
5 df = pd.read_csv("ml_insurance_challenge.csv")
6
7 # 2. Normalizeaza coloana 'description'
8 # Scop: sa fie folosita ca text pentru vectorizare (TF-IDF, embeddings etc.)
9 df['description_clean'] = (
10     df['description']
11         .fillna('')                                # inlocuieste valorile lipsa cu sir gol
12         .str.lower()                             # conversie la lowercase
13         .str.replace(r'^[a-z0-9\s]', '', regex=True) # elimina caractere speciale
14 )
15
16 # 3. Parseaza coloana 'business_tags' (care e un sir ce reprezinta o lista)
17 # Scop: sa devina un sir de cuvinte-cheie pentru clasificare
18 def parse_tags(tag_str):
19     try:
20         tags = ast.literal_eval(tag_str)      # converteste string-ul intr-o lista Python
21         if isinstance(tags, list):
22             return " ".join([str(tag).lower().replace(" ", "_") for tag in tags])
23         else:
24             return ""
25     except:
26         return ""
27
28 df['tags_clean'] = df['business_tags'].fillna([]).apply(parse_tags)
29
30 # 4. Curata campurile categorice: 'sector', 'category', 'niche'
31 # Scop: sa fie coerente, fara simboluri si cu valori complete
32 for col in ['sector', 'category', 'niche']:
33     df[col] = df[col].fillna('unknown') \
34         .str.lower() \
35         .str.replace(r'^[a-z0-9\s]', '', regex=True)
36
37 # 5. Creeaza campul 'full_profile'
38 # Scop: sa combine toate informatiile relevante intr-un singur camp textual
39 df['full_profile'] = (
40     df['description_clean'] + " " +
41     df['tags_clean'] + " " +
42     df['category'] + " " +
43     df['niche']
44 )
45
46 # 6. Salveaza rezultatul intr-un fisier CSV nou
47 df.to_csv("processed_data.csv", index=False)
```

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, golarile sunt completate cu şiruri 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 sir 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. Afisează pentru fiecare companie:
 - `description_clean`
 - `tags_clean`
 - `sector, category, niche`
4. Permite să scriem manual sau să alegi una sau mai multe etichete (`selected_labels`)
5. Salvăm progresul într-un fișier CSV (`annotated_training_set.csv`)

Initial, am explorat o abordare semi-automată de adnotare a unui subset de date folosind un Jupyter Notebook (`02_annotation_tool.ipynb`). Această metodă presupune 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.

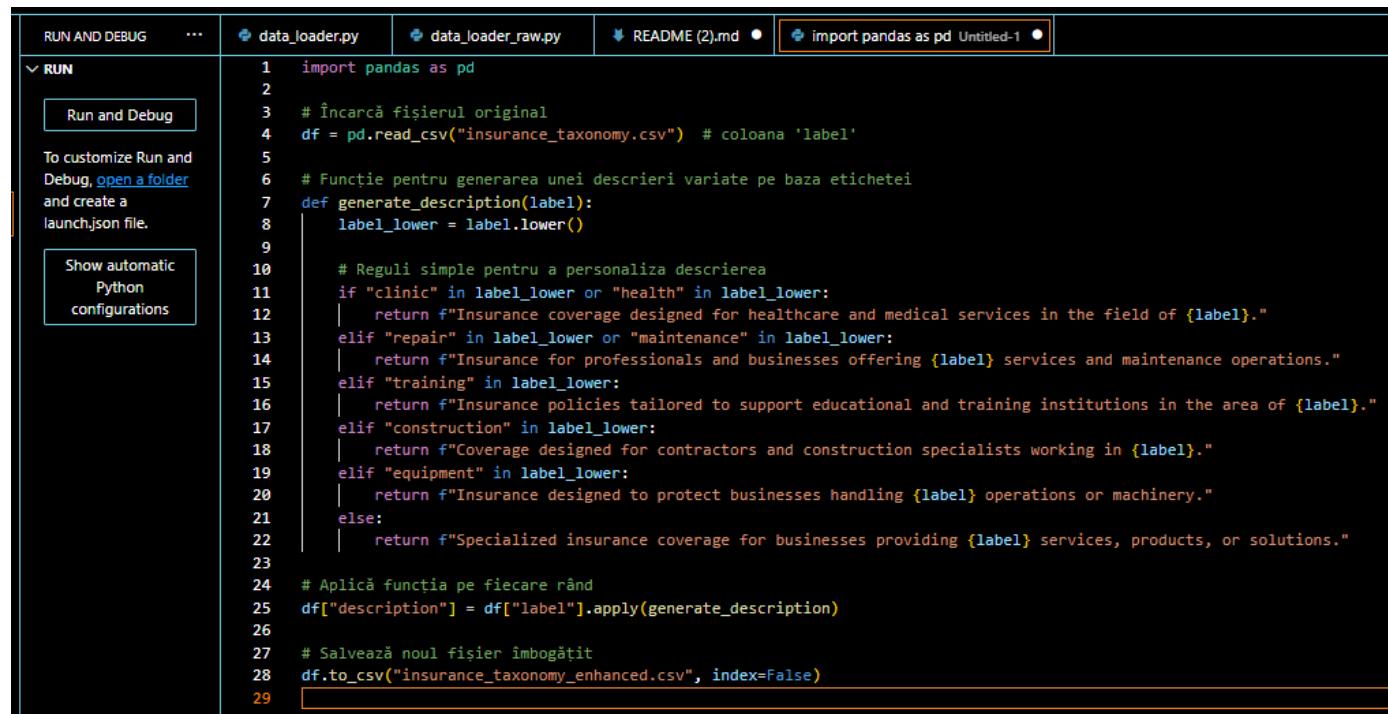
3. Selectăm top-N etichete (ex: 1–3) pentru fiecare companie în funcție de scor.
4. Salvăm într-un fișier annotated_training_set.csv.

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.



The screenshot shows a Jupyter Notebook interface with the following details:

- Run and Debug** button is highlighted.
- RUN** dropdown menu is open, showing **Run and Debug**.
- To customize Run and Debug, open a folder and create a launch.json file.**
- Show automatic Python configurations** button is highlighted.
- data_loader.py**, **data_loader_raw.py**, **README (2).md**, and **import pandas as pd Untitled-1** are listed in the notebook tabs.
- Untitled-1** tab contains the following Python code:

```

1 import pandas as pd
2
3 # Încarcă fișierul original
4 df = pd.read_csv("insurance_taxonomy.csv") # coloana 'label'
5
6 # Funcție pentru generarea unei descrieri variate pe baza etichetei
7 def generate_description(label):
8     label_lower = label.lower()
9
10    # Reguli simple pentru a personaliza descrierea
11    if "clinic" in label_lower or "health" in label_lower:
12        return f"Insurance coverage designed for healthcare and medical services in the field of {label}."
13    elif "repair" in label_lower or "maintenance" in label_lower:
14        return f"Insurance for professionals and businesses offering {label} services and maintenance operations."
15    elif "training" in label_lower:
16        return f"Insurance policies tailored to support educational and training institutions in the area of {label}."
17    elif "construction" in label_lower:
18        return f"Coverage designed for contractors and construction specialists working in {label}."
19    elif "equipment" in label_lower:
20        return f"Insurance designed to protect businesses handling {label} operations or machinery."
21    else:
22        return f"Specialized insurance coverage for businesses providing {label} services, products, or solutions."
23
24 # Aplică funcția pe fiecare rând
25 df["description"] = df["label"].apply(generate_description)
26
27 # Salvează noul fișier imbogătit
28 df.to_csv("insurance_taxonomy_enhanced.csv", index=False)
29

```

Ce face acest cod:

- Folosește reguli semantice simple pentru a personaliza descrierea în funcție de cuvintele cheie (ex: „clinic”, „repair”, „training”).
- Dacă nu se potrivește cu nicio regulă, folosește un şablon 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 parcursi:

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 apropriate 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 coherent, 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 apropriate 3 etichete. Rezultatul a fost salvat în annotated_training_set.csv, gata de folosit pentru antrenarea modelului.

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 from sklearn.metrics.pairwise import cosine_similarity
4
5 # 1. Încarcă datele preprocesate și taxonomia îmbunătățită
6 df_companies = pd.read_csv("processed_data.csv") # conține full_profile
7 df_taxonomy = pd.read_csv("insurance_taxonomy_enhanced.csv") # conține label + description
8
9 # 2. Prelucrarează câmpurile textuale
10 company_profiles = df_companies['full_profile'].fillna("").astype(str)
11 taxonomy_descriptions = df_taxonomy['description'].fillna("").astype(str)
12
13 # 3. Transformă textele în vectori numerică folosind TF-IDF
14 vectorizer = TfidfVectorizer(stop_words='english')
15 tfidf_matrix = vectorizer.fit_transform(company_profiles.tolist() + taxonomy_descriptions.tolist())
16
17 # 4. Separă matricea în două părți: companii și descrieri
18 tfidf_companies = tfidf_matrix[:len(company_profiles)]
19 tfidf_labels = tfidf_matrix[len(company_profiles):]
20
21 # 5. Calculează scorul de similaritate cosine între fiecare companie și fiecare etichetă
22 similarity_matrix = cosine_similarity(tfidf_companies, tfidf_labels)
23
24 # 6. Pentru fiecare companie, selectează top 3 etichete cele mai apropriate semantic
25 top_n = 3
26 top_labels = []
27 for i in range(similarity_matrix.shape[0]):
28     top_indices = similarity_matrix[i].argsort()[-top_n:][::-1]
29     best_matches = df_taxonomy.iloc[top_indices]['label'].tolist()
30     top_labels.append("; ".join(best_matches))
31
32 # 7. Salvează rezultatele într-un fișier final adnotat
33 df_companies['selected_labels'] = top_labels
34 df_companies[['full_profile', 'selected_labels']].to_csv("annotated_training_set.csv", index=False)
```

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 responsabil : tfidf_companies, tfidf_labels

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** compara 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

Înțial, am încercat să antrenăm modele pe întregul set de etichete din fișierul annotated_training_set.csv, care conține peste 200 de clase distințe. 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 învăță 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)

```
C:\Users\Lenovo\Desktop\Veridion> RegresieLogistica.py > ...
1  from sklearn.preprocessing import MultiLabelBinarizer
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.multiclass import OneVsRestClassifier
4  from sklearn.metrics import classification_report
5
6  # 1. Încarcă setul de date etichetat
7  df = pd.read_csv("annotated_training_set_filtered-regresie.csv")
8  df['filtered_labels'] = df['filtered_labels'].apply(ast.literal_eval)
9
10 # 2. Vectorizează textele (TF-IDF pe full_profile)
11 vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
12 X = vectorizer.fit_transform(df["full_profile"])
13
14 # 3. Binarizează etichetele
15 mlb = MultiLabelBinarizer()
16 Y = mlb.fit_transform(df["filtered_labels"])
17
18 # 4. Împarte în seturi de antrenare și testare
19 X_train, X_test, y_train, y_test = train_test_split(
20 |   X, Y, test_size=0.2, random_state=42
21 )
22
23 # 5. Antrenează Logistic Regression cu One-vs-Rest
24 clf = OneVsRestClassifier(
25 |   LogisticRegression(C=1.0, class_weight="balanced", solver="liblinear")
26 )
27 clf.fit(X_train, y_train)
28
29 # 6. Predicții
30 y_pred = clf.predict(X_test)
31
32 # 7. Raport de clasificare + salvare
33 report = classification_report(
34 |   y_test, y_pred, target_names=mlb.classes_, output_dict=True
35 )
36 report_df = pd.DataFrame(report).transpose()
37 report_df.to_csv("logistic_regression_report.csv")
38 print("Raport salvat în logistic_regression_report.csv")
```

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)

```
5  from sklearn.preprocessing import MultiLabelBinarizer
6  from sklearn.svm import LinearSVC
7  from sklearn.multiclass import OneVsRestClassifier
8  from sklearn.metrics import classification_report
9  import csv
10
11 # Încarcă setul de date filtrat
12 df = pd.read_csv("annotated_training_set_filtered-regresie.csv")
13 df[ "filtered_labels" ] = df[ "filtered_labels" ].apply(ast.literal_eval)
14
15 # Vectorizare TF-IDF
16 vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
17 X = vectorizer.fit_transform(df[ "full_profile" ])
18
19 # Binarizare etichete
20 mlb = MultiLabelBinarizer()
21 Y = mlb.fit_transform(df[ "filtered_labels" ])
22
23 # Împărțire în train/test
24 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
25
26 # Model LinearSVC cu One-vs-Rest
27 classifier = OneVsRestClassifier(LinearSVC(max_iter=1000))
28 classifier.fit(X_train, y_train)
29
30 # Predicții
31 y_pred = classifier.predict(X_test)
32
33 # Raport și salvare în CSV
34 report = classification_report(y_test, y_pred, target_names=mlb.classes_, output_dict=True)
35 report_df = pd.DataFrame(report).transpose()
36 report_df.to_csv("linear_svc_report.csv", index=True)
```

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)

```
1 # Încarcă datele
2 df = pd.read_csv("annotated_training_set_filtered-regresie.csv")
3 df[["filtered_labels"]] = df[["filtered_labels"]].apply(ast.literal_eval)
4
5 # Calculează frecvența etichetelor
6 all_labels = [label for sublist in df[["filtered_labels"]] for label in sublist]
7 label_freq = pd.Series(all_labels).value_counts()
8
9 # Selectează top 30 cele mai frecvente etichete
10 top_labels = label_freq.head(30).index.tolist()
11
12 # Păstrează doar rândurile relevante
13 df[["top_labels"]] = df[["filtered_labels"]].apply(lambda labels: [l for l in labels if l in top_labels])
14 df = df[df[["top_labels"]].map(len) > 0]
15
16 # Vectorizare TF-IDF
17 vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
18 X = vectorizer.fit_transform(df[["full_profile"]])
19
20 # Binarizare etichete
21 mlb = MultilabelBinarizer()
22 Y = mlb.fit_transform(df[["top_labels"]])
23
24 # Împărțire train/test
25 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
26
27 # Model Random Forest
28 classifier = OneVsRestClassifier(RandomForestClassifier(n_estimators=200, random_state=42))
29 classifier.fit(X_train, y_train)
30
31 # Predicții
32 y_pred = classifier.predict(X_test)
33
34 # Raport evaluare
35 report = classification_report(y_test, y_pred, target_names=mlb.classes_, output_dict=True)
36 df_report = pd.DataFrame(report).transpose()
37 df_report.to_csv("random_forest_report.csv", index=True)
```

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 (initial 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 și Linear SVC) pe intreg setul de companii anonte (aproximativ 9500) pentru a genera predictii asupra etichetelor (tipuri de asigurari) aferente fiecarii companii.

1. Pregatirea datelor pentru testare

- S-a folosit fisierul processed_data.csv, rezultat în etapele anterioare, care conține textul procesat al campului full_profile pentru toate companiile.
- Aceasta a fost încarcat în fiecare script de testare, urmand să fie vectorizat cu același TfidfVectorizer folosit la antrenare.

2. Încarcarea componentelor modelului antrenat

Pentru fiecare model, au fost încarcate urmatoarele componente salvate în etapa de antrenare:

- tfidf_vectorizer.pkl – folosit pentru a transforma textul în vectori TF-IDF
- multi_label_binarizer.pkl – folosit pentru a transforma etichetele binarizate în format uman citibil (inverse_transform)
- modelul în 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:

- În multe cazuri, Linear SVC a inclus etichete comune cu celelalte modele (ex: "Health Insurance", "Car Insurance"), dar a adăugat și etichete suplimentare, în funcție de continutul profilului.
- Logistic Regression a fost mai conservator în predicții, deseori prezicând mai puține etichete per companie decât Linear SVC.

Număr mediu de etichete prezise per companie (estimare):

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

Scor Jaccard (similaritate între modele):

- Am folosit scorul Jaccard pentru a măsura cât de mult se suprapun etichetele prezise de Logistic Regression și Random Forest față de Linear SVC.
- $Jaccard(A, B) = |A \cap B| / |A \cup B|$, unde A și B sunt multimi de etichete.
- Rezultatele arată o suprapunere parțială consistentă, dar și diferențe semnificative, ceea ce confirmă superioritatea acoperirii modelului Linear SVC.

6. Concluzii

- Linear SVC este singurul model capabil să acopere toate cele 220 de etichete, ceea ce îl face potrivit pentru a livra un set complet de predicții asupra companiilor.
- Logistic Regression și 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.
- Alegera de a folosi toate etichetele pentru Linear SVC și doar un subset pentru celelalte două modele a fost una strategic, având în vedere limitările de performanță observate anterior.

Recomandare:

- Folosirea rezultatului generat de Linear SVC ca output principal al proiectului.
- includerea unui tabel de comparatie (suprapunere parțială) între predicțiile celor trei modele în raportul final pentru a evidenția 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 substantiale în modul în care modelele învăță 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 slabă (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 sunt acceptabile, 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, permitând 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.