Universidade Federal de São Carlos
Ciência da Computação
Profº Murilo Coelho Naldi

Classificação Hierárquica:
DBPedia Classes

Ana Ellen Deodato P Silva 800206

Vinicius de Oliveira Guimarães 802431

Vinícius Gonçalves Perillo 800219

São Carlos, 02 de dezembro de 2023

1 - O PROBLEMA

O nosso desafio consiste na análise, processamento e classificação de uma base de dados de linguagem natural que possui uma hierarquia de classes, ou seja, realizar classificação hierárquica com a maior quantidade de acertos possíveis. Para isso, utilizamos de várias técnicas de pré-processamento, de modelos de linguagem natural, treinamento e classificação.

2 - SOBRE A BASE DE DADOS

A base de dados que será trabalhada consiste de vários textos de artigos do Wikipédia (Mais especificamente 60794 registros para teste e 240942 registros de treino), onde cada texto possui uma hierarquia de 3 classes: L1 (Classe mais genérica), L2 (Classe um pouco mais específica) e L3 (Classe ainda mais específica). O dataset pode ser encontrado no Kaggle.

Os atributos desta base são:

- “text”: Texto completo de uma página da Wikipédia
- “l1”: Categoria de nível 1
- “l2”: Categoria de nível 2
- “l3”: Categoria de nível 3

Imagem 1 - Primeiras 10 linhas do *dataset*

	text	l1	l2	l3
2486	Estampes (Prints), L.100, is a composition for...	Work	MusicalWork	ClassicalMusicComposition
64600	Ceratocystis oblonga is a plant-pathogenic sap...	Species	Eukaryote	Fungus
81657	In Greek mythology, Mermeros and Pheres were t...	Agent	FictionalCharacter	MythologicalFigure
140339	Kristen Veal (born 24 July 1981) is an Austral...	Agent	Athlete	BasketballPlayer
169079	Ishii Hisaichi's CNN (いしいひさいちのCNN Ishii Hisaic...	Work	Comic	Manga
71779	Witmer v. United States, 348 U.S. 375 (1955), ...	UnitOfWork	LegalCase	SupremeCourtOfTheUnitedStatesCase
170566	The Whitehorse Mountains are a mountain range ...	Place	NaturalPlace	MountainRange
149673	The Batasang Bayan (English: Legislative Advis...	Agent	Organisation	Legislature
171470	The Caspian tern (Hydroprogne caspia, formerly...	Species	Animal	Bird
142558	Debiprasad Chattopadhyaya (19 November 1918 – ...	Agent	Person	Philosopher

2.2 - ATRIBUTO “l1”

Considerando a nossa base de treino, a coluna L1 contém 9 valores diferentes com as frequências mostradas na imagem 2.

Imagem 2 - Valores e suas frequências da coluna L1

```
[26] data_train['l1'].value_counts()

Agent             124798
Place              45877
Species            21472
Work               21013
Event              19106
SportsSeason        5883
UnitOfWork          1761
TopicalConcept       784
Device               248
Name: l1, dtype: int64
```

2.3 - ATRIBUTO “l2”

Considerando também a base de treino, a coluna L2 contém 70 valores únicos, cada um com as frequências mostradas na imagem 3 (Devido a quantidade alta de valores únicos, alguns foram omitidos).

Imagem 3 - Valores e suas frequências da coluna L2

```
[27] data_train['l2'].value_counts()
```

```
Athlete             31111
Person              19504
Animal              14682
Building            10704
Politician           9504
                     ...
MusicalArtist         198
RaceTrack             172
ComicsCharacter       144
VolleyballPlayer      137
Database              129
Name: l2, Length: 70, dtype: int64
```

2.4 - ATRIBUTO “l3”

Já para a coluna L3, temos no total 219 valores únicos, igual ao mostrado na imagem 4, o que faz dela a classe mais específica dentre as 3 existentes.

Imagem 4 - Valores e suas frequências da coluna L3

```
[28] data_train['l3'].value_counts()
```

```
AcademicJournal          1924
Manga                    1924
FigureSkater             1924
OlympicEvent             1923
SoccerTournament         1922
                         ...
Cycad                     145
AnimangaCharacter         144
BeachVolleyballPlayer     137
CanadianFootballTeam      133
BiologicalDatabase        129
Name: l3, Length: 219, dtype: int64
```

3 - PRÉ-PROCESSAMENTO

Inicialmente, ao analisarmos cada uma das colunas, é perceptível a inexistência tanto de valores faltantes quanto de valores repetidos no atributo “text”, não necessitando de correções.

Para transformar o texto em algo útil para o processamento e treinamento é necessário que esse texto seja transformado em um array de vetores (Valores decimais) que representam as relações do texto com base em um conjunto de palavras de um modelo de PLN (Processamento de Linguagem Natural).

Primeiramente os textos foram *tokenizados*, ou seja, os textos foram quebrados em lista de palavras e símbolos significativos para se iniciar a análise. Após isso, foram removidas as *stopwords*, palavras que servem para conectar as palavras com mais significado do texto como preposições, artigos, pronomes e conjunções. Com essa lista mais limpa os tokens foram *lemmatizados*, isso significa desfazer todas as conjugações e qualquer outra transformação que substantivos possam ter para reduzir mais ainda a quantidade de *tokens* diferentes. Na tabela 1 temos o código da implementação dessas técnicas.

Tabela 1 - Técnicas de PLN

```
# Contém o conjunto de palavras irrelevantes do idioma inglês
stop_words = set(nltk.corpus.stopwords.words('english'))
lemmatizer = nltk.stem.WordNetLemmatizer()

# Função para processar o texto da coluna "text" e gerar um array contendo apenas os tokens mais relevantes
def process_text(sentence):
  global stop_words tokens = nltk.tokenize.word_tokenize(sentence, language='english')
  return [lemmatizer.lemmatize(word).lower() for word in tokens if word.lower().strip() not in stop_words]
```

Depois de ter o array que representa os tokens mais importantes, é preciso transformar essas palavras em inglês para valores numéricos de forma a conseguir realizar manipulações e treinamento.

Para tal ação, escolhemos a técnica de Word Embedding, essa técnica foi escolhida pois leva em consideração o significado das palavras já que ela permite que palavras estejam posicionadas em uma espaço em que cada dimensão representa uma escala de significado e não apenas sua frequência nos textos. Utilizamos um modelo de representação de palavras pré-treinado do GloVe (PENNINGTON, J; SOCHER, R; MANNING, C, 2014) para fazer essa extração de atributos. O modelo é um junção do Wikipedia 2014 e Gigaword 5, ele foi treinado com 6 bilhões de tokens e possui 400 mil palavras vetorizadas com a possibilidade de esp

Primeiramente optamos pela opção de 300 dimensões, entretanto estamos estourando a memória da máquina virtual do Google Colab na técnica de aprendizado plano. Com isso, migramos para a opção de 50 dimensões.

Como a técnica vetoriza cada token, em cada texto conseguimos um vetor de vetores, para que tenhamos apenas um vetor por texto fizemos uma média entre os vetores dos tokens de uma mesmo texto para obter a posição do texto no espaço. Na tabela 2 temos essa estratégia implementada.

Tabela 2 - Extração de atributos

```
def word2vec_mean(tokens):
  vectors = []
  for t in tokens:
    try:
      vectors.append(word_vectors[t.lower()])
```

```
    except KeyError:
      print(f'{t.lower()} not present')
  return np.mean(vectors, axis= 0)
```

4 - APLICAÇÃO DO MODELO

A fim de conseguir realizar uma comparação entre diferentes abordagens de classificação, optamos por fazer a classificação plana e a classificação hierárquica local por nível e em seguida analisar os seus resultados e diferenças.

4.1 - CLASSIFICAÇÃO PLANA

O objetivo da classificação plana é desconsiderar os níveis de hierarquia e ir direto para a determinação das classes mais específicas, que no nosso caso estão contidas na coluna L3. Através dessa forma de classificação conseguiremos saber dada uma classe de L3, a classe superior em L2 e a classe superior em L1.

Para começar o treinamento e classificação com a estratégia plana, foi preciso utilizar o arquivo glove.6B.50d.txt para transformar, para cada linha do dataset, o array de tokens em um array de vetores (Valores numéricos) que relacionam cada token com tópicos de palavras inglesas. Com isso, de forma a gerar comparação, executamos os modelos de classificação Random Forest e Regressão Logística somente para a coluna L3.

Seguindo para o modelo de Random Forest, foram utilizadas as configurações n_estimators e random_state com valores 100 e 42 respectivamente. Já para a Regressão Logística, utilizamos a propriedade max_iter como sendo 1000, de forma a encontrar uma conversão mais adequada aos dados. Um ponto a ser lembrado é que a coluna “l3” tem 219 classes diferentes, o que dificulta muito o treinamento. Na tabela 3 temos o código utilizado para o treinamento (240942 registros) e teste (60794) com regressão logística (O mesmo se aplica para Random Forest):

Tabela 3 - Treinamento da Random Forest

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import copy

copy train_x = copy.deepcopy(train)

# Separamos a base em treino e teste
# Estamos considerando apenas as classes da coluna “l3"
train_x train_y_l3 = data[data['sep'] == 'train']['l3']
train_y_l3 test_y_l3 = data[data['sep'] == 'test']['l3']
test_x = copy.deepcopy(test)

modelo_l3 = LogisticRegression(max_iter=1000, random_state=42)
modelo_l3.fit(train_x, train_y_l3) predicts_l3 = modelo_l3.predict(test_x)
```

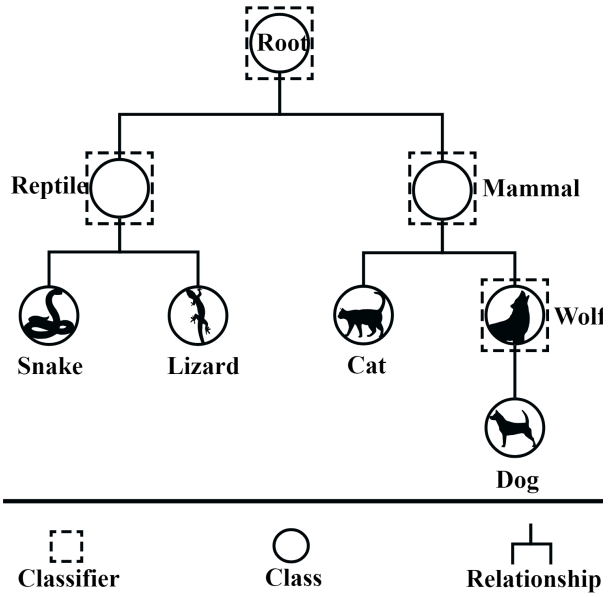
4.2 - CLASSIFICAÇÃO LOCAL POR PAI

Já o objetivo da classificação local por pai é conseguir obter todos os níveis de hierarquia. Os passos iniciais dessa forma de classificação é manipular os dados e criar modelos para cada hierarquia/nível de classes. Ou seja, primeiramente é criado um modelo para classificar todas as tuplas com relação a primeira coluna “l1” (9 classes diferentes) e em seguida, para cada conjunto de tuplas classificados para cada classe de “l1”, realizar uma construção de modelo para classificar esse conjunto de tuplas com relação agora à coluna “l2”. Seguindo a mesma ideia, um modelo é criado para cada valor diferente da coluna “l2”, classificando mais especificamente as tuplas com relação às classes de “l3” conectadas com a classe específica de “l2”.

Na imagem 5 temos um exemplo de representação da classificação por pai, onde será criado um modelo diferente para cada classe existente nas colunas

superiores à atual. Na imagem, primeiramente os dados são classificados entre répteis e mamíferos, com isso, o *dataset* será segmentado entre as duas classes. Com isso teremos um modelo para classificar os mamíferos entre lobos e gatos e um modelo para classificar répteis entre cobra e lagarto. Por fim, dos classificados como lobo, terá um modelo para classificar em cachorros.

Imagem 5 - Exemplo de Classificação Local por Pai



Como métodos de classificação escolhemos usar a Regressão Linear e a Random Forest:

- Regressão linear: melhor para hierarquias mais simples (relações lineares e tem interpretabilidade boa
- Random Forest: melhor para hierarquias com relações não lineares e interações mais complexas entre as variáveis

Por serem muito distintas, achamos que seria interessante aplicar as duas. A aplicação do modelo foi feita para as diferentes dimensões (50 e 300).

5 - RESULTADOS

Após todo o processo de treinamento e predição com base nos textos de texto precisa estimar a eficiência dos algoritmos para poder compará-los. Para isso utilizamos o módulo de métricas da biblioteca hiclass que utilizamos para o treinamento da classificação local por pai, isso porque ela consegue considerar a gravidade do erro em relação ao quão distante de uma raiz em comum ele está.

Para isso, tivemos que adaptar o resultado das predições planas gerando uma predição de L1 e L2 propagando da predição de L3 como apresentado na tabela 4.

Tabela 4 - Propagação de Labels

```
train = pd.read_csv('DBPEDIA_test.csv')

def set_labels():
    l2 = {}
    l1 = {}

    for i, l in enumerate(data['l3']):
        l2 |= {l: data.loc[i, 'l2']}

    for i, l in enumerate(data['l2']):
        l1 |= {l: data.loc[i, 'l1']}

    return l1, l2

l1, l2 = set_labels()

train['newl2'] = predicted.apply(lambda x: l2[x])
train['newl1'] = train['newl2'].apply(lambda x: l1[x])
train['newl3'] = predicted
```

Como métricas utilizamos Predição, que é uma métrica que avalia a qualidade das predições positivas feitas, Sensibilidade, que é a capacidade do modelo de identificar corretamente todos os exemplos positivos no conjunto de dados, e F1-score, que é uma métrica que combina precisão e sensibilidade em um único valor, sendo útil quando há um desequilíbrio entre as classes.

Assim, após realizar uma comparação entre os resultados obtidos para as técnicas de classificação hierárquica com relação ao pai e classificação hierárquica plana, tanto para o nosso modelo de transcrição de palavras para vetores com tamanho de 50 e 300 vetores, obtivemos o seguinte:

5.2 - CLASSIFICAÇÃO PLANA

5.2.1 - REGRESSÃO LOGÍSTICA

F1-Score: 0.8848954392429077

Precision : 0.8848954392429077

Recall: 0.8848954392429077

5.2.1 - RANDOM FOREST

F1-Score: 0.8364421927602504

Precision : 0.8364421927602504

Recall: 0.8364421927602504

5.2 - CLASSIFICAÇÃO HIERÁRQUICA COM RELAÇÃO AO PAI

5.2.1 - RANDOM FOREST

Considerando 300 dimensões:

- F1-Score: 0.7445189609876915
- Precision: 0.65145409086423
- Recall: 0.8686054544856401

Considerando 50 dimensões

- F1-Score: 0.8398252020484478
- Precision: 0.8398252020484478
- Recall: 0.8398252020484478

5.2.1 - REGRESSÃO LOGÍSTICA

Considerando 300 dimensões:

- F1-Score: 0.7786952659801953
- Precision: 0.681358357732671
- Recall: 0.908477810310228

Considerando 50 dimensões

- F1-Score: 0.7959063942713644
- Precision: 0.7959063942713644
- Recall: 0.7959063942713644

Considerando esse tipo de classificação, a regressão usando Random Forest com 50 dimensões se saiu melhor, provavelmente por conta da relação não-linear dos dados, e, por ter menos dimensões, fica mais fácil para o algoritmo classificar

6 - CONCLUSÃO

Pode-se perceber que a classificação plana teve um excelente desempenho, desempenho até melhor do que a classificação local por pai, entretanto seu custo computacional é gigantesco, levando em conta que não conseguimos treinar com 300 dimensões essas técnicas. Em relação aos modelos, foi inconclusivo, pois os modelos tiveram resultados muito parecidos.