Universidade do Minho
Mestrado Integrado em Engenharia Informática
Scripting no Processamento de Linguagem Natural

FLAIR - NLP Framework

João Gomes, A74033
Tiago Fraga, A74092

22 de Abril de 2019

Conteúdo

Capítulo 1

Introdução

Com a realização do presente trabalho prático, pretende-se estudar, conceber e implementar um modelo de processamento de linguagem natural.

Para o efeito, foi nos solicitado o aprofundamento dos conhecimentos sobre **FLAIR**. O *FLAIR* é uma ferramenta de processamento de linguagem natural desenvolvida pela *Zalando Research*.

Esta ferramenta foi desenvolvida em *Python*, com o auxilio de uma biblioteca de *machine learning* denominada por *PyTorch*, visto que esta é vocacionada para o domínio em questão.

A ferramenta *FLAIR* possui várias funcionalidades, que iremos caracterizar ao longo deste relatório, entre elas, salientamos a capacidade de reconhecimento de nomes e entidades, a etiquetação de palavras e o reconhecimento de texto. Além destas virtudes ainda é possivel criar e otimizar modelos de *machine learning* de forma a automatizar todo este processo.

Ao longo deste relatório iremos começar por abordar as componentes da ferramenta em estudo, indicando a forma de instalação e pormenores das duas funcionalidades. Em seguida, apresentaremos os casos de estudo elaborados e um resumo dos resultados que foram possíveis obter. Por fim, detalhamos um conjunto de pequenas conclusões sobre o trabalho efetuado, bem como medidas para serem implementadas como trabalho futuro.

Capítulo 2

Componentes

2.1 Instalação

A ferramenta em estudo, o *FLAIR* foi desenvolvida em ambiente *Python*, sendo que é suportada pela versão 3.6 ou superior. A biblioteca *PyTorch* foi também utilizada no seu desenvolvimento, sendo aceite a versão 0.4 ou superior. Por fim, é necessário executar o seguinte comando:

```
1 pip install flair
```

2.2 Funcionalidades

Neste secção vamos descrever as funcionalidades apresentadas pela ferramenta *FLAIR*. Estas funcionalidades têm como objetivo auxiliar o processamento de língua natural.

2.2.1 Classificação de Texto

A marcação de texto é um método que utiliza *machine learning* para conseguir realizar esta tarefa. Isto é o, *FLAIR* usa um modelo de treino para classificar o texto com base nesse modelo por exemplo classificar *ex: George Washington* como uma pessoa.

2.2.2 Marcação de Texto

Além da funcionalidade de classificação e com base na mesma temos a marcação do texto, em que marca o texto passado como parâmetro com a classificação realizada anteriormente.

Esta funcionalidade é muito abrangente usando os modelos disponibilizados pela ferramenta somos capaz de analisar sentimentos num texto, analisar semântica no texto somos ainda capazes

de analisar textos com mais que uma linguagem.

Mais a frente será apresentados código e o resultado gerado pelo mesmo de forma a ser mais percetíveis estas funcionalidades.

2.2.3 Criação do Modelo

Todas as funcionalidades descritas anteriormente são baseadas em machine learning, desta forma e com o auxílio da ferramenta em estudo é possível criar um modelo nesta área.

Para desenvolver o modelo é necessário carrega-lo com os dados que irão servir como base de treino.

Nesta etapa, podem ser usadas as bibliotecas **TaggedCorpus** e **NLPTaskDataFetcher**.

Após alguma pesquisa, notamos que a própria ferramenta oferece um conjunto de dados de teste em várias linguagens.

2.2.4 Treino do Modelo

Após executar o carregamento dos dados que irão ser usados para teste e da criação do modelo de machine learning, surge a fase do treino do mesmo.

As bibliotecas usadas para executar o treino são: **NLPTask**, **WordEmbeddings**, **FlairEmbeddings**, **DocumentRNNEmbeddings**, **TextClassifier** e **ModelTrainer**.

Durante o treino pudemos alterar parâmetros como o *learning rate*, o *batch size*, bem como o numero de épocas de treino.

2.2.5 Otimização do Modelo

Com o treino executado no modelo de machine learning é possivel fazer melhorias ao mesmo. Para o efeito, a ferramenta disponibiliza vários otimizadores disponíveis na biblioteca **flair.optim** onde encontramos o otimizador **ADAM** e **SGDW**.

Com o auxilio deste método, verifica-se uma melhoria na exatidão do modelo, bem como na rapidez e eficácia do mesmo.

2.2.6 Linguagens Disponíveis

Como abordado anteriormente, o *FLAIR* disponibiliza dados de treino em diversas linguagens para desenvolver modelos de *machine learning*. Apesar de ser o inglês a linguagem dominante, podemos desenvolver modelos de processamento de linguagem natural, com os *datasets* deles somos capazes de identificar texto em certas linguagens, os *datasets* são:

CONLL_2000	UD_DUTCH	UD_CROATIAN
CONLL_03_DUTCH	UD_FRENCH	UD_SERBIAN
CONLL_03_SPANISH	UD_ITALIAN	UD_BULGARIAN
WNUT_17	UD_SPANISH	UD_ARABIC
WIKINER_ENGLISH	UD_PORTUGUESE	UD_HEBREW
WIKINER_GERMAN	UD_ROMANIAN	UD_TURKISH
WIKINER_DUTCH	UD_CATALAN	UD_PERSIAN
WIKINER_FRENCH	UD_POLISH	UD_RUSSIAN
WIKINER_ITALIAN	UD_CZECH	UD_HINDI
WIKINER_SPANISH	UD_SLOVAK	UD_INDONESIAN
WIKINER_PORTUGUESE	UD_SWEDISH	UD_JAPANESE
WIKINER_POLISH	UD_DANISH	UD_CHINESE
WIKINER_RUSSIAN	UD_NORWEGIAN	UD_KOREAN
UD_ENGLISH	UD_FINNISH	UD_BASQUE
UD_GERMAN	UD_SLOVENIAN	IMDB
TREC_6	TREC_50	NER_BASQUE

Capítulo 3

Casos de Estudo

Neste capítulo vamos apresentar *scripts* simples de forma a mostrar as funcionalidades descritas acima, bem como o resultado gerado.

No fim vamos apresentar um *script* com as funcionalidades descritas mas para um modelo de treino em português, visto ser umas das lacunas a nível de linguagens disponibilizadas

3.1 Exemplos

3.1.1 Classificação de Texto

Apresentamos um exemplo de como classificar texto com esta ferramenta.

```
1    tagger = SequenceTagger.load('ner')
2    sentence = Sentence('George Washington went to Washington .')
3    # predict NER tags
4    tagger.predict(sentence)
5    # print sentence with predicted tags
```

3.1.2 Marcação de Texto

De modo a marcarmos o texto podemos seguir este pequeno exemplo.

```
1    from flair.data import Sentence
2    # Criamos uma sentence, usando tokenizer para partir a string
3    sentence = Sentence('The grass is green.', use_tokenizer=True)
4    # Adicionamos a tag a uma palavra na setence
5    sentence[3].add_tag('ner', 'color')
```

3.1.3 Criação do Modelo

Cada modelo passível de criação tem um propósito, podemos ler uma *a sequence labeling dataset*, ler *a text classification dataset* e *download a dataset*.

Para exemplo apresentamos um *text classification dataset* e como carregar de modo a criar o modelo.

Para utilizar este método o *dataset* tem de ser criado com base no formato *FastText*, no qual cada linha no ficheiro representa um documento de texto. Um documento pode ter um ou vários rótulos definidos no inicio da linha, começando sempre com o prefixo __label__ ficando com o seguinte aspecto:

```
__label__<label_1> <text>
__label__<label_1> __label__<label_2> <text>
```

O código para realizar este método é o seguinte:

```
1   from flair.data_fetcher import NLPTaskDataFetcher
2   from flair.data_fetcher import NLPTaskDataFetcher
3   from pathlib import Path
4
5   # Caminho para a pasta com os dados.
6   data_folder = Path('/resources/tasks/imdb')
7
8   # Carregar o corpus com teste, dev e treino.
9   corpus: TaggedCorpus = NLPTaskDataFetcher.load_classification_corpus(
        data_folder,
10                          test_file='test.txt',
11                          dev_file='dev.txt',
12                          train_file='train.txt')
```

3.1.4 Treino do Modelo

De modo a treinar o modelo temos vários modelos, conforme os tipos de *datasets*, tal como anteriormente vamos mostrar o exemplo correspondente ao ler *text classification dataset*.

```
from flair.data import TaggedCorpus
from flair.data_fetcher import NLPTaskDataFetcher, NLPTask
from flair.embeddings import WordEmbeddings, FlairEmbeddings,
    DocumentRNNEmbeddings
from flair.models import TextClassifier
from flair.trainers import ModelTrainer
# 1. Obter o corpus
corpus: TaggedCorpus = NLPTaskDataFetcher.load_corpus(NLPTask.AG_NEWS, '
    path/to/data/folder').downsample(0.1)
# 2. Criar um dicionario de label
label_dict = corpus.make_label_dictionary()
# 3. Criar uma lista Word Embeddings
word_embeddings = [WordEmbeddings('glove'),
                   # comment in flair embeddings for state-of-the-art
                       results
                   # FlairEmbeddings('news-forward'),
                   # FlairEmbeddings('news-backward'),
                   ]
# 4. inicializar um documento com a lista
# Can choose between many RNN types (GRU by default, to change use
    rnn_type parameter)
document_embeddings: DocumentRNNEmbeddings = DocumentRNNEmbeddings(
    word_embeddings,
                                                hidden_size=512,
                                                    reproject_words=True,
                                                reproject_words_dimension
                                                    =256,
                                                )
# 5. Criar o classificador do texto
classifier = TextClassifier(document_embeddings, label_dictionary=
    label_dict, multi_label=False)
# 6. Inicializar o modelo de treino
trainer = ModelTrainer(classifier, corpus)
# 7. Come ar o treino
trainer.train('resources/taggers/ag_news',learning_rate=0.1,
              mini_batch_size=32,anneal_factor=0.5,
              patience=5,max_epochs=150)
```

3.1.5 Otimização do Modelo

A otimização do modelo pode ser realizada recorrendo a dois otimizadores, sendo um deles o *Adam* e o outro o *SGDW*, mostramos como utilizar cada um.

```
from torch.optim.adam import Adam
trainer: ModelTrainer = ModelTrainer(tagger, corpus,
                                     optimizer=Adam, weight_decay=1e-4)

from flair.optim import SGDW
trainer: ModelTrainer = ModelTrainer(tagger, corpus,
                                     optimizer=SGDW, momentum=0.9)
```

3.2 Resultados Obtidos

Após elaborar uma análise sobre os casos de estudo descritos em cima, decidimos elaborar um pequeno *script* com o intuito de apresentar rapidamente e genericamente um resumo das funcionalidades da ferramenta em questão. Deste modo, e após a criação do *script* - que apresentamos no capitulo Apêndice - fomos capazes de tirar alguns testemunhos sobre os resultados obtidos.

3.2.1 NER - Classificação de Texto

Partindo com a frase de exemplo:

```
sentence = Sentence('George Washington went to Washington .')
```

foi possivel obter o seguinte resultado:

```
####################################
############    NER    ###############
####################################
2019-04-22 16:56:44,442 loading file /Users/tiagofraga/.flair/models/en-ner-conll03-v0.4.pt
George <B-PER> Washington <E-PER> went to Washington <S-LOC> .
PER-span [1,2]: "George Washington"
LOC-span [5]: "Washington"
{'text': 'George Washington went to Washington .', 'labels': [], 'entities': [{'text': 'George Washington', 'start_
pos': 0, 'end_pos': 17, 'type': 'PER', 'confidence': 0.9967881441116333}, {'text': 'Washington', 'start_pos': 26, '
end_pos': 36, 'type': 'LOC', 'confidence': 0.9993709921836853}]}
```

Figura 3.1: Resultado da Classificação de Texto

.

Deste modo, é possivel verificar que a ferramenta identificou como sendo uma **pessoa** as palavras: **George Washington**, e como sendo uma **localização**: **Washington**.

3.2.2 FRAME - Marcação de Texto

Partindo com as frases de exemplo:

```
sentence_1 = Sentence('George returned to Berlin to return his hat .')
sentence_2 = Sentence('He had a look at different hats .')
```

foi possivel obter o seguinte resultado:

```
####################################
##########    FRAME   ##############
####################################
2019-04-22 16:56:48,374 loading file /Users/tiagofraga/.flair/models/en-frame-ontonotes-v0.2.pt
George returned <return.01> to Berlin to return <return.02> his hat .
He had <have.LV> a look <look.01> at different hats .
```

Figura 3.2: Resultado da Marcação de Texto

.

A partir da observação dos resultados é possivel observar a forma como a ferramenta marcou as diferentes formas verbais. Na primeira frase, marcou o verbo *return* de duas maneiras diferentes. Na primeira o verbo é referente a uma localização enquanto que na segunda refere-se a obter alguma coisa de volta. Na segunda frase, marcou a forma verbal *had* como um *light verb* e marcou a forma verbal *look* como um verbo que necessita de um predicado.

3.2.3 TAGGER - Classificação de Texto

Partindo com a frase de exemplo:

```
text = "This is a sentence. This is another sentence. I love Berlin."
```

foi possivel obter o seguinte resultado:

```
####################################
##########  TAGGER    ##############
####################################
2019-04-22 17:41:55,009 loading file /Users/tiagofraga/.flair/models/en-ner-conll03-v0.4.pt
###
This is a sentence .
{'text': 'This is a sentence.', 'labels': [], 'entities': []}
###
This is another sentence .
{'text': 'This is another sentence.', 'labels': [], 'entities': []}
###
I love Berlin <S-LOC> .
LOC-span [3]: "Berlin"
{'text': 'I love Berlin.', 'labels': [], 'entities': [{'text': 'Berlin', 'start_pos': 7, 'end_pos': 13, 'type': 'LO
C', 'confidence': 0.9992183446884155}]}
```

Figura 3.3: Resultado da Classificação de Texto

.

Com é possivel observar, esta funcionalidade é parecida à primeira, com a vertente de analisar frase a frase. Nas duas primeiras frases, como não existe conteúdo relevante nao foi nada identificado. Na ultima frase identificou *Berlim* como uma localização.

3.2.4 EMOTION - Classificação de Texto

Outra das vertentes da classificação de texto, é a classificação do mesmo segundo as emoções que este transmite. Ou seja, segundo o sentimento que está implícito na frase, a ferramenta consegue percepcionar se está embebido um sentimento positivo ou negativo atribuído também, um valor de confiança para esta atribuição.

Partido da frase de exemplo:

```
1 sentence = Sentence('Porto wins. I am happy')
```

foi possivel obter o seguinte resultado:

```
####################################
##########  EMOTIONS   ##############
####################################
2019-04-22 17:34:09,706 loading file /
/anaconda3/lib/python3.7/site-packages
umentLSTMEmbeddings. (The functionalit
ion 0.4.
  result = unpickler.load()
[POSITIVE (1.0)]
```

Figura 3.4: Resultado da Classificação de Texto Positiva

.

A esta frase, a ferramenta atribui o sentimento positivo com um valor de confiança de **1.0**.

Por outro lado, partindo da frase de exemplo:

```
1 sentence = Sentence('Porto losed. I am sad')
```

foi possivel obter o seguinte resultado:

```
####################################
##########  EMOTIONS   ##############
####################################
2019-04-22 16:57:09,805 loading file
/anaconda3/lib/python3.7/site-package
umentLSTMEmbeddings. (The functionali
ion 0.4.
  result = unpickler.load()
[NEGATIVE (0.8180389404296875)]
```

Figura 3.5: Resultado da Classificação de Texto Negativa

.

A esta frase, a ferramenta atribui o sentimento negativo com um valor de confiança de **0.818**.

Capítulo 4

Conclusões e Trabalho Futuro

Damos por concluído o trabalho prático, onde abordarmos em forma de resumo o estudo sobre o **FLAIR**. Esta ferramenta de processamento de linguagem natural, faz parte da família de *NLP framework*.

Após concluirmos o nosso estudo, entendemos que esta ferramenta possui várias funcionalidades no reconhecimento e marcação de texto. Salientamos a capacidade de ser possível elabora um modelo de machine learning consoante um conjunto de dados á nossa escolha, para desta forma, ser capaz de identificar uma linguagem que não esteja disponível nativamente.

Deste modo, verificamos aqui a primeira evolução prática para o futuro. Um modelo capaz de identificar e marcar textos portugueses. Neste momento, não é possível fazê-lo pois os modelos disponíveis para carregamento que a ferramenta oferece-se, nenhum é em Português. Portanto, como trabalho futuro seria interessante, construir um modelo sólido e robusto capaz de identificar texto na nossa língua materna.

Em suma, apesar da falha em cima apontada, a ferramenta estudada é capaz de trabalhar sobre diversas outras linguagens e com grande eficácia e facilidade, o que a torna uma ferramenta bastante útil e prática no capítulo do processamento de linguagem natural.

Apêndice A

Anexo

Apresentamos o código das funções implementadas.

```
from flair.models import SequenceTagger
from flair.data import Sentence

print("##########################################")
print("############     NER     ################")
print("##########################################")

# load ner model
tagger = SequenceTagger.load('ner')

# example sample
sentence = Sentence('George Washington went to Washington .')

# predict NER tags
tagger.predict(sentence)

# print sentence with predicted tags
print(sentence.to_tagged_string())

for entity in sentence.get_spans('ner'):
    print(entity)

print(sentence.to_dict(tag_type='ner'))


print("##########################################")
print("###########     FRAME     ################")
print("##########################################")
```

```
29
30 # load frame model
31 tagger = SequenceTagger.load('frame')
32
33 # make sentence
34 sentence_1 = Sentence('George returned to Berlin to return his hat .')
35 sentence_2 = Sentence('He had a look at different hats .')
36
37 # predict NER tags
38 tagger.predict(sentence_1)
39 tagger.predict(sentence_2)
40
41 # print sentence with predicted tags
42 print(sentence_1.to_tagged_string())
43 print(sentence_2.to_tagged_string())
44
45
46
47
48 print("##########################################")
49 print("##########  TAGGER    ################")
50 print("##########################################")
51
52
53 # your text of many sentences
54 text = "This is a sentence. This is another sentence. I love Berlin."
55
56 # use a library to split into sentences
57 from segtok.segmenter import split_single
58 sentences = [Sentence(sent, use_tokenizer=True) for sent in split_single(
       text)]
59
60 # predict tags for list of sentences
61 tagger: SequenceTagger = SequenceTagger.load('ner')
62 tagger.predict(sentences)
63
64 for sentence in sentences:
65     print("###")
66     print(sentence.to_tagged_string())
67
68     for entity in sentence.get_spans('ner'):
69         print(entity)
70
```

```
71      print(sentence.to_dict(tag_type='ner'))
72
73
74  print("########################################")
75  print("##########  EMOTIONS  ##############")
76  print("########################################")
77
78  from flair.models import TextClassifier
79
80  # load sentiment model
81  classifier = TextClassifier.load('en-sentiment')
82
83  # example sentence
84  sentence = Sentence('Porto losed. I am sad')
85
86  # predict NER tags
87  classifier.predict(sentence)
88
89  # print sentence with predicted labels
90  print(sentence.labels)
```