fastText

LIU,TZU-CHUN

109 級畢業專題

5/9,2023

1. 簡介 fastText
2. Text Classification 文本分類
3. Word representations 詞表示

簡介 fastText

簡介 fastText

fastText is a library for efficient learning of word representations and sentence classification.

Text Classification

Text Classification

文本分類的目標是將文檔（如電子郵件、短信、產品評論等）分配到一個或多個類別。這些類別可以是評論分數、垃圾郵件與非垃圾郵件或文檔輸入的語言。如今，構建此類分類器的主要方法是機器學習，即從示例中學習分類規則。為了構建這樣的分類器，我們需要標記數據，它由文檔及其對應的類別（label, or tags）組成。

獲取和準備數據

如簡介中所述，我們需要標記數據來訓練我們的監督分類器。我們以構建一個分類器來自動識別關於烹飪的 stackexchange 問題為主題。

```
wget https://dl.fbaipublicfiles.com/fasttext/data/cooking.
stackexchange.tar.gz && tar xvzf cooking.stackexchange
.tar.gz
```

```
head cooking.stackexchange.txt
```

文本文件的每一行都包含一個標籤列表，後面是相應的文檔。所有標籤都以 label 前綴開頭，這就是 fastText 識別什麼是標籤或什麼是單詞的方式。然後訓練該模型以預測文檔中給定單詞的標籤。

獲取和準備數據

在訓練我們的第一個分類器之前，我們需要將數據分成訓練和驗證。我們將使用驗證集來評估學習的分類器在新數據上的表現

```
>> wc cooking.stackexchange.txt
   15404   169582 1401900 cooking.stackexchange.txt
```

我們的完整數據集包含 15404 個示例。讓我們將它分成 12404 個示例的訓練集和 3000 個示例的驗證集：

```
>> head -n 12404 cooking.stackexchange.txt > cooking.train
>> tail -n 3000 cooking.stackexchange.txt > cooking.valid
```

第一個分類器

```
>>> import fasttext
>>> model = fasttext.train_supervised(input="cooking.train")
Read 0M words
Number of words:  14598
Number of labels: 734
Progress: 100.0%  words/sec/thread: 75109  lr: 0.000000  loss: 5.708354  eta: 0h0m

\\Test
>>> model.predict("Which baking dish is best to bake a banana bread ?")
((u'__label__baking',), array([0.15613931]))
```

第一個分類器

```
>>> model.test("cooking.valid")
(3000L, 0.124, 0.0541)
```

輸出是樣本數（此處為 3000）、precision 精確度（0.124）Recall 召回率（0.0541）。
Precision 是 fastText 預測的標籤中正確標籤的數量。Recall 是在所有真實標籤中成功預測的標籤數。

優化模型-預處理數據

```
cat cooking.stackexchange.txt | sed -e "s/\([.\!?,'/()]\)/ \1
 /g" | tr "[:upper:]" "[:lower:]" > cooking.preprocessed.txt
```

將該文件中的特定字符（包括句號、問號、感嘆號、逗號、撇號、正斜線、左右括號）用空格隔開。這樣做是為了在將句子分成單詞時方便標記。接下來，腳本將所有大寫字母轉換為小寫字母，這是為了避免在處理文本時區分大小寫。

```
>> head -n 12404 cooking.preprocessed.txt > cooking.train
>> tail -n 3000 cooking.preprocessed.txt > cooking.valid
```

優化模型-預處理數據

```
>> ./fasttext supervised -input cooking.train -output model_co
Read 0M words
Number of words:  9012
Number of labels: 734
Progress: 100.0%  words/sec/thread: 82041  lr: 0.000000  loss:

>> ./fasttext test model_cooking.bin cooking.valid
N  3000
P@1  0.164
R@1  0.0717
Number of examples: 3000
```

我們觀察到，由於預處理，詞彙量更小（從 14k 個單詞減少到 9k 個）。
精度也開始上升 4%！

優化模型-預處理數據

More epochs and larger learning rate

默認情況下，fastText 在訓練期間只看到每個訓練樣例五次，考慮到我們的訓練集只有 12k 個訓練樣例，這是非常小的。可以使用以下選項增加每個示例出現的次數 (-epoch) 另一種改變我們模型學習速度的方法是增加（或減少）算法的學習率。這對應於處理每個示例後模型的變化量。學習率為 0 意味著模型根本不會改變，因此不會學習任何東西。良好的學習率在範圍內 0.1 - 1.0。

```
>> ./fasttext supervised -input cooking.train -output
 model_cooking -lr 1.0 -epoch 25
Read 0M words
Number of words:  9012
Number of labels: 734
Progress: 100.0%  words/sec/thread: 76394  lr: 0.000000
  loss: 4.350277  eta: 0h0m
```

優化模型-預處理數據

```
>> ./fasttext test model_cooking.bin cooking.valid
N  3000
P@1  0.585
R@1  0.255
Number of examples: 3000
```

優化模型-預處理數據

Word n-grams

最後，我們可以通過 word bigrams 來提高模型的性能，而不僅僅是 unigrams 。這對於詞序很重要的分類問題尤其重要，例如 sentiment analysis。

```
>> ./fasttext supervised -input cooking.train -output
 model_cooking -lr 1.0 -epoch 25 -wordNgrams 2
Read 0M words
Number of words:  9012
Number of labels: 734
Progress: 100.0%  words/sec/thread: 75366  lr: 0.000000
loss: 3.226064  eta: 0h0m

>> ./fasttext test model_cooking.bin cooking.valid
N  3000
P@1  0.599
R@1  0.261
Number of examples: 3000
```

Unigram & Bigram

Unigram

“unigram”指的是單個不可分割的單元或標記，通常用作模型的輸入。例如，一個 unigram 可以是一個單詞或一個字母，具體取決於模型。在 fastText 中，我們在單詞級別工作，因此 unigrams 是單詞。

Bigram

類似地，我們用“bigram”表示 2 個連續標記或單詞的串聯。同樣，我們經常談論 n-gram 來指代任何 n 個連續標記的串聯。例如，在句子“Last donut of the night”中，unigrams 是“last”、“donut”、“of”、“the”和“night”。雙字母組是：'Last donut'、'donut of'、'of the'和'the night'。

因為對於大多數句子，只需查看一袋 n-gram 就可以重建單詞的順序。

Word representations

什麼是 Word representations

現代機器學習中一個流行的想法是用向量來表示單詞。這些向量捕獲有關語言的隱藏信息，例如詞類比或語義。它還用於提高文本分類器的性能。

獲取數據

我們將研究限制在英語維基百科的前 10 億字節。

```
$ mkdir data
$ wget -c http://mattmahoney.net/dc/enwik9.zip -P data
$ unzip data/enwik9.zip -d data
```

原始維基百科轉儲包含大量 HTML / XML 數據。我們使用與 fastText 捆綁在一起的 wikifil.pl 腳本對其進行預處理（該腳本最初由 Matt Mahoney 開發，可以在他的網站上找到）。

```
$ perl wikifil.pl data/enwik9 > data/fil9
```

skipgram & cbow

fastText 提供了兩種用於計算單詞表示的模型：skipgram 和 cbow（'continuous-bag-of-words'）。
由於附近的單詞，skipgram 模型學習預測目標單詞。另一方面，cbow 模型根據上下文預測目標詞。上下文表示為包含在目標詞周圍固定大小窗口中的詞袋。

```
>>> import fasttext
>>> model = fasttext.train_unsupervised('data/fil9', "cbow")
```

在實踐中，我們觀察到 skipgram 模型比 cbow 更適用於子詞信息。

調整參數

該模型最重要的參數是它的維度和子詞的大小範圍。
維度：
維度 (dim) 控制向量的大小，它們越大，它們可以捕獲的信息越多，但需要學習更多的數據。但是，如果它們太大，訓練起來就會更難、更慢。默認情況下，我們使用 100 個維度，但 100-300 範圍內的任何值都很受歡迎。
子詞：
子詞是包含在最小大小 (minn) 和最大大小 (maxn) 之間的單詞中的所有子字符串。默認情況下，我們採用 3 到 6 個字符之間的所有子詞，但其他範圍可能更適合不同的語言

```
>>> import fasttext
>>> model = fasttext.train_unsupervised('data/fil9', minn=2,
 maxn=5, dim=300)
```

調整參數

根據擁有的數據量，我們可能想要更改訓練參數。epoch 參數控制模型將遍歷 (loop) 數據的次數。默認情況下，我們循環遍歷數據集 5 次。如果您的數據集非常龐大，您可能希望減少對它的循環。
另一個重要參數是學習率-lr。學習率越高，模型收斂到解決方案的速度就越快，但存在過度擬合數據集的風險。默認值為 0.05，這是一個很好的折衷。建議保持在 [0.01, 1] 的範圍內

```
>>> import fasttext
>>> model = fasttext.train_unsupervised('data/fil9', epoch=1, lr=0.5)
```

調整參數

最後，fastText 是多線程的，默認使用 12 個線程。如果您的 CPU 內核較少（例如 4 個），您可以使用線程標誌輕鬆設置線程數

```
>>> import fasttext
>>> model = fasttext.train_unsupervised('data/fil9', thread=4)
```

Printing word vectors

直接從文件中搜索和 print 詞向量 fil9.vec 很麻煩。幸運的是，fastText 中有一個 print-word-vectors 功能。

```
>>> [model.get_word_vector(x) for x in ["asparagus", "pidgey", "yellow"]]
```

Nearest neighbor queries

一個不錯的功能是您還可以查詢未出現在您的數據中的單詞！事實上，單詞是由其子串的總和表示的。只要未知單詞是由已知子串組成的，就有它的表示！
檢查詞向量質量的一種簡單方法是查看其最近的鄰居。這給出了向量能夠捕獲的語義信息類型的直覺。
這可以通過最近鄰（nn）功能來實現。例如，我們可以通過運行以下命令來查詢一個詞的 10 個最近鄰

```
>>> model.get_nearest_neighbors('asparagus')
[(0.812384, u'beetroot'), (0.806688, u'tomato'), (0.805928, u'horseradish'),
 (0.801483, u'spinach'), (0.791697, u'licorice'), (0.781507, u'lingonberries'),
 (0.780756, u'asparagales'), (0.778534, u'lingonberry'), (0.774529, u'celery'),
 (0.773984, u'beets')]
```

It seems that vegetable vectors are similar. Note that the nearest neighbor is the word asparagus itself, this means that this word appeared in the dataset.

Nearest neighbor queries

這邊再用神奇寶貝當例子看看。

```
>>> model.get_nearest_neighbors('pidgey')
[(0.891801, u'pidgeot'), (0.885109, u'pidgeotto'), (0.884739, u'pidge'),
(0.787351, u'pidgeon'), (0.781068, u'pok'), (0.758688, u'pikachu'),
(0.749403, u'charizard'), (0.742582, u'squirtle'), (0.741579, u'beedrill'),
 (0.733625, u'charmeleon')]
```

同一寶可夢的不同進化有相近的向量

Evolution chart

Nearest neighbor queries

如果我們拼錯單詞，它的向量是否會接近任何合理的值？

```
>>> model.get_nearest_neighbors('enviroment')
[(0.907951, u'enviromental'), (0.87146, u'environ'),
 (0.855381, u'enviro'), (0.803349, u'environs'), (0.772682, u'environnement'),
 (0.761168, u'enviromission'), (0.716746, u'realclimate'), (0.702706, u'environment
 (0.697196, u'acclimatation'), (0.697081, u'ecotourism')]
```

由於單詞中包含的信息，我們拼錯的單詞的向量與合理的單詞相匹配！
它並不完美，但已捕獲主要信息。

Measure of similarity

為了找到最近的鄰居，我們需要計算單詞之間的相似度分數。我們的單詞由連續的單詞向量表示，因此我們可以對它們應用簡單的相似性。特別是我們使用兩個向量之間的角度的餘弦（cosine）。計算詞彙表中所有單詞的相似度，並顯示 10 個最相似的單詞。當然，如果這個詞出現在詞彙表中，它就會出現在最前面，相似度為 1。

Word analogies 詞類比

In a similar spirit, one can play around with word analogies. For example, we can see if our model can guess what is to France, and what Berlin is to Germany.

```
>>> model.get_analogies("berlin", "germany", "france")
[(0.896462, u'paris'), (0.768954, u'bourges'), (0.765569, u'louveciennes'),
 (0.761916, u'toulouse'), (0.760251, u'valenciennes'), (0.752747, u'montpellier'),
  (0.744487, u'strasbourg'), (0.74143, u'meudon'), (0.740635, u'bordeaux'),
  (0.736122, u'pigneaux')]
```

Another example:

```
>>> model.get_analogies("psx", "sony", "nintendo")
[(0.803352, u'gamecube'), (0.792646, u'nintendogs'), (0.77344, u'playstation'),
 (0.772165, u'sega'), (0.767959, u'gameboy'), (0.754774, u'arcade'),
 (0.753473, u'playstationjapan'), (0.752909, u'gba'), (0.74907, u'dreamcast'),
 (0.745298, u'famicom')]
```

Importance of character n-grams

Using subword-level information is particularly interesting to build vectors for unknown words. For example, the word gearshift does not exist on Wikipedia but we can still query its closest existing words

```
>>> model.get_nearest_neighbors('gearshift')
[(0.790762, u'gearing'), (0.779804, u'flywheels'),
 (0.777859, u'flywheel'), (0.776133, u'gears'), (0.756345, u'driveshafts'),
  (0.755679, u'driveshaft'), (0.749998, u'daisywheel'), (0.748578, u'wheelsets'),
  ^^I (0.744268, u'epicycles'), (0.73986, u'gearboxes')]
```

Most of the retrieved words share "substantial substrings" but a few are actually quite different, like cogwheel.