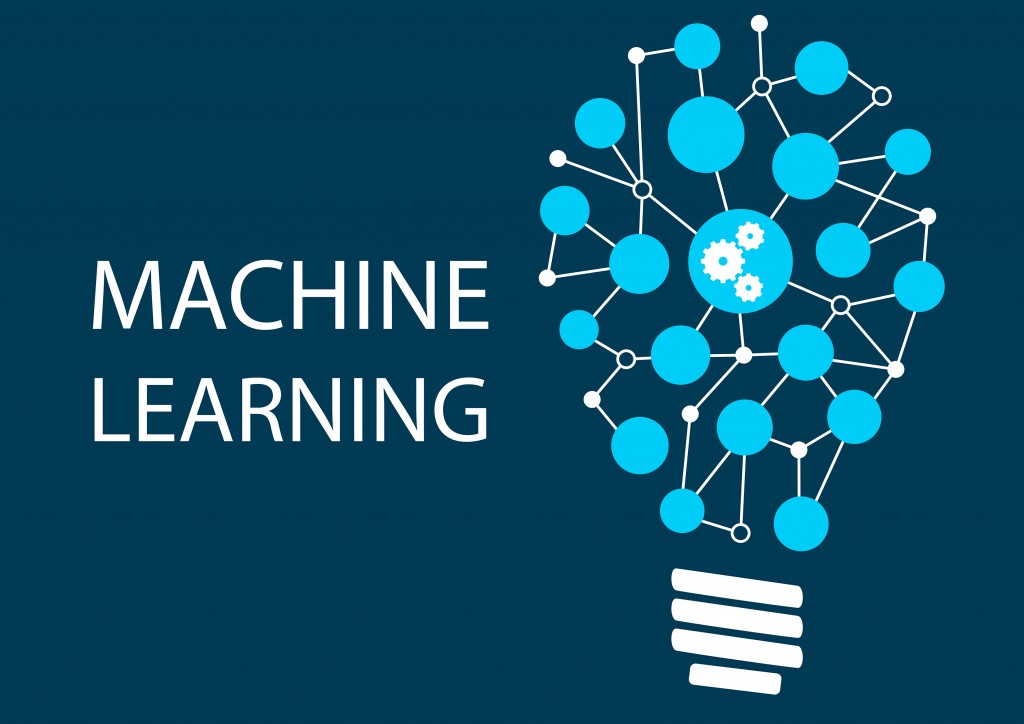
CASE STUDY 3

**Bài toán: “News Headlines Dataset For Sarcasm Detection - High quality dataset for the task of Sarcasm Detection” (Phát hiện tin bài châm biếm - Bộ dữ liệu chất lượng cao từ các trang tin chuyên nghiệp)**



***Môn học - Lớp:***

**MÁY HỌC - CS114.J21.KHTN**

***Giảng viên:***

**HUỲNH THỊ THANH THƯƠNG**

***Sinh viên:***

**17520689 - NGUYỄN THÙY LINH**

**17520015 - NGUYỄN HỮU PHONG**

# **I - GIỚI THIỆU CHUNG**

**Bài toán:**

“Phát hiện tin bài châm biếm” phát sinh từ bài toán gốc “phát hiện tin giả mạo” - thường dùng cho mạng xã hội (vốn tin bài trên mạng xã hội viết với văn phong không chính thống).

Đây là bài toán cũng đang khá hot, được nhiều người quan tâm (đặc biệt là Facebook đang gặp khó khăn trong vấn đề lọc tin)

**Dataset:**

Dataset được lấy từ trang kaggle:

<https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection>

# **II - XỬ LÝ DỮ LIỆU**

# **1 - Dữ liệu gốc ban đầu**

Bao gồm: 26709 dòng, mỗi dòng gồm các thông tin:

**<article\_link>** dẫn link nguồn của dữ liệu.

**<headline>** tiêu đề của bài viết.

**<is\_sarcastic>** mang giá trị nhị phân 0 - 1, biểu thị có phải là tin châm biếm hay không.

# **2 - Tiền xử lý dữ liệu:**

Chỉ xử lý và sử dụng **<headline>** + **<is\_sarcastic>**, trong đó:

**<headline>** là thuộc tính “thô” cần xử lý

**<is\_sarcastic>** là label chỉ mang 2 giá trị (0, 1)

**BƯỚC 1:**

**Công việc thực hiện:**

Loại bỏ các ký tự dấu câu, chữ số và chuyển về chỉ dùng các chữ cái lowercase.

|  |  |
| --- | --- |
| Ví dụ: | |
| Dữ liệu gốc | Dữ liệu sau khi xử lý |
| “This case study is so HARD, with me …” | this case study is so hard with me |
| “Giving up ? chaizoo, ... , fighting! | giving up chaizoo fighting |
| ... | ... |

**Sources code:**

Dữ liệu vào: training + testing lấy từ file **data.json**, stopwords lấy từ file **stop\_words.data**

Dữ liệu ra: (hiển thị headline sau khi xử lý) ghi ra file **fixed\_headline.data**

|  |
| --- |
| import pandas as pd #xử lý file json  import re #xử lý regular expression |
| df\_idf = pd.read\_json("data.json", lines = True) # đọc dữ liệu từ file data.json    def pre\_process(headline):  headline = headline.lower() # xử lý lowercase  headline = re.sub("&lt;/?.\*?&gt;"," &lt;&gt; ", headline) # xử lý ký tự dấu câu  headline = re.sub("(\\d|\\W)+"," ", headline) # xử lý ký tự đặc biệt và chữ số  return headline    df\_idf["headline"] = df\_idf["headline"].apply(lambda x:pre\_process(x)) |
| def get\_stop\_words(stop\_file\_path):  """load stop words """  stop\_set = set()  with open(stop\_file\_path, 'r', encoding = "utf-8") as fi: # đọc dữ liệu stopwords từ file  while True:  stopword = fi.readline()  if (len(stopword) == 0): break  stop\_set.update([stopword.strip()])  return frozenset(stop\_set)    stopwords = get\_stop\_words("stop\_words.data") # load stopwords  documents = df\_idf["headline"].tolist() # tạo documents cho mỗi headline |

**Kết quả sau khi xử lý headline:** (có 26709 dòng)

|  |
| --- |
| headline 1: former versace store clerk sues over secret black code for minority shoppers  headline 2: the roseanne revival catches up to our thorny political mood for better and worse  headline 3: mom starting to fear son s web series closest thing she will have to grandchild  headline 4: boehner just wants wife to listen not come up with alternative debt reduction ideas  headline 5: j k rowling wishes snape happy birthday in the most magical way  headline 6: advancing the world s women  headline 7: the fascinating case for eating lab grown meat  headline 8: this ceo will send your kids to school if you work for his company  headline 9: top snake handler leaves sinking huckabee campaign  ... |

**BƯỚC 2:**

**Công việc thực hiện:**

Ta sẽ xem xét mỗi headline là một document, các term là các từ đơn trong headline đó.

Nhận xét rằng, có một số từ sẽ không mang nhiều ý nghĩa, việc có chúng hay không thì ta cũng có thể nhận biết là bài viết đang xét có là châm biếm hay không? Do đó, ta sẽ xóa đi các từ này trong mỗi headline.

Danh sách từ cần loại bỏ (stopwords):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| the | off | and | a | to |
| in | that | this | his | it |
| i | is | he | with | was |
| as | were | all | for | at |
| but | by | not | him | from |
| be | on | so | whale | one |
| you | had | have | there | but |
| or | now | which | me | like |

**Sources code:**

Dữ liệu vào: documents , stopwords từ bước 1

Dữ liệu ra: danh sách từ điển (file **vocabulary.data**)

|  |
| --- |
| from sklearn.feature\_extraction.text import CountVectorizer # sử dụng thư viện |
| cv = CountVectorizer(max\_df=0.85, stop\_words = stopwords) # tạo một thư viện  word\_count\_vector = cv.fit\_transform(documents)  with open("vocabulary.data", "w+", encoding = "utf-8") as f: # ghi kết quả vào file  f.writelines(str(word\_count\_vector.shape) + "\n")  for i in cv.vocabulary\_.keys(): f.writelines(i + "\n") |

**Kết quả:**

Word\_count\_vector.shape = (26709, 24847) với ý nghĩa có 26709 headline + 24847 terms trong vocabulary

|  |
| --- |
| (26709, 24847)  former  versace  store  clerk  sues  over  secret  black  code  minority  shoppers  roseanne  revival  ... |

**BƯỚC 3:**

**Công việc thực hiện:**

Tính toán thống kê TF-IDF ([https://vi.wikipedia.org/wiki/Tf–idf](https://vi.wikipedia.org/wiki/Tf%E2%80%93idf))

Ý tưởng là với mỗi term trong vocabulary sẽ có trọng số riêng, đại biểu mức độ quan trọng (từ xuất hiện càng phổ biến thì thường ít quan trọng hơn) của term đó. Thì ta sẽ dùng IDF (inverse document frequency) làm giá trị để đánh giá

TF - term frequency, là thống kê tần số của từng từ, trong từng document

Vì từ điển có đến > 24k từ mà có 26k documents, nên việc tổ chức lưu trữ như thông thường là không khả thi (số chiều quá lớn). Do đó để giảm số chiều thì tổ chức lại với mỗi documents sẽ nhóm thành 1 vector 3 chiều.

|  |  |  |
| --- | --- | --- |
| 0: tổng trọng số những term chỉ xuất hiện ở bài viết là không châm biếm | 1: tổng trọng số những term chỉ xuất hiện ở bài viết châm biếm | 2: tổng trọng số những term xuất hiện ở cả 2 bài chấm biếm và không châm biếm |

**Source code:**

|  |
| --- |
| from sklearn.feature\_extraction.text import TfidfTransformer # thư viện sử dụng |
| tfidf\_transformer= TfidfTransformer(smooth\_idf = True, use\_idf = True)  tfidf\_transformer.fit(word\_count\_vector)  feature\_names=cv.get\_feature\_names() |
| wd = np.full(len(feature\_names), -1) #gán nhãn xem term thuộc dimension nào  setY = df\_idf["is\_sarcastic"] #lọc nhãn của dataset |
| # quá trình gán nhãn dimension  for i in range(len(documents)):  tf\_idf\_vector = tfidf\_transformer.transform(cv.transform([documents[i]]))  temp = tf\_idf\_vector.tocoo().col  label = setY[i]  for j in temp:  if wd[j] == 2: continue  if wd[j] == -1: wd[j] = label; continue  if wd[j] == 0 and label == 1: wd[j] = 2; continue  if wd[j] == 1 and label == 0: wd[j] = 2; continue |
| cnt0, cnt1, cnt2 = 0, 0, 0  for i in wd:  if i == 0: cnt0 += 1  if i == 1: cnt1 += 1  if i == 2: cnt2 += 1  print(cnt0, cnt1, cnt2) #đếm xem có bao nhiêu term của từng chiều: 7698, 7688, 9461 |
| # đã có setY, giờ tạo setX là tập các vector 3 chiều  setX = []  for i in range(len(documents)):  tf\_idf\_vector = tfidf\_transformer.transform(cv.transform([documents[i]]))  temp = tf\_idf\_vector.tocoo()  tempX = [0.0] \* 3  for idx, score in zip(temp.col, temp.data):  tempX[wd[idx]] += score  setX.append(tempX)  setX = np.array(setX) |

# **III - CHIA DATASET THÀNH TEST SET VÀ TRAINING SET**

Có 26709 datapoint, lấy 20000 datapoint đầu tiên là training set, 6709 datapoint còn lại làm test set

**Tổ chức gồm:**

|  |
| --- |
| Đầu vào:  setX: tập thuộc tính  setY: tập nhãn |
| trainingX, testX, trainingY, testY mang ý nghĩa tương ứng |

**Sources code:**

|  |
| --- |
| trainingX = np.array(setX[0:20000])  testX = np.array(setX[20000:26709])  trainingY =np.array(setY[0:20000])  testY = np.array(setY[20000:26709]) |

# **III - LOGISTIC REGRESSION**

# **1 – LV1 – Thủ công**

**Sources code:**

|  |
| --- |
| import time |
| start\_time = time.time()  w = np.zeros(4)  gamma = 0.000001  epsilon = 0.000001  predictY = np.zeros(6709)  def f(X):  fx = w[0]  for i in range(len(X)):  fx += w[i+1]\*X[i]  return fx  def sigmoid(fx):  return 1/(1+pow(np.e, -fx))  def cost():  E = 0  for i in range(len(trainingX)):  fx = f(trainingX[i])  px = sigmoid(fx)  E += -(fx\*np.log(px) + (1-fx)\*np.log(1-px))  return E/len(trainingX)  def gradient():  G = np.zeros(4)  for i in range(len(G)):  if i == 0:  for j in range(len(trainingX)):  fx = f(trainingX[j])  px = sigmoid(fx)  G[i] += px - trainingY[j]  else:  for j in range(len(trainingX)):  fx = f(trainingX[j])  px = sigmoid(fx)  G[i] += (px - trainingY[j])\*trainingX[j][i-1]  return G/len(trainingX)  def isConstant(w, w\_new):  for i in range(len(w)):  if abs(w\_new[i] - w[i]) > epsilon:  return False  return True  def LogisticRegression():  global w  for i in range(len(trainingX)):  # E = cost()  G = gradient()  w\_new = np.zeros(len(w))  for j in range(len(w)):  w\_new[j] = w[j] - gamma \* G[j]  if isConstant(w, w\_new) == True:  break  w = w\_new[:]  def PredictY():  for i in range(len(testX)):  if f(testX[i]) > 0:  predictY[i] = 1  LogisticRegression()  PredictY()  end\_time = time.time() |

**Đánh giá:**

|  |
| --- |
| from sklearn.metrics import classification\_report # khai báo thư viện  from sklearn import metrics |
| print(classification\_report(testY,predictY))  print("Accuracy:",metrics.accuracy\_score(testY, predictY))  print('total run-time: %f ms' % ((end\_time - start\_time) \* 1000)) |

**Kết quả:**

|  |
| --- |
|  |

# **2 - LV2 – Thư viện**

**Sources code:**

|  |
| --- |
| from sklearn.linear\_model import LogisticRegression #khai báo thư viện  import time |
| start\_time = time.time()  logreg = LogisticRegression()  logreg.fit(trainingX, trainingY)  predictY = logmodel.predict(testX)  end\_time = time.time() |

**Đánh giá:**

|  |
| --- |
| from sklearn.metrics import classification\_report # khai báo thư viện  from sklearn import metrics |
| print(classification\_report(testY,predictY))  print("Accuracy:",metrics.accuracy\_score(testY, predictY))  print('total run-time: %f ms' % ((end\_time - start\_time) \* 1000)) |

**Kết quả:**

|  |
| --- |
|  |

# **IV - CÁC MÔ HÌNH KHÁC:**

# **1 - K-nearest neighbors**

**Sources code:**

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier  import time |
| start\_time = time.time()  neigh = KNeighborsClassifier(n\_neighbors=5)  neigh.fit(trainingX, trainingY)  predictY = neigh.predict(testX)  end\_time = time.time() |
| print(classification\_report(testY,predictY))  print("Accuracy:",metrics.accuracy\_score(testY, predictY))  print('total run-time: %f ms' % ((end\_time - start\_time) \* 1000)) |

**Kết quả:**

|  |
| --- |
|  |

# **2 - Naive bayes (gaussian):**

**Sources code:**

|  |
| --- |
| from sklearn.naive\_bayes import GaussianNB  import time |
| start\_time = time.time()  clf = GaussianNB()  clf.fit(X, Y)  predictY = clf.predict(testX)  end\_time = time.time() |
| print(classification\_report(testY,predictY))  print("Accuracy:",metrics.accuracy\_score(testY, predictY))  print('total run-time: %f ms' % ((end\_time - start\_time) \* 1000)) |

**Kết quả:**

|  |
| --- |
|  |

# **V – ĐÁNH GIÁ**

**Kết quả:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Logistic Regression** | **K-nearest neighbors** | **Naïve Bayes** |
| **Accuracy** | 0.8521389178715159 | 0.8352958712177672 | 0.8521389178715159 |
| **Total run-time (ms)** | 168.937445 | 1167.747974 | 24.02401 |

**Nhận xét:**

Với bài toán này, Logistic Regression và Naïve Bayes đều cho độ chính xác cao, xấp xỉ 85%, còn K-nearest thấp hơn một chút, xấp xỉ 84%.

Trong đó Naïve Baives thực thi nhanh hơn gấp 7 lần so với Logistic Regression, gấp 48 lần so với K-nearest neighbors.