Analiza sentimentelor

Bogdan Macovei
Grupa 344

Universitatea din Bucuresti
Facultatea de Matematica si Informatica

6 mai 2018

Cuprins

Introducere in analiza sentimentelor

- analiza sentimentelor este procesul prin care sunt procesate automat opinii despre un anumit subiect, intr-o anumita limba;
- analiza sentimentelor a devenit un instrument care ne ajuta sa dam sens datelor;
- mai este cunoscuta sub denumirea de Opinion Mining si este o ramura a domeniului Natural Language Processing; se ocupa cu identificarea si extragerea opiniilor din texte.

Introducere in analiza sentimentelor

Pe langa identificarea opiniilor, se mai ocupa cu determinarea urmatoarelor informatii:

- polaritatea: daca subiectul exprima un sentiment pozitiv sau unul negativ;
- subiectul: ideea care reiese din textul procesat;
- titularul opiniei: persoana sau entitatea care exprima o anumita opinie.

In prezent, analiza sentimentelor este un topic de interes datorita multiplelor aplicatii practice, influentate de cresterea numarului de informatii disponibile pe Internet, astfel ca multe opinii exprimate de diversi utilizatori sunt disponibile pe forumuri, bloguri, site-uri de review-uri si social media. Informatiile extrase prin analiza sentimentelor sunt utile pentru aplicatiile comerciale, precum analiza de marketing, relatiile publice, review-urile de produse etc.

Opinia

Informatiile extrase din texte se pot clasifica in doua categorii: fapte si opinii. Faptele sunt obiective si exprima idei neimplicate afectiv despre un anumit lucru, in timp ce opiniile sunt subiective si descriu sentimentele oamenilor, evaluarile acestora si sentimentele relative la un anumit subiect. Analiza sentimentelor poate fi modelata cu ajutorul algoritmilor de clasificare, rezolvand urmatoarele doua subprobleme:

- clasificarea subiectiva: se determina daca un enunt este subiectiv sau obiectiv;
- polaritatea: se determina daca un enunt este pozitiv, negativ sau neutru din punctul de vedere al opiniei exprimate.

Tipuri de opinii

Exista mai multe tipuri de opinii, dintre care amintim:

- Opinii directe si comparative
- Opinii explicite si implicite

Exemple:

- opinie directa: "Calitatea fotografiei realizata cu ajutorul camerei A este slaba"
- opinie comparativa: "Calitatea forografiei realizata cu ajutorul camerei A este mai buna decat ce realizata cu ajutorul camerei B"
- opinie explicita: "Calitatea audio a acestui telefon este foarte buna"
- opinie implicita: "Aceste casti s-au stricat in doua zile"

Scopul analizei sentimentelor

Analiza sentimentelor poate fi aplicata pe diferite niveluri:

- pe documente: se obtine sentimentul unui document intreg sau al unui paragraf;
- pe enunturi: se obtine sentimentul exprimat de un singur enunt;
- pe parti ale enunturilor: se obtin sentimente pe subexpresii.

Tipuri de analiza de sentimente

Exista mai multe tipuri de rezultate ce pot fi obtinute, de la cele standard care se focuseaza pe polaritate, la cele care determina emotii (nervos, fericit, suparat etc.) sau care identifica intentii (interesat vs. neinteresat).

- Analiza sentimentelor "fine-grained": se aplica atunci cand suntem interesati sa obtinem mai mult decat nivelul de polaritate, si anume informatii precum "foarte bine", "bine", "neutru", "rau", "foarte rau", analiza asimilata cu rating-ul oferit in numar de stele (5 pentru "foarte bine", pana la 1 pentru "foarte rau");
- detectarea emotiei: se determina emotii precum fericire, frustrare, suparare, frica etc. In general, acestea se determina cu ajutorul unui lexicon sau cu ajutorul unor algoritmi de machine learning;
- analiza bazata pe o anumita componenta: se determina in special partile importante care determina polaritatea. Se aplica atunci cand nu vrem doar sa stim ca un comentariu oferit este negativ, ci vrem sa stim de ce anume este (de exemplu, un comentariu negativ pentru un telefon mobil poate fi datorat faptului ca are o camera slaba)

Tipuri de analiza de sentimente

- analiza intentiei: se determina ce isi doreste o persoana prin enuntul exprimat (o plangere, o cerere, o intrebare).

Analiza sentimentelor este importanta, deoarece se considera ca aproximativ 80% din informatia existenta in lume este nestructurata, iar o mare parte din aceasta provine din texte, precum emailuri, chat-uri, social media, articole, documente etc. Aceste texte sunt greu de analizat si costisitoare ca timp.

Avantaje ale analizei sentimentelor

- scalabilitate: sunt sortate automat review-uri ale cumparatorilor, conversatiile de la contactele cu clientii etc.;
- analiza in timp real a situatiilor care pot aparea (se pot identifica anumiti clienti nemultumiti ce pot lua anumite masuri impotriva unei companii etc);
- determinarea unor criterii clare de analiza, pe care oamenii nu le pot sesiza din cauza factorilor subiectivi.

Cum functioneaza analiza sentimentelor

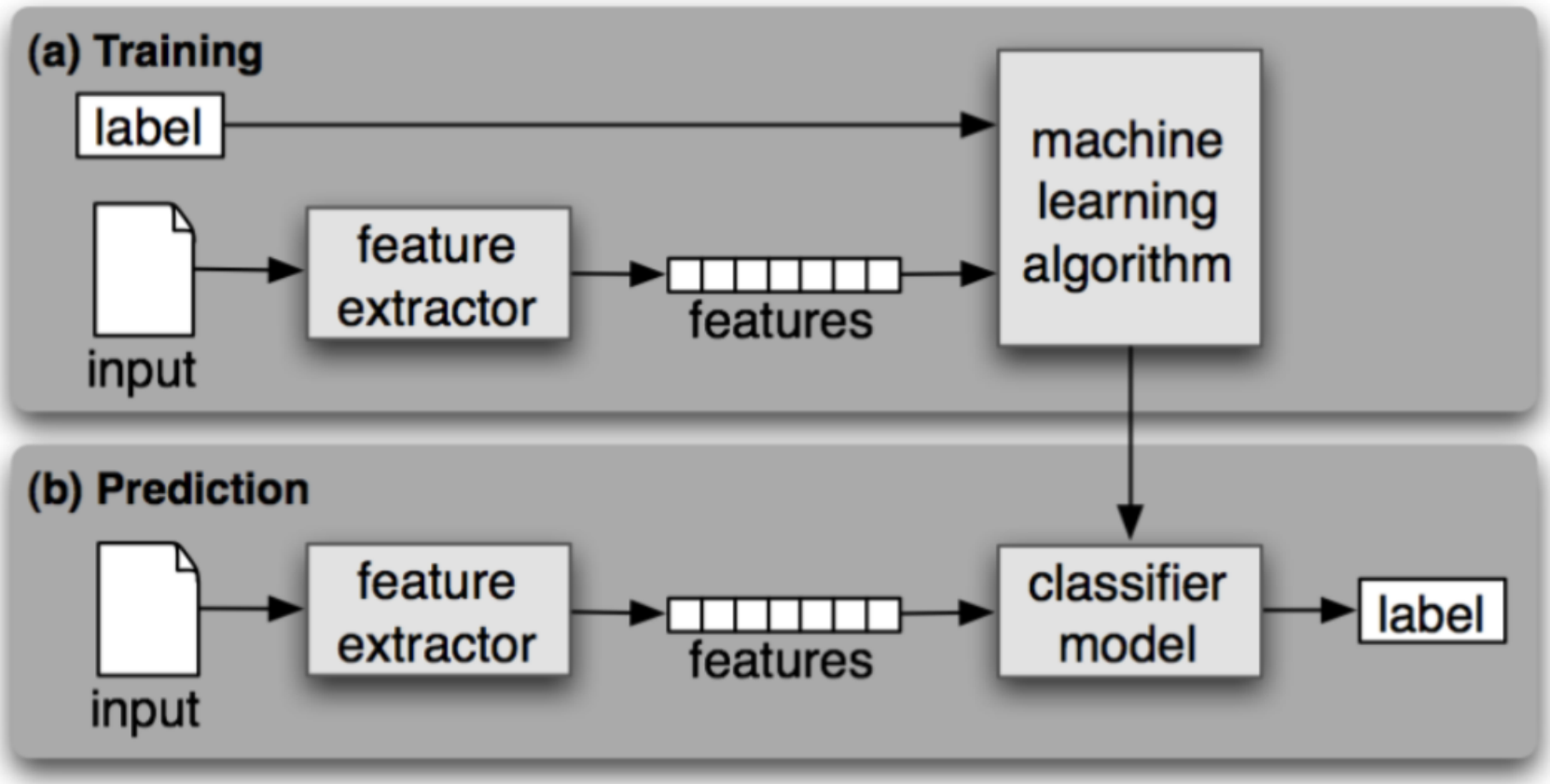
Exista mai multe moduri in care functioneaza analiza sentimentelor, si anume:

- abordare bazata pe reguli: se determina sentimentele in functie de anumite constrangeri definite manual;
- automat: se utilizeaza algoritmi de machine learning;
- hibrid.

Pentru prima abordare, cea cu reguli definite manual, exista tehnici clasice de NLP, precum parsare, tokenizare, etichetarea partilor din discurs, cat si utilizarea lexiconului. O abordare standard este aceea de a se defini doua liste, una de cuvinte negative ("rau", "urat"...) si una de cuvinte pozitive ("bun", "frumos"...). Dat fiind un text, se numara cate cuvinte pozitive, respectiv negative apar, iar rezultatul polaritatii este dat de comparatia intre aceste doua numere. Varianta este una naiva pentru ca nu tine cont de modul in care sunt conectate cuvintele, nu este inteles contextul general.

Abordarea automata

Abordarea automata este bazata pe algoritmii standard de Machine Learning, schema generala fiind:



Implementarea analizei sentimentelor utilizand Pyhotn

In cele ce urmeaza, vor fi descrise etapele pe care le urmarim cand vrem sa implementam automat analiza sentimentelor, cu scopul de a obtine o polaritate binara (pozitiv vs. negativ). Limbajul pe care l-am utilizat pentru a descrie aceste operatii este Python 3, iar toate abordarile sunt "from scratch". Exemplele de implementare vor fi pentru a determina daca un review de film este pozitiv sau negativ.

Vectorizarea inputului. Tf-idf

Primul pas pentru a putea aplica un algoritm de Machine Learning este acela de a transforma input-ul intr-un feature numeric. Tehnica utilizata este Tf-idf (term frequency, inverse document frequency).

- Tf: se determina frecventa de aparitie a fiecarui cuvant in fragmentul din care face parte $(\frac{numarul\ \ de\ \ aparitii}{numarul\ \ de\ \ cuvinte\ \ din\ \ text})$
- Idf: se determina cat de semnificativ este un cuvant pentru un text dat, metoda aceasta ajuta ca acele cuvinte care apar foarte des sa aiba o pondere cat mai mica. Formula utilizata este $\log \frac{numarul\ \ total\ \ de\ \ texte}{numarul\ \ de\ \ texte\ \ in\ \ care\ \ apare\ \ cuvantul\ \ curent}$

Marimea tf-idf pentru un cuvant va fi produsul dintre *tf* si *idf*.

Exemplu in Python

Fie *corpus* structura in care retinem textele (o lista) si *words* toate cuvintele distincte care apar in toate textele. Definim X ca fiind rezultatul final (matricea care va contine ponderile numerice in urma aplicarii tf-idf). Initial, vom adauga frecventele cuvintelor pentru fiecare text in parte:

Example

```
X = np.zeros((len(corpus), len(words)))

for i in range(0, len(corpus)):
    for j in range(0, len(words)):
        X[i][j] = corpus[i].lower().count(words[j])
```

Dupa ce avem *tf* si *idf* calculate, X va fi

Example

```
X = tf * idf
```

Definitii pentru tf si idf

Avand X calculat ca mai sus, definim:

Example

```
tf = list(map(lambda x: list(map(lambda y: y/len(x), x)), X))

N = len(corpus)
idf = np.zeros((len(corpus), len(words)))
for j in range(0, len(words)):
    count = 0
    for i in range(0, N):
        if X[i][j] != 0:
                count = count + 1
        if count != 0:
            for i in range(0, N):
                if X[i][j] != 0:
                    idf[i][j] = math.log(N / count)
```

Alegerea algoritmului de clasificare

Intrucat polaritatea este binara (pozitiv / negativ), vom alege ca model de implementare Regresia Logistica. Pentru aceasta vom oferi doua abordari, si anume cand label-ul este $y \in \{0, 1\}$ si cand label-ul este $y \in \{-1, 1\}$. Pentru prima abordare vom utiliza Gradient Descent, iar pentru cea de-a doua vom utiliza un model Perceptron.

Descrierea modelului logistic

Metoda de regresie logistica este o metoda de clasificare pentru care se foloseste o functie logistica,

$$\sigma(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

(un caz special al functiei sigmoid). Pentru modelul care urmeaza, utilizam functia standard cu $L = 1$, $k = 1$ si $x_0 = 0$:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Cum $0 < \sigma(x) < 1$, functia poate fi interpretata ca o probabilitate.

Descrierea modelului logistic

Utilizand aceasta functie logistica si un feature vector $x \in \mathbf{R}^n$, obtinem ca

$$\hat{y} = \sigma(w_0 + \sum_{i=1}^{n} w_i x_i)$$

i.e.

$$\hat{y} = \frac{1}{1 + e^{-w^T x}}$$

ceea ce inseamna ca vom aplica o functie logistica pe un model liniar. Predictia $\hat{y}$ va fi interpretata ca o probabilitate de forma

$$\hat{y} = P(y = 1|x, w)$$

Pentru a putea utiliza modelul, trebuie sa calculam vectorul w (weights), iar acesta va fi obtinut minimizand loss function-ul. In cazul regresiei logistice, utilizam

$$L = -\sum_i [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})] + \lambda ||w||_2^2$$

Vrem sa determinam $w^* = \arg\min_w L(w)$.

Descrierea modelului logistic

$$\arg\min_w\{L(w)\} =$$

$$\arg\min_w \Big\{ -\sum_i [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})] + \lambda ||w||_2^2 \Big\} =$$

$$\arg\min_w \Big\{ -\frac{1}{\lambda} \sum_i [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})] + ||w||_2^2 \Big\} =$$

$$\arg\min_w \Big\{ -C \sum_i [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})] + ||w||_2^2 \Big\}$$

Ultima varianta este cea mai comuna, deoarece este usor de interpretat C-ul ca fiind costul pentru o greseala de antrenare.
Alte formulari ale loss function-ului sunt date de interpretarea diferita a variabilei de prezis: daca vom considera ca $y^{(i)} \in \{-1, 1\}\ \forall i$, atunci avem

$$P(y^{(i)}|x^{(i)}, w) = \frac{1}{1 + e^{-y^{(i)} \cdot w^T x^{(i)}}}$$

Descrierea modelului logistic

Se defineste, astfel, Log-likelihood:

$$\log\left(\prod_{i=1}^{m}\frac{1}{1+e^{-y^{(i)}\cdot w^T x^{(i)}}}\right) = -\sum_{i=1}^{m}\log\left(1+e^{-y^{(i)}\cdot w^T x^{(i)}}\right)$$

Vrem sa maximizam log-likelihood, astfel ca problema este echivalenta cu minimizarea (cu regularizare):

$$\min_w C\sum_{i=1}^{m}\log\left(1+e^{-y^{(i)}\cdot w^T x^{(i)}}\right) + ||w||_2^2$$

Implementarea utilizand Gradient Descent

Utilizam forma generala de aplicare pentru Gradient Descent, utilizand notatiile problemei pe care incercam sa o rezolvam,

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} L(w)$$

Avem de calculat

$$\alpha \frac{\partial}{\partial w_j} L(w) = \frac{\partial}{\partial w_j} \big[-\alpha C \sum_i [y^{(i)} \log \sigma_w(x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma_w(x^{(i)}))]$$

$$+ ||w||_2^2] = -\alpha C \sum_{i=1}^{m} \big(\sigma_w(x^{(i)}) - y^{(i)}\big) x_j^{(i)} = -\frac{\alpha}{m} \sum_{i=1}^{m} \big(\sigma_w(x^{(i)}) - y^{(i)}\big) x_j^{(i)}$$

Utilizand o descriere vectoriala, obtinem ca

$$w := w - \frac{\alpha}{m} X^T (\sigma(X \cdot w) - y)$$

Implementarea utilizand Gradient Descent

```
def GradientDescentFit(X_train, y_train):
    m = len(X_train); alpha = 1e-3; coef = alpha/m
    oldW = np.random.rand(len(X_train[0]) + 1)
    oldW = oldW.reshape((len(X_train[0]) + 1, 1))

    localX_train = list(X_train)
    localX_train = np.array(list(map(lambda x: np.concatenate([[1], x]), localX_train)))

    localy_train = np.array(y_train)
    localy_train = localy_train.reshape((len(y_train), 1))

    while 1:
        product = localX_train.dot(oldW)
        applied_sigmoid = np.array(list(map(lambda x: sigmoid(x), product)))
        applied_sigmoid = applied_sigmoid.reshape(localy_train.shape)

        newW = oldW - coef * localX_train.T.dot(applied_sigmoid - localy_train)

        if np.linalg.norm(newW) > np.linalg.norm(oldW):
            break

        oldW = newW

    return newW
```

Implementarea utilizand Gradient Descent

In urma antrenarii si evaluarii modelului se obtine un accuracy score de 0.83.

```
weights = GradientDescentFit(X_train, y_train)
```

```
X_t = list(X_test)
X_t = np.array(list(map(lambda x: np.concatenate([[1], x]), X_t)))
```

```
pred_init = X_t.dot(weights)
```

```
y_pred = list(map(lambda x: round(sigmoid(x)), pred_init))
```

```
precision, recall, accuracy = model_evaluation(y_pred, y_test)
```

```
accuracy
```

```
0.83
```

Implementarea utilizand modelul Perceptron

Pentru modelul Perceptron, optimizarea se realizeaza actualizand vectorul de weights w la fiecare pas,

$$w_{new} = w_{old} + y^{(i)} x^{(i)}$$

Cum valorile pe care le prezicem iau valori in multimea $\{-1, 1\}$, folosindu-ne doar de feature-ul curent, vom avea

$$\hat{y} = sgn(w^T x^{(i)})$$

Vom opri iteratiile dupa un numar setat de pasi (epoci de antrenare), sau dupa ce nu vom mai avea nicio eroare la un anumit pas. Definim eroarea

$$\varepsilon^{(i+1)} = \varepsilon^{(i)} + 1$$

daca $y^{(i)} \hat{y}^{(i)} < 0$.

Implementarea utilizand modelul Perceptron

```
def perceptron(X_train, y_train):
    m = len(X_train); n = len(X_train[0])

    weights = np.zeros((n, 1))
    weights = weights.reshape((n, 1))
    bias = np.zeros((1, 1))

    max_iter = 50

    for iteration in range(0, max_iter):
        print(iteration, '/', max_iter - 1)
        errors = 0
        for i in range(0, m):
            xi = np.array(X_train[i])
            xi = xi.reshape((n, 1))
            predicted = np.sign(weights.T.dot(xi) + bias)

            if y_train[i] * predicted <= 0:
                weights = weights + y_train[i] * xi
                bias = bias + y_train[i]
                errors = errors + 1

        if errors == 0:
            break

        y_pred = np.sign(X_t.dot(weights) + bias)
        precision, recall, accuracy = model_evaluation(y_pred, y_test)
        print('Errors:', errors, 'Train accuracy:', (m - errors) / m, 'Test accuracy:', accuracy)
        print('')

    return weights, bias
```

Implementarea utilizand modelul Perceptron

Se antreneaza modelul utilizand:

Example

```
weights, bias = perceptron(X_train, y_train)
```

Si se obtine, in urma aplicarii a 50 de epoci, rezultatul

```
Errors: 23543 Train accuracy: 0.8114266948609509 Test accuracy: 0.8030565167243368
```

Concluzii

Rezultatele obtinute in urma aplicarii ambilor algoritmi sunt in jurul valorii de 0.8 pentru acuratete. Tinta a fost de a determina daca un anumit text exprima o parere pozitiva sau o parere negativa, textele fiind review-uri de filme (dataset-ul poate fi gasit aici: https://www.kaggle.com/c/movie-review-sentiment-analysis-kernels-only/data).

Datele prelucrate au fost echilibrate, fapt care a influentat obtinerea unui accuracy ridicat utilizand tehnici nu foarte performante (Gradient descent cu pas constant si un model perceptron cu putine epoci de antrenare).

Tehnica tf-idf a ajutat la obtinerea unui feature vector cu informatie sugestiva, fata de ce am fi obtinut daca utilizam doar vectori de frecventa (am reusit sa punctam cuvintele cele mai semnificative pentru clasificare - cele care apar in cat mai putine texte, si cat mai mult intr-un anumit text).

Implementarea completa

Jupyter Notebook-ul se gaseste la adresa:
https://github.com/bogdanmacovei/NLP-Sentiment-Analysis/blob/master/NLPSentimentAnalysis.ipynb.

Sfarsit

Va multumesc pentru atentie!