Enzo POGGIO

Projet 2016 MELS

1 Baseline

Ce que j'ai fait:

Démarche adoptée:

Afin de répondre au problème de manière exhaustive je décide d'adopter la démarche scientifique. Ainsi après une observation minutieuse des données j'ai formulé un ensemble d'hypothèse que j'ai testé par rapport à une affectation aléatoire de la valeur des tweets (Moyenne pos & neg de 33%). Ainsi j'écarte premièrement les traits trop peu distinctifs (par exemple: disinguer les tweets contnant des smileys). Ayant obtenu de bons résultats avec l'addition de la valeurs sentimentales de chaque mot des tweets (avec premièrement Sentiword 40%), je tente corroboré cette hypothése en utilisant une autre liste de mots positifs et négatifs en me basant sur les travaux de la NRC-SentimentAnalysis ayant fini premier dans la "Building the State-of-the-Art in Sentiment Analysis of Tweets". J'utilise alors maintenant la pmilexicon.

Gestion et prétraitement des données de base:

Systèmes modulaire & Dictionnaire de données:

Je choisis de lire une fois le fichier testé et de le mettre en mémoire dans un dictionnaire de données. Cette variable sera modifié, voir rééditer dans la majorité des modules de ce programme. Le systèmes modulaires est plus simple pour essayer une hypothèses portant sur un trait. Je conditionne l'information ainsi :

```
d[i]=(firstNum , secondNum, value, tweet, posScore, negScore)
```

Où i est un int (différent pour chaque tweet)
firstNum, secondNum sont les deux premiers numéros (désignant le numéro utilisateur et le numéro du tweet)
value est la valeur du tweet ["positive", "negative", "neutral"]
tweet est un tableau contenant toutes les chaines de caractère du tweet
posScore, negScore sont des variables de types float qui permetront de définir la valeur du tweet lors de l'affectaion

Description des modules:

readFile(file)
Permet de lire un fichier de tweet
Crée une variable dictionnaire contenant l'enregistrement
de chaque tweet (firstNum , secondNum, value, tweet, posScore, negScore)

writeData(data)
Permet d'écrire les données de façon à ce que le script scoredev.py puisse les utiliser

sentiWordNet(swntxt) & pmilexicon(pmitxt):
Crée et retourne des dictionnaires à partir des ressources entrées.
Ils sont ainsi constitué (voir le code pour plus de détails)
swn[wordkey]=[posScore,negScore, occWord]
pmi[key]=(score, occPos, occNeg)

termeAnalyzerSWN(swn, data) & termeAnalyzerPMI(pmi, data):
Utilise les dictionnaires créé pour pondéré les tweets
Change les valeurs de posScore, negScore
Et retourne le dictionnaire changé

affectationPNN(data) & affectationPNNCoef(data,coef):
Affecte une valeur ["positive", "negative", "neutral"] au tweet
selon posScore, negScore et les coef calculer dans coefficateur(data, version)
Retourne le dictionnaire évalué

randomizer(data)
Permet d'attribuer une valeur aléatoire a chaque tweet
Et retourne un dictionnaire évalué aléatoirement

understandData(data)
Affiche des informations qui peuvent aider à trouver un critère
d'évaluation à partir des données générées

coefficateur(data, version)
Crée des coefficiant de à partir de données empiriques developper avec des tests.

```
        Procédure d'évluation :
                Explication de mes critères d'affectation

Dans affectationPNNCoef(data,coef) nous avons la clé de notre affectation !
Procedure:
Tout d'abord on va initialisé la valeur à neutre

                value="neutral"

Ensuite on s'occupe des tweets ambigues qui pourraient être positif ou négatif il vont dans un
ensemble très tolérant et sont ensuite séléctionner selon leur plus haut score entre negScore et
posScore:

                if (negScore>negAmb and posScore>posAmb ):
                        if negScore>posScore:
                                value="negative"
                        elif negScore<posScore:
                                value="positive"

Et pour finir nous traitons les tweets non-ambigues si leur valeurs dépassent les seuils suivant ils
sont déclarés comme positif ou négatif:

                elif negScore>negNonAmb:
                        value="negative"
                elif posScore>posNonAmb:
                        value="positive"

La valeur de séléction on était trouvé en étudiant les données grâce à la fonction understandData
(data) et coefficateur(data, version) :

Moyenne Positive = 0.6791802897448531    Moyenne Négative = 0.39006549775717836
Minimum Positif = -1.6969848655423443    Minimum Négatif = -3.126057203751694
Maximum Positif = 12.126889641038899     Maximum Négatif = 9.599009709434053

C'est coefficient on était calculé grace au valeur de affecatation(data) qui elle provienne
understandData(data) puis ajuster empiriquement jusqu'à avoir un bon score satisfesant.

                        coef = moyenneAjusté / moyenne

coefposAmb =0.868694232         coefposNonAmb = 0.927588756
coefnegAmb =0.743464884         coefnegNonAmb = 0.820375044
coefmoyenneAmb = 0.806079558    coefmoyenneNonAmb = 0.8739819

Ici j'utilise les moyenne positive et négative comme critère de sélection. Pour l'améliorer de
quelques pourcent je les ai un peu diminuer afin de traiter beaucoup plus de tweets ambigues et un peu
plus de tweets non-ambigues!

        Comment ça marche:
                Comment lancer ma baseline?

Pour lancer ma baseline il vous suffit décrire dans le bash du dossier contenant ma baseline, le
dossier pmilexicon (, le fichier sentiWordNet.txt) et le fichier à traiter:
python3 baseline.py -*le nom du fichier à traiter*- > -*le nom du fichier que vous voulez créer*-
Remplacer les -*balises*- par les noms de fichiers appropriés.
```

Résultats & Conclusion

Avec les données de développement j'arrive à :

Confusion table:

gs \ pred	positive	negative	neutral
positive	259	102	91
negative	46	179	36
neutral	206	223	154

Scores:

class	precision	recall	fscore
positive	(259/511) 0.5068	(259/452) 0.5730	0.5379
negative	(179/504) 0.3552	(179/261) 0.6858	0.4680
neutral	(154/281) 0.5480	(154/583) 0.2642	0.3565
average(pos and neg)			0.5029

Et avec les données de test j'arrive à:

Scoring T13:
- positive: P=48.62, R=51.50, F1=50.02
- negative: P=74.04, R=26.89, F1=39.45
- neutral: P=27.89, R=58.22, F1=37.71
- OVERALL SCORE : 44.73

Scoring T14:
- positive: P=58.04, R=59.92, F1=58.97
- negative: P=63.33, R=21.99, F1=32.65
- neutral: P=25.22, R=46.13, F1=32.61
- OVERALL SCORE : 45.81

Scoring TS1:
- positive: P=42.42, R=33.33, F1=37.33
- negative: P=35.00, R=50.00, F1=41.18
- neutral: P=23.08, R=18.75, F1=20.69
- OVERALL SCORE : 39.25

On remarque une baisse significative entre les résultats d'entraînement et les résultats du test. Certainement une adaptation plus générique au données est nécessaire. De plus on s'aperçoit que les tweets neutre sont largement sous représentés avec mon système une augmentation de leurs rappel serait bénéfique. Il me faudrait de meilleur facteur de dicernement pour ne pas sur-estimer les tweets neutres.

Sources:

pmiLexicon tiré de cette page :
http://saifmohammad.com/WebPages/Abstracts/NRC-SentimentAnalysis.htm

Enzo POGGIO

Projet 2016 MELS

2 Perceptron

Ce que j'ai fait:

Démarche adoptée:

Conception du perceptron:

Premièrement, je me suis attardé sur la conception d'un perceptron pour un aussi grand nombre de données. Je me basais sur mon neurone réalisant la porte & qui donnait des résultats justes, mais qui avait un biais de confirmation à l'intérieur. En effet, je testais ma condition de sortie à chaque entrée de l'ensemble d'entraînement! L'ensemble était de quatre éléments, c'est pour cela que la différence n'était pas majeure et que la convergence arrivait vite. Le perceptron ainsi créé arrivait à convergence mais les résultats avaient complètement perdu leur puissance significative. Les poids descendaient parfois jusqu'à 1*10^-232. De plus, j'utilisais comme condition de sortie la proximité (np.allclose()) entre les anciens et les nouveaux poids. Ceci rendait la convergence très lente et une mise à jour continuelle des poids. Le seul avantage de cette solution était de ne pas utiliser de fonction pour déterminer si l'apprentissage avait été assimilé.
Sur ce premier échec, je décide de repartir sur de nouvelles bases. J'ai donc réalisé un perceptron qui utilise le taux d'erreur commis lors de l'apprentissage pour juger la pondération des poids. J'ai ajouté un compteur d'erreurs et une fonction pour les retourner lors de la mise à jour des poids.

Recherche d'un modèle:

Ma première hypothèse était que si le vecteur traité était de taille minimale, on arriverait plus vite à convergence! J'ai donc créé, premièrement, des vecteurs où les mots étaient rentrés en minuscule sans les hashtags, le linkat et les liens. Je n'arrivais seulement qu'à une convergence avec un taux de 6% d'erreurs, soit environ 42% de réussite avec les données de développement.
Mon erreur était dans mon choix de modèle. En effet, plus un ensemble est varié, plus les traits spécifiques ressortent. Avec mon ensemble réduit, je retardais la convergence en plus de devoir accepter un fort taux d'erreur: beaucoup de variables du vecteur du tweet se recoupaient et donc la discrimination devenait plus difficile.
Deuxièmement, j'ai adapté mon modèle pour que celui-ci accepte la capitalisation et prenne en compte les hastags et les linkats (car même un utilisateur tweeter peut être connoté sentimentalement). Seuls les liens et la ponctuation ne sont pas pris en compte. Avec ce nouvel ensemble vectoriel, j'arrive à faire descendre mon taux d'erreurs à moins de 0,25% soit à peine une cinquantaine d'erreurs pour l'ensemble actuel. J'obtiens donc 48% de réussite avec les données de développement.

Description des modules:

tools(train)
crée les outils nécessaires à l'analyse à partir des données d'entraînement:
dictonary: dictionnaire<mot,indice du tableau vectoriel> créé sur la base de train
vectors: set d'entraînement créé à partir des tweets de train et de dictionary

checkWord(word)
la fonction checkWord(word) sert à trier les mots à évaluer
elle exclut les liens https://...
elle sépare le croisillon ou l'arobase des hashtags et des linkats
elle prend en compte la capitalisation

readFile(file, dictonary)
Permet de lire un fichier de tweets
Crée une variable dictionnaire contenant l'enregistrement
de chaque tweet (firstNum , secondNum, value, tweet, vector)

weightsValue(vectors)
weightsValue calcule les différents vecteurs de poids avec une tolérance d'erreurs
selon leurs valeurs.

neurone(vectors,weight, SelecValue, pourcentErreur)
neurone calcule les poids à partir des poids aléatoires, des vecteurs d'apprentissage
de la valeur sélectionnée et du pourcentage d'erreur.

weightCalculation(entry,weight,t)
weightCalculation est la suite d'opérations pour mettre à jour un poids
entry est le vecteur traité
et t est sa valeur: 1 si est la valeur sentimentale désirée sinon 0

binarisation(x)
binarisation transforme en 1 le paramètre s'il est plus grand de 0
sinon en 0

```
affectationPNN(dictTw,weights)
        affectationPNN affecte une valeur à chaque tweet selon les poids calculés.
        Selon le produit interne du vecteur de chaque tweet à traiter avec les
        différentes valeurs de poids possibles,on sélectionne la valeur max des
        produits internes.
        Et on affecte cette valeur au tweet.

writeData(dictTw,nomfichier)
        Permet d'écrire les données dans un fichier de façon à ce que le script
        scoredev.py puisse les utiliser

        Procédure d'apprentissage & d'évaluation :

                Explication du fonctionnement de mon perceptron:

        def neurone(vectors,weight, SelecValue, pourcentErreur):
                while True :
                        erreur=0
                        for entry in vectors:
                                value=entry[1][SelecValue]
                                weight, e = weightCalculation(entry[0],weight, value)
                                erreur+=e
                        if  erreur<pourcentErreur*len(weight):
                                break
                return weight

        def weightCalculation(entry,weight,t):
                theta=0.17
                o=binarisation(np.dot(entry,weight))
                prediction=np.subtract(t,o)
                deltaWeight= entry*prediction*theta
                newWeight=np.sum([weight, deltaWeight], axis=0)
                erreur=0
                if not prediction==0:
                        erreur=1
                return newWeight, erreur

Mon neurone marche grâce à ces deux modules. Tant que le taux d'erreur n'est pas satisfaisant, on
recommence à lire le set d'entraînement et on met à jour les poids. Le taux d'erreurs influe
directement sur le temps d'exécution et la précision des réponses.

                Explication de mon affectation:

On ne peut pas négliger le fait que les tweets neutres ont un trait distinctif particulier - des mots
peuvent être sentimentalement neutres - au même titre que les positifs et les négatifs. Je ne
considère donc pas deux ensembles de tweets positifs, négatifs et les tweets restants comme neutres,
mais je décide de calculer le potentiel neutre de chaque tweet. Le produit interne le plus grand entre
le vecteur du tweet et les poids sentimentaux détermine la valeur du tweet.

        Comment ça marche:
                Comment lancer mon perceptron2?

Pour lancer mon perceptron2 il suffit de lancer la commande suivante dans un bash UNIX:

        python3 perceptron2.py

Ensuite, le script vous demandera quel fichier vous voulez traiter.
Puis, le nom de fichier de sortie que vous voulez donner.
Après 1 à 3 minutes selon les tolérances d'erreurs choisies, un fichier sera créé
dans le dossier contenant le script.

Exemple:
enzo@EnzoP300:~/.../poggioe0perceptron$ python3 perceptron2.py
Nom du fichier à traiter : development.input.txt
Nommer votre fichier de sortie : development.output (pas besoin de mettre le ".txt")
```

Résultats & Conclusion

Avec les données de développement j'arrive à :

Confusion table:

gs \ pred	positive	negative	neutral
positive	326	23	103
negative	88	69	104
neutral	193	46	344

Scores:

class	precision	recall	fscore
positive	(326/607) 0.5371	(326/452) 0.7212	0.6157
negative	(69/138) 0.5000	(69/261) 0.2644	0.3459
neutral	(344/551) 0.6243	(344/583) 0.5901	0.6067
average(pos and neg)			0.4808

Et avec les données de test j'arrive à:

Scoring T13:
- positive: P=63.47, R=61.77, F1=62.61
- negative: P=24.60, R=43.60, F1=31.46
- neutral: P=66.45, R=59.49, F1=62.77
- OVERALL SCORE : 47.03

Scoring T14:
- positive: P=62.81, R=70.03, F1=66.23
- negative: P=20.00, R=31.91, F1=24.59
- neutral: P=65.08, R=52.34, F1=58.02
- OVERALL SCORE : 45.41

Scoring TS1:
- positive: P=84.85, R=45.90, F1=59.57
- negative: P=10.00, R=50.00, F1=16.67
- neutral: P=84.62, R=64.71, F1=73.33
- OVERALL SCORE : 38.12

Conclusion et remarque:

Aussi bien pour les valeurs de développement que pour celles de test, on remarque que la valeur négative est largement sous-représentée et mal discriminée! Il faudrait peut-être ajouter un biais pour obtenir plus de résultats négatifs (comme augmenter légèrement les poids négatifs).

Pour conclure, on peut dire que ce système est consistant. En effet, entre les résultats du test et du développement, il n'y a pas tant de différence, les résultats restent similaires. En comparaison avec ma baseline, les résultats du test sont assez similaires. En revanche, les résultats des données de développement sont inférieurs de 3% à ceux trouvés ici. Mais cela est dû au fait que ma baseline était trop spécifique.
Maintenant, les seuls facteurs pour avoir de meilleurs résultats sont : le modèle adopté et la tolérance d'erreur acceptée.

Sources:

stopwords tirés de cette page :
http://www.ranks.nl/stopwords