Mini Projet 2

GLO-7050
Été 2020

Gabriel Rieger Junqueira

Résumé

Ce projet a pour but classifier des textes du site web reddit.
Nous allons tester 4 modèles, à savoir:

- Bernoulli Naive Bayes, implémenté avec numpy et pandas
- LinearSVC de Scikit Learn
- DecisionTreeClassifier de Scikit Learn
- LogisticRegression de Scikit Learn

Nous avons utilisé aussi de libraires comme NLTK, Sapcy pour le prétraitement du texte. Ici ce la partie qui fait toute la différence, un bon prétraitement peut faire monter le score.
Comment il s'agit de textes il faut utiliser des fonctions comme CountVectorizer.
J'ai découvert que la normalisation n'a rien changé
Je pense que je pourrais avoir un meilleur score entre 65%-75% d'exactitude avec une très bien fait *feature engineering,* cependant aucune mot a été dit par rapport à ce sujet dans ce cours.
Très difficile monter le score, je soupçonne de l'interprétation des chiffres, liens comme *http://* commentaires qui contiennent *.gif* ou *.jpg* etc.
J'ai essayé faire un GridSearch et Random Search et un Bayes Search pour monter mon score, le résultat a été infime.

Introduction

On commence par charger les données.
Ensuite on procède ou traitements manuels ou avec des libraires comme NLTK.
On continue avec les transformations du texte en chiffre .
On fait la séparation en ensemble d'entraînement et ensemble de test
On exécute l'algorithme de prédiction, ajustement sur l'ensemble d'entraînement et prédiction sur l'ensemble de validation.
Finalement, on évalue les résultats.
Il faut encore générer la prédiction sur le fichier de teste et le soumettre dans Kaggle.

Travail Liée

J'ai utilisé les cours donnés par le Professeur Brahim Chaib-draa, ainsi que des liens envoyées par lui.
À chaque point de difficulté je m'adressait à des sites comme *stackoverflow* ou simplement des recherches des erreurs et des explications sur *Google.*

Jeux de données et environnement

L'ensemble d'entraînement contient 60 milles registres divisé également en 20 catégories, soit 3000 registres par catégorie, j'ai travaillée seulement dans *jupyter lab.*
Comme prétraitement pour Bernoulli Naive Bayes j'ai utilisé un approche customisé plus *tweettokenizer* plus *stopwords*, pour les 3 autres j'ai seulement essayé un prétraitement customisé.
Dans mon prétraitement customisé j'ai essayé de corriger les problèmes causés par des commentaires avec beaucoup de chiffres ou des liens web par exemple.

J'ai fait une analyse descriptive pour voir ou mon modèle était pire et j'ai créé des variables pour ajouter l'information dans ces cas.
Comme résultat j'ai eu un sur ajustement, avec un très bon score dans l'ensemble de validation et un faible dans l'ensemble de test

Approche Proposé

J'ai essayé les 4 modèles suggérés dans le document.
En commençant par le pire:

DecisionTreeClassifier avec un prétraitement customisé, j'ai eu un résultat d'environs 25% d'exactitude.
LosgistRegression avec un prétraitement customisé, j'ai eu un résultat d'environs 43% d'exactitude.
Bernoulli Naive Bayes avec un prétraitement customisé, plus tweettokenizer, plus stopwords j'ai eu un résultat d'environs 45% d'exactitude.
LinearSVC avec un prétraitement customisé, j'ai eu un résultat d'environs 49% d'exactitude.

Après avoir une idée de la performance de chaque modèle je me suis concentré dans le meilleur en essayant monter mon score avec prétraitement, *feature engineering* ou *Grid/Random Search* sans avoir beaucoup de succès.

Après j'ai tenté d'ajouter l'information au modèle en créant des nouvelles *features* comme décrit dans .. et le résultat a été un *overfit*.

Résultats

J’ai essayé avec et sans prétraitement, des fois j’ai eu de meilleurs score sans prétraitement.
On a pas d’espace pour mettre tous mes résultats, alors j’ai décide de montrer l’exactitude de chaque modèle avec et sans prétraitement, en affichant les meilleurs résultats obtenus.

Les résultats sont les suivants:

Decision Tree:

-Avec prétraitement

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 10.4min finished
0.28099999999999997
{'mean_fit_time': array([109.53653725,  78.25729593,  70.16194574,  80.4855183 ,
        58.46328163,  60.53703634,  60.54092733,  61.12919172,
        90.86585164,  84.41392016]), 'std_fit_time': array([ 0.67646766, 17.79949636,  8.16119288,  0.67793147,  0.2942137 ,
        1.62034554,  0.29106425,  1.38394375,  1.22073467,  4.20848265]), 'mean_score_time': array([1.4823854 , 1.37764843, 1.47404965, 1.53144844, 1.42962011,
       1.41660412, 1.37138255, 2.0242219 , 2.19510587, 1.99014982]), 'std_score_time': array([0.1018871 , 0.02819107, 0.1103874 , 0.01523926, 0.06072799,
       0.09494082, 0.10242028, 0.10251352, 0.10093275, 0.65521162]), 'param_clf__criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                   'gini', 'gini', 'gini'],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
```

-Sans prétraitement

```
previsoes_test=grid_search.predict(features_test)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  5.4min finished
0.21385
{'mean_fit_time': array([115.29111938]), 'std_fit_time': array([26.91406565]), 'mean_score_time': array([0.47483046]), 'std_score_time': array([0.23469801]), 'params': [{}], 'split0_
test_score': array([0.20516667]), 'split1_test_score': array([0.218]), 'split2_test_score': array([0.2145]), 'split3_test_score': array([0.21333333]), 'split4_test_score': array([0.2
065]), 'split5_test_score': array([0.2185]), 'split6_test_score': array([0.21816667]), 'split7_test_score': array([0.21966667]), 'split8_test_score': array([0.20833333]), 'split9_tes
t_score': array([0.21633333]), 'mean_test_score': array([0.21385]), 'std_test_score': array([0.00507962]), 'rank_test_score': array([1], dtype=int32)}
```

Losgistic Regression:

-Avec prétraitement

```
print("GridSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start_time), len(random_search.cv_results_['params'])))
report(random_search.cv_results_)
```

```
GridSearchCV took 1462.70 seconds for 5 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.455 (std: 0.005)
Parameters: {'clf__C': 1.3250000000000002, 'clf__penalty': 'l2', 'clf__solver': 'liblinear'}

Model with rank: 2
Mean validation score: 0.453 (std: 0.005)
Parameters: {'clf__C': 2.5500000000000003, 'clf__penalty': 'l2', 'clf__solver': 'liblinear'}

Model with rank: 3
Mean validation score: 0.450 (std: 0.006)
Parameters: {'clf__C': 3.7750000000000004, 'clf__penalty': 'l2', 'clf__solver': 'liblinear'}
```

-Sans prétraitement

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  3.4min finished
/home/gabriel/.local/lib/python3.6/site-packages/sklearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
0.47783333333333344
{'mean_fit_time': array([71.05292385]), 'std_fit_time': array([14.59525171]), 'mean_score_time': array([0.76388927]), 'std_score_time': array([0.2430821]), 'param
est_score': array([0.4805]), 'split1_test_score': array([0.48266667]), 'split2_test_score': array([0.475]), 'split3_test_score': array([0.4595]), 'split4_test_scor
67]), 'split5_test_score': array([0.48033333]), 'split6_test_score': array([0.48316667]), 'split7_test_score': array([0.49133333]), 'split8_test_score': array([0.4
test_score': array([0.48]), 'mean_test_score': array([0.47783333]), 'std_test_score': array([0.00812233]), 'rank_test_score': array([1], dtype=int32)}
```

```
'''
results cv=5 ...0.4705333333333333

results cv=10 ... 0.47783333333333344'''
```

```
'\nresults cv=5 ...0.4705333333333333\n\nresults cv=10 ... 0.47783333333333344'
```

Bernoulli Naive Bayes:

-Avec prétraitement

```
]: acuracia=accuracy_score(classe_valid,label_predict)
   print('Exactitude: ',acuracia)

   acuracia:  0.49725
```

-Sans prétraitement

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
/home/gabriel/.local/lib/python3.6/site-packages/sklearn/model_selection/_search.py:282: UserWarning: The total space of parameters 1 is smaller than n_iter=10. Running 1 iterations.
For exhaustive searches, use GridSearchCV.
  % (grid_size, self.n_iter, grid_size), UserWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  3.8min finished
0.4945833333333334
{'mean_fit_time': array([81.26296055]), 'std_fit_time': array([16.82986061]), 'mean_score_time': array([0.43079157]), 'std_score_time': array([0.11149226]), 'params': [{}], 'split0_t
```

LinearSVC:

-Avec prétraitement

```
grid_search = RandomizedSearchCV(pipeline_LinearSVC, param_dist, n_jobs=-1, verbose=1, cv=3, scoring='accuracy')
grid_search.fit(clean_data, clean_labels)

import time

start_time = time.time()


print(grid_search.best_score_)
print(grid_search.cv_results_)

previsoes_test_pre_proc=grid_search.predict(clean_test)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.3min finished
0.44108333333333327
{'mean_fit_time': array([53.05908418, 17.06678152, 80.83308268, 25.01031343, 99.83305629,
       30.36058156, 91.03038128, 30.93642847, 75.75728003, 23.35302178]), 'std_fit_time': array([1.46971987, 1.39998632, 2.55404485, 0.57691967, 2.41240064,
       2.41454276, 1.14542526, 1.96733421, 0.47638651, 5.01881578]), 'mean_score_time': array([3.17057268, 3.40252924, 4.40425237, 3.54931251, 4.17923951,
       3.78457212, 3.43580214, 3.42118247, 2.78780047, 2.45341412]), 'std_score_time': array([0.19167176, 0.0781308 , 0.17497332, 1.16934937, 0.13059424,
       0.59601594, 0.20097794, 0.87832855, 0.29233179, 0.71186117]), 'param_clf__penalty': masked_array(data=['l2', 'l2', 'l2', 'l2', 'l2', 'l2', 'l2', 'l2', 'l2',
                  'l2'],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
```

-Sans prétraitement

```
'''{}= aqui tem q colocar os parametros a testar
parameters ={}
testar random grid'''
#clf = RandomizedSearchCV(logistic, distributions, random_state=0)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  1.6min finished
0.4946833333333333
{'mean_fit_time': array([53.91757889]), 'std_fit_time': array([10.58227763]), 'mean_score_time': array([0.68023701]), 'std_score_time': array([0.23779654]), 'params': [{}], 'split0_test_score': array([0.49516667]), 'split1_test_score': array([0.48891667]), 'split2_test_score': array([0.49216667]), 'split3_test_score': array([0.50216667]), 'split4_test_score': array([0.495]), 'mean_test_score': array([0.49468333]), 'std_test_score': array([0.00438001]), 'rank_test_score': array([1], dtype=int32)}
'{}= aqui tem q colocar os parametros a testar\nparameters ={}\ntestar random grid'
```

Discussion et Conclusion

J’ai appris que le prétraitement est la f*eature engineering* sont beaucoup plus importants que la normalisation, techniques similaires à normalisation et le *fine tuning* de modèles.
L’expérience de participer dans un compétition Kaggle aussi a été très enrichissante, j’ai trouvé très motivant et même j’ai eu de la difficulté pour savoir l’heure d’arrêter d’essayer d’améliorer mon score. J’ai soumis jusque des teste jusqu’au dernier moment et fini par livrer le rapport une semaine plus tard, j’ai appris qu’il faut être plus pratique et rapide dans mes décisions.
Comme investigations je pense que le seul moyen effective de monter le score est faire une *feature engineering* plus élaborée.

Déclarations de Contributions

Je crois que j’ai passé 70% du temps avec des essais pour le prétraitement, la *feature engineering* et une analyse descriptive détaillée (images en annexe) pour comprendre ou mon modèle était mauvais et monter le score.
J’ai passé 20% du temps pour faire les étapes *fit* et *predict,* avec la validation croisée et des procédures comme TF-IDF, et Normalisation
J’ai utilisé 10% du temps en cherchant la meilleure combinaison de hyper-paramètres avec *Grid search, Randomized Search* ou *Random Bayesian Search*

Annexes

Score par catégorie

```
acertos.sort_values(by='%_acertos',ascending=True)
```

Out[170]:

subreddit	Id	igual	count_subreddit	%_acertos
funny	18543759	114	609	0.187192
AskReddit	17636754	117	590	0.198305
conspiracy	16230416	173	541	0.319778
worldnews	17133584	220	590	0.372881
canada	17330563	275	588	0.467687
Music	19110665	304	630	0.482540
europe	17760953	288	595	0.484034
gameofthrones	17883785	302	589	0.512733
hockey	19345930	323	623	0.518459
trees	18210804	330	618	0.533981
movies	17244947	314	584	0.537671
soccer	18678996	331	613	0.539967
nfl	17980673	332	604	0.549669
GlobalOffensive	18140416	333	592	0.562500
anime	18732233	348	608	0.572368
nba	17411862	348	599	0.580968
Overwatch	18454745	370	608	0.608553
baseball	17815772	376	601	0.625624
leagueoflegends	17762439	389	618	0.629450
wow	18027439	380	600	0.633333

Score par catégorie avec *overfiting*

```
acertos.sort_values(by='%_acertos',ascending=True)
```

t[242]:

subreddit	Id	igual	count_subreddit	%_acertos
funny	18543759	177	609	0.290640
AskReddit	17636754	186	590	0.315254
conspiracy	16230416	245	541	0.452865
worldnews	17133584	286	590	0.484746
Music	19110665	339	630	0.538095
trees	18210804	350	618	0.566343
canada	17330563	338	588	0.574830
gameofthrones	17883785	344	589	0.584041
hockey	19345930	373	623	0.598716
movies	17244947	354	584	0.606164
europe	17760953	368	595	0.618487
soccer	18678996	382	613	0.623165
anime	18732233	380	608	0.625000
nba	17411862	384	599	0.641068
nfl	17980673	388	604	0.642384
GlobalOffensive	18140416	392	592	0.662162
baseball	17815772	409	601	0.680532
wow	18027439	410	600	0.683333
leagueoflegends	17762439	430	618	0.695793
Overwatch	18454745	424	608	0.697368

Règles pour ajouter l'information, création de nouvelles variables, *Feature Engineering*

```
train_Ask['new_comment'] = np.where((train_Ask.prep.str.contains('how')|
                                     train_Ask.prep.str.contains('since') |
                                     train_Ask.prep.str.contains('http') |
                                     train_Ask.prep.str.contains('https') |
                                     train_Ask.prep.str.contains('ask') |
                                     train_Ask.prep.str.contains('who') |
                                     train_Ask.prep.str.contains('why') |
                                     train_Ask.prep.str.contains('which') |
                                     train_Ask.prep.str.contains('what') |

                                     train_Ask.prep.str.contains('jpg')|
                                       train_Ask.prep.str.contains('jpeg') |
                                       train_Ask.prep.str.contains('http') |

                                       train_Ask.prep.str.contains('https') |

                                       train_Ask.prep.str.contains('picture') |
                                       train_Ask.prep.str.contains('gif') |



                                     mexer pouco e so nas que estao ruins


                                      np.where((train.prep.str.contains('http') |
                                    # train.prep.str.contains('https') |    #nao precisa pois esta contido no http

                                       train.prep.str.contains('jpg')|
                                       train.prep.str.contains('jpeg') |


                                       train.prep.str.contains('picture') |
                                       train.prep.str.contains('gif') |

                                    ( train.prep.str.contains('http') & train.prep.str.contains('num')) |


                                      #((train.prep.str.contains('https') & (train.prep.str.contains('num')) |

                                        (train.prep.str.contains('jpg') & train.prep.str.contains('num')) |
                                          (train.prep.str.contains('jpeg') & train.prep.str.contains('num')) |
                                            (train.prep.str.contains('picture') & train.prep.str.contains('num')) |
                                              (train.prep.str.contains('gif') & train.prep.str.contains('num'))




                                      ),'seems' + ' ' + train.subreddit, '.')
```

```
train_funny['new_comment'] = np.where((train_funny.prep.str.contains('jpg')|
                                       train_funny.prep.str.contains('jpeg') |
                                       train_funny.prep.str.contains('http') |

                                       train_funny.prep.str.contains('https') |

                                       train_funny.prep.str.contains('picture') |
                                       train_funny.prep.str.contains('gif') |
                                       train_funny.prep.str.contains('lmao') |


                                       train_funny.prep.str.contains('lol') |

                                       train_funny.prep.str.contains('omg') |

                                       train_funny.prep.str.contains('haha') |









                                       train_funny.comment.str.contains('!!') |
                                       train_funny.prep.str.contains('laugh') |









                                       train_funny.prep.str.contains('num')




                                      ),'is funny', '.')

train_funny
```

```
train_conspiracy=train.loc[train['subreddit']=='conspiracy']


train_conspiracy['new_comment'] = np.where((train_conspiracy.prep.str.contains('how')|
                                    train_conspiracy.prep.str.contains('since') |
                                    train_conspiracy.prep.str.contains('http') |
                                    train_conspiracy.prep.str.contains('https') |
                                    train_conspiracy.prep.str.contains('what') |

                                      train_conspiracy.prep.str.contains('jpg')|
                                      train_conspiracy.prep.str.contains('jpeg') |
                                      train_conspiracy.prep.str.contains('http') |

                                      train_conspiracy.prep.str.contains('https') |

                                      train_conspiracy.prep.str.contains('picture') |
                                      train_conspiracy.prep.str.contains('gif') |


                                      train_conspiracy.prep.str.contains('num')



                                    ),'is conpiracy', '.')

train_conspiracy
```

```
train_new=train


train_new['new_comment'] = np.where((train.prep.str.contains('http') |
                                # train.prep.str.contains('https') |    #nao precisa pois esta contido no http

                                   train.prep.str.contains('jpg')|
                                   train.prep.str.contains('jpeg') |


                                   train.prep.str.contains('picture') |
                                   train.prep.str.contains('gif') |

                                ( train.prep.str.contains('http') & train.prep.str.contains('num')) |


                                  #((train.prep.str.contains('https') & (train.prep.str.contains('num')) |

                                    (train.prep.str.contains('jpg') & train.prep.str.contains('num')) |
                                     (train.prep.str.contains('jpeg') & train.prep.str.contains('num')) |
                                       (train.prep.str.contains('picture') & train.prep.str.contains('num')) |
                                         (train.prep.str.contains('gif') & train.prep.str.contains('num'))




                                  ),'seems' + ' ' + train.subreddit, '.')

train_new
```