

Rapport projet 2 Advanced machine learning

Simon Hervé

Jean-Louis Delebecque

Lien du github: https://github.com/jdelebec/Project_2_advanced_machine_learning

Lien du google collabs:

<https://colab.research.google.com/drive/1iPTIzO4hxZL4bAtYkUHRhbk9hpLmUm32#scrollTo=hG51ftLyyPaD>

I) Introduction

Pour ce projet nous avons dû répartir notre précédent projet, l'ancien objectif était de classer les articles par rapport à leur thème, essayer de les résumer en quelques mots clés.

Pour ce projet nous devons maintenant prédire l'année de publication d'un article, en utilisant toujours l'analyse de langage.

Nous avons donc extrait tous les corpus grâce à beautiful soup (les fichiers étaient en xml), puis nous avons mis toutes les données dans un dataframe pandas contenant de base 13 colonnes:

Voici le déroulé du code:

Projet Advanced Machine Learning: Text synthesis

- 1) Extraction des textes
- 2) Librairies
- 3) Processing et features engineering
- 4) Suite du processing (projet 2)

Modeling

- 1) Data split
- 2) Word2Vec
- 3) TF-IDF + Random_Forest_Classifier
- 4) Optimisation des features
- 5) Optimisation des paramètres
- 6) SVM

II) Processing and features engineering

Pour ce qui est du texte nous avons repris de notre précédent processing (nous en avons testé 8) et avons gardé le processing 8:

1) Processing du texte

```
def processing8(text):  
  
    text = str(text)  
    text = drop_brackets(text)  
    text = lower_letters(text)  
    text = lemmatize(text)  
    text = remove_stop_words(text)  
    text = remove_accents(text)  
    text = drop_slash_n(text)  
    text = drop_special_carac(text)  
    text = drop_double_space(text)  
    text = drop_single_char(text)  
  
    return text
```

On y fait beaucoup de cleaning (retirer les crochets, majuscules, les stop words français, accent, caractère spéciaux, espace en trop et les caractères seuls (causés par les autres traitements)).

On lemmatize aussi le tout pour changer la forme des mots.

Le texte de départ:

```
df["title"][0]  
  
'\nÉléments de conception d'un système d'interprétation automatique de textes par des images\n'
```

Texte après le processing:

```
df["title"] = df.apply(lambda x: processing8(x["title"]), axis=1)  
df["title"][0]  
  
'element conception systeme interpretation automatique texte image'
```

On applique ce traitement aux colonnes "title" et "abstract_fr".

2) Processing des autres features

Pour les autres features nous avons:

- Supprimé la colonnes "monogr_date" : car elle contenait les mêmes informations que publication_date.

```
df[df["publication_date"] == df["monogr_date"]].count()
# la colonne monogr_date et publication_date sont toujours les mêmes
```

title	1602
-------	------

- Supprimé les colonnes "keywords_fr", "keywords_eng" et "abstract_eng" car elles contenaient beaucoup trop de valeurs 'None'.
- Converti la colonne cible "publication_date" en int, ce qui nous a donné des info grâce à la fonction describe.

```
[339] df["publication_date"].describe()

count    1602.000000
mean     2009.431960
std       5.414435
min      1997.000000
25%      2005.000000
50%      2010.000000
75%      2014.000000
max      2019.000000
Name: publication_date, dtype: float64
```

Ici on apprend que le dataset est réparti sur 22 ans de 1997 à 2019 et la médiane est vers 2010

- On a aussi split les auteurs en liste d'auteur, mais cela n'a aucun impact au final.
- On crée aussi une nouvelle colonne: "text_title_proc" qui est une fusion des colonnes title et abstract_fr sur lesquelles est appliqué le processing 8.

III) Modeling

1) Split du dataset

Pour split le dataset nous avons utilisé la fonction train_test_split de scikit-learn, en utilisant le paramètre stratify sur la colonne des années pour obtenir une répartition homogène en fonction des années.

```
#train test validation split
x_train, x_test, y_train, y_test = train_test_split(df, df.iloc[:,4], stratify = df.iloc[:,4], test_size=0.1, random_state=1) # 90 % train et 10 % test
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, stratify = y_train, test_size=0.11, random_state=1) # 0.9 * 0.11 = 0.099
```

Pour avoir des sets respectant le 80/10/10 alors que cette fonction ne permet que de split en deux sets différents nous avons effectué cette fonction deux fois.

Une fois 90 % train set et 10 % test set.

Une deuxième fois sur le train set 89 % train et 0.11 validation set.

Ce qui nous donne finalement: 80 % train set, 10 % test set et 9.9 % validation set.

Pour avoir une idée de l'efficacité du paramètre stratify dans le split voici une comparaison:

Sans stratify					Avec stratify = publication_date				
	dataset	train	test	val		dataset	train	test	val
1997	0.187266	0.156006	NaN	0.628931	1997	0.187266	0.234009	NaN	NaN
1998	0.811486	0.858034	0.621118	0.628931	1998	0.811486	0.858034	0.621118	0.628931
1999	2.871411	3.120125	3.105590	0.628931	1999	2.871411	2.808112	3.105590	3.144654
2000	1.435705	1.326053	1.863354	1.886792	2000	1.435705	1.482059	1.242236	1.257862
2001	3.308365	3.510140	3.726708	1.257862	2001	3.308365	3.354134	3.105590	3.144654
2002	3.558052	3.510140	3.105590	4.402516	2002	3.558052	3.510140	3.726708	3.773585
2003	3.995006	4.056162	4.347826	3.144654	2003	3.995006	4.056162	3.726708	3.773585
2004	5.368290	5.382215	6.211180	4.402516	2004	5.368290	5.304212	5.590062	5.660377
2005	5.430712	5.382215	3.726708	7.547170	2005	5.430712	5.382215	5.590062	5.660377
2006	5.056180	4.802184	6.211180	7.547170	2006	5.056180	5.070203	4.968944	5.031447
2007	5.493134	5.148206	4.968944	8.805031	2007	5.493134	5.460218	5.590062	5.660377
2008	4.119850	3.900156	4.347826	5.660377	2008	4.119850	4.056162	4.347826	4.402516
2009	6.491885	7.020281	3.105590	5.660377	2009	6.491885	6.552262	6.211180	6.289308
2010	6.491885	5.382215	11.801242	10.062893	2010	6.491885	6.474259	6.832298	6.289308
2011	7.303371	7.566303	4.347826	8.176101	2011	7.303371	7.254290	7.453416	7.547170
2012	5.430712	5.538222	4.968944	5.031447	2012	5.430712	5.382215	5.590062	5.660377
2013	6.429463	6.474259	8.695652	3.773585	2013	6.429463	6.474259	6.211180	6.289308
2014	5.930087	6.240250	5.590062	3.773585	2014	5.930087	5.928237	6.211180	5.660377
2015	5.742821	5.850234	6.211180	4.402516	2015	5.742821	5.772231	5.590062	5.660377
2016	3.370787	3.276131	4.347826	3.144654	2016	3.370787	3.432137	3.105590	3.144654
2017	3.558052	3.900156	1.863354	2.515723	2017	3.558052	3.510140	3.726708	3.773585
2018	5.243446	5.460218	5.590062	3.144654	2018	5.243446	5.304212	4.968944	5.031447
2019	2.372035	2.340094	1.242236	3.773585	2019	2.372035	2.340094	2.484472	2.515723

On remarque que les différences de proportion ont disparu quand on a ajouté le paramètre stratify.

Comme nous n'avons que 0.187 % de données en 1997 pour le dataset original, il se trouve que toutes ces données ne peuvent être à la fois dans les 3 sets (train, test et val) pour une raison de logique mathématique.

On a donc supprimé les lignes contenant l'année 1997.

2) Word2Vec

Nous avons essayé un word2Vec sur la colonne "text_title_proc", mais le vocabulaire résultant de la première étape n'était pas satisfaisant nous sommes donc directement passé au TF-IDF.

```
words = list(w2v.wv.vocab)
print(words)

['the', 'mitkov', 'algorithm', 'for', 'anaphora', 'resolution', 'in', 'portuguesenone', 'o', 'r',
```

3) TF-IDF

Pour pouvoir répondre à ce problème, il est indispensable de vectoriser notre texte, le word2Vec ne nous a pas suffi alors nous avons essayé le TF-IDF.

Après plusieurs essais nous avons trouvé qu'utiliser une pipeline pour count_vectorizer et puis transformer les données étaient le mieux.

```
pipe = make_pipeline(CountVectorizer(), TfidfTransformer())
pipe.fit(x_train["text_title_proc"])
feat_train = pipe.transform(x_train["text_title_proc"])
feat_train.shape

(1280, 8201)
```

On retrouve bien 1280 lignes (comme dans le set train).

4) Random Forest Classifier

Premier modèle testé le random forest pour la classification. On l'utilise sur nos données vectorisées par le TF-IDF.

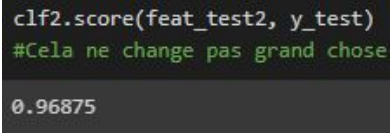
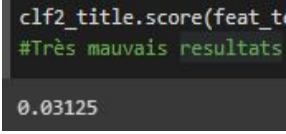
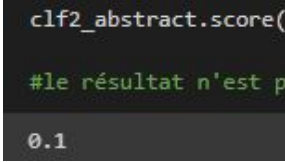
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50)
clf.fit(feat_train, y_train)
```

La précision sur la colonne "text_title_proc" est de 0.1125.

```
clf.score(feat_test, y_test)

0.1125
```

Nous avons essayé la même chose sur “publication_place”, “title” et “abstract_fr_proc8” afin de déterminer quels textes est le mieux à utiliser.

Features	publication_place	title	abstract
Screen			
Précision	0.96875	0.03125	0.1

Aucune de ces précisions n’est supérieure, de plus publication_place contient très peu d’informations et montre juste le fait qu’une précision d’ environ 0.1 ne veut rien dire de pertinent.

5) Optimisation des hyper-paramètres

Nous allons donc optimiser les paramètres du modèle random Forest sur la feature “text_title_proc”.

Pour cela nous avons effectué un GridSearchCV:

```
[387] #On effectue donc notre gridsearch sur notre premier modèle: clf

param_grid = {
    'n_estimators': [50 , 100, 150 , 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [50,100,150,200
    ]
}
GSC = GridSearchCV(estimator=clf, param_grid=param_grid, cv= 5)
GSC.fit(feet_train, y_train)
print(GSC.best_params_)
```

Nous avons choisis 3 paramètres:

- n_estimators (nombres d’estimateurs, on privilégie un grand nombre)
- max_features (taille des sous-datasets aléatoires a considérer quand on split un noeud)
- max_depth (profondeur de l’arbre).

Les best paramètres trouvés sont les suivants:

```
{'max_depth': 100, 'max_features': 'sqrt', 'n_estimators': 200}
```

On a donc fit un nouveau modèle avec ces hyper-paramètres.

```
clf_final = RandomForestClassifier(max_depth=100, max_features = 'sqrt', n_estimators = 200)
clf_final.fit(feet_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=100, max_features='sqrt',
                        max_leaf_nodes=None, max_samples=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=200,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

```
clf_final.score(feet_test, y_test)
```

```
#On a essayé plusieurs paramètres dans le gridsearch ça n'a jamais dépassé 0.1125 ce qui est notre m
```

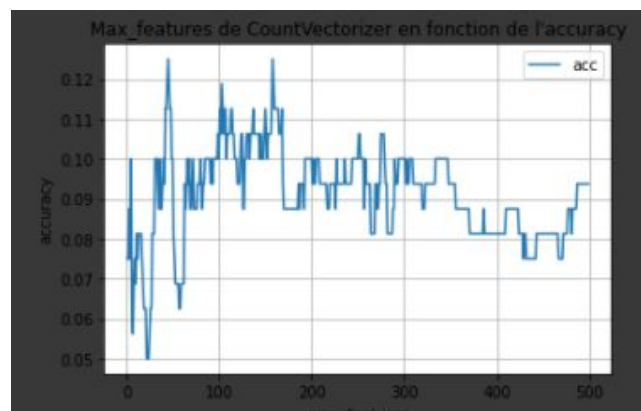
```
0.1125
```

La précision n'a pas changé malgré un changement des hyper-paramètres.

Nous avons remarqué (pas dans le notebook rendu) qu'en augmentant le nombre d'estimateurs (`n_estimators`) la précision augmentait légèrement (comme la forêt était plus grande).

6) SVM

Le deuxième modèle de machine learning que nous utilisons est SVM. On cherche à optimiser cet algorithme sur l'hyper-paramètre `max_features` entre 1 et 500 pour obtenir la meilleure accuracy. L'accuracy est testée sur le testset. On s'aide du graphique pour avoir une bonne interprétation



IV) Conclusion

En conclusion nous avons utilisé deux modèles de machine learning, RandomForestClassifier et SVC, pour une accuracy maximale de 12, 5% sur le testset. Dans ce projet nous avons utilisé des techniques de preprocessing de texte pour simplifier leur transformation et pouvoir utiliser les algorithmes cités ci-dessus.