Text classification

CONSULTING BSTORM

Comment traiter du texte?

1. La tokenization
2. Tf-Idf
3. Preprocessing/nettoyage
4. Construction d'un modèle
5. Imblearn, ensemble, spaCy

1. Tokenization

Découper le texte en unité

- “Je mange une pomme !” => [‘je’, ‘mange’,’une’,’pomme’,’!’]
- 1 token == 1 bout de phrase

Les stop words = les mots qui n’apportent pas de sens:

- Les déterminants, sujets, la ponctuation, *et*
  - “**Je** mange **une** pomme **!**” => [‘mange’,’pomme’]

1. Tokenization

Stemming vs lemmatization

Le stemming: prendre la plus petite racine commune

e.g.: jouez, jouons, joue, jouet => Jou

La lemmatization: forme non fléchie des mots et verbes:

=> jouer et jouet

1. Tokenization

“manger une pomme” != “manger un homme”

Unigram: mot unique

bigram:

- e.g.: [‘je mange’,’mange une’,’une pomme’,’pomme !’]

On peut mélanger uni et bi gram:

- e.g.: [‘Je’, ‘mange’,’Je mange’, ‘une’,‘pomme’,‘une pomme’,‘!’,’pomme ! ’]

2. Tf-Idf

Méthode pour transformer les tokens en données exploitables:

Tf-Idf: '*Term Frequency - Inverse Document Frequency*'

e.g.: ['Je mange une pomme', 'Je mange une pomme verte']

TF:

je	mange	pomme	une	verte
0.25	0.25	0.25	0.25	0
0.2	0.2	0.2	0.2	0.2

2. Tf-Idf

Quel est le mot qui apporte le plus d'information dans l'exemple?

=> le mot 'vert' car il permet de discriminer les deux

=> il a le même poids que les autres

Idf permet de ne prendre en compte ce problème en apportant plus de poids au mots qui ne sont pas présents partout

idf = log [(1 + n) / (1 + df(t))] + 1

où **n**=nombre total de documents

df(t)=nombre de documents où l'on retrouve un mot

2. Tf-Idf

IDF: e.g.: ‘vert’ idf = log[(1+2)/(1+1)]+1 = 1,4055

je	mange	pomme	une	verte
1	1	1	1	1.40547

Il suffit de multiplier **TF** par **IDF**:

je	mange	pomme	une	verte
0.5	0.5	0.5	0.5	0
0.40909	0.40909	0.40909	0.40909	0.574962

2. Tf-Idf

[sklearn.feature_extraction.text.CountVectorizer]

[sklearn.feature_extraction.text.TfidfVectorizer]

2. Tf-Idf

Shape d'une matrice tf-idf = (n_docs, n_tokens)

Une matrice remplie de 0 == sparsity élevée

2 conséquences:

- matrice volumineuse remplie de non information:
  - sparse array permet de ne stocker que les éléments non nulls
- matrice d'une très grande dimension:
  - 'Dimensionality curse'
  - Trop d'information tue l'information

2. Tf-Idf

Réduire les dimensions:

- prendre moins de mots:
  - TfIdfVectorizer(max_features=None): permet de ne garder que les X mots les plus courants
- sklearn.decomposition.TruncatedSVD:
  - Permet de réduire les dimensions comme une PCA
- Un meilleur preprocessing:
  - Retirer les caractères spéciaux
  - Retirer toutes les choses pas utiles: Regex, spaCy

3. Meilleur preprocessing

Deux outils pour aborder le problème des fautes de frappes:

- Distance de Hamming:
  - nombre d'opérations nécessaires pour faire correspondre deux mots
  - Comparaison lettre par lettre
  - e.g.: 'abcdefg' et 'bcdefgh' distance de Hamming = 7
    - Totalement différent
- Distance de Levenshtein:
  - Nombre d'insertions, suppressions ou substitution
  - e.g.: 'abcdefg' et 'bcdefgh' distance de Levenshtein = 2
    - Suppression du **a** et ajout du **h**

4. Modèles

Quels modèles pour la classification de textes?:

- Eviter les arbres de décisions:
  - Très lents et très mauvaises performances sur des matrices de grandes dimensions
- Naive Bayes
- Régression logistique
- Analyse discriminante linéaire
- SVM

5. Imblearn

On a rarement des datasets avec des classes équilibrées:

- Les modèles peuvent avoir des mauvaises performances sur les classes minoritaires

Méthode pour gérer les déséquilibres:

- dans sklearn l'hyper paramètre *class_weight* peut aider. Utiliser *priors* pour les méthodes basées sur le théorème de bayes
- Le resampling => imblearn

5. Imblearn

3 méthodes pour le resampling:

- L'over sampling: créer des copies de la classe minoritaire
  - Ajoute de l'information et du bruit
  - Augmente les temps d'entraînement
- L'under sampling: retirer des individus de la classe majoritaire
  - Retire de l'information
- Une combinaison des deux

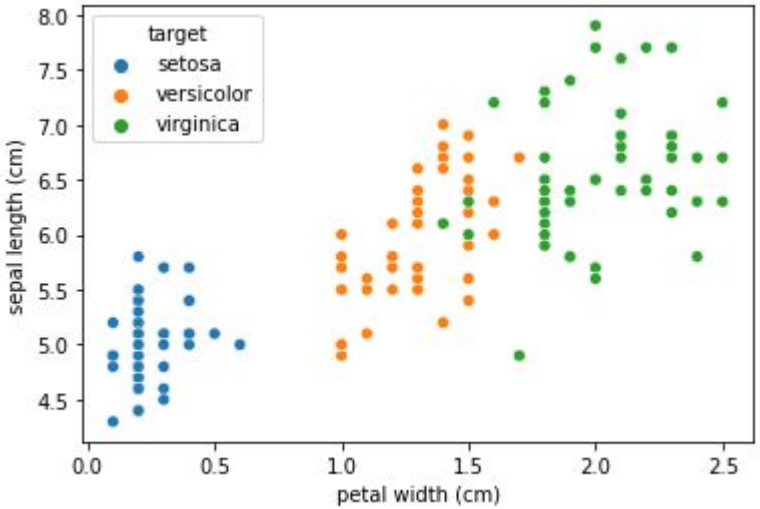
pip install imblearn

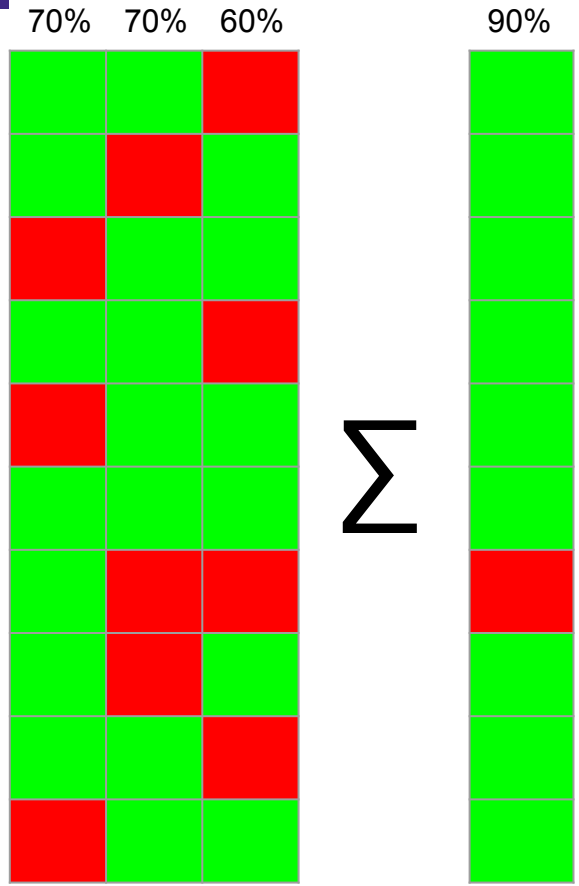
5. Imblearn

- RandomOverSampling:
  - On copie au hasard des individus de la classe minoritaire
- SMOTE:
  - KNN sur chaque individu
  - Choisir aléatoirement un KNN
  - Pour chaque feature:
    - différence entre le KNN et l'individu qui a servi à générer le KNN
    - générer un nombre entre 0 et 1
    - multiplier la différence par ce nombre
- ADASYN

5. Imblearn

- SMOTE Tomek
  - SMOTE + suppression des “tomek links”
  - Tomek links = pair d’individus proches mais de classes différentes





Aucun des 3 modèles ne dépassent les 70% de bonnes classifications

Si on combine les résultats on passe à 90%

Wisdom of the crowd - sagesse de la foule

Une foule de personnes éclairées aura plus souvent raison qu'un seul expert

Applicable en machine learning

Attention: la performance de chaque modèle doit au moins être égal à 51%

Sinon convergence vers 0

6. Bagging

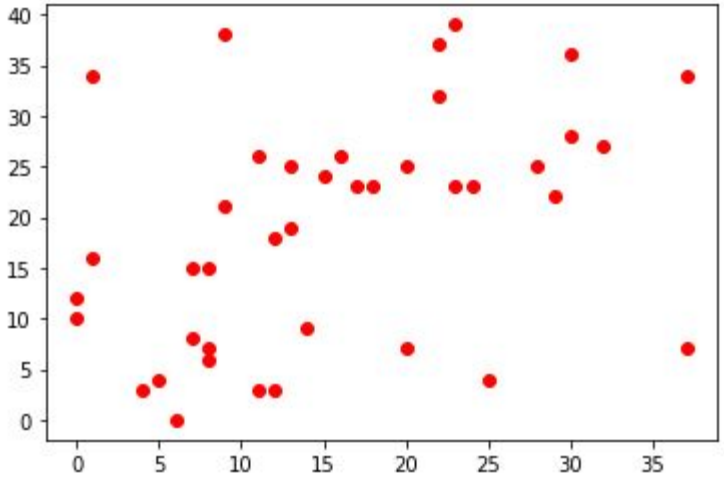
Bootstrap **Agg**regat**ing**

L’idée ici est d’entraîner une foule d’un même modèle mais en utilisant le bootstrapping

Bootstrapping: méthode d'échantillonnage où l’on tire au hasard des individus et que l’on replace après

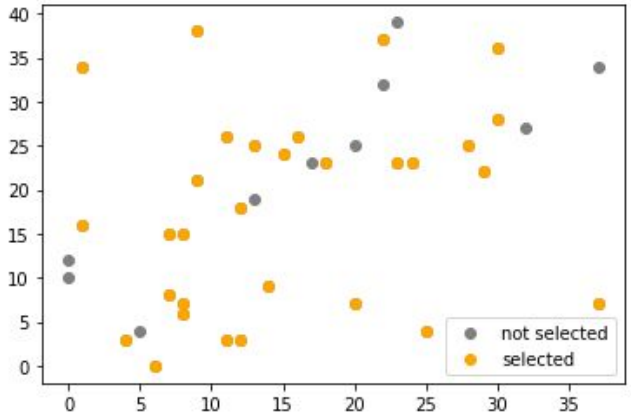
6. Bagging

Dataset original

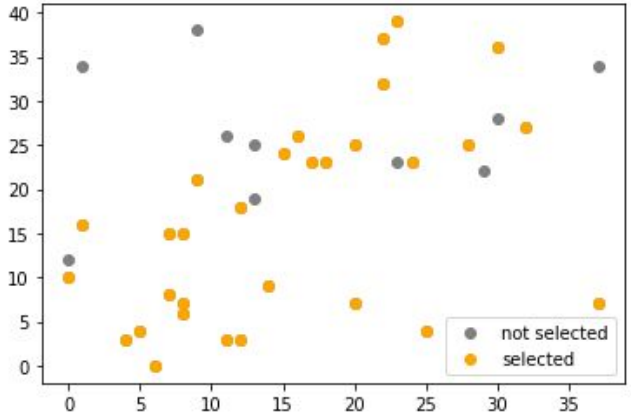


Bootstrapping

modèle 1



modèle 2



6. Bagging

Les modèles partagent donc une partie de l'information du dataset

L'idée est de "**moyenner**" le résultat de tous les modèles

Un des plus connus: sklearn.ensemble.RandomForestClassifier:

- L'idée est d'entraîner une série d'arbres de décision

sklearn.ensemble

6. Boosting

On entraîne une succession de modèle faible

Sur base des erreurs du précédent on modifie le poids des individus

Plus de poids donné aux erreurs

Star du boosting: XGBoost (e**X**treme **G**radient **B**oosting):

- Compatible sklearn
- Calcul sur unité graphique

6. Bagging vs Boosting

Bagging:

- entraînement en **parallèle**
- foule d'**experts**
- pris individuellement, les modèles sont en **overfitting**
- la foule permet de réduire la **variance**

Boosting:

- entraînement **successif**
- foule de modèle **faible**
- pris individuellement, les modèles sont en **underfitting**
- permet de réduire le **biais**

6. Stacking

On entraîne des modèles **différents**

Ensuite on entraîne **par dessus** ces modèles un modèle qui doit déterminer qui a raison

Si les modèles ne sont pas différents ça ne sert à rien

7. spaCy

pip install spacy

nltk = pédagogie, recherche

spaCy = pour la production

C pour cython => plus rapide

17 langues supportées + multi langue

Tokenisation, preprocessing, classification(deeplearning), *etc*.

7. spaCy

modèles pré entraînés

Il faut les ajouter manuellement:

- spacy download fr_core_news_sm

démo: *cfr.* Jupyter Notebook

Tutoriel très complet sur spaCy