Rapport d’expériences en machine learning

Laurane Castiaux

Contexte

En classe, nous avons entraîné trois modèles grâce au module Scikit-learn de python sur les données issues du fichier « all_tweets.csv ». Le but était de retrouver le « party » (« EM » ou « FN ») sur base du « text ». Voici les scores d’exactitude (accuracy) obtenus pour chaque modèle vu en classe (l’approximation du score est due au fait que le corpus total est redécoupé en sous-corpus d’entrainement, de développement et de test à chaque nouveau lancement du programme) :

- Decision Tree : c. 0.74
- Support Vector Classificator : c. 0.85
- Multi-layer Perceptron classifier : c. 0.84

Pour tous les résultats suivant, je me baserai toujours sur le même corpus d’entrainement (80 % du corpus total et j’utiliserai une cross-validation en 5 passes. Mon code est disponible, joint à ce document ou sur mon git-hub (github.com/Reyla98/ScikitLearnExperiences).

Expérience 1 : paramétrisation de CountVectorizer()

J’ai commencé par définir token_pattern comme la regex que nous a partagée M. Watrin (voir dans le code). Elle est faite pour le français, elle devrait donc mieux fonctionner que la fonction utilisée par défaut.

Voici les scores obtenus :

- Decision Tree : 0.76
- Support Vector Classificator : 0.84
- Multi-layer Perceptron classifier : 0.83

On ne voit pas vraiment d’amélioration.

J’ai ensuite ajouté une liste de stop-words (voir dans le code). Je n’y ai inséré que des mots grammaticaux étant donné que, pour notre tâche, des mots lexicaux, même très courants, pourraient être plus utilisés par les adhérents d’un parti ou de l’autre (j’ai fait une exception pour les formes « a », « est » et « été »).

Scores d’exactitude:

- Decision Tree : 0.77
- Support Vector Classificator : 0.84
- Multi-layer Perceptron classifier : 0.83

A nouveau, pas ou peu d’amélioration.

Après, j'ai ajouté ma propre fonction utilisée comme argument preporcessor de CountVectorizer(). Celle-ci filtre les « RT » qui signalent qu'un tweet est un retweet. Je ne désire pas filtrer autre chose (par exemple les hashtags ou les mentions) car tout le reste me semble contenir des informations pertinente pour différencier les adhérents d'un parti ou de l'autre.

Scores d'exactitude:

- Decision Tree : 0.75
- Support Vector Classificator : 0.84
- Multi-layer Perceptron classifier : 0.83

Pas de grand changement, si ce n'est une légère baisse pour l'arbre de décision. Je décide donc de ne pas garder ce paramètre pour la suite.

Expérience 2 : tester un autre algorithme

Nearest Neighbors

Pour cette seconde expérience, j'ai choisi d'utiliser le KNeighborsClassifier implémenté dans sklearn.

Le fonctionnement est le suivant : pour chaque nouveau document à classer, l'algorithme recherche les k documents les plus proches parmi le corpus d'apprentissage ; la classe la plus représentée est alors choisie en sortie.

Une variante de cet algorithme, RadiusNeighborsClassifier, est également implémentée dans sklearn ; au lieu de sélectionner les k plus proches voisins, il sélectionne tous les voisins présents dans un rayon définit. Cette approche est mieux adaptée lorsque les données ne sont pas uniformément échantillonnées, mais elle fonctionne moins bien lorsque le nombre de dimensions augmente (« malédiction de la dimension », comme vue au cours avec M. Naets), il vaut donc mieux ici privilégier la première approche.

J'ai fait une grid search sur les paramètres leaf_size et n_neighbors. Les meilleures valeurs retournées étaient respectivement 1 et 5. J'ai donc utilisé ces valeurs pour entrainer le classifieur. J'ai obtenu un score d'exactitude de 0.70.

Par défaut, l'algorithme donne autant de poids à chaque voisin sélectionné, sans tenir compte de la distance de chacun par rapport au document à classer. Il est possible de modifier cela et de donner un poids proportionnel à la distance entre le document à classer et chacun des voisins. Étant donné que je ne considère que les 3 plus proches voisins, je ne pense pas avoir de différence significative dans les résultats, mais j'ai tout de même testé. Cela m'a effectivement donné le même score de 0.70.

Naive Bayes

J'ai voulu tester une approche bayésienne. Ces approches font l'hypothèse (dite bayésienne) d'indépendance, c'est à dire que chaque caractéristique du jeu de donnée est indépendante des autres. En linguistique, on sait bien que le choix d'un mot n'est pas indépendant de son contexte, mais pourtant cette hypothèse fonctionne parfois. En lisant la documentation sur les différentes approches bayésiennes de sklearn, il me semble (mais sans certitude) que celle qui correspond le mieux à notre problème est le ComplementNB. Je teste donc ce nouvel algorithme.

Il y a un paramètre ($0 \leq \theta \leq 1$), pour lequel je fais une grid search. J'obtiens 0.5 comme meilleure valeur.
Je teste une nouvelle fois la classification et j'obtiens 0.85 d'exactitude, résultat similaire au SVM et au multi-layer perceptron.

Expérience 3 – Classification par ensembles – Voting Classifier

Le but des méthodes d'ensemble est de combiner des prédictions qui se basent sur plusieurs estimateurs produits par des algorithmes d'apprentissage différents, et ce afin améliorer la robusté d'un estimateur. Il existe deux familles de méthodes par ensembles :

- les méthodes de moyennes : on fait la moyenne des estimations de chaque « sous-algorithme », cela a pour effet de réduire la variance et donc de produire un résultat plus précis que chaque algorithme pris séparément.
- Les méthodes de « boosting » : chaque estimateur est construit après le précédent et tente d'en réduire le biais.

Le VotingClassifier fait partie de la première famille. Il va sélectionner la classe attribuée par le plus grand nombre d'algorithmes (hard voting) ou qui a la plus grande probabilité en additionnant la probabilité attribuée par chaque modèle (soft voting). Il est précisé dans la documentation de sklearn que cet algorithme est utile lorsque plusieurs modèles ont des performances similaires. Je vais donc d'abord essayer de combiner uniquement le SVM, le perceptron et le modèle bayésien, puisqu'ils obtiennent tous les trois des résultats très similaires. J'ajouterai les deux autres modèles dans un second temps pour voir quelle influence cela aura sur le résultat.

Premier résultat obtenu (VotingClisifier avec SVM, perceptron et Naive Bayes) : 0.844 d'exactitude.
Second résultat obtenu (idem que ci-dessus, avec arbre de décision et Kneighbors en plus) : 0.846.

A mon grand étonnement, le résultat est pratiquement le même dans les deux cas., et malheureusement, ce résultat n'est pas meilleur que ceux qu'on avait déjà obtenus auparavant.

J'ai voulu voir la différence entre le « hard » et le « soft voting », mais lorsque j'ai entré « soft » en paramètre, l'erreur suivant m'a été retournée : 'LinearSVC' object has no attribute 'predict_proba'. Puisque le « soft voting » se base sur la probabilité attribuée à chaque classe par chacun des modèles, il est logique que cela ne fonctionne pas avec le LinearSVC puisqu'apparamment, celui-ci n'attribue pas de probabilité à chaque classe (ce n'est d'ailleurs peut-être pas le seul classifieur qui serait problématique ici, mais c'est en tout cas celui-là qui m'a systématiquement retourné l'erreur). Je ne pourrai donc pas tester ce « soft voting ».

Évaluation finale

Après avoir fait toutes ces expériences sur mon corpus de développement, j'ai testé mon dernier classifier sur le corpus de test. Pour cela, j'ai sélectionné 20 % de mon corpus de développement comme corpus d'apprentissage (afin d'avoir la même taille de corpus d'apprentissage que lors de mes cross-validations) et j'ai classifié le corpus d'apprentissage avec le VotingClassifier. Le résultat obtenu est de 0.85.