

Présentation du projet de recherche : What's
cooking ? Use recipe ingredients to categorize the
cuisine

Ayman HAMZAOUÏ & Julien DELAUNAY

9 avril 2019

Table des matières

1	Introduction	3
2	Les données	3
3	Traitement des données	4
4	Les méthodes utilisées	4
4.1	Les K plus proches voisins	4
4.2	Réseaux de neurones	5
4.3	Machine à Vecteur de Support	5
4.4	Forêt aléatoire	5
4.5	Naive Bayes	6
4.6	Regression logistique	6
4.7	Adaboost	6
5	Les résultats	7
6	Conclusion	7
A	Choix du design	8
B	Utilisation de Git	8

1 Introduction

Nous avons choisi la base de données *"What's cooking? Use recipe ingredients to categorize the cuisine"*¹ parmi les bases de données de Kaggle. Le domaine de cette base de données nous est apparu intéressant et nous avons remarqué que les données fournies semblent pouvoir être séparés et ainsi analysées avec les outils appris en cours.

2 Les données

Les données fournies par Kaggle sont sous la forme de deux documents json. Le fichier de test et le fichier d'entraînement. Pour le fichier d'entraînement, nous avons un id de la recette, le type de cuisine pour cette recette et les ingrédients nécessaires à sa fabrication.

Nous avons effectué des recherches sur les données fournies afin d'obtenir quelques informations. Nous avons ainsi découvert que les cuisines utilisent entre 1 et 65 ingrédients par recette. Les ingrédients les plus utilisés varient en fonction des cuisines, il est donc intéressant d'observer si un modèle peut prévoir en fonction des ingrédients à quelle cuisine la recette appartient.

Nous avons observé que la base de données comportait 20 recettes, c'est donc un problème de classification avec des classes multiples. Il y a plus que 2 catégories à prédire.



FIGURE 1 – Repartition des ingrédients

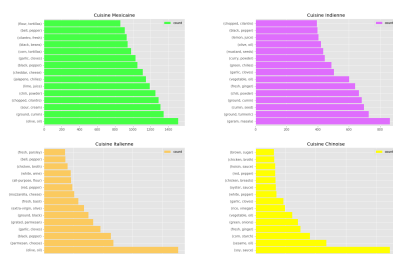


FIGURE 2 – Repartition des ingrédients par cuisine

1. <https://www.kaggle.com/c/whats-cooking-kernels-only>

3 Traitement des données

La partie "processing" de notre projet s'est basée sur TF-IDF. C'est une méthode de pondération qui permet d'évaluer l'importance d'un terme contenu dans un document, relativement à son ensemble. Pour ce faire, nous avons commencé par effectué une tokenisation sur le nom des ingrédients afin de bien les séparer et ne pas risquer d'erreurs par la suite dans le traitement des données. Nous avons pensé particulièrement au traitement de TF-IDF qui nécessite un token de séparation.

Par la suite, nous avons remplacé les ingrédients par des indices grâce à la fonction `"label_train_transform"` qui peuvent ensuite être retourné sous leurs noms originaux avec la fonction `"label_test_inverseTransform"`

4 Les méthodes utilisées

Pour le traitement des données et la recherche d'un modèle, nous avons mis en place 7 méthodes différentes à partir de scikit learn. Nous avons utilisé chaque méthode sous la forme d'une classe différente que nous avons appelée tour à tour dans notre programme principal en cherchant les meilleurs hyper-paramètres. Pour la recherche d'hyper-parametres, nous avons utilisé *GridSearch*.² Nous avons donc implémenté manuellement différentes valeurs, et *GridSearch* nous aide à trouver les meilleurs hyper-paramètres. Pour chacune de ces méthodes, dans le cas d'utilisation de *GridSearch*, nous avons initialisé le nombre de processus à utiliser à 4 pour calculer le modèle plus rapidement.

Les hyper-paramètres optimaux retournés sont localisés dans le dossier "Data/CVresult". Nous avons placé les résultats obtenus pour chaque méthode dans un fichier csv que nous avons pu ensuite soumettre sur la page de *Kaggle*.

4.1 Les K plus proches voisins

La méthode des K plus proches voisins³ que nous avons vu en cours de IFT501 nécessite la mise en place de certains arguments pour bien fonctionner :

1. Le nombre de voisins.
2. La manière dont les poids sont distribués parmi les points d'un cluster.

2. https://scikit-learn.org/0.16/modules/generated/sklearn.grid_search.GridSearchCV.html

3. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

3. La fonction de distance utilisée, si nous utilisons la distance Euclidienne, la distance de Manhattan ou la distance de Minkowski.

4.2 Réseaux de neurones

La méthode des réseaux de neurones⁴ de scikit learn nécessite la mise en place de certains paramètres :

1. La taille des couches cachées.
2. La fonction d'activation.
3. Le choix du solveur pour l'optimisation des poids.
4. La taille des lots (*batch size*.)

4.3 Machine à Vecteur de Support

La méthode des machines à vecteur de support⁵ nécessite la mise en place de certains paramètres pour bien fonctionner :

1. La pénalité pour le terme d'erreur.
2. Le type de noyau qui sera évalué, tel que linéaire, rbf ou encore polynomiale.
3. La mesure du critère d'arrêt.
4. Le degré de la fonction polynomiale.
5. Le coefficient pour les noyaux rbf, polynomiale et sigmoïde.

Certains paramètres tel que le critère d'arrêt ou le maximum d'itération servent à accélérer les temps de calcul.

4.4 Forêt aléatoire

La méthode des forêts aléatoires⁶ nécessite la mise en place de certains paramètres pour une utilisation optimale :

1. Le nombre d'arbres généré dans la forêt.
2. La profondeur maximale.

4. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.fit

5. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

6. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

4.5 Naive Bayes

La méthode naive Bayes⁷ se base sur un paramètre principal pour construire le modèle : la variance maximale pour rendre le calcul plus stable. Nous avons donc lancé Grid Search avec deux variables différentes pour cet hyper-paramètre.

4.6 Regression logistique

La méthode de la regression logistique⁸ est celle qui nécessite l'initialisation du plus grand nombre d'hyper paramètres.

1. Le critère d'arrêt.
2. La force de régularisation.
3. On décide qu'un biais doit être ajouté à la fonction de décision.
4. L'algorithme utilisé dans les problèmes d'optimisations. Etant dans un problème multi classe, nous avons le choix entre sag, saga, newton-cg et lbfgs.

4.7 Adaboost

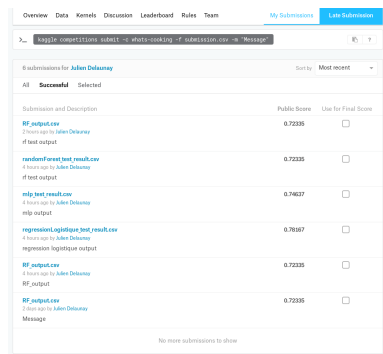
La méthode Adaboost⁹ que nous avons implémenté s'est basé sur la méthode de la regression logistique avec les paramètres optimaux que nous avons trouvés à l'aide de Grid Search. Nous avons essayé de l'implémenter avec 200 estimateurs, cependant le temps de calcul étant trop long, nous avons baissé le nombre d'estimateurs.

7. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

8. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

9. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

5 Les résultats



Submission and Description	Public Score	Use for Final Score
RF_output.csv 7 rows up to Adaboost of test output	0.72235	<input type="checkbox"/>
randomForest_test_result.csv 4 rows up to Adaboost of test output	0.72235	<input type="checkbox"/>
mlp_test_result.csv 4 rows up to Adaboost mlp output	0.74627	<input type="checkbox"/>
regression_logistique_test_result.csv 4 rows up to Adaboost regression logistique output	0.76167	<input type="checkbox"/>
RF_output.csv 4 rows up to Adaboost RF output	0.72235	<input type="checkbox"/>
RF_output.csv 7 rows up to Adaboost Message	0.72235	<input type="checkbox"/>

FIGURE 3 – Score lors de la soumission sur Kaggle en fonction des méthodes

Nous obtenons un score de 19,3% pour Adaboost et 33,8% avec naive Bayes. Nous avons commencé par implémenter naive Bayes et ce mauvais résultat nous à amené à essayer une autre méthode dans l'espoir d'avoir de meilleurs résultats. C'est ainsi que nous avons implémenté une septième méthodes et au vu des résultats obtenus avec celle-ci, nous en concluons que ce sont deux méthodes qui ne sont pas adaptés aux données fournies. Les erreurs obtenues sur ces modèles sont probablement dû à la trop forte proportion de recettes italienne et indienne car nous pouvons voir sur les données de adaboost que la totalité des prévisions sont des recettes italiennes.

6 Conclusion

Au vu des résultats retourné par nos méthodes et des scores correspondants donnés par *Kaggle*, nous pouvons conclure que la méthode la plus efficace pour ce jeu de données parmi celles que nous avons implémenté est la méthode des machines à vecteur de support. Cependant, nous avons remarqué que la méthode de la regression logistique à trouvé son modèle très rapidement et à obtenu un score de 78,2% tandis que la méthode SVM à finit au bout de 5h. C'est une durée moyenne comparé à KNN par exemple qui a prit plus de 30h pour terminer.

A Choix du design

Pour le design du code, nous avons utilisé ce format qui nous est inspiré de l'article écrit par Edward Ma¹⁰. Nous avons donc utilisé jupyter pour commencer à travailler sur les données afin de faciliter et rendre plus rapide le début du projet.

Ainsi, nous avons mis en place une partie Data contenant les données récupérées sur *Kaggle*, et les données que nous avons fournies à partir des 6 méthodes mentionnées plus haut.⁴ De plus, nous avons pu commencer par travailler sur des notebooks, et donc commencer certains travaux d'une façon moins rigoureuse.

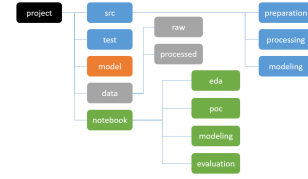


FIGURE 4 – Design d'un projet de Data Science

B Utilisation de Git

Nous avons mis en place un github¹¹ afin de répartir le travail et indiquer le travail restant à faire. Pour cela, nous avons créé des *issues*¹² au fur et à mesure de l'avancée du projet.

10. <https://goo.gl/a8cffw>

11. https://github.com/hamzaouiayman/IFT_712_projet_session.git

12. Tâches à effectuer avant la fin du projet