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

Aiman Hamzaoui 18 131 996
&
Julien Delaunay 18 155 167

12 avril 2019

*Lien vers le git : `https://github.com/hamzaouiayman/IFT\protect_712\protect_projet\protect_session.git`

Table des matières

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.

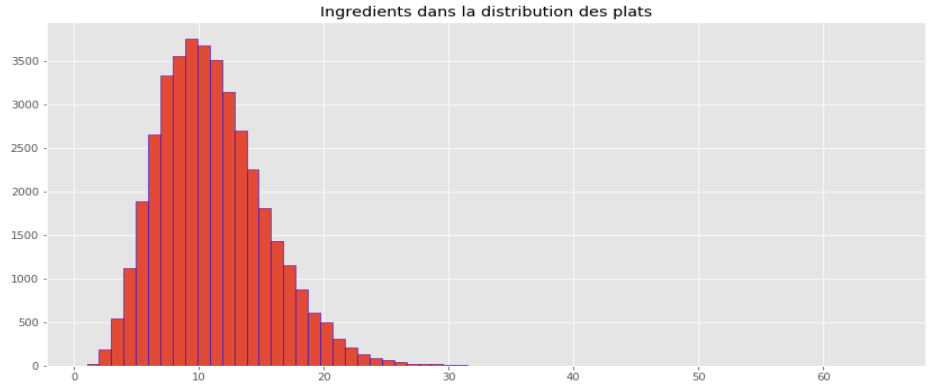


FIGURE 1 – Nombre d'ingrédients par recette

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 (figure 1). On peut observer que en moyenne, la majorité des recettes utilisent entre 5 et 20 ingrédients. Les ingrédients les plus utilisés varient en fonction des cuisines (figure 2), il est donc intéressant d'observer si un modèle peut prévoir en fonction des ingrédients à quelle cuisine ils appartiennent.

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

Nous avons observer que la base de données comportait 20 type de cuisines. C'est donc un problème de classification avec des classes multiples puisqu'il y a plus que 2 catégories à prédire.

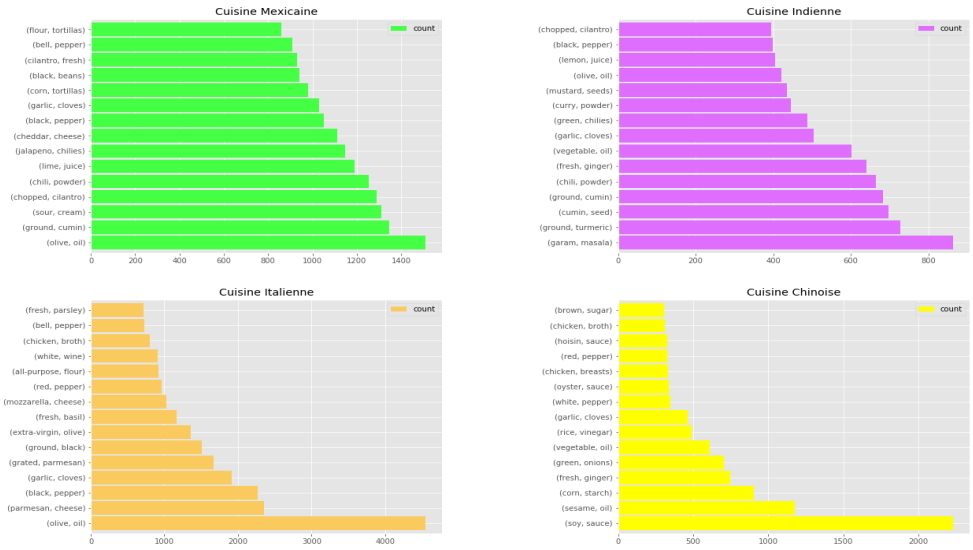


FIGURE 2 – Repartition des ingrédients par cuisine

Afin de commencer l'analyse et le traitement des données nous avons décidé de modifier le format d'entrée et transformer la liste des ingrédients sous la forme d'un tableau, avec une recette par ligne, et un ingrédients par colonne (figure 3).

	cuisine	ingredients
0	greek	[romaine lettuce, black olives, grape tomatoes...
1	southern_us	[plain flour, ground pepper, salt, tomatoes, g...
2	filipino	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	indian	[water, vegetable oil, wheat, salt]
4	indian	[black pepper, shallots, cornflour, cayenne pe...

FIGURE 3 – Transformation des données sour la forme d'un tableau

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.
Ainsi, nous avons enlevé tous les espaces afin que chaque ingrédients soit prit comme un token unique. Par la suite, il a fallu transformer ce tableau en une chaîne de caractère puisque TF-IDF ne supporte que des *String* (figure 4). Pour vectoriser TF-IDF nous avons implémenté la fonction TF-IDF de sklearn [2] qui retourne une matrice creuse qui contient les valeurs TF-IDF de chaque ingrédient par rapport à une recette.

Par la suite, nous avons remplacer les cuisines par des indices grâce à la fonction *"label_train_transform"* qui peuvent ensuite être retourné sous leurs noms originels 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

2. `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

	cuisine	ingredients
0	greek	romainelettuce blackolives grapetomatoes garli...
1	southern_us	plainflour groundpepper salt tomatoes groundbl...
2	filipino	eggs pepper salt mayonaise cookingoil greenchi...
3	indian	water vegetableoil wheat salt
4	indian	blackpepper shallots cornflour cayennepepper o...

FIGURE 4 – Transformation des ingrédients sous la forme d'un string unique

la recherche d'hyper-parametres, nous avons utilisé *GridSearch*[3] avec une validation croisée qui divise l'échantillon par 5. Nous avons donc implémenté manuellement différentes valeurs, et *GridSearch* nous aide à trouver les meilleurs hyper-paramètres. Pour chacunes 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 voisins[4] 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.
3. La fonction de distance utilisée, si nous utilisons la distance Euclidienne ou la distance de Manhattan.

3. `https://scikit-learn.org/0.16/modules/generated/sklearn.grid_search.GridSearchCV.html`
4. `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

4.2 Réseaux de neuronnes

La méthode des réseaux de neuronnes [5] de scikit learn nécessite la mise en place de certains paramètres tel que :

1. La taille des couches cachées.
2. La fonction d'activation.
3. Le choix du solver 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 support [6] aura de meilleurs résultat avec le bon choix de certains paramètres :

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 sigmoide.

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

4.4 Forêt aléatoire

La méthode des forêts aléatoires [7] nécessite la mise en place de certains paramètre pour une utilisation optimale :

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

5. `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.fit`
6. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`
7. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

4.5 Naive Bayes

La méthode naive Bayes [8] 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 logistique [9] est la méthode pour laquelle nous avons inscris le plus 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.

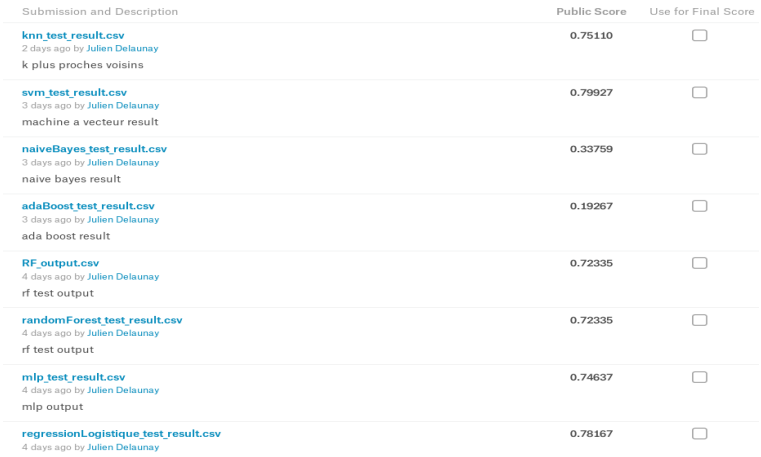
4.7 Adaboost

La méthode Adaboost [10] 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.

8. `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html`
9. `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`
10. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`

5 Les résultats

Pour la lecture des meilleurs hyper-paramètre, si nous prenons le cas de knn, il faut regarder quel est le modèle qui à le rank_test_score le plus élevé. Nous pouvons observer que le meilleur modèle est celui qui possède 20 voisins, la fonction de mesure des poids dans un cluster est distance, et la mesure de distance est Euclidienne. Le score d'entraînement associé à ce modèle est de 0,99%.



Submission and Description	Public Score	Use for Final Score
knn_test_result.csv 2 days ago by Julien Delaunay k plus proches voisins	0.75110	☐
svm_test_result.csv 3 days ago by Julien Delaunay machine a vecteur result	0.79927	☐
naiveBayes_test_result.csv 3 days ago by Julien Delaunay naive bayes result	0.33759	☐
adaBoost_test_result.csv 3 days ago by Julien Delaunay ada boost result	0.19267	☐
RF_output.csv 4 days ago by Julien Delaunay rf test output	0.72335	☐
randomForest_test_result.csv 4 days ago by Julien Delaunay rf test output	0.72335	☐
mlp_test_result.csv 4 days ago by Julien Delaunay mlp output	0.74637	☐
regressionLogistique_test_result.csv 4 days ago by Julien Delaunay	0.78167	☐

FIGURE 5 – Score lors de la soumission sur Kaggle en fonction des méthodes

Nous avons donc soumis ces 7 techniques sur *Kaggle* et nous avons obtenus pour 5 d'entre elles des scores supérieurs à 70%. Les différences de score pour ces 5 méthodes ne sont pas très élévés. Cependant pour certaines, le temps de calcul est très grand et pour d'autres, beaucoup plus court.

La méthode de la forêt aléatoire à obtenu un score de 72,3% en utilisant grid search. Pour la technique du perceptron multi couches, nous avons obtenu un score de 74,6%, celle des k plus proches voisins est arrivée à 75%. Les deux meilleurs techniques sont la méthode de la régression logistique qui nous montre un score de 78,2% et celle des machines à vecteur de support un score de 80%.

Le meilleur score observé sur cette plateforme et cette base de données est de 83,2%, ce qui nous place à la 200e place.

Les méthodes pour lesquels nous avons eus de mauvais résultat sont Adaboost et naive Bayes. 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 designe du code, nous avons utilisé ce format qui nous est inspiré de l'article écrit par Edward Ma [11]. Nous avons donc utiliser 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 récuperé sur *Kaggle*, et les données que nous avons fournis à partir des 6 méthodes mentionnées plus haut.4 De plus, nous avons pu commencer par travailler sur des notebook, et donc commencer certains travaux d'une façon moins rigoureuse.

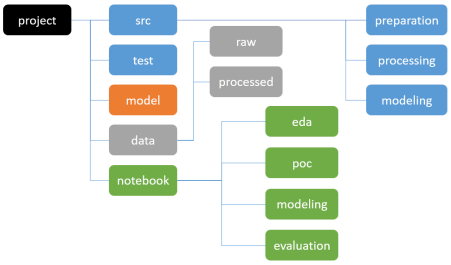


FIGURE 6 – Design d'un projet de Data Science

B Utilisation de Git

Nous avons mis en place un github [12] afin de répartir le travail et indiquer le travail restant à faire. Pour cela, nous avons crée des *issues* [13] au fur et à mesure de l'avancée du projet.

Pour la répartition du travail, nous avons par exemple séparé d'un côté la préparation du git et du rapport et d'un autre côté la génération des données. L'avantage de git est que nous avons pu travailler chacun de notre côté sur une partie du programme et nous avons rassembler tous nos travaux à la fin.

11. `https://goo.gl/a8cffw`
12. `https://github.com/hamzaouiayman/IFT_712_projet_session.git`
13. Tâches à effectuer avant la fin du projet