UQÀM | Faculté des sciences
Université du Québec à Montréal

INF7370 –**Apprentissage automatique**

Rapport du Travail Pratique #1

Session d'Hiver 2023

Table des matières

1. Introduction

Dans ce présent travail, nous allons mettre en pratique cinq (5) modèles de classification en faisant une analyse tout en comparant les performances observées. Ce sont entre autres : Arbre de décision, Forêt d'arbres décisionnels, Classification bayésienne naïve, bagging et adaboost.

L'objectif principal est d'utiliser python et sckit-learn pour entrainer ces modèles et classifier des restaurants (fermeture définitive / restaurant encore ouvert) à travers des données collectées à partir de l'application Yelp. Il faut savoir que Yelp est une application mobile qui permet à des commerciales de mettre à la disposition du public (utilisateurs) des informations sur leur entreprise et sur leurs services. Ces informations peuvent être par exemple : l'emplacement (zone, ville), l'horaire (des jours et heures d'ouvertures et de fermeture). Dans le cadre de ce travail, nous nous sommes intéressés à un seul type d'entreprise : les restaurants. Pour ce type, en plus de trouver l'emplacement et l'horaire, yelp permet aux utilisateurs de rechercher ces derniers par catégorie (Café, Bar, …) et par type de cuisine (chinoise, Italienne, Française, ...). Les utilisateurs peuvent y trouver également la gamme de prix et les services offerts (bon pour les groupes, avec une terrasse, possibilité de livraison, bon pour les enfants, …). Yelp permet aux utilisateurs, en retour, de réagir en donnant des avis avec un nombre d'étoiles, un nombre de réaction (useful, funny, cool, …) sur chaque restaurant et également rédiger un conseil avec un nombre de compliments (meilleur moment de visite, plats du jour, …).

Pour atteindre cet objectif de classification, nous passerons par quelques étapes :

1. Calcul des attributs (features engineering) : à partir des huit (8) fichiers de données collecter (utilisateurs, horaires, avis, services, …), on calcule pour chaque restaurant un certain nombre d'attributs qui permet de distinguer les restaurants.
2. Pré-traitement : durant cette étape, nous allons traiter les valeurs manquantes et faire une transformation sur les données afin de les rendre cohérentes et prêtes pour l'analyse.
3. Entraînement des modèles et analyse comparative : nous utiliserons pour un départ l'ensemble total des attributs puis une sélection des dix (10) meilleurs attributs seulement pour :
    a. Appliquer les cinq (5) algorithmes pour classifier les restaurants et faire une comparaison de performance en utilisant les métriques suivantes : le taux des vrais positifs, le taux des faux positifs, la F-mesure et la surface sous la courbe ROC (AUC)
    b. Illustrer nos résultats dans des tables
    c. Afficher la matrice de confusion
    d. Analyser et commenter nos résultats (points faibles et forts constatés)

Le travail est reparti sur trois fichiers principaux :

- ✓ features.py pour le calcul des attributs, les données obtenues (34268 Lignes et 39 attributs) sont stockées dans un fichier features.csv dans le dossier de données initial.
- ✓ processing.py pour les traitements et transformations sur les données. Une fois terminée, ces données sont stockées dans un fichier features_finaux.csv dans le dossier de données initial également.
- ✓ learning.py pour faire l'apprentissage automatique (entrainement des modèles).

2. Calcul des attributs

Dans cette partie, nous allons utiliser les fichiers de données utilisateurs.csv, avis.csv, conseils.csv, checkin.csv, restaurant.csv, horaire.csv pour calcul des attributs. Ces attributs seront calculés et stockés au fur et à mesure dans le fichier features.csv.

Au total nous aurons 39 attributs et 34268 lignes de données.

Nb : features.csv a été initialisé à partir du fichier de données restaurants.csv avec les colonnes suivantes : restaurant_id, moyenne_etoiles, ville, zone, ferme. Cela dit, nous calculerons 34 attributs au total.

Pour « nb_restaurants_zone » : Le nombre de restaurants dans la zone associée au restaurant en question

- ✓ Compter le nombre de restaurant par zone

Pour « zone_categories_intersection » : Le nombre de restaurants dans la même zone qui partagent au moins une catégorie avec le restaurant en question.

- ✓ D'abord, on associe les zones et les catégories de chaque restaurant en faisant une jointure
- ✓ Ensuite on calcule le nombre de restaurant(s) par zone qui partage au moins une catégorie

Pour « ville_categories_intersection » : Le nombre de restaurants dans la même ville qui partagent au moins une catégorie avec le restaurant en question

- ✓ D'abord, on associe les villes et les catégories de chaque restaurant en faisant une jointure
- ✓ Ensuite on calcule le nombre de restaurant(s) par ville qui partage au moins une catégorie

Pour « nb_restaurant_meme_annee » : Le nombre de restaurants qui ont ouverts leurs portes dans la même année que le restaurant en question.

- ✓ On trie puis on filtre les données sur les avis pour récupérer le premier avis publié pour chaque restaurant
- ✓ A travers une fonction, on vérifie et on compte ensuite les avis qui sont publiés pendant la même année que le restaurant en question

Pour « ecart_type_etoiles » : L'écart type de la moyenne des étoiles par année.

- ✓ On calcule d'abord la moyenne des étoiles par année pour chaque restaurant
- ✓ Puis on calcule, à partir du résultat précédent, l'écart type des étoiles pour chaque restaurant

Pour « tendance_etoiles » : La différence entre la moyenne des étoiles de la dernière année et la moyenne des étoiles de la première année d'un restaurant.

- ✓ Sur le jeu de données des avis, on commence par concaténer la colonne "annee" et le «restaurant_id" pour chaque restaurant puis le sauvegarder dans une colonne "ref"
- ✓ Puis on filtre les avis pour obtenir seulement les étoiles de la première et dernière année de chaque restaurant et concaténer comme précédemment la colonne "annee" et le «restaurant_id" pour chaque restaurant puis le sauvegarder dans une colonne "annee_f" et "annee_l" respectivement
- ✓ On calcule ensuite la moyenne des étoiles de la première et dernière année de chaque restaurant récupéré à l'étape 1
- ✓ On calcule enfin, après la jointure des moyennes des étoiles de la première et dernière année par restaurant, la différence des deux moyennes

Pour « nb_avis » : Le nombre total d'avis pour ce restaurant

- ✓ On compte le nombre d'avis publiés par restaurant

Pour « nb_avis_favorables » : Le nombre total d'avis favorables et positifs pour ce restaurant.

- ✓ On filtre puis on compte pour chaque restaurant le nombre d'avis qui ont un nombre d'étoiles supérieur ou égal à 3 (etoiles ≥ 3)

Pour « nb_avis_defavorables » : Le nombre total d'avis défavorables et positifs pour ce restaurant

- ✓ On filtre puis on compte pour chaque restaurant le nombre d'avis qui ont un nombre d'étoiles inférieur à 3 (etoiles < 3)

Pour « ratio_avis_favorables » : Le nombre d'avis favorables et positifs (nb_avis_favorables) sur le nombre total d'avis (nb_avis) pour chaque restaurant.

Pour « ratio_avis_defavorables » : Le nombre d'avis défavorables (nb_avis_defavorable) sur le nombre total d'avis (nb_avis) pour chaque restaurant.

Pour « nb_avis_favorables_mention » : Le nombre total d'avis qui ont reçu au moins une mention "useful : (>0)" ou "funny : (>0)" ou "cool : (>0)" ET dont le nombre d'étoiles de l'avis est supérieur ou égal à 3.

- ✓ On commence par filtrer les avis qui répondent à la condition puis on les compte par restaurant

Pour « nb_avis_defavorables_mention » : Le nombre total d'avis qui ont reçu au moins une mention "useful : (>0)" ou "funny : (>0)" ou "cool : (>0)" ET dont le nombre d'étoiles de l'avis est inférieur à 3.

- ✓ On commence par filtrer les avis qui répondent à la condition puis on les compte par restaurant

Pour « nb_avis_favorables_elites » : Le nombre total d'avis favorables pour un restaurant qui sont rédigés par des utilisateurs élites.

- ✓ A travers une jointure on associe les 'conseils' et les 'utilisateurs'

- ✓ On filtre pour récupérer seulement les conseils des utilisateurs élites (élite =1) qui ont rédigé au moins 100 avis (nb_avis>0) et dont au moins 100 (nb_avis_mention>100) avis avec mention
- ✓ On compte le nombre de conseils par restaurant

Pour « nb_checkin » : Le nombre total de visite

- ✓ On compte le nombre de visites (ckeck-in) par restaurant

Pour « moyenne_checkin » : La moyenne de visites par année

- ✓ On calcule d'abord le nombre visite par année pour chaque restaurant
- ✓ On calcule ensuite la moyenne en fonction du nombre de visites par année pour chaque restaurant

Pour « ecart_type_checkin » : L'écart type de visites par année

- ✓ On calcule d'abord le nombre visite par année pour chaque restaurant
- ✓ On calcule ensuite l'écart type en fonction du nombre de visites par année pour chaque restaurant

Pour « chaine » : Vérifier si un restaurant est une chaine. Un restaurant est une chaine s'il existe un autre avec le même nom

- ✓ Dans la liste des restaurant on compte le nombre de fois que le nom d'un restaurant apparait puis on ne garde que ceux qui apparaissent plus d'une fois
- ✓ On crée une fonction qui retourne 0 si le nom du restaurant courant (paramètre) n'apparait pas dans la liste calculée à l'étape précédente sinon on retourne 1

Pour « nb_heures_ouvertures_semaine » : Le nombre total d'heures d'ouverture du restaurant par semaine.

- ✓ On calcule d'abord la différence entre l'heure de fermeture et l'heure d'ouverture de chaque jour de la semaine à travers une fonction que nous avons définie. Il faut noter qu'on avait des horaires avec des formats différents, alors on a ajouté 12h pour une conversion vers le format 24h. Le résultat de chaque différence pour chaque jour est stocké dans une colonne différente. (ex : lundi è lundi_diff)
- ✓ On fait enfin la somme des colonnes calculées précédemment pour chaque restaurant

Pour « ouvert_samedi » : Si le restaurant est ouvert les samedis (valeur booléenne : 0 ou 1).

- ✓ On vérifie la valeur de la colonne 'samedi_diff' calculée lors du calcul du nombre total d'heures d'ouverture par semaine, si elle est supérieure ou égale à 1 alors le restaurant est ouvert les samedis (1) sinon il est est fermé (0).

Pour « ouvert_dimanche » : Si le restaurant est ouvert les dimanches (valeur booléenne : 0 ou 1).

- ✓ On vérifie la valeur de la colonne 'dimanche_diff' calculée lors du calcul du nombre total d'heures d'ouverture par semaine, si elle est supérieure ou égale à 1 alors le restaurant est ouvert les dimanches (1) sinon il est est fermé (0).

Pour « ouvert_lundi » : Si le restaurant est ouvert les lundis (valeur booléenne : 0 ou 1).

- ✓ On vérifie la valeur de la colonne 'lundi_diff' calculée lors du calcul du nombre total d'heures d'ouverture par semaine, si elle est supérieure ou égale à 1 alors le restaurant est ouvert les lundis (1) sinon il est est fermé (0).

Pour « ouvert_vendredi » : Si le restaurant est ouvert les vendredis (valeur booléenne : 0 ou 1).

- ✓ On vérifie la valeur de la colonne 'vendredi_diff' calculée lors du calcul du nombre total d'heures d'ouverture par semaine, si elle est supérieure ou égale à 1 alors le restaurant est ouvert les vendredis (1) sinon il est fermé (0).

Pour « emporter » : Si le restaurant offre le service à emporter (valeur booléenne : 0 ou 1)

- ✓ On récupère directement l'information sur la possibilité d'emporter qui est disponible dans le fichier de données services.csv

Pour « livraison » : Si le restaurant offre le service de livraison (valeur booléenne : 0 ou 1).

- ✓ On récupère directement l'information sur l'offre de la livraison qui est disponible dans le fichier de données services.csv

Pour « bon_pour_groupes » : Si le restaurant est approprié pour les groupes (valeur booléenne : 0 ou 1).

- ✓ On récupère directement l'information sur la qualité de service de groupe qui est disponible dans le fichier de données services.csv

Pour « bon_pour_enfants » : Si le restaurant est approprié pour les enfants (valeur booléenne : 0 ou 1)

- ✓ On récupère directement l'information sur la qualité de service pour des enfants qui est disponible dans le fichier de données services.csv

Pour « reservation » : Si on a besoin de faire une réservation au restaurant (valeur booléenne :0 ou 1).

- ✓ On récupère directement l'information sur la possibilité de faire une réservation qui est disponible dans le fichier de données services.csv

Pour « prix » : Le niveau de prix du restaurant. Il existe trois niveaux, 1 (abordable), 2 (moyen) et 3 (coûteux).

- ✓ On récupère directement l'information sur les prix qui est disponible dans le fichier de données services.csv

Pour « terrasse » : Si le restaurant a une terrasse (valeur booléenne : 0 ou 1).

- ✓ On récupère directement l'information sur la colonne terrasse qui est disponible dans le fichier de données services.csv

3. Traitement du dataset

Pour le traitement des données, nous avons utilisé la méthode de remplacement des valeurs manquantes par certaines valeurs en nous basant sur la nature de chaque colonne. Voici notre démarche de traitement des valeurs manquantes :

- ✓ Vérification du nombre de valeurs manquantes pour chaque colonne

- ✓ Remplacer par **0** les valeurs manquantes des colonnes calculées par nombre d'occurrences (toutes les variables commençant par « **nb_** »), parce que ces variables résultent d'un calcul d'effectif.
- ✓ Afficher une statistique des variables à virgules (résultant d'un calcul décimal) afin de voir comment sont les plages de valeurs pour voir la méthode de remplacement appropriée
- ✓ Remplacer par la **moyenne (mean)** les valeurs manquantes des colonnes qui sont distribuées presque normalement à savoir « **ecart_type_etoiles, ratio_avis_favorables, ratio_avis_defavorables** »
- ✓ Remplacer par la **médiane (median)** les valeurs manquantes des colonnes qui ne sont pas distribuées normalement à savoir « **moyenne_checkin**, **ecart_type_checkin** » pour éviter la présence des valeurs aberrantes dans les données.
- ✓ Remplacement de la dernière colonne « Zone » comportant des valeurs manquantes par le mode vu que c'est une colonne catégorielle.

4. Entrainement des modèles et analyse comparative

1. Utilisation de tout l'ensemble des attributs

Description des différentes étapes effectuées dans cette section :

a. Importation des bibliothèques nécessaires :
   - StandardScaler : pour la mise en échelle des données
   - Mutual_info_classif : pour la selection des attributs importants (gain d'information)
   - DecisionTreeClassifier, RandomForestClassifier, GaussianNB, BaggingClassifier, AdaBoostClassier : la construction des modèles
   - Train_test_split : pour le partitionnement des données
   - Metrics : pour l'évaluation des performances des modèles et la visualisation des courbes ROC

b. Normalisation des données après leur initialisation

c. Partage des données en données d'entraînement et de test (le pourcentage des données de tests utilisé est **20%**)

d. Création et entraînement des 5 modèles de classification demandés

e. Evaluer les performances des modèles en faisant des prédictions sur les données de test.

f. Affichage des résultats des différentes métriques utilisées pour évaluer les modèles ainsi que la courbe ROC de chaque modèle.

Affichage des résultats obtenus lors des évaluations des différents modèles

Métriques	Decision Tree	Random Forest	Naive Bayes	Bagging	AdaBoost
TPR	0.722 = 72%	0.771 = 77%	0.947 = 95%	0.715 = 72%	0.747 = 75%
FPR	0.245 = 25%	0.198 = 20%	0.882 = 88%	0.149 = 15%	0.241 = 24%
F1_SCORE	0.736 = 74%	0.785 = 79%	0.674 = 67%	0.768 = 77%	0.754 = 75%
AUC	0.71 = 71%	0.87 = 87%	0.68 = 68%	0.86 = 86%	0.84 = 84%

Légende
TPR = True Positive Rate (Taux des Vrais Positifs)

FPR = False Positive Rate (Taux des Faux Positifs) F1_SCORE = F-mesure de la classe Restaurants fermés définitivement AUC = Area Under Curve (L'aire sous la courbe)

Analyse des résultats

Si nous nous limitons aux différents résultats obtenus par les modèles décrits dans le tableau ci-dessus et plus particulièrement aux métriques (F1_Score et AUC), nous voyons que **les modèles d'ensembles** à savoir (Random Forest, Bagging et AdaBoost) donnent des bons résultats dans des grandes dimensions (37 attributs) comparativement au modèle Naive Bayes qui donne des résultats moins bons. Dans ce travail, l'arbre de décision donne également des bons résultats.

Note : les 4 modèles les plus performants de ce travail font un bon compromis entre sensibilité (TPR) et la précision.

- ✓ Decision Tree : permet une lisibilité du résultat claire et facilement paramétrable.
  Les résultats de la classification montrent qu'avec les 37 attributs il peut avec une certitude de 74% (f-score) prédire la bonne classe d'appartenance du restaurant. La courbe ROC ne contredit pas notre observation, celle-ci s'étire vers la classe positive et augmente ainsi l'aire sous la courbe.
  Mais malheureusement, nous avons un taux de faux positifs élevés. Le modèle fait peu d'hypothèses sur les données d'entraînement. Sans restriction, il explore tous les cas possibles. Lorsque la profondeur de l'arbre n'était pas fixée à 10, nous avions eu un résultat moins bon.
- ✓ Random Forest : très performant dans des grandes dimensions, parce qu'il combine la sélection aléatoire d'objets et d'attributs. Détermine l'importance des attributs dans le processus de classification et choisit un sous ensemble à chaque fois pour former un ensemble d'apprentissage.
  Nous observons dans notre cas une mesure f-score de 79% et un taux de faux positifs de 20 %, ceci justifie une amélioration du résultat de l'arbre de décision. La courbe ROC s'étire également vers la classe positive et est un bon signe de performance du modèle. Mais il faut noter que sans la limite de la profondeur de l'arbre que nous avons fixé ici à 16, le TFR était plus élevé.
- ✓ Naive Bayes :
  Nous observons qu'il donne pas de bons résultats avec nos données, c'est peut-être parce que la plupart de nos attributs sont dépendamment liés (calcul de certains attributs à partir d'autres). Toutes les mesures de performances utilisées sont en concordance sur la faible performance du modèle : f-score et courbe roc.
  Il faut tout de même noter que Naive Bayes utilisés sur des données indépendantes est un modèle simple et efficace.
- ✓ Bagging : réduit l'erreur de généralisation du classifieur de base dans le but d'obtenir une erreur faible. Il combine différents classifieurs obtenus pendant l'entraînement pour classer correctement les différents objets. D'ailleurs nos résultats affichent un taux de

faux positifs (15%) plus faibles que tous nos autres modèles et c'est un bon signe de performance. La courbe ROC et la f-score nous permettent d'observer également de bons résultats.
Mais avec cet algorithme il faut savoir que le fait que les échantillons sont indépendants entre eux, en cas de présence d'objets bruités, il peut se répercuter tout au long du processus et entraîner un overfitting.

- ✓ AdaBoost : Il permet de pondérer les échantillons à chaque étape pour voir les objets mal classés afin de les corriger pour la seconde étape.
  Bien qu'il ne soit pas le meilleur modèle comparativement aux randomForest et le bagging. Les mesures f-score et la courbe ROC montrent une bonne performance de classification du modèle.
  Mais lorsque les échantillons sont indépendants, adaBoost n'est pas très performant parce que la performance du modèle dépend de la séquence d'échantillonnage

2. Sélection des attributs

Description des différentes étapes effectuées dans cette section

- ✓ Identification et affichage des 10 meilleurs attributs du jeu de données en utilisant le gain d'information.
- ✓ Normalisation des 10 meilleurs attributs de données
- ✓ Partitionnement des données en données d'entraînement et de test (le pourcentage des données de test utilisé est **20%**)
- ✓ Création et entraînement des 5 modèles de classification demandés à partir des 10 meilleurs attributs
- ✓ Evaluer les performances des modèles en faisant des prédictions sur les données de test des 10 meilleurs attributs.
- ✓ Affichage des résultats des différentes métriques utilisées pour évaluer les modèles ainsi que la courbe ROC de chaque modèle.

Affichage des tops 10 attributs

```
Les 10 meilleurs features sont:
                    features      rank
22          moyenne_checkin  0.060088
23       ecart_type_checkin  0.053780
6   nb_restaurant_meme_annee  0.030192
21               nb_checkin  0.028997
12    ratio_avis_favorables  0.028018
13  ratio_avis_defavorables  0.027434
7        ecart_type_etoiles  0.027138
9                    nb_avis  0.026876
10       nb_avis_favorables  0.026488
27          ouvert_dimanche  0.023491
```

Affichage des résultats

Métriques	Decision Tree	Random Forest	Naive Bayes	Bagging	AdaBoost
TPR	0.738 = 74%	0.775 = 78%	0.947 = 95%	0.740 = 74%	0.739 = 74%
FPR	0.200 = 20%	0.174 = 17%	0.883 = 88%	0.161 = 16%	0.251 = 25%
F1_SCORE	0.762 = 76%	0.796 = 80%	0.671 = 67%	0.779 = 78%	0.743 = 74%
AUC	0.74 = 74%	0.87 = 87%	0.64 = 64%	0.86 = 86%	0.82 = 82%

Légende
TPR = True Positive Rate (Taux des Vrais Positifs)
FPR = False Positive Rate (Taux des Faux Positifs)
F1_SCORE = F-mesure de la classe Restaurants fermés définitivement
AUC = Area Under Curve (L'aire sous la courbe)

Analyse des résultats (comparaison avec l'expérimentation qui implique l'utilisation de tout l'ensemble d'attributs)

En nous basant sur les métriques F1_Score et AUC des différents classifieurs, nous pouvons voir qu'il y a eu une amélioration de performances pour l'arbre de décision (**Decision Tree**) avec cette réduction des attributs, cependant les modèles d'ensemble n'ont pas eu d'amélioration de performances avec la réduction de la dimensionnalité. Sur la base de ces résultats, nous pouvons dire que l'arbre de décision a tendance à bien performer dans les dimensions faibles comparativement aux modèles d'apprentissage par ensemble. Pour ce travail, avec ou sans sélection des meilleurs attributs, le classifieur **Random Forest** est le meilleur modèle, suivi du **Bagging** avec les métriques (F1_Score et AUC) les plus élevées.

Note : Dans les 2 cas (réduction ou non de la dimensionnalité), le classificateur **Naive Bayes** ne donnent pas des résultats convaincants.

5. Conclusion

La réalisation de ce travail nous a permis d'approfondir nos connaissances en ingénierie des caractéristiques et la mise en place de quelques modèles de classification parmi les plus utilisés. D'une manière générale, le problème rencontré est l'amélioration des performances de nos différents algorithmes. Les démarches possibles que nous pouvons considérer pour les améliorer sont :

- Régulariser les paramètres de chaque modèle jusqu'à obtenir une amélioration de performance
- Varier le pourcentage des données d'entrainement et de test
- Avant la mise en place des modèles, s'assurer qu'il n'y as pas de dépendance entre les prédicteurs (colinéarité) et supprimer les caractéristiques non pertinentes, puis entraîner et tester les modèles sur ces nouvelles données.