Rapport Projet DataScience

👨🏼 par Fellahi Tarik et Lepretre Matys

Introduction

Dans le cadre de notre projet de Data Science, nous avons été sollicités par une compagnie d'assurance automobile fictive, On the Road, afin de construire un modèle prédictif capable d'estimer si un client effectuera une demande d'indemnisation au cours de sa période d'assurance. Le contexte métier repose sur l'optimisation de la tarification et de la gestion du risque client, éléments clés dans le secteur assurantiel.

Pour cela, nous avons travaillé sur un jeu de données contenant diverses informations sociodémographiques, comportementales et financières relatives à un panel de clients. L'objectif de ce projet est de concevoir un modèle de classification supervisée permettant de prédire efficacement la variable cible outcome, indiquant la réalisation ou non d'une demande d'indemnisation.

Nous avons structuré notre démarche en plusieurs étapes : exploration et préparation des données, entraînement et évaluation de différents modèles de classification, comparaison de leurs performances, et enfin sauvegarde du meilleur modèle en vue d'une intégration potentielle en production. Ce rapport présente en détail chacune de ces étapes, les choix méthodologiques effectués, ainsi que les résultats obtenus.

L'environnement technique utilisé pour ce projet comprend Python comme langage principal, avec les bibliothèques suivantes :

- Pandas pour la manipulation de données,
- NumPy pour les opérations numériques,
- Matplotlib pour la visualisation,
- Scikit-learn pour la modélisation, la validation croisée, le prétraitement et les métriques d'évaluation,
- Pickle pour la sauvegarde du modèle entraîné.

Etapes du projet

1. Compréhension et exploration des données

La première étape du projet consiste à importer et analyser le jeu de données car_insurance.csv, fourni sous forme d'un fichier tabulaire. L'import a été réalisé avec la bibliothèque Pandas via la fonction read_csv(), ce qui a permis de charger les données dans une structure de type DataFrame.

Structure et typage des données

Le dataset contient 18 colonnes et 10 000 lignes :

#	Colonne	Non-Null Count	Type	Description
0	id	10000	int64	Identifiant unique du client
1	age	10000	int64	Tranche d'âge du client : 16-25, 26-39, etc.
2	gender	10000	int64	Sexe du client : 0 = femme, 1 = homme
3	driving_experience	10000	object	Tranche d'années de conduite : 0-9, 10-19, etc.
4	education	10000	object	Niveau d'éducation : No education, High school, University
5	income	10000	object	Tranche de revenu : Poverty, Working class, Middle class, Upper class
6	credit_score	9018	float64	Score de crédit du client entre 0 et 1
7	vehicle_ownership	10000	float64	Possession du véhicule : 0 = non, 1 = oui
8	vehicle_year	10000	object	Année du véhicule : 0 = avant 2015, 1 = 2015 ou après
9	married	10000	float64	Statut marital : 0 = non marié, 1 = marié

10	children	10000	float64	Nombre d'enfants du client
11	postal_code	10000	int64	Code postal du client
12	annual_mileage	9043	float64	Distance annuelle parcourue en miles
13	vehicle_type	10000	object	Type de véhicule : Sedan ou Sports car
14	speeding_violations	10000	int64	Nombre d'infractions pour excès de vitesse
15	duis	10000	int64	Nombre de conduites sous influence d'alcool
16	past_accidents	10000	int64	Nombre d'accidents antérieurs
17	outcome	10000	float64	**Target** : 0 = pas de réclamation, 1 = réclamation faite

Données manquantes

Une analyse via df.isna().sum() a permis d'identifier la présence de valeurs manquantes dans certaines colonnes, dont

Données aberrantes

L'observation d'histogrammes (df.hist()) et de valeurs extrêmes a permis d'identifier certaines données incohérentes, comme des valeurs anormalement élevées dans speeding_violations ou encore children.

Distribution et qualité des données

Les histogrammes et la méthode describe() ont permis de :

- visualiser la distribution des variables numériques,
- détecter les asymétries éventuelles,
- mieux comprendre les ordres de grandeur.

2.Préparation des données

Suppression de colonne

La suppression de la colonne *id* a été effectué en raison de son inutilité dans l'apprentissage de notre classification étant aléatoire

Le champ *postal_code* n'apporte pas d'information utile pour la prédiction. Il a donc été supprimé aussi.

Remplacement des valeurs aberrantes

Le remplacement des valeurs aberrantes (la colonne *children* avec plus de 10 enfants et la colonne *speed_violations* avec plus de 50 violations) ont été remplacé par la médianne permettant de garder des données cohérentes par rapport à la moyenne qui peut contenir un nombre a virgule ce qui n'est pas cohérent d'avoir par exemple 0.8 enfants ou avoir été flasher 2.4 fois.

Complétions des données manquantes

Les variables présentant un taux de valeurs nulles supérieur à 30 % ont été supprimées. Pour les autres :

- les variables numériques manquante des colonnes

credit_score (float) ont été remplacées par la moyenne et *annual_mileage* (int) ont été imputées par la médiane permettant de suivre une logique comme celle cité dans le point du dessus,

Encodage des variables qualitatives

Les algorithmes de machine learning nécessitent des données numériques. Ainsi, les variables catégorielles (object) ont été converties :

- Pour les variables binaires ou ordinales facilement encodables (ex. gender, vehicle_year, married, vehicle_ownership), un simple remplacement via replace() a été appliqué.
- Pour les variables multicatégorielles (ex. education, income, driving_experience, vehicle_type), la classe LabelEncoder de Scikit-learn a été utilisée pour convertir les modalités en entiers uniques.

Normalisation des variables

Les variables numériques présentaient des échelles très différentes (annual_mileage en milliers, credit_score entre 0 et 1, etc.), ce qui peut nuire à l'efficacité de certains modèles. La bibliothèque Scikit-learn propose la classe StandardScaler, utilisée ici pour standardiser les données : on ramène toutes les variables à une moyenne nulle et un écart-type unitaire.

3. Analyse des corrélations

Une étape essentielle avant l'entraînement des modèles consiste à analyser les **corrélations statistiques** entre les variables. Cette analyse permet d'identifier :

- les **variables les plus liées à la variable cible outcome**,
- les **meilleurs features pour la classification**.

Corrélation avec la variable cible

outcome

Nous avons utilisé la méthode corr() de Pandas pour calculer la **corrélation de Pearson** entre toutes les variables numériques et la variable cible outcome. Ce coefficient varie entre –1 (corrélation parfaitement négative) et +1 (corrélation parfaitement positive).

Voici les résultats obtenus :

Variable	Corrélation avec outcome
outcome	1.000000
vehicle_year	0.294178
annual_mileage	0.177575
gender	0.107208
vehicle_type	0.005620
income	-0.047560
education	-0.092643
duis	-0.189352
children	-0.232237
married	-0.261807
speeding_violations	-0.291862
credit_score	-0.309010
past_accidents	-0.311495
vehicle_ownership	-0.378921
age	-0.448463
driving_experience	-0.497431

Quelles variables sont les plus corrélées à la variable de sortie ?

Les variables les plus corrélées à outcome sont :

- **Négativement** :
  - driving_experience (–0.497)
  - age (–0.448)
  - vehicle_ownership (–0.379)
  - past_accidents, credit_score, speeding_violations

Ces résultats indiquent qu'un profil expérimenté, plus âgé et propriétaire a moins de risque d'indemnisation.

- **Positivement** :
  - vehicle_year (+0.294)
  - annual_mileage (+0.178)

Les véhicules récents et les conducteurs qui roulent beaucoup sont **plus à risque**.

Quelles variables d'entrée sont les plus corrélées entre elles ?

L'analyse de la matrice de dispersion (scatter_matrix) permet d'observer visuellement les relations entre plusieurs variables numériques. Voici les principales observations tirées du graphique fourni :

- *age* et *driving_experience* : on observe une forte linéarité croissante. Cela indique une corrélation positive forte et attendue : plus un client est âgé, plus il a d'années d'expérience de conduite.
- *age* et *vehicle_year* : relation négative légère, les conducteurs plus jeunes ont tendance à posséder des véhicules plus récents. Cette dépendance est visible par une légère pente inversée.
- *driving_experience* et *vehicle_year* : même observation que précédemment, confirmant une corrélation inversée modérée entre l'ancienneté du véhicule et l'expérience du conducteur.
- *annual_mileage* semble faiblement corrélé aux autres variables. Les points sont largement dispersés, sans tendance claire, indiquant une indépendance partielle avec age ou driving_experience.

Quelles sont les variables les plus pertinentes pour la classification ?

Les variables suivantes apparaissent comme les plus informatives pour la prédiction de outcome :

- **Très pertinentes** :

  driving_experience, age, vehicle_ownership, past_accidents, credit_score

- **Pertinentes à surveiller** :

  speeding_violations, annual_mileage, vehicle_year, duis

- **Moins utiles (corrélation très faible)** :

  income, education, vehicle_type

4. Extraction des jeux d'apprentissage et de test

Après la phase de préparation et d'analyse des données, il est essentiel de procéder à la division du jeu de données en deux sous-ensembles distincts. Cette étape permet de construire un modèle sur une partie des données (jeu d'apprentissage) et de l'évaluer de manière rigoureuse sur une autre partie indépendante (jeu de test). Cette séparation garantit une estimation fiable de la capacité de généralisation du modèle sur de nouvelles données.

La séparation a été réalisée à l'aide de la fonction train_test_split() de la bibliothèque Scikit-learn. Nous avons opté pour un découpage standard avec une proportion de 80 % des données pour l'apprentissage et 20 % pour le test.

Le jeu de données initial comporte 10 000 lignes. La répartition appliquée donne :

- 8 000 individus dans le jeu d'apprentissage (X_train, y_train)
- 2 000 individus dans le jeu de test (X_test, y_test)

5. Entraînement du modèle de régression logistique

Dans cette partie, nous avons entraîné un modèle de classification binaire fondé sur la régression logistique. L'algorithme de régression logistique a été implémenté à l'aide de la classe LogisticRegression de la bibliothèque Scikit-learn. L'entraînement du modèle a été réalisé sur le jeu d'apprentissage (X_train, y_train) à l'aide de la méthode fit().

La régression logistique repose sur une adaptation de la régression linéaire pour les problèmes de classification. Plutôt que de prédire directement une valeur continue, elle modélise la probabilité d'appartenance à une classe (y = 1) à l'aide d'une fonction logistique (sigmoïde).

Hypothèse

L'algorithme suppose que le logarithme du rapport des probabilités conditionnelles (fonction logit) est une fonction linéaire des variables d'entrée :

$$\log\left(\frac{P(y=1 \mid x)}{P(y=0 \mid x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

Autrement dit, la probabilité qu'un individu appartienne à la classe 1 dépend linéairement des variables explicatives, mais est transformée par la fonction sigmoïde pour produire une sortie comprise entre 0 et 1.

Minimisation de la fonction de coût

L'algorithme repose sur le principe du maximum de vraisemblance. Il cherche à ajuster les paramètres du modèle (β) de manière à maximiser la probabilité que le modèle reproduise les étiquettes observées sur les données d'apprentissage.

Pour ce faire, il minimise une fonction de coût appelée log-loss (ou entropie croisée), à l'aide de techniques numériques d'optimisation. Par défaut, Scikit-learn utilise un solveur de descente de gradient (ex. lbfgs, liblinear) selon le paramétrage choisi.

Paramètres estimés pendant l'apprentissage

Pendant la phase d'entraînement, l'algorithme estime les coefficients (β) associés à chaque variable explicative, ainsi qu'un biais (β_0). Ces paramètres définissent le plan de séparation entre les deux classes dans l'espace des caractéristiques.

Chaque coefficient peut ensuite être interprété comme la contribution (positive ou négative) de la variable correspondante à la probabilité de passer dans la classe 1.

Optimisation par validation croisée et recherche d'hyperparamètres

Afin d'optimiser les performances du modèle, nous avons eu recours à une recherche sur grille (Grid Search) combinée à une validation croisée à 5 plis (cv=5). Cela permet d'évaluer plusieurs configurations de paramètres et de sélectionner celle qui maximise la performance moyenne du modèle.

Le code suivant a été utilisé :

```
gs = GridSearchCV(
   model_logistic_regression,
   {
      'C': [0.01, 0.1, 1, 10, 100],
      'penalty': ['l1', 'l2'],
      'solver': ['liblinear'],
      'max_iter': [100, 500, 1000]
   },
```

```
    cv=5
).fit(X, y)
```

Cette grille explore différentes valeurs de :

- C : inverse du terme de régularisation (plus C est grand, moins il y a de pénalité),
- penalty : type de régularisation (l1 ou l2),
- max_iter : nombre maximal d'itérations pour la convergence,
- solver : algorithme utilisé pour l'optimisation (liblinear, compatible avec l1 et l2).

Le modèle final utilisé a été celui renvoyé par GridSearchCV avec les meilleurs paramètres identifiés automatiquement sur le jeu d'apprentissage.

Remarque sur le paramètre *random_state=42*

Lors de l'entraînement du modèle de régression logistique, nous avons utilisé le paramètre random_state=42 dans la fonction LogisticRegression. Ce paramètre permet de fixer la graine aléatoire utilisée lors de l'initialisation, garantissant ainsi la reproductibilité des résultats à chaque exécution.

Le choix du nombre 42 et fréquemment utilisé dans les exemples et la documentation scientifique. Intrigué par cette constante, nous avons consulté plusieurs sources en ligne : il s'avère que ce choix est souvent en référence au roman *The Hitchhiker's Guide to the Galaxy* de Douglas Adams, dans lequel "*42*" est présenté comme la réponse à la question ultime sur la vie, l'univers et le reste. Bien que cela n'ait aucune incidence technique, ce nombre est une convention humoristique dans la communauté scientifique et informatique.

6. Évaluation du modèle

Une fois le modèle de régression logistique entraîné sur le jeu d'apprentissage, il est essentiel d'évaluer sa performance sur des données indépendantes. Pour cela, nous avons utilisé le jeu de test précédemment isolé, comportant 20 % des données totales.

Résultats obtenus

Le modèle a été évalué à l'aide des métriques suivantes :

- **Accuracy** : 0.8440

  Le taux de bonnes classifications atteint 84,4 %, ce qui est satisfaisant dans le cadre de ce problème.

- **Matrice de confusion** :

  [[1556 167]
   [223 554]]

- Interprétation :
  - 1556 vrais négatifs : le modèle a correctement prédit l'absence de réclamation.
  - 554 vrais positifs : le modèle a correctement détecté une réclamation.
  - 167 faux positifs : le modèle a prédit une réclamation là où il n'y en avait pas.
  - 223 faux négatifs : le modèle a manqué certaines réclamations.

- **Precision** : 0.7684

  Parmi toutes les réclamations prédites, 76,8 % étaient réellement positives.

- **Recall** : 0.7130

  Le modèle a identifié 71,3 % des réclamations réelles.

- **F1-Score** : 0.7397

  Moyenne harmonique entre précision et rappel, synthétisant la performance globale du modèle.

Analyse

Le modèle présente une bonne capacité de généralisation, avec un équilibre raisonnable entre la précision et le rappel. Cela montre qu'il est capable :

- de détecter une part importante des clients à risque (bon rappel),
- tout en limitant les fausses alertes (précision correcte).

7. Évaluation finale par validation croisée

Afin de confirmer la robustesse du modèle de régression logistique optimisé, une évaluation approfondie a été menée à l'aide d'une validation croisée en 5 plis. Cette méthode, plus fiable qu'une simple séparation en apprentissage/test, permet d'estimer la performance moyenne du modèle sur différentes partitions du jeu de données.

Résultats de la validation croisée (régression logistique)

Le tableau ci-dessous synthétise les performances du modèle sur les 5 plis, avec la moyenne et l'écart-type observés pour chaque métrique :

Métrique	Moyenne	Écart-type
Accuracy	0.8400	0.0052
Précision	0.7574	0.0155
Rappel	0.7206	0.0156
F1 Score	0.7383	0.0081

Interprétation

- L'accuracy montre que 84% des prédictions sont correctes.
- Le score de précision indique que près de 76 % des prédictions positives sont correctes.
- Le rappel atteint environ 72 %, ce qui reflète une bonne capacité du modèle à détecter les cas de réclamation.
- Le F1 Score, équilibre entre précision et rappel, est relativement élevé (0.7383), ce qui indique une bonne performance globale et peu de déséquilibre entre faux positifs et faux négatifs.
- L'écart-type faible des scores témoigne d'une bonne stabilité du modèle sur les différentes partitions du jeu de données.

Ces résultats confirment que le modèle de régression logistique, optimisé par Grid Search, est fiable, stable et performant.

9. Comparaison avec d'autres modèles de classification

Afin de mesurer la pertinence de la régression logistique dans ce contexte, nous avons comparé ses performances à celles de deux autres algorithmes de classification supervisée : le Perceptron et le K-Nearest Neighbors (KNN).

Chacun de ces modèles a été testé avec deux configurations :

- Paramètres par défaut, sans ajustement spécifique,
- Paramètres optimisés, obtenus via une recherche d'hyperparamètres (GridSearchCV).

8.1 Résultats comparatifs

Les performances ont été évaluées à l'aide des métriques classiques : accuracy, précision, rappel et F1 score. Les tableaux suivants présentent les moyennes des scores obtenus sur les 5 plis, ainsi que les écarts-types.

Régression logistique

Métrique	Moyenne	Écart-type
Accuracy	0.8400	0.0052
Précision	0.7574	0.0155
Rappel	0.7206	0.0156
F1 Score	0.7383	0.0081

Perceptron

Configuration	Accuracy	Précision	Rappel	F1 Score
Défaut	0.7748	0.6504	0.6166	0.6274
Paramètres optimisés	0.7852	0.6683	0.6763	0.6602

K-Nearest Neighbors (KNN)

Configuration	Accuracy	Précision	Rappel	F1 Score
Défaut	0.8045	0.6985	0.6623	0.6797
Paramètres optimisés	0.8304	0.7365	0.7152	0.7254

8.2 Analyse comparative

- La régression logistique présente les meilleurs résultats globaux, avec un excellent compromis entre précision (0.7574) et rappel (0.7206), ce qui se reflète dans son F1 Score élevé (0.7383). Elle reste le modèle le plus robuste, à la fois stable et interprétable.
- Le Perceptron, même après optimisation, affiche des performances moindres sur toutes les métriques. Son F1 Score plafonne à 0.6602, indiquant une difficulté à capturer les bons équilibres entre classes.
- Le KNN, quant à lui, se montre compétitif après tuning, atteignant un F1 Score de 0.7254, proche de celui de la régression logistique. Toutefois, sa précision (0.7365) reste légèrement inférieure, et il est potentiellement plus coûteux en temps de prédiction, notamment sur de grands jeux de données.

8.3 Conclusion

Au regard des résultats obtenus, la régression logistique optimisée reste le meilleur choix pour ce projet, en termes de performance, de stabilité, et d'interprétabilité. Le KNN peut constituer une alternative valable, notamment si l'on envisage des approches hybrides

ou ensemblistes. Le Perceptron, en revanche, semble moins adapté à ce jeu de données.

9. Sauvegarde du modèle entraîné

Une fois le modèle de régression logistique sélectionné et optimisé, il est important de prévoir sa mise en production ou sa réutilisation future sans avoir à le réentraîner. Cela passe par la sérialisation du modèle, c'est-à-dire son enregistrement dans un fichier exploitable ultérieurement.

Nous avons utilisé le module Python pickle, une bibliothèque standard permettant de sérialiser et désérialiser des objets Python. La méthode dump() permet d'enregistrer le modèle dans un fichier .pkl, tandis que la méthode load() permet de le recharger.

Points à améliorer

Bien que le modèle de régression logistique ait obtenu de bonnes performances globales, plusieurs axes d'amélioration peuvent être envisagés pour renforcer la précision, la robustesse et la pertinence des résultats.

Améliorations possibles du modèle

➤ Ingénierie des variables

Certaines variables ont été simplement encodées sans transformation avancée. Il serait intéressant de :

- Créer des variables composites (ex. : ratio accidents / années de conduite),
- Discrétiser des variables continues comme annual_mileage ou credit_score,
- Grouper les tranches d'âge ou d'expérience en classes plus pertinentes selon leur impact.

➤ Ajout de variables externes

Le modèle n'utilise que les données fournies. On pourrait envisager d'intégrer plus de données comme des données historiques si disponibles.

➤ Traitement du déséquilibre éventuel

- Si la classe outcome=1 était minoritaire (ce que l'on pourrait confirmer avec un value_counts()), on aurait pu :
  - Sur-échantillonner cette classe (SMOTE, RandomOverSampler),
  - Ajuster le poids des classes (class_weight='balanced' dans LogisticRegression),
  - Utiliser des métriques plus adaptées (AUC, courbe ROC) pour les problèmes déséquilibrés.

➤ Exploration de modèles supplémentaires

- Bien que la régression logistique fonctionne bien, nous aurions pu tester d'autres modèles.

➤ Meilleure gestion des valeurs aberrantes

- Bien que certaines valeurs extrêmes aient été corrigées, une détection plus fine via les z-scores, l'IQR, ou des boxplots aurait permis un traitement plus rigoureux.

Conclusion

Ce projet de classification supervisée appliqué au domaine de l'assurance automobile nous a permis de mettre en œuvre une démarche complète de data science, allant de l'exploration des données à la mise en œuvre d'un modèle optimisé.

À partir d'un jeu de données nous avons mené une série d'étapes rigoureuses :

- Nettoyage et préparation des données : gestion des valeurs manquantes, encodage, normalisation.
- Analyse exploratoire : étude des distributions, corrélations et pertinence des variables.
- Modélisation : entraînement de plusieurs classifieurs (régression logistique, Perceptron, KNN) avec validation croisée et optimisation d'hyperparamètres.
- Évaluation comparative : le modèle de régression logistique s'est révélé le plus performant, atteignant un F1-score de 0.7383 avec une bonne stabilité.
- Sauvegarde du modèle : le modèle final a été sérialisé avec pickle en vue d'une réutilisation future.

Ce travail nous a permis de constater l'importance de l'équilibre entre performance, interprétabilité et robustesse dans un contexte réel. Il a également mis en lumière les limites possibles d'un projet de classification, notamment en matière de qualité des données, de complexité algorithmique et d'optimisation.

Enfin, plusieurs pistes d'amélioration ont été identifiées : ingénierie de variables, exploration de modèles plus complexes, traitement du déséquilibre de classes, ou encore enrichissement des données.

Ce projet constitue une expérience concrète et structurante, et fournit les bases nécessaires pour aborder des problématiques similaires en entreprise ou dans des environnements plus complexes.