Projet Intégrateur

Compte rendu
Comparaison de plusieurs modèles de feature selection et de machine learning dans le cas de la détection d'anomalies dans les logs de serveurs

https://github.com/NathSimon/projet_integrateur

Nathanaël Simon
Hamza Zarfaoui
Ziyi Liu
nathanael.simon@telecom-paris.fr
hamza.zarfaoui@telecom-paris.fr
ziyi.liu@telecom-paris.fr

Télécom Paris, Institut Polytechnique de Paris

June 25, 2023

Abstract

Dans le cadre de la formation d'ingénieur de Télécom-Paris, les étudiants ont l'opportunité en deuxième année de réaliser un projet intégrateur.

Celui-ci à pour but de mettre en pratique les compétences acquises au cours de l'année, mais également de développer et croiser de nouvelles compétences, non abordées direcretement dans leur cursus, afin de se former dans des domaines étendus.

Le but de ce document est de rendre compte du projet mené par Hamza Zarfaoui, Nathanaël Simon et Ziyi Liu, intitulé "Comparaison de plusieurs modèles de feature selection et de machine learning dans le cas de la détection d'anomalies dans les logs de serveurs".

Les objectifs de la réalisation de ce projet sont la découverte et la mise en pratique de méthodes et modèles utilisé en science des données, appliqué au domaine de la fillière Sécurité des Réseaux et des Infrastructures Informatiques (SR2I). Il sera ici question d'expliquer le projet et ses objectifs, les méthodes utilisées, puis de discuter les résultats obtenus. Nous verrons ainsi l'implémnetation de différents modèles populaires de feature selection et d'intelligence artificielle, en comparant leurs différentes efficacité et fonctionnement dans le contexte de la cybersécurité. Dans un premier temps nous réaliserons une classification, puis une detection d'anomalie et enfin la réalisation d'un détecteur.

Contents

1 Présentation

En cybersécurité, la detection d'attaques et d'intrusions fait partie des domaines les plus importants dans la réponse à un incident. La détection la plus rapide et fiable possible permettra une réponse efficace. Dans la réalisation de cette tâche de détection, l'intelligence artificielle connaît, au même titre que dans les autres domaines scientifiques, un essort de plus en plus important, poussé par des progrès techniques et théoriques.

Les acteurs industriels, gouvernementaux ou associatifs sont particulièrement sujet à l'augmentation des cyberattaques sur leurs réseaux, et donc à la recherche de moyens de détections fiables, complémentaires aux Host Intrusion Detection System (HIDS). Celle-ci peut se faire à 3 niveaux différents.

- Niveau réseau : Analyse du traffic sur un réseau et détection de paquets suspects ou dangereux.
- Niveau logs : Analyse des logs du réseaux, aussi bien des logs sur les automates, postes de travail ou serveurs.
- Niveau électrique : Cas présent dans les industries possédant un réseau OT. Analyse des signaux électriques entre automates pour analyser les valeurs au plus proche du terrain et détecter les anomalies sans se fier aux outils de supervision.

Alors que le monde industriel connaît une révolution grâce aux techniques d'intelligences artificielles, et que la detection d'anomalie est de plus en plus primordiales dans la réponse aux incidents, nous avons choisi dans ce projet de nous intéresser a cette problématique. Nous souhaitons ainsi, par l'accomplissement de ce dernier, nous former sur la science des données appliqué à un cas concret de cybersécurité, et les problématiques liées à la detection des anomalies.

Le but de ce projet est de réaliser une étude sur les performances de trois algorithmes de machine learning, et trois techniques de features selections, adaptés à deux cas d'usages.

Le premier cas d'usage est de réaliser de la classification de logs d'un webserveur en trois catégories, normal, suspect et dangereux. Le but est d'évaluer les différents algorithmes dans un contexte d'utilisation où nous pouvons les entraîner sur un dataset complet.

Le second cas d'usage est de réaliser un detecteur d'intrusion sur les logs d'un webserveur, en se basant sur son fonctionnement normal. A partir de ce dernier, nous tenterons de voir si il est possible de réaliser une detection d'anomalie lors de l'injection de logs suspects et dangereux.

Les productions de ce projet sont réferencées dans les annexes. Cela contient, en plus de ce document, les deux notebooks python utilisés et une application de detection d'intrusion.

Au cours de ce document, nous présenterons le dataset que nous avons utilisé. Nous expliquerons ensuite les modèles de feature selection et de machine learning implémentés. Nous détaillerons les méthodes d'évaluation des différents modèles, avant d'analyser les résultats obtenus et d'explorer les pistes d'amliorations.

2 Dataset

Pour réaliser ce projet, nous avons utilisé un dataset représentant les logs d'un webserveur Apache. [1]

Les serveur Apache représente actuellement les serveurs les plus répendus mondialement, et partir de leur analyse dans le cadre de la détection d'anomalie nous a paru comme un choix pertintent.

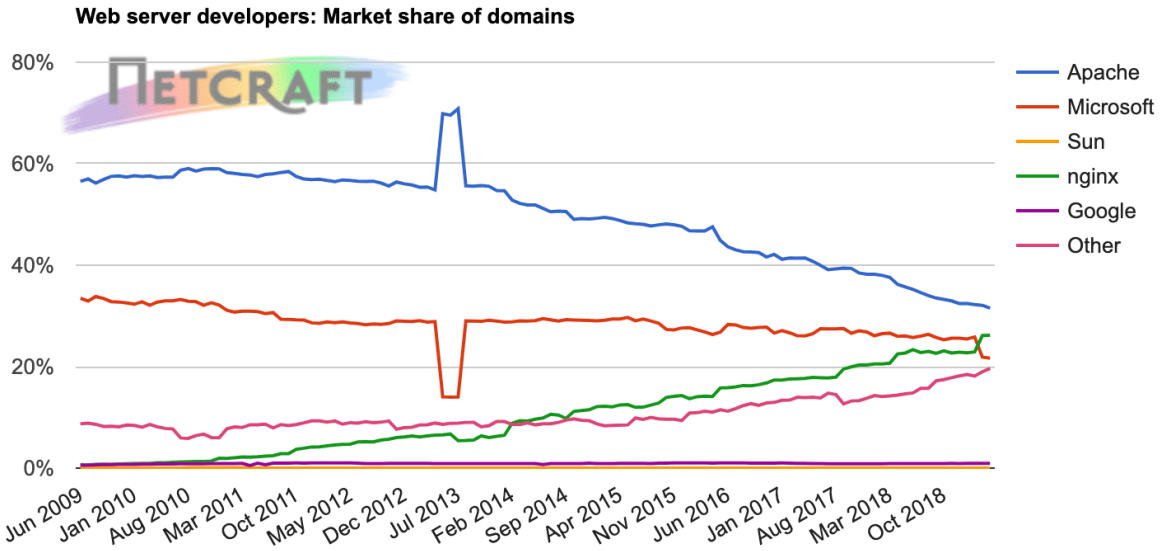

Figure 1: Part de marché des domaines - Source : Netcraft

Une des difficulté majeure dans tout projet de science des données, est de trouver un dataset permettant de travailler sur les problèmatiques qui nous intéresse. Il existe de nombreux dataset de logs disponnible, tel que [2] ou [3]. De plus, des outils tel que LogParseur [4], utililsant des techniques diversifié commme le Drain [5] [6], permettent de préparer les dataset pour l'utilisation d'algorithmes de machine learning. Cependant, ceux-ci ne présente pas de labels sur la nature des logs qu'ils référencent, ce qui complexifie grandement l'analyse de ces derniers. De plus, ceux-ci référencent souvent des datasets sain, qui peuvent être utilisé à des fins d'entraînement, mais non pas sur de l'évaluation, où un dataset présentant des anomalies référencées est nécessaire.

2.1 Préparation du dataset

La prochaine étape importante dans la récupération des datasets est leur préparation pour l'entraînement des modèles. Il faut dans un premier temps identifier les features du dataset.

Cette étape peut se faire à l'aide des dataframes de la bibliothèque Pandas [7]. Ensuite, il est nécessaire de vectoriser les données pour les rendre traitable par un algorithme.

En effet, les données sous formes de chaînes de caractères ne peuvent servir d'entrainement. Nous avons retenus ici le choix d'utiliser la factorisation [8], qui permet d'associer à chaque chaine de caractères unique un entier unique. Nous avons préféré cette méthode à CountVectorizer [9], utisant une réprésentation avec une matrice creuse, pour sa facilité de compréhension de la donnée vectorisée.

2.1.1 Classification

Pour la classification, nous avons associé à chaque label du dataset, `AMAN`, `DICURIGAI` et `BAHAYA`, se traduisant du malaysien par bénin, suspect et danger, un entier unique, en utilisant LabelEncoder [10].

Ainsi, les labels précédents ont respectivement les codes `0`, `1` et `2`.

Afin de pouvoir entrainer et tester nos différents modèles, il est necessaire de séparer le dataset en 2 sous-sets distincts. Un premier servira à l'entrainment, tandis que l'autre à l'évaluation. Ainsi, les modèles ne seront pas testés sur des valeurs sur lesquelles ils se sont entrainés.

Le ratio de ces deux sous-sets peut varier dans les implémentations, mais il est généralement compris entre 70/30 ou 80/20. Dans notre cas nous sommes avons choisi une répartition 80/20, permettant un nombre important de données d'entrainement, tout en évitant les problèmes d'overfitting. [11]

2.1.2 Détection d'anomalies

Pour la détection d'anomalies, nous avons suivi les mêmes étapes, mais en les adaptants aux besoins de la problématique.

Dans un premier temps, nous avons adapté nos labels, pour n'avoir que deux catégories, normal et anomalie.

`AMAN` obtient le label `0`, pour son fonctionnement normal, tandis ce que `DICURIGAI` et `BAHAYA` ont le labels `1`, pour leur fonctionnement déviant. Ensuite, nous avons créé deux sous sets; avec une répartition de 70/30, pour conserver un nombre plus important de données de tests.

Dans le set d'entrainement, nous avons conservée uniquement les entrées étant labélisées `0`, pour pouvoir entrainer nos algorithmes sur le fonctionnement nomial de la machine cible.

3 Modèles utilisés

3.1 Feature selection

La feature selection est une étape de pré-traitement des données avant l'entrainement d'un modèle d'intelligence artificielle. Le but est d'identifier quelles sont les features de

notre dataset pertinentes pour la prédiction. En se faisant, on réduit la dimension de nos données et donc la complexité d'entraînement et de prédiction. De plus, on peut éliminer une partie du bruit des données, afin d'avoir une meilleure généralisation. Enfin cela permet de réduire les risques d'overfitting.

3.1.1 Principal Component Analysis

Le modèle de feature selection PCA (Principal Component Analysis), est une méthode couramment utilisée dans le domaine de l'apprentissage automatique pour réduire la dimensionnalité des données. L'idée principale derrière son fonctionnement est de transformer un ensemble de variables d'origine en un nouvel ensemble de variables appelées composantes principales, qui sont des combinaisons linéaires des variables d'origine.

Le PCA cherche à trouver les directions dans lesquelles les données varient le plus, appelées axes principaux. Ces axes principaux sont ordonnés en fonction de leur variance, ce qui signifie que les premiers axes principaux capturent la plus grande partie de la variance des données. En choisissant un nombre réduit d'axes principaux, on peut réduire la dimensionnalité des données tout en conservant une grande partie de leur information. PCA peut également être utilisé pour visualiser les données en projetant les observations sur les axes principaux.

PCA présente également des inconvénients. Il suppose que les données sont linéairement corrélées, ce qui peut être une limitation si les relations entre les variables sont non linéaires. De plus, le PCA ne tient pas compte des étiquettes de classe lors de la sélection des axes principaux, ce qui peut être problématique dans les problèmes de classification où la séparation des classes est importante. [12]

3.1.2 Kendall Correlation

Le modèle de feature selection Kendall correlation, ou Kendall's tau, permet d'évaluer les relations d'ordre entre les caractéristiques et une variable cible. Kendall's cherche à mesurer la corrélation non linéaire entre les rangs des caractéristiques et les rangs de la variable cible.

La corrélation de Kendall évalue la similarité entre les ordres des valeurs des caractéristiques et les ordres des valeurs de la variable cible. Elle mesure la proportion de paires de données dont l'ordre des rangs est concordant (c'est-à-dire que les rangs sont dans le même ordre relatif) ou discordant (c'est-à-dire que les rangs sont dans un ordre différent) par rapport à toutes les paires de données. Une corrélation de Kendall proche de +1 indique une corrélation positive forte, tandis qu'une corrélation proche de -1 indique une corrélation négative forte.

Les avantages du modèle de feature selection Kendall correlation résident dans sa capacité à détecter des relations d'ordre non linéaires entre les caractéristiques et la variable cible. Il peut être utilisé pour sélectionner des caractéristiques qui ont un impact significatif sur la variable cible, même en l'absence de relations linéaires. De plus, la

corrélation de Kendall est moins sensible aux valeurs aberrantes que d'autres mesures de corrélation, ce qui en fait une méthode robuste.

Cependant, la corrélation de Kendall ne capture que les relations d'ordre et ne tient pas compte des relations numériques précises entre les caractéristiques et la variable cible. Elle peut également être sensible à la taille de l'échantillon, car les estimations de la corrélation peuvent être instables avec de petits échantillons. [13] [14]

3.1.3 Analysis of Variance

Le modèle de feature selection ANOVA (Analysis of Variance), est une méthode utilisée dans le domaine de l'apprentissage automatique pour sélectionner les caractéristiques les plus informatives pour un problème donné. Le but est de mesurer la variation entre les groupes de données en utilisant l'analyse de la variance et de sélectionner les caractéristiques qui présentent une variation significative par rapport à la variable cible.

L'ANOVA compare les moyennes des différentes classes ou groupes de données et évalue si ces moyennes sont significativement différentes les unes des autres. Pour chaque caractéristique, on calcule la somme des carrés des différences entre les valeurs réelles et les moyennes de chaque groupe. Si cette somme est importante par rapport à la variabilité totale des données, la caractéristique est considérée comme significative.

Les avantages du modèle de feature selection ANOVA résident dans sa simplicité et sa facilité d'interprétation. Il permet de sélectionner les caractéristiques les plus discriminantes en termes de variation entre les groupes de données, ce qui peut améliorer les performances des modèles d'apprentissage automatique. De plus, l'ANOVA est relativement rapide à calculer, même pour des ensembles de données de grande taille.

Cependant, l'ANOVA présente également des inconvénients. Il suppose que les données suivent une distribution normale et que les variances sont homogènes entre les groupes, ce qui peut ne pas être le cas dans certaines situations réelles. De plus, l'ANOVA ne tient pas compte des relations non linéaires entre les caractéristiques et la variable cible, ce qui peut conduire à une sélection sous-optimale des caractéristiques dans certains cas. [15] [16]

Cependant, lors de nos travaux, nous n'avons pas reussi a implementer ANOVA avec des resultats exploitables, et avons donc décider d'implémenter une autre méthode de feature selection.

3.1.4 kBests

Le modèle de feature selection kBests permet de sélectionner les k meilleures caractéristiques d'un ensemble de données. Son but est d'évaluer chaque caractéristique individuellement en utilisant une mesure de mérite spécifique, puis de sélectionner les k caractéristiques les plus performantes.

Les mesures de mérite couramment utilisées dans le modèle de feature selection kBests comprennent le score F, le score chi-carré, l'information mutuelle et d'autres métriques

statistiques. Chaque mesure évalue l'association ou la dépendance entre la caractéristique et la variable cible. Les caractéristiques sont ensuite classées en fonction de leur score et les k caractéristiques avec les scores les plus élevés sont sélectionnées.

Les avantages du modèle de feature selection kBests résident dans sa simplicité et sa facilité d'utilisation. Il permet de sélectionner un sous-ensemble de caractéristiques les plus informatives pour un problème d'apprentissage automatique, ce qui peut améliorer les performances des modèles en réduisant la dimensionnalité des données. De plus, cette méthode est rapide à calculer, même pour de grands ensembles de données.

Cependant, le modèle de feature selection kBests ne prend en compte que les relations entre chaque caractéristique et la variable cible individuellement, sans considérer les interactions ou les dépendances entre les caractéristiques. Par conséquent, il peut ne pas capturer les relations complexes et non linéaires présentes dans les données. De plus, la sélection des k meilleures caractéristiques peut ne pas être optimale pour tous les problèmes, ce qui peut entraîner une perte d'information importante. [17]

3.1.5 Application a la detection d'anomalie

Pour l'application à la détection d'anomalies sur des données saines, les méthodes de feature selection ne peuvent être utilisées.

En effet, celles-ci cherchent à évaluer l'importance des différentes features pour la prédiction d'un modèle. Il est nécessaires d'avoir des données comportant plusieurs labels différents, permettant d'évaluer l'importance de chaque feature pour prédire les différences entre chaque label.

Dans le cas de la détection d'anomalie, nous ne possédons alors que des labels sains. Il n'est donc pas possible de déduire des liens entre chaque features et la prédiction, et donc, nous ne pourrons appliquer ces méthodes.

3.2 Machine Learning

Lors de la réalisation de ce projet, nous nous sommes également concentrés sur 3 modèles de machines learning.

3.2.1 Support Vector Machine

Le modèle d'IA Support Vector Machine (SVM) est une méthode d'apprentissage supervisé utilisée pour la classification et la régression. L'idée principale derrière son fonctionnement est de trouver un hyperplan optimal qui sépare les différentes classes de données. L'objectif est de maximiser la marge entre les points de données les plus proches de chaque classe, appelés vecteurs de support, et l'hyperplan de décision. Cela permet de garantir une bonne généralisation et une capacité à traiter des données non linéaires, en utilisant la technique du kerneling.

Le concept de kernelling, également connu sous le nom de fonction de noyau, est une composante essentielle des SVM et d'autres méthodes d'apprentissage automatique. Il permet de transformer les données d'entrée dans un espace de dimension supérieure, où la séparation des classes devient plus facile.

Les SVM utilisent différents types de kernels pour transformer les données d'entrée dans un espace de dimension supérieure, où la séparation des classes peut être plus facilement réalisée. Les kernels les plus couramment utilisés sont le linéaire, le polynomial et le gaussien (ou RBF). Le kernel linéaire est utilisé lorsque les données sont linéairement séparables, tandis que les kernels polynomial et gaussien sont plus adaptés aux données non linéaires.

Les avantages des SVM résident dans leur capacité à traiter efficacement des ensembles de données de grande dimension et dans leur résistance aux problèmes de sur-ajustement. Ils peuvent également fonctionner avec des données non linéaires grâce à l'utilisation de kernels. De plus, les SVM peuvent gérer des ensembles de données contenant des valeurs manquantes.

Cependant, le choix du kernel approprié peut être délicat, et il peut y avoir des compromis entre la complexité du modèle et sa capacité de généralisation. Une mauvaise sélection de kernel peut entraîner un sur-ajustement ou une sous-ajustement du modèle aux données d'entraînement.

Les SVM possèdent enfin une complexité d'entraînement importante, entre $O(n^{2p})$ et $O(n^{3p})$ pour un modèle avec n samples et p features. De plus, l'interprétation des résultats d'un SVM peut être complexe, car il est difficile de comprendre l'influence de chaque variable sur la décision finale.

En résumé, les SVM offrent un cadre puissant pour la classification et la régression en utilisant des hyperplans de décision optimaux et des kernels pour traiter des données non linéaires. Malgré certains inconvénients, ils restent très utilisé pour leur versatilité. [18]

Dans le cadre de la détection d'anomalie, nous utiliserons OneClassSVM [19]

3.2.2 Extreme Gradient Boost

Le modèle d'IA XGBoost, abréviation de "Extreme Gradient Boosting", est une méthode d'apprentissage automatique qui appartient à la famille des modèles de gradient boosting. L'idée principale derrière son fonctionnement est de construire un modèle prédictif en combinant plusieurs modèles de prédiction faibles, tels que des arbres de décision, de manière itérative.

Le fonctionnement d'XGBoost repose sur l'optimisation d'une fonction de perte en utilisant le gradient boosting. À chaque itération, un nouvel arbre de décision est ajouté au modèle pour capturer les erreurs résiduelles du modèle précédent. L'algorithme recherche l'arbre optimal qui minimise la fonction de perte en utilisant une technique d'optimisation appelée "gradient boosting".

Les avantages d'XGBoost résident dans sa capacité à fournir des modèles de prédiction de haute qualité avec une grande précision. Il est capable de gérer efficacement des

ensembles de données volumineux et complexes grâce à des techniques d'optimisation avancées et à un parallélisme efficace. De plus, XGBoost fournit des fonctionnalités telles que la gestion des valeurs manquantes et la régularisation pour prévenir le sur-ajustement.

Cependant, XGBoost peut nécessiter un réglage minutieux des hyperparamètres pour obtenir les meilleurs résultats, ce qui peut être un processus complexe et nécessiter une expertise supplémentaire. De plus, l'entraînement d'XGBoost peut être relativement lent en raison de la construction itérative de nombreux arbres de décision. [20] [**?**]

XGBoost est difficile à adapte à un contexte de détection d'anomalies, et n'utiliserons pas l'algorithme dans le cadre de cette problématique.

3.2.3 Random Forest

Le modèle de random forest, ou "forêt aléatoire", est une méthode d'apprentissage automatique basée sur l'ensemble d'arbres de décision. L'idée principale derrière son fonctionnement est de construire un grand nombre d'arbres de décision indépendants et de combiner leurs prédictions pour obtenir un résultat final.

La random forest fonctionne en créant un échantillon aléatoire avec remplacement à partir de l'ensemble d'entraînement pour chaque arbre de décision. Chaque arbre est construit en utilisant un sous-ensemble différent des caractéristiques d'entrée, ce qui favorise la diversité des arbres. Lors de la phase de prédiction, les prédictions de chaque arbre sont agrégées, par exemple en utilisant un vote majoritaire pour les problèmes de classification ou une moyenne pour les problèmes de régression.

Les avantages de la random forest résident dans sa capacité à fournir des modèles robustes et précis, avec une bonne généralisation. Elle est moins sujette au sur-ajustement que les arbres de décision individuels, car elle utilise des moyennes et des votes majoritaires pour prendre des décisions. De plus, la random forest est capable de gérer des ensembles de données de grande dimension et des caractéristiques non linéaires.

Cependant, la random forest peut être plus complexe à interpréter que les arbres de décision individuels en raison de la combinaison de plusieurs arbres. La complexité d'entrainement est de $O(t * u * n * log(n))$ où t est le nombre d'arbres, u est le nombre de features et n le nombre de samples. Elle peut également être plus lente à entraîner et à prédire, en raison du grand nombre d'arbres et du calcul supplémentaire requis pour agréger les résultats. La prédiction se fait en $O(t * log(n))$. [21]

Dans le cadre de la détection d'anomalies, nous utiliserons IsolationForest [22].

4 Evaluation

Dans cette partie, nous discuterons les différentes méthodes d'évaluations des modèles cité précedement, et le choix de celles-ci dans notre problème.

Nous prendrons principalement les valeurs suivantes pour construire nos méthodes d'évaluations :

- TP - True Positive : les prédictions positives justes.
- TN - True Negative : les prédictions négatives justes.
- FP - False Positive : les prédictions positives fausses.
- FN - False Negative : les prédictions négatives fausses.

Remarque : Dans un cas avec plus de deux classes, nous nous réferrons à la matrice de confusion de notre modèle. Celle ci décrit les prédictions attendues et obtenues par classes, et permet d'adapter les valeurs vues précedement. Nous pouvons l'illustrer avec un exemple pour une classification à trois élements.

Class / Prediction	Class1	Class2	Class3
Class1	T1	F1	F1
Class2	F2	T2	F2
Class3	F3	F3	T3

Matrice de confusion pour une classification sur 3 labels distincts

Cette matrice met en évidence les prédictions réalisées par le modèle par rapport aux attentes. Dans ce modèle, il n'existe plus de FP et FN, mais seulement des valeurs justes où fausses. Les valeurs diagonnales a_{ij} avec $i = j$ représente les prédictions justes, tanfis que les termes a_{ij} avec i différent de j représente les prédictions fausses. [23]

4.1 Accuracy

La première méthode pour réaliser une évaluation est l'accuracy de notre modèle. Celle ci représente le nombre de précisions correcte sur l'ensemble des prédictions réalisées.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Remarque : Dans un modèle avec N classes, il est necessaire d'adapter la formule précedente de la sorte en utilisant la matrice de confusion A :

$$Accuracy = \frac{\sum_{i=1}^{N} a_{ii}}{\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij}}$$

Elle présente un résultat simple à interpréter et qui permet d'avoir une idée du fonctionnement général du modèle, mais qui ne permet pas de rentrer dans les détails du multi-class. En analysant l'ensemble des prédictions faites, il n'y a donc plus de distinctions, et il peut être compliqué de savoir comment améliorer le modèle.

4.2 Recall

Le recall est calculé en réalisant la fraction des TP par l'ensemble des prédictions positives du set de données initial.

$$Rec = \frac{TP}{TP + FN}$$

Le but du recall est de mesurer la capacité du modèle à trouver tous les positifs dans le set originel. [23]

4.3 Precision

La précision est calculé en réalisant la fraction des TP par l'ensemble des prédictions positives du modèle.

$$Prec = \frac{TP}{TP + FP}$$

Le but de la précision est de mesurer la proporition de prédiction que notre modèle a réalisé positives de manière adéquate, par rapport à l'ensemble de ses prédictions positives. Cela permet d'avoir une idée sur la confiance que l'on peut donner à notre modèle lorsque celui-ci réaliser une prédiction positive. [23]

4.4 F1-Score

Le F1-Score est la moyenne harmonique du recall et de la précision, calculée de la manière suivante :

$$F1score = \frac{2 * (Prec * Rec)}{Prec + Rec}$$

Cette valeur peut être vue comme une moyenne pondérée entre la précision et le recall, où les 2 ont le même poids. Le F1-score est égal à 1 si on a une précision et un recall parfait, et à 0 si soit le recall, soit la précision est nulle.

4.5 Macro-average F1-Score

Le Macro-average F1-Score calcule la moyenne non pondérée des F1-Score pour chaque classe du modèle. Elle est calculée de la manière suivante :

$$MacroAverageF1Score = \frac{\sum_{i=1}^{N} F1Score_{class(i)}}{N}$$

Cette formule est adaptable au Recall et à la Précision également. Elle permet d'avoir une vue d'ensemble du modèle en utilisant les valeurs précdement décrites.

4.6 Weighted-average F1-Score

Le Weighted-average F1-Score vise à donner une vue d'ensemble du modèle, de la même manière que le Macro-average F1-Score, mais en prenant en compte la taille des échantillons pour pondérer son calcul.

Chaque F1-Score est multiplié par sa proportion tel que :

$$WeightedF1Score = \frac{2 * (Prc(Weight) * Rec(Weight)}{(Prc(Weight) + Rec(Weight)}$$

avec :

$$Prc(Weight) = \frac{\sum_{i=1}^{N} Prc(class(i)) * class(i)}{\sum_{i=1}^{N} class(i)}$$

$$Rec(Weight) = \frac{\sum_{i=1}^{N} Rec(class(i)) * class(i)}{\sum_{i=1}^{N} class(i)}$$

Cette mesure permet ainsi de pondérer l'évalution de notre modèle en fonction des données d'entrées, et ainsi tenter d'avoir une mesure relativement fiable pour son évaluation. Celle ci peut cependant produire des résultats peut exploitable dans le cas de dataset déséquilibré, où la partie minoritaire est celle que l'on souhaite mesurer. Le choix du Macro-F1-Score est alors plus adapté dans ces problématiques.

5 Resultats

Cette partie vise à présenter les résultats obtenus après l'entrainement des différents modèles. Nous détaillerons et expliquerons ces résultats dans la partie suivante.

Pour chaque problématique, nous présentons les résultats à l'aide d'un tableau général, et les différents tableaux détaillés par la suite.

5.1 Classification

5.1.1 Sans feature selection

Method	Accuraccy	Macro F1-Score
SVM	0.91033	0.62716
Random Forest	0.99933	0.99825
XGBoost	0.99946	0.99844

Résultats généraux des algorithmes mis en place pour la classification sans feature selection

	Precision	Recall	F1-Score
Normal	0.91468	0.98538	0.94871
Suspect	0.88564	0.98521	0.93277
Danger	0.00000	0.00000	0.00000
Accuracy	0.91033		
Macro-average	0.60011	0.65686	0.62716
Weighted-average	0.84113	0.91033	0.87434

Détail des résultats de SVM pour la classification sans feature sélection

	Precision	Recall	F1-Score
Normal	0.99983	0.99966	0.99975
Suspect	0.99802	0.99606	0.99704
Danger	0.99480	1.00000	0.99739
Accuracy	0.99920		
Macro-average	0.99755	0.99857	0.99806
Weighted-average	0.99921	0.99920	0.99920

Détail des résultats de Random Forest pour la classification sans feature sélection

	Precision	Recall	F1-Score
Normal	1.00000	0.99983	0.99992
Suspect	0.99901	0.99704	0.99803
Danger	0.99480	1.00000	0.99739
Accuracy	0.99947		
Macro-average	0.99794	0.99896	0.99845
Weighted-average	0.99947	0.99947	0.99947

Détail des résultats de XGBoost pour la classification sans feature sélection

5.1.2 Avec feature selection

Feature Selection	Method	Accuraccy	Macro F1-Score
PCA	SVM	0.90994	0.62645
PCA	Random Forest	0.98899	0.97360
PCA	XGBoost	0.98713	0.97009
kBests	SVM	0.88898	0.59787
kBests	Random Forest	0.99814	0.99653
kBests	XGBoost	0.99841	0.99718
Kendall's	SVM	0.78976	0.29876
Kendall's	Random Forest	0.99204	0.97526
Kendall's	XGBoost	0.99337	0.97800

Résultats généraux des algorithmes mis en place pour la classification avec feature selection

Feature selection	Measure	Precision	Recall	F1-Score
PCA	Normal	0.91624	0.98336	0.94861
PCA	Suspect	0.87500	0.99408	0.93075
PCA	Danger	0.00000	0.00000	0.00000
PCA	Accuracy		0.90994	
PCA	Macro-average	0.59708	0.65915	0.62645
PCA	Weighted-average	0.84093	0.90994	0.87399
kBests	Normal	0.90101	0.97278	0.93552
kBests	Suspect	0.81957	0.90039	0.85808
kBests	Danger	0.00000	0.00000	0.00000
kBests	Accuracy		0.88898	
kBests	Macro-average	0.57353	0.62439	0.59787
kBests	Weighted-average	0.82146	0.88898	0.85388
Kendall's	Normal	0.78975	0.99983	0.88246
Kendall's	Suspect	0.00000	0.00000	0.00000
Kendall's	Danger	0.80000	0.00697	0.01382
Kendall's	Accuracy		0.78976	
Kendall's	Macro-average	0.52992	0.33560	0.29876
Kendall's	Weighted-average	0.68431	0.78976	0.69763

Détail des résultats de SVM pour la classification avec feature sélection

Feature selection	Measure	Precision	Recall	F1-Score
PCA	Normal	0.99280	0.99614	0.99446
PCA	Suspect	0.97654	0.98521	0.98085
PCA	Danger	0.97064	0.92160	0.94549
PCA	Accuracy		0.98899	
PCA	Macro-average	0.97999	0.96765	0.97360
PCA	Weighted-average	0.98892	0.98899	0.98890
kBests	Normal	0.99933	0.99866	0.99899
kBests	Suspect	0.99408	0.99408	0.99408
kBests	Danger	0.99308	1.00000	0.99653
kBests	Accuracy		0.99814	
kBests	Macro-average	0.99550	0.99758	0.99653
kBests	Weighted-average	0.99815	0.99814	0.99814
Kendall's	Normal	0.99933	0.99899	0.99916
Kendall's	Suspect	0.97992	0.96252	0.97114
Kendall's	Danger	0.93939	0.97213	0.95548
Kendall's	Accuracy		0.99204	
Kendall's	Macro-average	0.97288	0.97788	0.97526
Kendall's	Weighted-average	0.99215	0.99204	0.99207

Détail des résultats de Random Forest pour la classification avec feature sélection

Feature selection	Measure	Precision	Recall	F1-Score
PCA	Normal	0.99047	0.99513	0.99279
PCA	Suspect	0.97754	0.98718	0.98234
PCA	Danger	0.96828	0.90418	0.93514
PCA	Accuracy		0.98713	
PCA	Macro-average	0.97876	0.96216	0.97009
PCA	Weighted-average	0.98704	0.98713	0.98700
kBests	Normal	0.99933	0.99882	0.99908
kBests	Suspect	0.99507	0.99507	0.99507
kBests	Danger	0.99480	1.00000	0.99739
kBests	Accuracy		0.99841	
kBests	Macro-average	0.99640	0.99796	0.99718
kBests	Weighted-average	0.99841	0.99841	0.99841
Kendall's	Normal	1.00000	0.99950	0.99975
Kendall's	Suspect	0.98592	0.96647	0.97610
Kendall's	Danger	0.93970	0.97735	0.95816
Kendall's	Accuracy		0.99337	
Kendall's	Macro-average	0.97520	0.98111	0.97800
Kendall's	Weighted-average	0.99351	0.99337	0.99340

Détail des résultats de XGBoost pour la classification avec feature sélection

5.2 Détection d'anomalies

La méthode que nous avons choisi d'appliquer pour obtenir les résultats de la détection d'anomalie est de récupérer, pour chaque prédiction du modèle, les scores d'anomalies. Ceux-ci sont compris entre -1 et 1, et le modèle applique aux valeurs considérées comme des anomalies une note négative. Il est donc possible ensuite, d'associer, pour chaque valeur prédite, le label 0 ou 1 en fonction des scores d'anomalies.

Cependant, on peut également décider du seuil pour lequel nous appliquons ce label. On parle alors d'un seuil de confiance. Il est possible de chercher le meilleur seuil qui maximise une des variables d'évaluation, et nous avons alors choisi de chercher celui qui maximise le Macro-F1-Score. En effet, il y a un déséquilibre dans le dataset entre les logs normaux et anormaux. Maximiser l'accuracy reviendrait donc à pousser le modèle à maximiser sa prédiction de labels normaux, ce qui ne réponds pas à notre problématique. De même maximiser le Weighted-F1-Score revient à maximiser les labels normaux, du au déséquilibre du dataset.

Pour répondre à cette problématique, il aurait alors été possible de rééquilibrer le dataset, ou bien d'utiliser en Macro-F1. En effet, celui ci n'étant pas pondéré par la taille des échantillons, il peut alors nous donner un bon équilibre entre la détection de logs normaux et des anomalies.

Tous les résultats suivants ont été réalisé en choisissant un treshold suivant cette logique. Pour la réalisation d'un détecteur fonctionnel, nous détaillerons notre méthode dans une partie suivante. De plus, comme expliqué précedement, nous n'appliquerons pas d'algorithmes de feature selections dans cette partie.

Method	Accuraccy	Macro F1-Score
SVM	0.66599	0.64399
Random Forest	0.72807	0.59783

Résultats généraux des algorithmes mis en place pour la classification sans feature selection

	Precision	Recall	F1-Score
Normal	0.99672	0.57899	0.73249
Anomalie	0.38562	0.99285	0.55549
Accuracy	0.66599		
Macro-average	0.69117	0.78592	0.64399
Weighted-average	0.86827	0.66599	0.69528

Détail des résultats de OneClassSVM pour la classification sans feature sélection

	Precision	Recall	F1-Score
Normal	0.83227	0.82118	0.82669
Anomalie	0.36018	0.37821	0.36897
Accuracy	0.72807		
Macro-average	0.59623	0.59970	0.59783
Weighted-average	0.73304	0.72807	0.73048

Détail des résultats de Isolation Forest pour la classification sans feature sélection

5.3 Réalisation du détecteur

5.3.1 Réaliation

Pour réaliser le détecteur, nous avons choisi d'utiliser la bibliothèque python CustomTkinter. Cette dernière permet de réaliser de facon très efficace une application graphique répondant à nos besoins.

L'application se présente sous la forme suivant:

- une section "Information" présentant le logiciel (objectif, techniques utilisées, ...)
- une section "Entrainement" permettant d'entrainer un modèle de machine learning (Isolation Forest ou One Class Support Vector Machines) en lui fournissant un fichier de logs normaux.

- une section "Prédiction" permettant de détecter des anomalies en fournissant le fichier de logs concerné.

Comme évoqué précedemment, nous avons choisi d'implémenter deux modèles de machine learning à savoir Isolation Forest et One Class Support Vector Machines. Il s'agit des principaux modèles permettant de détecter une déviation du comportement dans un jeu de donnée.

Après avoir analysé un jeu de donnée, quatre cas sont possibles :

- Auncune anomalie suspecte ou dangereuse n'est détectée : dans ce cas, rien de particulier n'est affiché
- Des anomalies suspectes sont détectées mais aucune anomalie dangereuse : dans ce cas, une fenetre contenant l'ensenmble des requetes concerné va s'afficher.
- Des anomalies dangereuses sont détectées mais aucune anomalie suspecte : dans ce cas, une fenetre contenant l'ensenmble des requetes concerné va s'afficher.
- Des anomalies dangereuses et suspectes sont détectées : dans ce cas, deux fenetres vont s'afficher, une contenant les requetes suspectes et une autre les requetes dangereuses.

5.3.2 Evaluation

Pour réaliser l'évaluation du detecteur qui implémente les modèles entrainés pour la détection d'anomalie, nous allons adapté notre prédiction.

Dans un premier temps, nous allons évaluer 1000 logs à la fois, en tentant de savoir si les logs sont sains, supects, ou dangereux.

Pour se faire, nous allons réutiliser les résultats d'anomalies des logs, et adapter nos seuils de confiance (treshold), pour en avoir 3.

Le premier sera très proche de la position maximisant le F1 Score, et permettra d'analyser globalement les résultats. Le deuxième sera légèrement plus élevé, et diminuera le nombre de prédictions positives. Seront alors marquées anormales seulement les anomalies légèrement plus fortes. Le troisième sera encore plus élevé, et diminuera encore le nombre de prédictions positives. Seront alors marquées que les anomalies les plus importantes.

Si le troisième treshold repère des anomalies, alors nous indiquerons les 1000 logs évalués comme dangereux. Si le deuxième treshold repère des anomalies, alors nous indiquerons les 1000 logs comme suspects. Enfin, si seulement le premier treshold est activé, nous indiquerons les 1000 logs comme normaux.

Il est important de préciser que cette méthode part de l'hypothèse que les logs dangeureux ont un déviation plus forte des logs normaux que ceux suspects.

5.3.3 Résultats

Nous avons utiliser 3 sets de 1000 samples pour tester l'application. Le premier contient uniquement des logs normaux, le deuxième des logs normaux et suspects, et enfin le dernier des logs normaux, suspects, et dangereux.

Nous avons utilisé le dataset normal pour calibrer le premier treshold, nous assurant de ne pas detecter de logs suspects, et le dataset suspects pour régler le second treshold, nous assurant de ne pas détecter de logs dangereux.

Cette méthode n'a pas prouvé son efficacité au cours de nos tests, ne parvenant pas à réaliser une séparation parfaite, et les modèles prédisant de nombreux logs normaux comme suspects.

6 Discussion

Dans cette partie nous discuterons des résultats détaillés précedement, en mettant en avant les princpaux indicateurs de performances, tout en explorant des pistes d'améliorations. Nous analyserons en particulier l'accuracy de chaque modèle et son Macro-F1-Score, du à la nature déséquilibré du dataset.

6.1 Classification

Pour la classification, on observe une différence très forte entre SVM et les deux autres algorithmes utilisés.

XGBoost et RandomForest possèdent tous deux des résultats proches de `1` dans chaque mesures d'évaluations. Ceci signifie que la préparation du dataset à été efficace et permet à ces deux algorithmes des performances très élevées, avec une accuracy de 0.999.

SVM n'a pas produit de résultats aussi performants. En particulier, SVM n'a produit aucune prédiction juste pour la catégorie `DANGER`.

Les raisons pour cette incapacité nous ont été difficiles à identifier. Aussi bien le choix du kernel (ici `rbf`) que la paramétrisation du model peuvent être en defaut ici. Une piste d'amélioration serait alors de réaliser une analyse des données poussées pour identifier le kernel le plus adapté, ainsi que les paramètres associés.

En implémentant maintenant la feature selection, toutes les mesure d'évaluation pour XGBoost et Random Forest ont légèrement diminué. On peut identifer que kBest est la méthode qui donne les meilleurs résultats, suivi de Kendall's puis PCA de manière presque identique.

On peut alors supposer que, dans notre situation, toutes les features jouent un rôle important dans la prise de décision. Les données sont alors peu bruitées, et réduire la dimensions de celles-ci réduisent légèrement nos capacitées de classification.

Pour SVM, l'application de la feature selection augmente l'analyse précédente. En particulier pour Kendall's, qui provoque un effondrement du Macro-F1-Score en plus de l'accuracy. Ceci est du au fait que, lors de l'application de Kendall, SVM n'a pas su prédire la classe `suspect`, en plus de la classe `danger`. Les mêmes hypothèses que précédement peuvent alors être faites sur la paramétrisation du modèle. Pour pousser l'analyse sur les détails de la feature selection, il faudrait avoir un modèle SVM performant sans feature selection pour avoir une base de comparaison solide avec les autres algorithmes.

En résumé, dans le cas de la classification de logs, les modèles Random Forest et XGBoost s'avèrent être performants. Toutes les features des données d'entrées semblent cependant jouées un rôles dans la prédiction du modèle, et l'application d'algorithmes de feature selection ne permet pas d'observer d'améliorations des résultats.

Nous ne pouvons pas conclure sur la performance d'SVM, tant qu'une analyse plus approfondie de la paramétrisation adaptée à notre cas d'usage n'a pas été effectuée.

6.2 Détection d'anomalies

Pour évaluer les modèles SVM et Random Forest sur la détection d'anomalie, nous nous concentrerons principalement sur l'étude du Macro-F1-Score, de la precision et du recall de chaque classes, l'accuracy pouvant être à 0.8 facilement dans le cas d'une prédiction uniquement composées de `normal`, ce qui rendrait le detecteur parfaitement inefficace.

Pour SVM, on peut voir que l'algorithme à été capable de retrouver une grande partie des anomalies avec un recall à 0.99285, mais en échange d'une précision assez faible de 0.38562. Cela indique que l'on peut difficilement avoir une confiance aveugle dans ses prédiction, car il a tendance à déclarer trop rapidement une anomalie.

Pour Random Forest, on observe un recall beaucoup plus faible de 0.37821 avec une precision de 0.36018. On voit alors que notre modèle n'a pas été capable de retrouver les anomalies de manière fiable, tout en produisant des erreurs sur ses prédictions.

Nous pensons que cela est du principalement à un manque de données d'entrainement. La détection d'anomalie demande un nombre beaucoup plus important de données d'entrainement pour pouvoir être performante, et il nous aurait alors été possible de fusionner le dataset avec un autre dataset sain pour multiplier la quantité de données d'entrainement.

Dans le cadre d'un déploiement sur une machine réelles, il faudrait alors être particulièrement attentif à la période d'entrainement, pour s'assurer d'avoir une couverture complète du serveur dans une utilisation saine.

6.3 Détecteur

Les résultats du détecteur ont été mitigé. Dans un premier temps, il repose sur les algorithmes précédents, et donc de leurs défauts.

Le premier defaut que l'on remarque en testant l'application, est le nombre de logs normaux prédits comme des logs suspects ou dangeureux. On remarque également que

la forme de ceux-ci est toujours similaire, et cela nous pousse à penser à un manque de données d'entrainement.

Le second défaut que l'on remarque, est la difficulté pour l'applciation à détecter les logs dangeureux. L'hypothèse selon laquelle les logs dangereux ont une déviation plus importante que les logs suspects ne semble pas se vérifier en pratique, et pourrait donc expliquer les résultats précédents.

Pour améliorer cette application, nous envisageons plusieurs pistes.

La première est d'automatiser la détection des tresholds optimaux lors de la phase d'entraînement.

La seconde est de pouvoir utiliser des techniques d'apprentissage adverse ou supervisé, en se servant du retour de d'administrateur systeme. Lors de la présentation des résultats à la suite de l'analyse, l'administrateur pourrait confirmer manuellement le choix du modèle, ou bien le réfuter. On ajouterais alors ces logs avec le bon label au dataset d'entrainement, pour perfectionner ses résultats.

Enfin, on peut remettre en question la pertinence des techniques de détection d'anomalies dans ce cas d'usage. En effet, la classification a présenté des résutlats intéressants, et on gagnerait alors à implémenter ces algorithmes, pour réaliser une classification des logs. Cela suppose cependant de modifier le dataset d'entraînement.

Celui-ci étant supposé être consitité du fonctionnement nominal d'un serveur, on pourrait manuellement injecter des exemples de logs suspects et dangereux aux logs récupéré sur la machine. Bien que ceux-ci necessiterait un travaille de pré-traitement important pour être adapté aux ip et chemins nominaux de la machine, sela pourrait permettre d'accroître considérablement la précision des modèles et donc leurs capacités de détection.

7 Annexes

En plus de ce document, les annexes suivantes sont fournies :

- `log_anomaly_detection.ipynb` : notebook retracant nos travaux sur la partie detection de malware
- `log_classification.ipynb` : notebook retracant nos travaux sur la partie classification de malware
- `anomaly_detection.py` : application peremettant d'entrainer un modèle sur un fichier cible et de prédire les anomalies sur des nouveaux logs.

Cest ressources sont trouvables au lien suivant `https://github.com/NathSimon/projet_integrateur`.

References

[1] M. A. A. Hilmi, K. A. Cahyanto, and M. Mustamiin, "Apache web server - access log pre-processing for web intrusion detection," 2020.

[2] C. by LogPAI, "Loghub: A Large Collection of System Log Datasets for Log Analysis Research," Sept. 2021.

[3] F. Zaker, "Online Shopping Store - Web Server Logs," 2019.

[4] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," 2019.

[5] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 654–661, IEEE, 2016.

[6] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*, pp. 33–40, IEEE, 2017.

[7] "Pandas.Dataframe." `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`.

[8] "Pandas.factorize." `https://pandas.pydata.org/docs/reference/api/pandas.factorize.html`.

[9] "Scikit-learn countvectorizer." `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html`.

[10] "sklearn.LabelEncoder." `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html`.

[11] "Scikit-learn TrainTestSplit." `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.

[12] "Scikit-Learn PCA." `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`.

[13] "Kendall Rank Correlation Coefficient." `https://www.geeksforgeeks.org/python-kendall-rank-correlation-coefficient/`.

[14] "scipy.stats.kendalltau." `https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.kendalltau.html`.

[15] "Feature Selection using ANOVA." `https://www.kaggle.com/code/yoshifumimiya/feature-selection-using-anova`.

[16] "Scikit-Learn ANOVA." `https://scikit-learn.org/stable/auto_examples/svm/plot_svm_anova.html#sphx-glr-auto-examples-svm-plot-svm-anova-py`.

[17] "sklearn.featureselection.SelectKBest." https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html.

[18] "Scikit-learn SVM." https://scikit-learn.org/stable/modules/svm.html#svm.

[19] "sklearn.svm.OneClassSVM." https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[20] "Scikit-learn GradientBoostingClassifier." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[21] "Scikit-learn RandomForestClassifier." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[22] "sklearn.ensemble.IsolationForest." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html.

[23] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," 2020.

[24] G. Z. V. S. Min Du, Feifei Li, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," Nov. 2017.

[25] "Towards Data Science micro, macro weighted averages of f1 score, clearly explained." https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f.

[26] "Towards Data Science deep dive into multi-label classification..! (with detailed case study)." https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff.

[27] "Towards Data Science evaluating multi-label classifiers." https://towardsdatascience.com/evaluating-multi-label-classifiers-a31be83da6ea.

[28] "Scikit-learn multioutput module." https://scikit-learn.org/stable/modules/multiclass.html.