Projet de session

Rapport 2 : Traitement des données

Réalisé par:

El ASS Walid (2ème cycle)

DRIOUICH Meryam (2ème cycle)

MAAOU Marouane (2ème cycle)

Contents

1 Description des algorithmes implémentés **3**
1.1 Un contre Un (OneVsOneClassifier) . . . 3
1.2 Méthode par ensemble Adaboost . . . 3
1.3 RandomForestClassifier . . . 4

2 Tests **5**

3 Version finale **7**

4 Études de cas **9**
4.1 Les cas positifs . . . 9
4.2 Les cas négatifs . . . 9

5 Rétrospective **10**

1 Description des algorithmes implémentés

Pour ce projet, nous avons accès aux données de l'étude électorale canadienne produite par le Consortium de la démocratie électorale de plus de 37 822 réponses au sondage par des personnes. De nombreux paramètres furent à prendre en compte comme les différents problèmes (économique, sociale, etc..) des individus, les impressions sur les différents partis politiques et leurs chefs. Ces paramètres se combinent pour former un profil global qui rend potentiellement chaque parti à des caractéristiques uniques par rapport aux autres.

Nous avons testé principalement trois algorithmes de machine learning.

1.1 Un contre Un (OneVsOneClassifier)

One-vs-one divise notre ensemble de données de classification multi-classes à un problème de classification binaire. Cette approche divise l'ensemble de données en un ensemble de données pour chaque classe par rapport à toutes les autres classes. Dans notre cas, on a 8 classes ('Liberal Party', 'Conservative Party', 'Bloc Qu<e9>b<e9>cois', 'Green Party', 'ndp', 'People's Party', 'Don't know/ Prefer not to answer', 'Another party (please specify)') cela pourrait être divisé en 28 (k_classes*(k_classes-1)/2) ensembles de données de classification binaire. Chaque modèle de classification binaire peut prédire une étiquette de classe. Et le modèle ayant le plus de prédictions ou de votes est prédit par la stratégie un contre un. Nous avons utilisé *support vector machines* (SVM) comme classificateur binaire avec le kernel 'poly' qui est plus avancé que le noyau linéaire et qui utilise les équations non linéaires pour les entrées de transition afin de présenter l'espace avec un temps computationnel optimal par rapport au kernel 'rbf'. On prévoyait avoir un score plus élevé avec notre jeu de données en utilisant ce modèle de prédiction, mais on a obtenu un score de 73.3% (moins que le score prévu). Ce résultat est probablement dû aux données ayant trop d'erreurs.

```
Test Set Accuracy : 73.30142063001853 %


Classification Report :
```

	precision	recall	f1-score	support
Another party (please specify)	0.10	0.21	0.13	38
Bloc Qu<e9>b<e9>cois	0.77	0.72	0.74	292
Conservative Party	0.84	0.91	0.87	1903
Don't know/ Prefer not to answer	0.51	0.36	0.42	768
Green Party	0.68	0.42	0.52	496
Liberal Party	0.76	0.85	0.81	1942
People's Party	0.43	0.60	0.50	116
ndp	0.67	0.63	0.65	921
accuracy			0.73	6476
macro avg	0.60	0.59	0.58	6476
weighted avg	0.73	0.73	0.72	6476

Figure 1: Rapport de classification de l'algorithme OneVsOneClassifier

Ce rapport de classification guide notre évaluation de nos modèles à choisir F1 score, car on a une distribution de classe déséquilibrée comme illustré dans la figure 7. L'exactitude fonctionne mieux si les faux positifs et les faux négatifs ont un coût similaire.

1.2 Méthode par ensemble Adaboost

AdaBoost signifie Adaptive Boosting, adaptant le boosting dynamique à un ensemble de modèles afin de minimiser l'erreur commise par les modèles individuels. Comme on a l'hypothèse qu'il y a des erreurs potentielles dans notre ensemble de données, utiliser un seul classificateur n'est pas une bonne idée. En effet, on a pensé à utiliser l'approche Adaboost qui combine plusieurs modèles pour produire des résultats plus précis et cela se fait en autorisant à plusieurs apprenants faibles (weak learner) à apprendre sur les données d'entrainement. Puis, en combinant ces modèles pour générer un méta-modèle qui vise à résoudre les erreurs telles qu'elles sont effectuées par les apprenants faibles individuels.

Nous avons essayé de classifier les données à l'aide d'un arbre de décision *DecisionTreeClassifier* avec une profondeur maximale de 3 pour éviter le surapprentissage. Considérant un modèle composé d'un intervalle de 1 à 50 arbres de décision, et avec un taux d'apprentissage de 0.01, on voit dans la figure 2 les scores sur

l'ensemble de données d'entrainement et de test selon le nombre des arbres de décision.

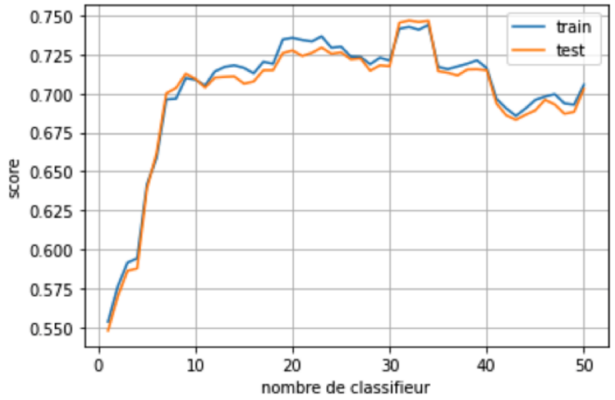


Figure 2: Score de Adaboost en fonction de nombre des classifieurs

On peut remarquer que le nombre des classificateurs optimaux est 35, quand on dépasse ce nombre le modèle performe moins.

Pour trouver le nombre optimal d'apprenants faibles à nos données d'entrainement, on a effectué une recherche par grille avec la méthode *GridSearchCV* implémenter dans la bibliothèque *sklearn* qui implémente la validation croisée sur une grille de paramètres. On a choisi un intervalle de 10 à 300 d'apprenants, ainsi on a effectué une recherche du *learning_rate* parce qu'il affecte la contribution de chaque apprenant. Les paramètres optimaux obtenus sont 44 apprenants et 0.1 comme learning rate. Avec ces paramètres, on a pu avoir un score de 77.03% (+3,73% du classifieur *OneVsOne*).

1.3 RandomForestClassifier

Comme son nom l'indique, ce classifieur se compose d'un grand nombre d'arbres de décision individuels qui fonctionnent comme un ensemble. Chaque arbre individuel dans la forêt aléatoire donne une prédiction de classe et la classe avec le plus de votes devient la prédiction de notre modèle. Ce modèle est bon pour notre jeu de donnée. En effet, en échantillonnant nos attributs, on évite le fléau de dimensionnalité de nos données. Ainsi, les arbres faiblement corrélés se protègent mutuellement de leurs erreurs individuelles (tant qu'ils ne se trompent pas constamment tous dans le même sens) et peuvent produire des prédictions d'ensemble plus précises que leur prédiction individuelle. Alors que certains arbres peuvent avoir de mal prédictions, de nombreux autres arbres auront raison, de sorte qu'en tant que groupe, les arbres sont capables de se déplacer dans la bonne direction.

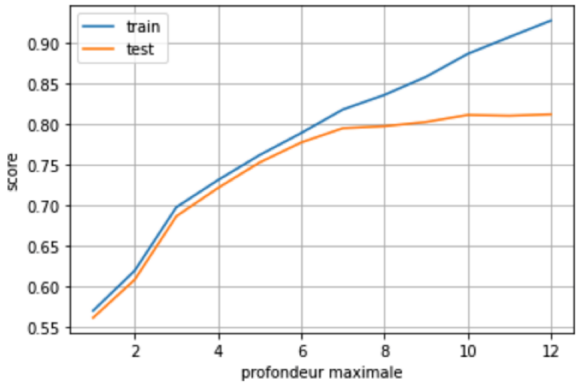


Figure 3: score de l'algorithme RandomForestClassifier en fonction de la profondeur maximale

Nous entrainons nos données d'entrainement avec *RandomForestClassifier* avec 50 arbres et sur un intervalle de 1 à 12 comme profondeur maximale des arbres. Comme illustré dans la figure, le modèle **surapprend**

après une profondeur maximale de 7. Avec 50 estimateurs (optimiser par une recherche en grille) et une profondeur de 6, on a pu avoir un score de 78.53% (+5,23% du classifieur *OneVsOne*).

2 Tests

Les hyperparamètres des modèles utilisés sont optimisés par une recherche de grille à validation croisée sur une grille de paramètres. Nous répartissons notre jeu de données étiquetées en 70% (données d'entrainement) et 30% (données de test), nous appliquerons une validation croisée à **5 folds** sur nos modèles d'apprentissage afin d'éviter le surapprentissage et avoir une bonne estimation du modèle sur de nouvelles données. La figure ci-dessous illustre les scores des folds de chaque modèle.

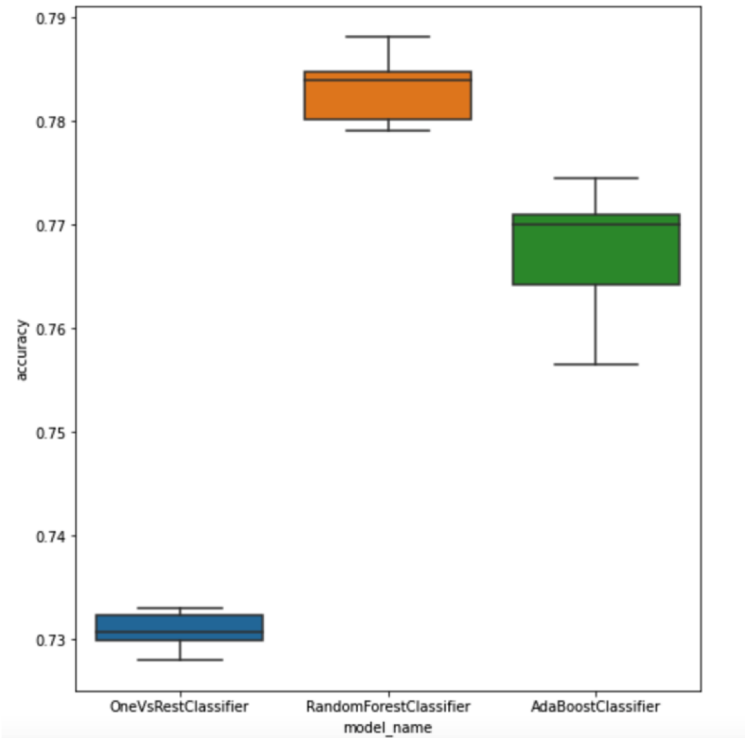


Figure 4: box plot des scores des folds de chaque algorithme

	classifier	train_score	test_score	train_time
1	Random Forrest	0.830265	0.785361	0.417754
2	Adaboost	0.761505	0.770383	2.427978
0	OnevsRestClassifier	0.732742	0.720970	39.153892

Figure 5: Performance de chaque algorithme

Selon les deux figures ci-dessus, on remarque qu'en termes de performance, le classificateur *'Random forest'* est le plus performent avec un temps d'exécution optimal, mais l'exactitude n'est pas une grande mesure de rendement des classificateurs lorsque les classes sont déséquilibrées. Nous avons besoin de plus d'information pour comprendre à quel point le modèle s'est bien comporté. A-t-il donné de bons résultats pour chaque classe ?

Une bien meilleure façon d'évaluer les performances d'un classifieur est d'examiner la matrice de confusion. Elle compare les données réelles pour une variable cible à celles prédites par le modèle. Elle permet de connaître d'une part les différentes erreurs commises par un algorithme de prédiction, mais plus important encore, de connaître les différents types d'erreurs commis.

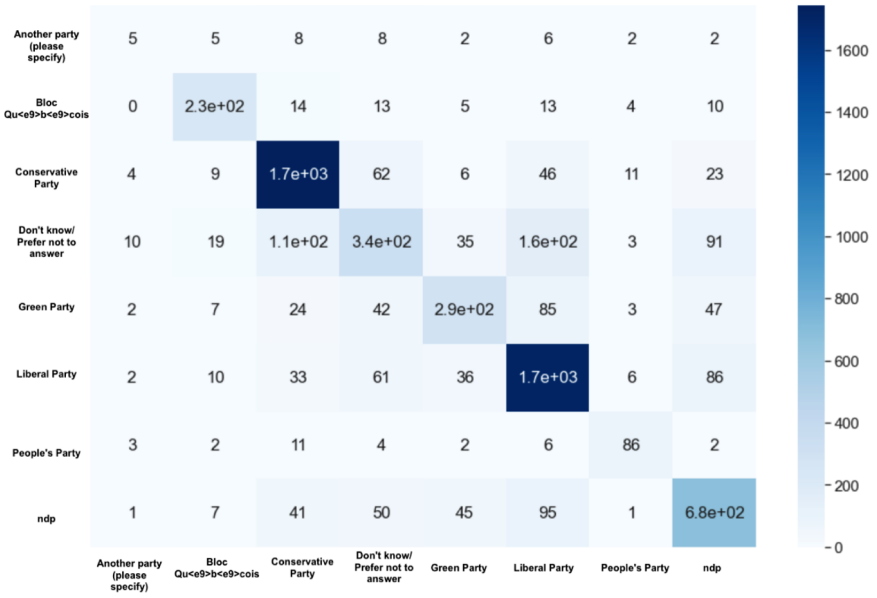


Figure 6: Matrice de confusion des données test avec RandomForrestClassfier

Afin d'analyser notre matrice, on présente dans le tableau suivant les différentes métriques (precision, recall et f1-score).

	Precision	Recall	F1_score
Another Party (please specify)	0.19	0.13	0.15
Bloc Qu<e9>b<e9>cois	0.80	0.80	0.80
Conservative party	0.88	0.92	0.90
Don't know/ Prefer not to answer	0.59	0.44	0.50
Green Party	0.69	0.58	0.63
Liberal Party	0.80	0.88	0.84
People's Party	0.74	0.74	0.74
Ndp	0.72	0.74	0.73

On remarque que les métriques des différents partis sont généralement bonnes, surtout 'Liberal Party' et 'Conservative Party', alors que les classes 'Another party' et 'Don't know/ Prefer not to answer' sont moins bonnes. Nous pouvons imaginer que le débalancement entre les classes a probablement effet sur ces résultats, surtout que 'Liberal Party' et 'Conservative Party' ont une grande proportion par rapport aux autres classes comme illustrer dans la figure.

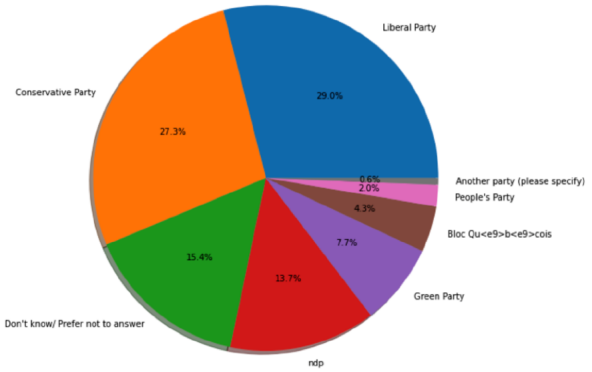


Figure 7: Distribution des partis fédéraux

Pour cela et avec la bibliothèque **imblearn**, nous avons essayé de faire un oversampling aux classes minoritaires, mais ceci a donnée des résultats moins bons, puis nous avons pensé à utiliser un ré-échantillonnage

hybride en ajoutant des échantillons aux classes minoritaires et supprimant des échantillons aux classes majoritaires. Ceci a pu améliorer un peu les métriques de la classe 'Another party' cependant cela n'était pas suffisant. À ce moment, on a pensé de combiner les attributs que l'utilisateur note chaque chef d'un parti politique selon son intelligence (cps19_lead_int), cares (cps19_lead_cares), trustworthy (cps19_lead_trust), leadership (cps19_lead_strong) en attributs qui représentent les choix de réponses des participants. Cela nous a permis d'une part de réduire la dimensionnalité de 32 à 8, d'autre part de nous donner une idée sur quel parti le participant peut voter. Par exemple, si l'attribut 'Justin Trudeau' dans un objet a la plus grande note parmi les autres attributs, il y a une grande probabilité que ce participant va voter sur 'Liberal Party'. Dans le pseudocode ci-dessous, on peut remarquer que 3988 participants parmi 5373 dont l'attribut 'Justin Trudeau' a une note supérieure à 3 ont voté sur 'Liberal Party'.

```
Entrée [309]: data_train[data_train['Justin Trudeau'] >= 3]['cps19_votechoice'].value_counts()

Out[309]: Liberal Party                        3988
          ndp                                   440
          Don't know/ Prefer not to answer      351
          Conservative Party                    228
          Green Party                           225
          Bloc Qu<e9>b<e9>cois                   88
          People's Party                         45
          Another party (please specify)          8
          Name: cps19_votechoice, dtype: int64
```

Figure 8: Distribution des participants qui ont donné une note supérieure à 3 à Justin Trudeau

Cela nous a permis un peu plus d'améliorer les métriques de la classe 'Another Party' avec une précision de 0.22, un recall de 0.47 et f1-score de 0.30. Il n'y avait pas un changement sur les métriques des autres classes. Pour la classe 'Don't know/ Prefer not to answer', on peut imaginer que même on a un bon pourcentage d'échantillons pour cette classe ou même si un participant a donné une bonne note à un ou plusieurs chefs fédéraux, l'incertitude humaine l'a poussé de ne pas avoir une décision finale (surtout une certitude entre 'Liberal Party' et 'Conservative Party'), c'est ce qui a augmenté le nombre des faux négatifs dans cette classe (faible recall).

3 Version finale

Les attributs qu'on a choisi pour notre système sont : cps19_imp_iss_party, cps19_cand_rating, cps19_win_local, cps19_party_rating, cps19_lead_rating, cps19_lead_trust, cps19_lead_int, cps19_lead_cares, cps19_lead_strong, cps19_province.

Pour les attributs cps19_cand_rating, cps19_win_local, cps19_party_rating, cps19_lead_rating, les participants donnent une note de 0 à 100 aux partis politiques ou aux chefs fédéraux. On a pensé de fixer une note qui sera un seuil pour choisir le parti ou le chef fédéral pour lequel le participant peut voter. Si la note n'était pas supérieure à ce seuil, on peut supposer que le participant préfère un autre parti ou chef fédéral et qui ne figure pas dans les choix de réponse du sondage ('Another party'). Cette idée vient de l'observation que par exemple dans l'attribut cps19_party_rating, on a parmi 4436 participants qui ont donné une note supérieure ou égale à 80 à Conservative Party (cps19_party_rating_24), 3845 ont voté sur Conservative Party. C'est le même cas pour les autres attributs.

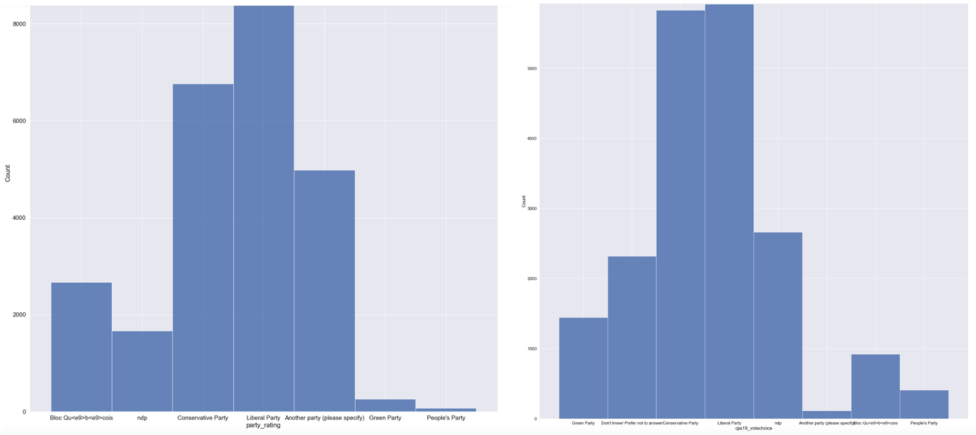
```
Entrée [373]: data_train[data_train['cps19_party_rating_24'] >= 80]['cps19_votechoice'].value_counts()

Out[373]: Conservative Party                   3845
          Liberal Party                         193
          Don't know/ Prefer not to answer      129
          People's Party                         83
          ndp                                    78
          Green Party                            61
          Bloc Qu<e9>b<e9>cois                   39
          Another party (please specify)          8
          Name: cps19_votechoice, dtype: int64
```

Figure 9: Distribution des participants qui ont donnée une note supérieur à 80 au parti conservatif

Cette approche nous a permis de réduire la dimensionnalité de nos données, mais le problème est le choix

du seuil de la note. En effet, si on a choisi un seuil trop élevé, l'attribut sera biaisé à choisir l'option 'Another party' ce qui va nous donner de mauvaises prédictions. D'autre côté, un seuil trop bas va donner un attribut avec beaucoup d'erreurs ainsi l'option 'Another party' sera dominé par les autres options. La solution que nous avons choisie est de fixer un seuil qui mène à une distribution des nouveaux attributs des partis (ou des chefs) similaire à celle de l'attribut cible 'cps19_votechoice'. (Seuil = 70)



Nous entrainons nos données d'entrainement avec *RandomForestClassifier* avec 50 arbres et une profondeur maximale de 7. La figure ci-dessous la valeur prédictive de chaque attribut.

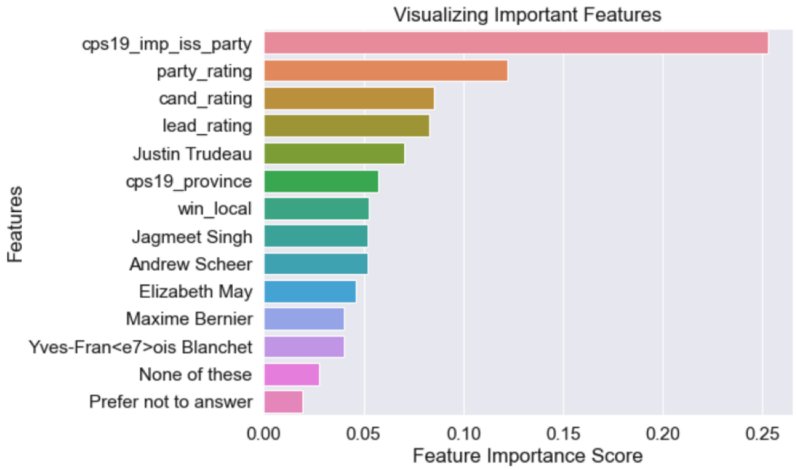


Figure 10: Distribution de nos attributs selon leur valeur prédictive

On remarque que l'attribut cps19_imp_iss_party a la plus grande valeur prédictive, ce résultat est tout à fait logique, car le participant va probablement voter sur le parti politique qui pense va résoudre son enjeu. En effet, les participants ont exprimé les enjeux les plus importants selon eux dans cette élection fédérale (l'attribut cps19_imp_iss). À la phase prétraitement de données, nous avons catégorisé ces enjeux en 8 catégories principales (économique, santé, environnement, sociale, politique, immigration et minorité, éducation, crime et justice) et on a observé que les enjeux 'économique', 'santé', 'environnement' sont les majoritaires. Ainsi, selon l'attribut cps19_imp_iss_party, 'Liberal Party' et 'Conservative Party' sont les plus choisis par les participants pour les traiter et ce qui explique la dominance de ces deux partis par rapport aux autres partis dans notre ensemble de données.

Le deuxième attribut important est party_rating. Les participants donnent une note à chaque parti politique et on peut imaginer qu'ils vont donner une bonne note au parti qu'ils pensent vont voter. Comme décrit précédemment et à partir de ces notes, on a prédit le parti que les participants favorisent, le problème avec cette approche est qu'un participant peut donner une note de 80 par exemple à deux ou plus partis ce qui peut avoir un effet négatif sur le parti qui favorise et le parti sur lequel a voté, c'est ce qui peut expliquer sa mauvaise valeur prédictive par rapport à l'attribut cps19_imp_iss_party. (Même remarque pour les attributs cand_rating, trust_rating, win_local)

4 Études de cas

4.1 Les cas positifs

9_province	cps19_imp_iss_party	Justin Trudeau	Andrew Scheer	party_rating	lead_rating	win_local	cand_rating	Jagmeet Singh	Yves-Fran<e7>ois Blanchet	Elizabeth May	Maxime Bernier	None of these	Prefer not to answer
Ontario	Conservative Party	0	4	Conservative Party	Andrew Scheer	Conservative Party	Conservative candidate	0	0	0	0	0	0

Dans cet exemple, le participant choisit 'Conservative Party' comme parti qui croie qu'il peut résoudre son enjeu et il est aussi son favorable. Aussi, le total de notes est donné à 'Andrew Scheer' qui est le chef fédéral du parti conservatif. Ce participant a choisi de voter sur 'Conservative Party' et ce qui était prédit par notre système.

province	cps19_imp_iss_party	Justin Trudeau	Andrew Scheer	party_rating	lead_rating	win_local	cand_rating	Jagmeet Singh	Yves-Fran<e7>ois Blanchet	Elizabeth May	Maxime Bernier	None of these	Prefer not to answer
Quebec	Bloc Qu<e9>b<e9>cois	0	0	Bloc Qu<e9>b<e9>cois	Yves-François Blanchet	Liberal Party	Liberal candidate	2	4	0	0	0	0

Dans cet exemple, le participant choisit 'Bloc Qu<e9>b<e9>cois' comme parti qui croie qu'il peut résoudre son enjeu et il est aussi son favorable. Aussi, le total de notes est donné à 'Yves-Frab< $e7$ >ois Blancher' qui est le chef fédéral du parti bloc québécois, mais il pense que 'Liberal Party' a le plus de chances de gagner le siège, aussi il favorise le candidat libéral. Ce participant a choisi de voter sur 'Bloc Qu<e9>b<e9>cois' et ce qui était prédit par notre modèle, ce qui explique que les attributs cps19_imp_iss_party et party_rating ont une valeur prédite plus que win_local.

4.2 Les cas négatifs

ps19_province	cps19_imp_iss_party	Justin Trudeau	Andrew Scheer	party_rating	lead_rating	win_local	cand_rating	Jagmeet Singh	Yves-Fran<e7>ois Blanchet	Elizabeth May	Maxime Bernier	None of these	Prefer not to answer
ritish Columbia	Don't know/ Prefer not to answer	0	0	Another party (please specify)	Another candidate (please specify)	Liberal Party	Liberal candidate	0	0	0	0	3	1

Dans cet exemple, le participant préfère de ne pas répondre à quel parti peut résoudre son enjeu ainsi qu'il a choisi autre parti que ceux présenter dans le sondage pour son parti et chef favorable. Notre modèle a prédit 'Don't know/ Prefer not to answer' alors que le répondant a choisi 'Liberal Party'.

9_province	cps19_imp_iss_party	Justin Trudeau	Andrew Scheer	party_rating	lead_rating	win_local	cand_rating	Jagmeet Singh	Yves-Fran<e7>ois Blanchet	Elizabeth May	Maxime Bernier	None of these	Prefer not to answer
Ontario	Don't know/ Prefer not to answer	0	0	Conservative Party	Another candidate (please specify)	Conservative Party	Liberal candidate	0	0	0	0	1	3

Dans cet exemple, le participant fait un multiple choix dans les différents attributs, par exemple, il préfère 'Conservative Party' alors que son candidat favori est 'Liberal candidate' ainsi il ne préfère pas répondre à quel parti capable de résoudre son enjeu, ce qui est fort probable que notre modèle engendre une fausse prédiction. Ici le répondant a choisi de voté sur 'ndp' alors que nous avons prédit 'Conservative Party'.

En guise de conclusion, la performance de notre modèle varie selon la précision des réponses du participant. En effet, quand le participant hésite entre plusieurs partis (ou chefs des partis) le modèle sera incapable de faire une bonne prédiction, ce qui explique le faible score (f1-score) des classes 'Don't know/ Prefer not to answer', 'Another Party (please specify)'.

5 Rétrospective

Notre projet s'est basé sur nos prédictions du premier rapport. Les attributs qu'on a choisis ont donné des résultats acceptables. On a pu obtenir un modèle capable de prédire le vote d'un participant à partir ces attributs avec une précision de 79% obtenue par *'Random Forest'*.

En effet, les attributs cps19_imp_iss_party et l'attribut party_rating semblent être judicieux vu qu'ils jouent un rôle important dans la prédiction du score.

Au fil de projet, l'équipe a fait face à certaines difficultés. Notamment, le prétraitement des attributs était une étape délicate vue la complexité des données. Il y avait des champs manquants dans des attributs qui ont une signification. Cela nous a demandé du temps pour bien comprendre les données et faire un prétraitement adéquat.

Pour limiter les pertes d'informations, on a effectué plusieurs essais et géré les exceptions afin de former une base de données constituée des attributs significatifs choisis préalablement. En effet, on a extrait les valeurs des champs ayant le moins de valeurs nulles, ainsi que les champs que nous avons prévus qu'ils seront significatifs pour notre modèle. On a aussi reformulé totalement certains champs en fixant un seuil ou en transformant des attributs textuels en attributs catégoriels, au but de simplifier nos données pour augmenter la performance du modèle. Finalement, le choix est fait de telle manière à avoir des valeurs d'attribut qui ne biaisent pas le modèle et faire un ré-échantillonnage des données pour équilibrer les classes prédites, dans l'objectif d'avoir une version finale du modèle performant dans toutes autres échantillons de données.

Notre modèle performait bien dans certains cas, mais en contrepartie, sa performance diminue dans d'autres. Comme vu dans l'étude de cas, le modèle devient incapable de faire une bonne prédiction quand le participant n'est pas sûr de ces choix, ce qui peut montrer la difficulté de prédire la pensée humaine.

Si le projet était à refaire, nous aurions mis plus de temps sur le prétraitement des attributs, et essayer de procéder d'une autre manière, choisir d'autres attributs, tester et faire une comparaison. Un immense dataset donne plusieurs chemins d'y procéder et qui demande plus de temps pour conclure quel prétraitement qui sera le plus optimal, quels sont les attributs les plus significatifs et quel modèle est le plus adéquat.