1. Modélisation
   1. Méthodologie
      1. Phases
         1. Phase A : travail sur neuf sous-tableaux

A l’issue de la première phase de notre travail, nous avons construit neuf sous tableaux différents, chacun correspondant à un ensemble de données non mesurée par les stations. Cela répondait à une problématique de gestion des valeurs manquantes, où nous préférerions ne pas biaiser notre jeu de données initiales. Par exemple, la station Adélaïde ne mesure pas la couverture nuageuse : ainsi, mettre une valeur dans cette colonne nous paraissait peu pertinent.

Voici un résumé visuel de l’ensemble des étapes de cette première phase de préparation des données :

* + - * + Illustrer préparation des données (data\_prep\_pipeline.png)

1.1.1.2 Phase B : retour à un tableau unique

L’objectif pour la phase de modélisation était ensuite de se répartir le travail à trois, chacun d’entre-nous ayant en charge la modélisation de trois tableaux. Il s’est avéré que cette méthodologie était peu fructueuse pour plusieurs raisons :

1. Chercher le bon modèle pour chaque tableau prend déjà beaucoup de temps. Le chercher pour trois en prend encore plus, car nous voulons choisir le modèle le plus adapté à chaque tableau.
2. Notre groupe est passé de trois personnes à deux personnes en cours de travail, ce qui alourdit le travail pour les deux personnes restantes.
3. Un des neuf tableaux mesure l’intégralité des grandeurs, et nous avons analysé quelles grandeurs sont les plus importantes pour le modèle. Parmi ces grandeurs, on retrouve l’humidité, la pression, la couverture nuageuse, la vitesse du vent et parfois la température. Les tableaux qui ne mesurent pas ces grandeurs donnent des résultats médiocres.

Compte tenu de toutes ces contraintes, nous avons changé de méthode en cours de route, et sommes partis sur la gestion d’un tableau entier dont nous allons rapidement vous présenter les choix de production.

Nous avons choisi, comme précédemment, de supprimer les valeurs manquantes des colonnes RainToday et RainTomorrow. Cela règle le problème de la colonne RainFall.

Pour les autres valeurs manquantes, nous avons choisi de les remplacer par la valeur moyenne sur l’ensemble des tableaux pour les variables numériques. Nous avons fait de même avec les variables catégorielles, en remplaçant les valeurs manquantes par la modernité la plus fréquente.

Nous avons ajouté au tableau d’autres données, notamment les coordonnées gps des stations météo (latitude et longitude), leur région d’appartenance, la liste concaténée des grandeurs non mesurées, et trois colonnes : une pour jour, une pour le mois, et une pour l’année.

Enfin, nous avons convertit toutes les grandeurs catégorielles en variable numériques. Nous disposons à présent d’un tableau complet prêt à être modéliser.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

* + - 1. Phase C : choix de métrique et optimisation

Notre travail consiste en une classification binaire visant à prédire, à partir des données journalières, s’il pleuvra le lendemain ou non. En l’absence de cahier des charges qui pourrait correspondre à des souhaits particuliers de clients, nous avons dû faire un choix de métrique, tant pour l’optimisation des modèles que pour leur évaluation. Notre jeu de données étant déséquilibré, se baser sur une simple accuracy nous a initialement paru peu pertinent. Ainsi, nous avons choisi d’entraîner nos modèles sur le f1 score moyen.

Après analyse une certaine quantité de matrices de confusion, nous nous sommes rendus compte que ce qui importait le plus pour nous était la précision des prédictions des classes positives et négatives. Nous avons donc fixé un seuil de 83% pour ces deux classes, ce qui correspond à cinq prédictions juste sur six. Comme nous le verrons, cela conduit parfois avoir des rappels sur la classe positive quelque peu décevants ce qui sera un critère supplémentaire pour choisir un modèle.

Notre jeu de données initiale est déséquilibré : il fait beau dans 78 % des cas, et il pleut les 22% restants. Il nous a paru important d’en tenir compte, en employant deux méthodes :

1. La première consiste à s’assurer que les proportions des deux classes soient identiques dans le jeu d’entraînement et de test. Pour cela, nous indiquons dans le train\_test\_split l’argument stratify = y.
2. La deuxième vise à ré échantillonner les données, soit par under sampling, soit par over sampling. Nous avons choisi de tester les deux méthodes afin de voir leur impact sur la modélisation et de choisir la meilleure des deux. Nous partons sur SMOTE pour l’oversampling, et sur ClusterCentroid pour l’undersampling.

Nous avons aussi normalisé nos données en utilisant la fonction StandardScaler.

Au final, le bilan est mitigé, certains modèles fonctionnant mieux sans ré-échantillonage (les times series et les réseaux de neurones notamment).

* + 1. Approches
       1. Classification : algorithmes classiques

Notre première approche a consisté à réutiliser les premiers algorithmes de classification que nous avons étudié en cours. Nous les avons classés en fonction de leur interprétabilité, du plus simple au plus complexe, à savoir :

* + - * + LogisticRegression
        + DecisionTreeClassifier
        + RandomForestClassifier
        + KNeighborsClassifier
        + SVC

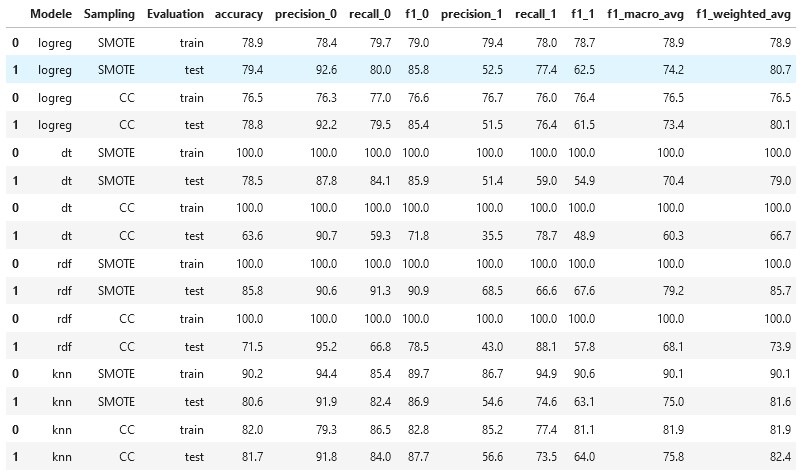
Premier essai :

Nous avons structuré notre premier travail autour de la régression logistique.

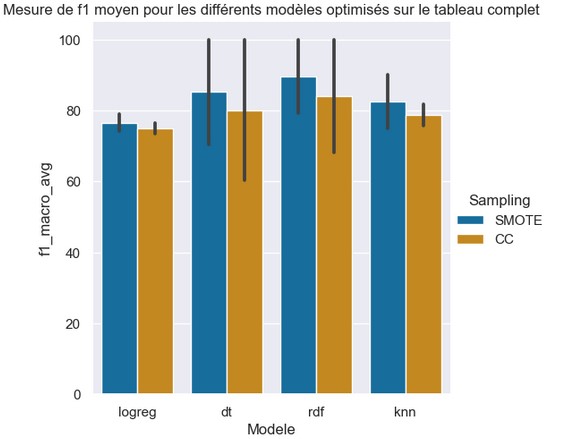
Nous avons appliqué un GridSearchCV pour trouver les meilleurs paramètres de ce modèle. Une fois la sauvegarde du GridSearchCV effectuée, nous l’utilisons pour créer le meilleur modèle de régression logistique, qui sera entraîné deux fois : une première fois sur les données sur échantillonnées par SMOTE, une deuxième fois sur les données sous échantillonnées par ClusterCentroid. Ces deux modèles seront ensuite sauvegardés afin d’éviter d’avoir à les refaire tourner. Nous évaluons ensuite les performances des modèle grâce à une matrice de confusion et grâce à un classification report, sur les ensembles d’entraînement et de tests. L’objectif ici étant de choisir la meilleure méthode de ré-échantillonnage, et de détecter un éventuel sur-apprentissage.

Deuxième essai :

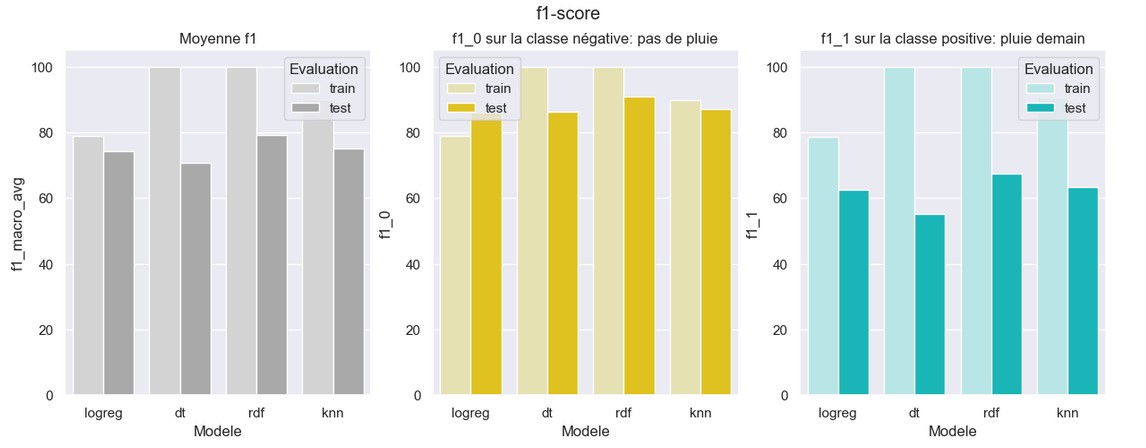
Une fois cette méthode élaborée pour un modèle, il est aisé d’automatiser la procédure pour les quatre autres modèles, afin de créer un tableau regroupant tous les résultats de tous les modèles entraînés pour chaque méthode de ré échantillonnage. Nous obtenons ainsi le tableau suivant :



Remarque : le modèle svm a été étudié sur les petits tableaux, et ne donne généralement pas de meilleurs résultats que les autres. Nous avons estimé qu’appliquer cette méthode sur le tableau entier prendrait 8h de temps de calcul, aussi avons-nous choisi d’exclure svm de l’analyse sur le tableau entier.



Résultats :

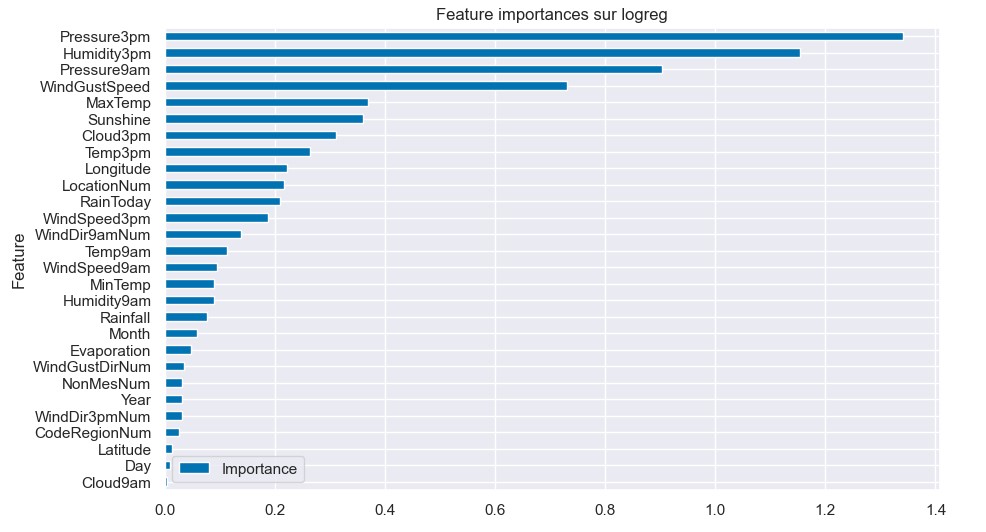


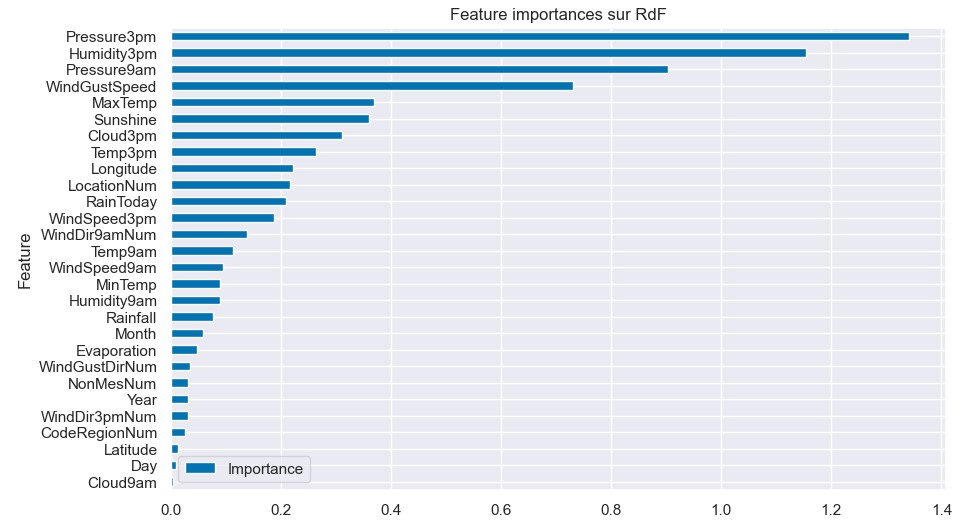
Il apparaît de cette analyse que les meilleurs modèles sont la régression logistique et les forêts aléatoires, avec le ré échantillonnage SMOTE. Pour autant, ils ne donnent pas le résultat attendu de 83 % de précision sur les deux classes. On constate un certain déséquilibre : la classe positive (il pleut demain) à une précision modeste, et aux alentours de 67 %.

Nous constatons aussi un fort sur-apprentissage des modèles, le modèle des forêts aléatoires allant même jusqu’à obtenir un score de 100 % sur le jeu d’entraînement.

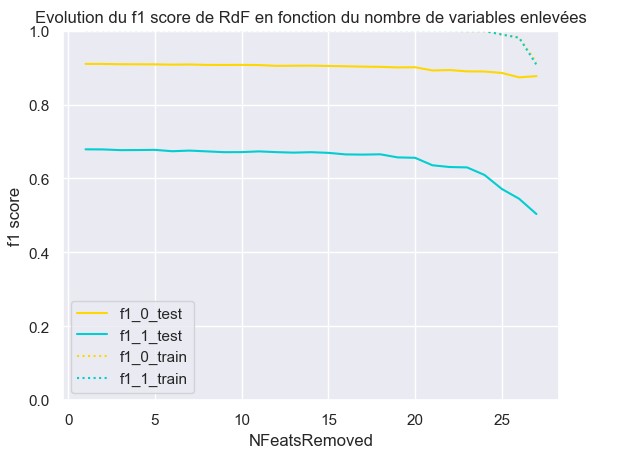
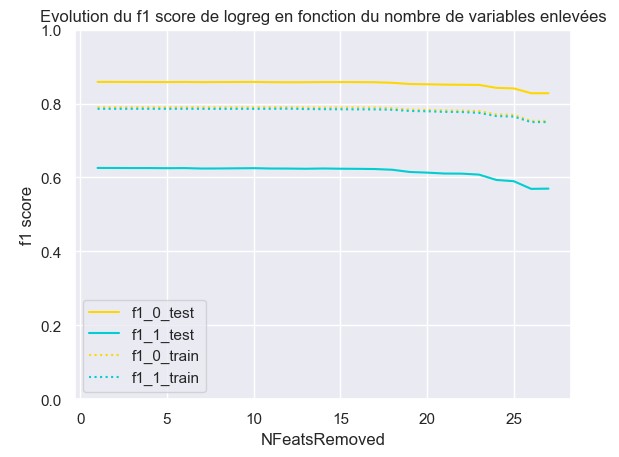
Nous pensons à ce stade que réduire le sur-apprentissage permettrait sans doute d’améliorer les prédictions du modèle sur la classe positive. Notre prochaine étape consiste à analyser Pour grandeurs le modèle considère comme pertinentes, afin d’effectuer une sélection sur les variables explicatives. Notre hypothèse est qu’enlever les variables les moins pertinentes permettrait aux modèles de gagner en précision dans une certaine mesure jusqu’à un optimum au-delà duquel continuer à enlever des variables conduirait à une baisse significative de la précision. C’est ce que nous analyseront dans la partie suivante.

Nous nous focalisant maintenant sur le modèle de régression logistique et sur le modèle des forêts aléatoires. Pour chacun des deux modèles, nous affichons la liste des variables explicatives, de la plus importante à la moins importante.





Nous construisons ensuite un programme qui va enlever les variables explicatives une par une, de la moins importante à la plus importante. A chaque étape, les modèles sont entraînés sur les variables restantes, et nous affichons le f1-score moyen sur le jeu de tests et le jeu d’entraînement. Nous obtenons les graphiques suivants :



Notre hypothèse de départ est invalidée : de manière très surprenante, nous constatons que le score est quasiment constant, jusqu’à subir une baisse brutale lorsque nous enlevons les variables explicatives les plus importantes. De plus, cette sélection progressive ne réduit pas le sur-apprentissage. Il nous faut donc une autre approche afin de régler ce problème.

* + - 1. Méthodes d’ensemble

Une autre façon de réduire le sur-apprentissage consiste à utiliser un ensemble de classifieurs faibles et à agréger leurs résultats. Nous étudierons ainsi par la suite des techniques de Voting, Stacking et XGBoost.

Voting :

Nous utilisons les quatre modèles optimisés précédents, à savoir la régression logistique, les arbres de décision, les forêts aléatoires et la méthode des plus proches voisins. Deux choix s’offrent ensuite à nous : le voting ‘hard’ où chaque classifieur vote pour une classe et c’est la majorité qui l’emporte, ou le voting ‘soft’, où chaque classifieur rend les probabilités d’appartenance à chaque classe, et le voting choisit celui qui renvoie la plus forte probabilité : c’est celui-là qui déterminera la classe d’appartenance finale.

Stacking :

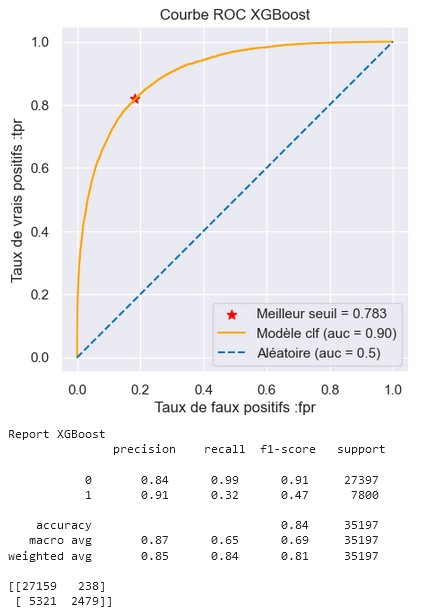
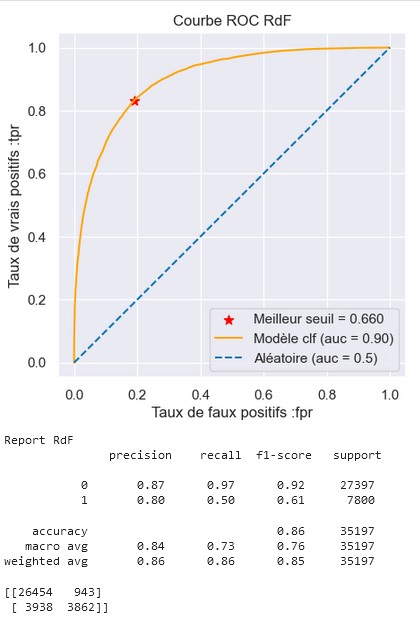
C’est la même idée que la méthode précédente, mais on envoie en sortie un autre classifieur. Nous avons choisi la régression logistique comme estimateur final.

XGBoost :

Nous avons aussi essayé d’implémenter l’algorithme XGBoost, mais sa complexité en termes d’hyper paramètres et le peu de temps à notre disposition ne nous ont pas permis d’optimiser pleinement les fonctionnalités de cet algorithme. Pourtant comme on va le voir, les résultats sont plutôt satisfaisants.

Ajustement des prédictions par recherche du meilleur seuil :

Malgré toutes ces recherches, un déséquilibre persiste entre les classes positives et négatives. N’ayant pas pu réduire le sur-apprentissage autant que nous l’espérions, nous choisissons de jouer sur le seuil (qui par défaut vaut 0,5) permettant d’affecter un résultat à une classe donnée. Pour cela nous utilisant la méthode ROC pour chacun de nos classifieurs, et cherchons le seuil optimal, c’est-à-dire celui qui se rapproche le plus du point de coordonnées (0, 1) dans le diagramme. Ensuite, nous implémentons ce seuil aux classifieurs déjà obtenu. Nous illustrons les résultats sur deux de nos meilleurs modèles : Les forêts aléatoires et XGBoost .



A ce stade, trois modèles retiennent désormais notre attention :

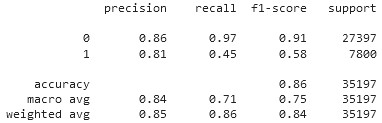


Figure : Résultats du Voting Hard

1. Les forêts aléatoires optimisées et seuillées
2. Le Voting Hard avec les modèles optimisés
3. XGBoost seuillé et légèrement optimisé
   * + 1. Apprentissage profond
          - Réseaux de neurones profonds
       2. Bonus : régression linéaire
   1. Résultats
      1. Décrire scores.xlsx (+ illustrer ?)
   2. Discussion
      1. Interprétation des résultats
         * + Les rapporter à leurs jeux de données respectifs (9df vs 1df)
           + Évoquer les effets des optimisations
           + Indiquer que nous avons décidé de retenir 1 algorithme par famille d’approches (RandomForestClassifier, TimeSeriesForestClassifier, VotingClassifier (hard), réseaux de neurones profonds)
           + Pourquoi ces algorithmes semblent-ils mieux adaptés à notre jeu de données que les autres ?
      2. Remise en contexte
         * + Quels sont les algorithmes d’apprentissage automatique / profond qui sont réellement utilisés dans le domaine de la prévision météorologique ?
           + Sont-ils les mêmes que ceux que nous avons retenus ?
           + Quel rôle joue l’interprétabilité des algorithmes / modèles dans le choix de ces derniers dans le domaine de la prévision météorologique ?
   3. Perspectives
      1. Preprocessing

L’approche consistant à remplacer les valeurs manquantes pour les grandeurs qui ne sont jamais mesurée par certaines stations par la valeur moyenne de ces grandeurs sur le reste du tableau nous paraît quelque peu brutale.

Nous aimerions pouvoir gérer ça plus finement, et deux pistes s’offrent à nous :

* La première concerne des grandeurs dont la mesure s’interrompt au cours de la période étudiée. Pour ses valeurs manquantes, il pourrait être envisagé de les remplacer en faisant de l’interpolation temporelle, à partir des données déjà présentes.
* La deuxième concerne les grandeurs non mesurées. Nous avons réalisé quelques modèles simples de régression linéaire sur les variables numériques, comme la pression, l’humidité où la température. Pour certaines d’entre elles, les résultats sont très bons, et on pourrait envisager entraîner un modèle par grandeur, et les utiliser pour prévoir les valeurs manquantes, afin d’avoir un tableau avec des variables explicatives de meilleure qualité. Ce travail semble très long, puisqu’il implique de rechercher un modèle par grandeur, et de ce que les optimiser. Une piste serait de travailler sur un tableau où nous supprimerions brutalement toutes les valeurs manquantes pour entrainer les modèles.
  + - * + Supprimer des données, voire des variables
    1. Feature engineering
    2. Optimisations
    3. Régression linéaire