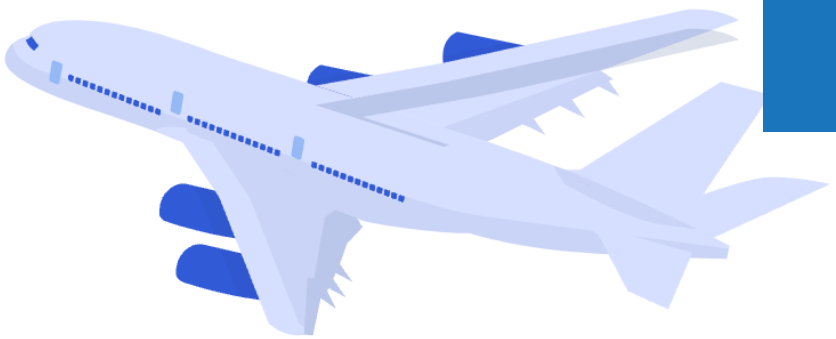
Dauphine | PSL
UNIVERSITÉ PARIS

Exécutive Master Intelligence Artificielle et Science des données Promo 5

SCALABLE DATA MINING POUR LES RETARDS D'AVION



Samy BEJI
Siham EL GHAZI
Benjamin HIVERT
Johann MAUCHAND

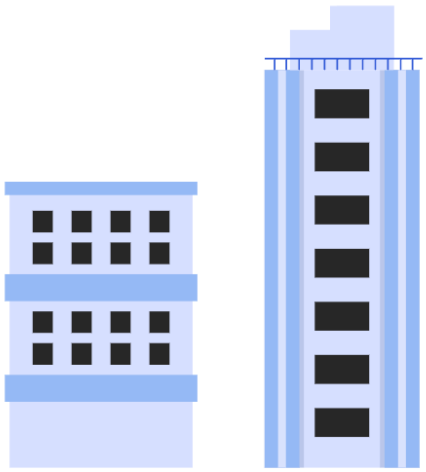
Décembre 2024

Projet Flight

Dans un monde où le transport aérien joue un rôle clé dans l'économie mondiale, la gestion des retards de vols représente un enjeu majeur pour les compagnies aériennes, les aéroports, et les passagers. Ces retards, souvent perçus comme des inconvénients mineurs, engendrent des coûts considérables et une frustration croissante, affectant l'expérience client et l'efficacité opérationnelle. Ce projet s'inscrit dans cette problématique avec pour objectif principal de prédire les retards de vols en s'appuyant sur des données de vols et des informations météorologiques.

Pour ce faire, une analyse approfondie a été réalisée à partir de deux jeux de données couvrant une période significative de deux ans : les données de vols et les données météorologiques aux États-Unis. Le projet a nécessité une série d'étapes de retraitement des données, incluant le nettoyage, le formatage, et la jointure des tables pour créer des ensembles cohérents et exploitables. Les choix méthodologiques ont été orientés par le besoin de garantir une compatibilité entre les différents formats, de minimiser le biais introduit par les données manquantes, et de maximiser la fiabilité des modèles prédictifs.

L'un des aspects innovants de cette étude réside dans l'application de techniques d'optimisation des jointures et l'exploitation de modèles d'apprentissage automatique, tels que le **Random Forest** et le **LightGBM**, pour améliorer les prédictions. Ce rapport présente une vue d'ensemble des méthodologies employées, des difficultés rencontrées, et des solutions apportées pour surmonter les défis techniques. Il explore également l'impact potentiel de ces prédictions dans un contexte réel, en tenant compte des besoins des compagnies aériennes et des passagers.



SOMMAIRE

I. DATASETS UTILISES DANS L'ANALYSE

1. DESCRIPTION DES DONNEES

Les données sont réparties en deux jeux différents :

- **Les données relatives aux données de vol** regroupant l'ensemble des vols sur la période allant du 1er janvier 2012 au 31 décembre 2014. Ce jeu de données comprend notamment les informations concernant l'aéroport de départ, l'aéroport d'arrivée, le temps de vol prévu, l'heure de départ, l'heure d'arrivée, le retard potentiel. L'ensemble de ces données représentent environ 1,26 Go de données.
- **Les données relatives à la météo** : Ces fichiers au format texte inclus l'ensemble des données de météo aux Etats-Unis sur la période allant du 1er janvier 2012 au 30 novembre 2013 pour environ 3,1 Go de données.

L'ensemble de ces données ont été chargées dans des dataframes, un pour les données de vols et l'autre pour les données de météo.

La suite de l'analyse a été réalisée sur des datasets représentant 0,1% des données pour faciliter l'exécution du programme en local. Ces datasets ont été ensuite enregistrés au format parquet par années et par dates.

```
// Sample creation
if (sample) {
  logger.info("createParquetFile: Sample creation")
  val withReplacement = false // Without replacement
  val fraction = 0.001
  val seed = 42 // Seed for reproducibility
  flight = flight.sample(withReplacement, fraction, seed)
  weather = weather.sample(withReplacement, fraction, seed)
}
```

Il a ensuite été procédé à un premier filtre pour homogénéiser les périodes. En effet, il n'est pas nécessaire de conserver, pour l'analyse ultérieure, des données de vol ne correspondant à aucune donnée météo.

Finalement, les données étudiées peuvent être résumées dans le tableau ci-dessous :

1ère date	Dernière date	Nbs lignes	dataset
01/01/2012	31/12/2014	18476	flight_initial
01/01/2012	30/11/2013	24533	weather_initial
01/01/2012	30/11/2013	12163	flight_filtered
01/01/2012	30/11/2013	24533	weather_filtered

2. CONTENU DES DONNEES

Les données prises en compte permettent d'étudier 12 163 vols sur les années 2012 et 2013. Ces données semblent assez cohérentes et représentatives puisque 80% des vols sont à l'heure pour un pourcentage de vol annulé d'environ 1,36%.

Année	nbs de vol	% On-time	% Delayed	% Cancelled	% Diverted
2013	5907	79.36	18.93	1.53	0.16
2012	6256	81.25	17.28	1.19	0.27

Si l'on s'intéresse plus précisément aux retards de vols sur les deux années précitées, on constate que les retards de vols ne sont pas principalement dû aux conditions météo ; ce type de retard ne représentant que 4% des retards. 21% des retards sont dû au système NAS.

Année	% Weather_delay	% Nas_delay	% Other_delay
2013	4.39	22.93	72.67
2012	4.12	20.97	74.90

II. PIPELINE DE RETRAITEMENT DES DONNEES ET DE JOINTURE DES TABLES

Cette partie du document décrit les retraitements mis en place pour nettoyer les données et améliorer leur qualité.

En préambule, il convient de préciser qu'une première analyse des données a été réalisée à l'aide de la bibliothèque ***Deequ*** afin d'avoir une analyse complète et automatisé des données. Cette première analyse a permis d'identifier les points importants dans le retraitement des données et l'ensemble des corrections à faire par la suite dans le programme.

1. RETRAITEMENT DES DONNEES

Une première étape a consisté à nettoyer et formatter les données proposées afin de générer deux tables :

- La table *« FT Table »*, contenant les données de vols et
- La table *« OT Table »* contenant l'ensemble des données météorologiques.

Opération de création de la table FT Table

La fonction mise en place permet de nettoyer les données de vol. Les opérations réalisées sont :

- Formatage des heures et des dates
- Ajout d'un traitement pour tenir compte des décalages horaires
- Traitement des valeurs manquantes de certaines colonnes
- Filtrage des vols annulés ou détournés ; l'objectif étant de conserver uniquement les données de vol retardé en raison de la météo et les vols à l'heure.
- Identifier chaque colonne de dataframe (via l'ajout d'un préfixe *« FT_ »* sur chaque colonne.

L'ensemble de ces retraitements se trouvent dans la fonction *"createFlightTable"* de la bibliothèque *"Restatement"* du projet.

Opération de création de la table OT Table

Cette fonction a pour objectif de créer une table des données météorologiques propres. Les opérations réalisées sont les suivantes :

- Jointure avec la table « *WBAN* »[1] afin d'identifier et ne conserver que les données météorologiques relatives à un aéroport[2].
- Conversion de certaines colonnes en format numérique
- Complétion des valeurs manquantes sur ces colonnes via un imputeur permettant de remplacer les valeurs manquantes par la moyenne des observations d'une colonne[3].
- Correction des valeurs erronées marquées par des flags « s » (ce qui signifie que les valeurs sont suspectes[4]) en les remplaçant soit par la moyenne (pour les colonnes numériques), soit par la valeur la plus fréquente (pour les colonnes non numériques).
- Identifier chaque colonne de dataframe (via l'ajout d'un préfixe « *OT_* » sur chaque colonne.

L'ensemble de ces retraitements se trouvent dans la fonction *"CreateFlightTable"* de la bibliothèque *"Restatement"* du projet.

2. JOINTURE DES TABLES

Les tables ont été ensuite jointe suivant plusieurs modalités. En effet, il convient de rappeler que pour chaque vol doit être associé une donnée météo à l'heure du départ puis toutes les heures jusqu'à 12h en amont pour les aéroports de départ mais également à l'heure d'arrivée moins 12h en amont pour les aéroports d'arrivées.

Commenté [JM1]: Il faut preciser que pour avoir l'heure d'arrivee reelle on a du se baser sur WBAN afin de traiter l'heure comme heure de depart+trajet+jet lag (sinon on serait en heures locales)

En ce qui concerne l'heure d'arrivée réelle, le fichier WBAN a permis de tenir compte des heures locales d'arrivée. Ainsi, l'heure d'arrivée se définit comme l'heure de départ à laquelle on ajoute le temps de trajet. Puis on applique, le bon fuseau horaire pour ne pas rester en heure locale d'arrivée, ce qui fausserait l'analyse.

Deux jointures ont été mise en place :

- **Une jointure par colonne :** Cette option a consisté à rajouter, dans la table des vols, autant de colonne correspondant aux heures de départ jusqu'à 12h en amont et autant de colonnes pour les aéroports d'arrivée suivant les mêmes modalités. Ensuite à chacune de ces nouvelles colonnes, il a été joint la donnée météo correspondante de la table des données météorologiques.
- **Une jointure par ligne :** Chaque ligne du dataframe de vol a été dupliquée 24 fois pour correspondre aux heures des aéroports de départ et d'arrivée. A chaque ligne a été ajoutée une donnée météo correspondante.

[1] Cette table contient les données d'aéroports

[2] On exclut les autres données car elles ne servent pas à la prédiction des retards

[3] Seules les colonnes ayant moins de 20% de données manquantes ont été corrigées

[4] Ces informations sont contenues dans les colonnes « flag ». Une fois les retraitements effectués, les colonnes flag sont supprimées

3. OPTIMISATION DES JOINTURES

La première jointure réalisée concerne la table météo avec le fichier WBAN. Cette première jointure a été optimisée en réalisant une opération de broadcast. En effet, cette table étant relativement petite, elle peut être copié sur l'ensemble des nœuds du cluster puis être jointe à la table météo. La table étant copiée, cela évite au moment de la jointure de déplacer physiquement les fichiers et donc de gagner en efficacité.

Cette opération est réalisée au moment de la création de la table "OT_Table" car cela permet également d'éliminer toutes les données de météo qui ne sont pas reliées à un aéroport.

```
newdf = newdf.join(broadcast(wban_df), Seq("WBAN"), "inner")
```

Concernant les autres opérations de jointure, elles se trouvent toute dans la libraire *"JoinOperations"*.

Afin d'optimiser les jointures, les deux datasets ont été réorganisés par date sur chaque nœud afin d'éviter les mouvements de données au moment des jointures proprement dite :

```
// Sort datetime in ascending order
val df_weather_sorted =
OT_weather.sort(asc("OT_WEATHER_TIMESTAMP")).repartitionByRang
e(numPartitions, col("OT_WEATHER_TIMESTAMP"))
val df_flights_sorted =
FT_flights.sort(asc("FT_TIMESTAMP")).repartitionByRange(numPar
titions, col("FT_TIMESTAMP"))
```

La création des colonnes supplémentaires et la jointure proprement dite s'en trouvent optimisées ainsi que le montre le tableau ci-dessous des temps d'exécution des différentes étapes du programme.

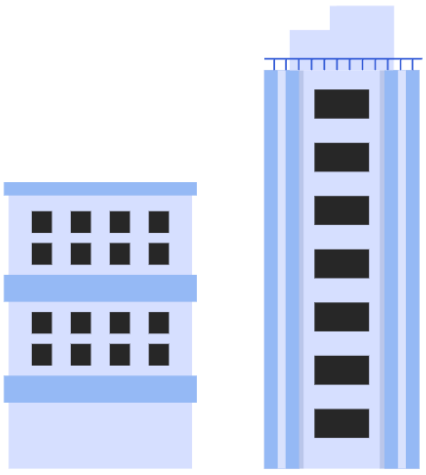
4. CONCLUSION

En conclusion, il apparaît que les différents traitements évoqués sont réalisés en 8,43 minutes sur un Mac M1 et sur 0,1% des données. On constate que les taches d'écriture sont les plus couteuses en termes de temps (création des fichiers en début et fin de traitement). En revanche, les taches de jointures sont relativement rapides alors qu'elles sont normalement couteuses en temps de calcul.

Task	Duration (seconds)
create_parquet	100.12
read_parquet	14.35

Projet FLIGHT

Create_tables	25.68
Join_tables_first_step	0.16
Join_tables_second_step_column	7.18
Join_tables_second_step_line	1.66
Store_parquet_FinalDf	357.187



III. FILTRES DES DONNEES

La fonction associée génère un dataset de vols classifiés en vols *"retardés"* et *"à l'heure"* selon des seuils de retard. Il a été testé un seuil de 15 min et un seuil de 30 minutes.

Quatre jeux de dataset sont créés : DS1, DS2, DS3 et DS4. Ils incluent tous les vols dont le retard à l'arrivée est supérieur ou égale à un seuil spécifié.

- **DS1** contient les vols où le retard total à l'arrivée est dû à la somme des retards liés à la météo et des retards dû au contrôle aérien :

```
FT_ARR_DELAY_NEW >= in_DelayedThreshold et FT_ARR_DELAY_NEW ==
(FT_WEATHER_DELAY + FT_NAS_DELAY)
```

- **DS2** contient tous les vols où le retard à l'arrivée est causé par la météo ou où le retard de contrôle aérien est au moins égal au seuil de retard spécifié :

```
FT_ARR_DELAY_NEW >= in_DelayedThreshold et (FT_WEATHER_DELAY >
0 || FT_NAS_DELAY >= in_DelayedThreshold)
```

- **DS3** contient les vols où le retard à l'arrivée est supérieur ou égal au seuil spécifié et où la somme des retards météorologiques et de contrôle aérien est supérieure à zéro :

```
FT_ARR_DELAY_NEW >= in_DelayedThreshold et (FT_WEATHER_DELAY +
FT_NAS_DELAY > 0)
```

- DS4 contient tous ls vols ayant un retard à l'arrivée supérieur ou égal au seuil spécifié, sans autre condition :

```
FT_ARR_DELAY_NEW >= in_DelayedThreshold
```

Les vols sont ensuite divisés aléatoirement en ensembles d'entrainement et de test selon une répartition de 75% pour les jeux d'entrainement et 25% pour les jeux de test. L'ensemble des jeux de données sont ensuite enregistrés au format parquet avant d'être utilisé dans le pipeline de l'apprentissage.

Cette phase d'enregistrement en parquet des fichiers était initialement conçue pour une utilisation sur le cluster. En effet, il était prévu de réaliser plusieurs scénarios incluant notamment des seuils de retard différents (15 minutes puis 30 minutes). Cette phase générait de nombreuses données puisqu'il était combiné un seuil avec quatre datasets différents (DS1 à DS4). Ainsi, pour la stabilité du programme, il était intéressant de faire une étape de sauvegarde à ce moment.

IV. AXES D'AMELIORATION DU POUVOIR PREDICTIF DE L'ALGORITHME

Plusieurs questions ont émergé durant l'analyse.

1. FIXATION DU SEUIL DE RETARD

La première d'entre elles est la question de la fixation du seuil de retard[5]. En effet, est ce que ce seuil est le meilleur pour définitif la ponctualité d'un vol ? Autrement dit, cela fait-il sens de prendre en considération le même seuil pour spécifier si un vol est à l'heure ou en retard ?

En effet, si l'on considère deux vols, dont les aéroports de départ et d'arrivée sont les mêmes, avec des conditions météorologiques équivalentes et que le seuil de retard est fixé à une heure, un vol avec 59 min de retard serait catégorisé « *On-Time* ». Cela induit que l'algorithme de prédiction le prendra comme tel et donc apprendra sur cette base en ajustant les caractéristiques de chaque vol.

On pourrait également s'interroger sur la fiabilité du relevé des données et la présence de bruit non pertinent qui pourrait compromettre l'apprentissage.

Dans ce cas, il pourrait être pertinent de définir un seuil pour fixer le caractère à l'heure ou non d'un vol comme cela est fait pour les vols retardés.

Ce point a été traité dans le cadre de l'analyse et au moment de de la génération des différents datasets en préalable à la mise en place de l'algorithme de prédictions.

Il est à noter toutefois que les auteurs de l'étude initiale n'ont peut-être pas aperçu ce biais du fait du caractère déséquilibré des échantillons *OnTime/Delayed* : en effet ayant parfois plus de 80% de vols « *OnTime* », les auteurs ont réalisé des échantillonnages aléatoires, pouvant ainsi court-circuiter le problème évoqué.

2. PROBLEMATIQUE A FIXER PREPRODUCTION

La deuxième question liée à l'analyse est liée à la possibilité d'une application en temps réel de la prédiction. En effet, l'intérêt pour les compagnies aériennes comme pour les clients est très important. Ainsi, en ayant calibré le modèle, peut-on se baser sur les données temps réels afin de prédire le délai ou non des vols à l'arrivée. A première vue, il est possible de penser que c'est effectivement possible.

Cependant, à la lecture de l'étude, un point étonnant apparait dans la construction des caractéristiques climatiques et la jointure de ces dernières aux horaires des vols,

[5] Fixer à 15 minutes dans l'étude initiale

empêchant toute mise en production future, ou en tout cas, sans avoir conscience du biais introduit lors de l'apprentissage.

En effet, il est mentionné que, lors de la jointure, les données climatiques sélectionnées par rapport aux horaires de départs et d'arrivées des vols sont celles les plus proches de ces derniers, et donc potentiellement collectées après ces horaires (maximum 29 min). L'apprentissage de l'algorithme est ainsi légèrement biaisé, ayant une potentielle connaissance de données futures.

Certes, pour l'apprentissage et prédiction, les plages horaires des relevés peuvent aller jusqu'à douze heures. Cependant, comme le mentionne les auteurs ou comme nous pouvons le voir sur les chiffres, l'apport de la première donnée météorologique (i.e. la plus proche) est celle qui contribue le plus dans la prédiction. Par ailleurs, les auteurs soulignent l'amélioration limitée lorsque l'on considère plus de trois heures de d'observations, même si la contribution existe.

Il parait donc intéressant, si ce n'est nécessaire, de prendre en considération cette remarque à des fins de mise en production.

Pour résoudre cette problématique, deux approches potentielles existent :

1. Utiliser des prédictions de données climatiques pré-horaires
2. Prendre les données les plus proches mais antérieures.

Pour des raisons pratiques, il a été privilégié la deuxième solution ; la première se révélant être elle-même une vraie problématique de recherche nécessitant beaucoup plus de temps de recherche.

Ainsi, nous avons donc laissé la possibilité de paramétrer, lors de la jointure, le choix *'closest'*, comme dans l'article, ou *'less'* pour régler le problème cité.

Dans ce cadre, il aurait été intéressant de saisir l'impact d'une telle contribution. Bien que nous puissions penser que celle-ci n'aurait pas d'apport significatif (et ce du fait de la régularité du climat, des relevés…), il est important de noter son intérêt principal : **la confiance** que l'on donnerait plus raisonnablement par la suite au modèle calibré pour une mise en production temps réel.

3. ERREUR ALGORITHMIQUE

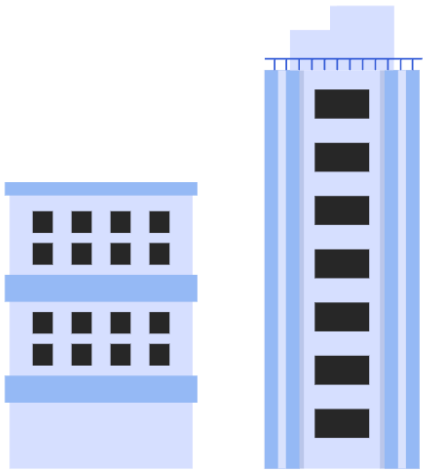
Dans ce cas, le cas '*else'* traite des observations de vols auxquelles il doit être ôté douze heures. Or, il faut considérer le cas possible où des données climatiques de la veille seraient nécessaire. Pour cela, nous avons remplacé dans notre algorithme plusHours() et plusDays() par minHours() et minDays(), rendant ainsi le code compatible avec l'algorithme sous-jacent.

ALGORITHM 1: MapReduce Pseudocode for the First Join Step

```
Map(K: null, V: a tuple from a split of either OT or FT)
if V is a tuple from OT then
    join_key ← ⟨V.A, Date(V.t)⟩
    table_tag ← "OT"
    tagged_tuple ← add a tag "OT" to V
    composite_key ← ⟨join_key, table_tag⟩
    emit(composite_key, tagged_tuple)
else
    join_key ← ⟨V.A_a, Date(V.t_ad)⟩
    table_tag ← "FT"
    tagged_tuple ← add a tag "FT" to V
    composite_key ← ⟨join_key, table_tag⟩
    emit(composite_key, tagged_tuple)
    if Date(V.t_ad).plusHours(12) is Date(V.t_ad).plusDays(1) then
        join_key ← ⟨V.A_a, Date(V.t_ad).plusDays(1)⟩
        composite_key ← ⟨join_key, table_tag⟩
        emit(composite_key, tagged_tuple)
    end
end

Partition(K': a composite key)
hashcode ← hash_function(K'.join_key)
return hashcode mod #reducers

Reduce(K': a composite key, LIST_V': a list of tagged tuples for K' first from OT then FT)
create an array of observations A_O ordered by time
create a temporary array of observations A_T
for each OT tuple o in LIST_V' do
    put o in A_O
end
for each FT tuple f in LIST_V' do
    A_T ← get_hourly_observations(A_O, f.t_ad)
    emit(null, merge(f, A_T))
end
```



V. ALGORITHMES DE PREVISIONS DES RETARDS DE VOLS

1. MISE EN ŒUVRE DU RANDOM FORREST

La deuxième partie du programme est consacré à l'utilisation du jeu de données pour l'apprentissage d'un algorithme de prévisions des retards de vols et plus précisément l'utilisation d'un algorithme de Random Forrest.

Le modèle a appris sur le jeu de données en colonnes et uniquement, à ce stade, sur le dataset « *DS1* » c'est-à-dire le dataset contenant les vols où le retard total à l'arrivée est dû à la somme des retards liés à la météo et des retards dû au contrôle aérien. Le seuil de retard testé est de 15 minutes[6].

L'algorithme a été entrainé avec des paramètres relativement petits pour tenir compte d'une exécution en local qui même avec ces paramètres[7] ne permet pas d'obtenir des résultats probants.

2. LIGHTGBM

L'algorithme *LightGBM* a été choisi en raison de ses performances supérieures dans le traitement de grands ensembles de données, comme l'a démontré son succès dans de nombreux concours Kaggle.

En outre, de nombreux articles (Grinsztajn, Oyallon, & Varoquaux, 2022) abondent dans ce sens, documentant l'apport supérieur des modèles arborescents comme le *gradient boosting*. Ces derniers étant de meilleur choix pour les ensembles de données tabulaires, ce qui est le cas présent.

Par ailleurs, d'autres modèles type *XGBoost* ou *CATBoost* auraient également pu être étudiés. Cependant, le choix d'un modèle alternatif s'est porté vers le modèle LightGBM pour des raisons de temps de calcul.

Pour résumer, *LightGBM* a été choisi à titre de comparaison au *Random Forest* du fait des critères suivants :

- **Performance et efficacité :** LightGBM est réputé pour son efficacité dans le traitement de grands ensembles de données et sa rapidité d'apprentissage.
- **Succès dans les compétitions :** LightGBM gagne régulièrement ou se classe bien dans de nombreuses compétitions Kaggle, ce qui démontre sa robustesse et ses

[6] Voir la partie « *Axes d'amélioration* »

[7] Nombre d'arbres : 20, profondeur : 5 et nombre de plis : 3

performances supérieures à celles d'autres modèles d'apprentissage automatique.

- **Adaptation aux données tabulaires :** De nombreux articles académiques et implémentations pratiques ont démontré la supériorité des modèles basés sur les arbres pour les ensembles de données tabulaires.
- **Traitement de grandes quantités de caractéristiques :** LightGBM est conçu pour traiter un grand nombre de caractéristiques et peut gérer efficacement des données à haute dimension, ce qui est intéressant si ce n'est crucial dans le cadre de notre étude.

Cette approche s'appuie sur la capacité à traiter de vastes ensembles de données et à identifier des relations complexes et non linéaires entre diverses caractéristiques afin d'établir des prévisions précises sur les retards des vols.

VI. DIFFICULTES RENCONTREES AVEC LE CLUSTER

1. PROBLEMES D'ACCES INITIAL ET D'ADMINISTRATION

Retard dans l'Attribution des Clés

La première difficulté majeure a été le délai dans l'attribution des clés d'accès au cluster. Ce retard a eu plusieurs conséquences :

- Report du début de la phase de développement sur le cluster
- Compression du planning initial
- Nécessité de réaliser des tests préliminaires en local

Problèmes de Reconnaissance sur le Cluster

Une fois les clés obtenues, nous avons fait face à des problèmes d'authentification :

- Le cluster ne reconnaissait pas certains utilisateurs malgré des credentials valides
- Incohérences dans les droits d'accès
- Nécessité de réinitialisation multiple des authentifications

Plusieurs jours ont été nécessaire pour permettre le bon accès au cluster.

Problèmes de configuration et d'accès

Durant la phase de déploiement de l'application sur le cluster Spark, nous avons rencontré plusieurs obstacles techniques qui ont nécessité une attention particulière. Le principal défi concernait la configuration correcte des chemins d'accès et la gestion des ressources du cluster.

En effet, l'application a été développé avec la bibliothèque « *log4j* » permettant de gérer de façon efficace l'ensemble des données de configurations de l'application dont notamment l'ensemble des chemins vers les fichiers d'entrée et de sortie. L'adaptation de cette logique au cluser a entrainé la perte de plusieurs jours pour bien comprendre le fonctionnement du serveur et s'adapter en conséquence.

Par ailleurs, l'utilisation de cette librairie a également nécessité d'utiliser la librairie « *typesafe* ». Or, il semble que cette dernière n'est pas disponible sur le serveur. Il a donc fallu faire en sorte de l'embarquer directement dans le fichier JAR de l'application en changeant la méthode de création de l'archive JAR[8]. Cette opération a nécessité de nouvelles configurations.

[8] Utilisation de la méthode « assembly «

Enfin, plusieurs solutions ont été testé pour finalement simplifier le code afin de le rendre fonctionnel pour une utilisation sur le cluster. Là, encore, détricoter le code et le rendre fonctionnel a nécessité plusieurs jours.

2. PROBLEMES DE GESTION DES DONNEES HDFS

Difficultés de chargement des données

La gestion des fichiers sur HDFS s'est révélée problématique car il a été rencontré à la fois des erreurs fréquentes lors du téléchargement des fichiers avec des temps de transfert parfois très importants.

Il a également été constaté des problèmes de permissions lors des transferts des données.

Problèmes de reconnaissance des liens

Il a également été constaté des difficultés avec les liens des fichiers :

- Liens symboliques non reconnus correctement
- Problèmes de résolution des chemins absolus vs relatifs
- Incohérences dans la reconnaissance des chemins entre différents nœuds du cluster

Erreurs de chemin d'accès

L'une des premières difficultés rencontrées était liée à l'accès aux fichiers dans HDFS. L'erreur suivante était régulièrement observée :

PATH_NOT_FOUND: Path does not exist:
hdfs://10.40.178.80:9000/students/execiasd5_2024/bhivert/data/airport/*.csv

Cette erreur indiquait un problème de configuration dans le chemin d'accès aux données, malgré la présence effective des fichiers dans le système HDFS.

3. PROBLEMES DE CONFIGURATION

Changements de Configuration Inattendus

Les modifications de configuration ont posé plusieurs défis :

- Pertes inattendues des paramètres configurés
- Nécessité de reconfigurer régulièrement certains paramètres
- Incohérences entre les configurations des différents nœuds

Configuration du script de lancement

La résolution de ce problème a nécessité plusieurs ajustements dans le script spark-run.sh :

- Correction du format du chemin d'accès
- Suppression des lignes de configuration redondantes ou conflictuelles
- Optimisation de la structure du script pour une meilleure lisibilité et maintenance

4. CONFLITS DE VERSIONS ET COMPATIBILITE

Problèmes de version de Java

- Conflits entre différentes versions de JDK installées
- Incompatibilités entre les versions requises par Spark et nos modèles
- Nécessité de reconfigurer les variables d'environnement JAVA_HOME

Optimisation des ressources

À la suite de la résolution des problèmes d'accès, l'attention s'est portée sur l'optimisation des performances de l'application. Les paramètres de configuration suivants ont été implémentés pour maximiser l'utilisation des ressources du cluster :

```
--executor-cores 15 # Nombre de cœurs par exécuteur
--executor-memory 32G # Mémoire allouée par exécuteur
--num-executors 4 # Nombre total d'exécuteurs
```

- Ces paramètres ont été choisis pour :
- Assurer une distribution efficace des tâches sur le cluster
- Optimiser l'utilisation de la mémoire disponible
- Maintenir un équilibre entre parallélisme et surcharge du système

VII. CONCLUSION

Le projet a mis en lumière la complexité et les opportunités liées à la prédiction des retards de vols. À travers une analyse systématique des données de vols et météorologiques, les résultats obtenus illustrent la capacité des algorithmes d'apprentissage automatique à identifier des schémas significatifs et à fournir des prédictions fiables.

Les défis rencontrés au cours de ce projet, notamment liés à la gestion des volumes de données, aux limites du cluster utilisé et aux problèmes de configuration, ont permis de renforcer la robustesse des pipelines de traitement.

Cependant, certaines limites persistent, notamment en ce qui concerne l'utilisation de données météorologiques futures lors de l'apprentissage, introduisant un biais subtil mais notable. Ce point aurait pu être partiellement corrigé en privilégiant l'utilisation de données antérieures pour une mise en production fiable.

En conclusion, ce projet ouvre la voie à de nombreuses perspectives, telles que l'intégration de prédictions météorologiques en temps réel ou l'application de ces modèles à d'autres secteurs du transport. L'ensemble de ces contributions représente une avancée significative vers une gestion plus efficace et préventive des retards dans le secteur aérien.

References

Grinsztajn, Oyallon, & Varoquaux. (2022). Why do tree-based models still outperform deep learning on tabular data?