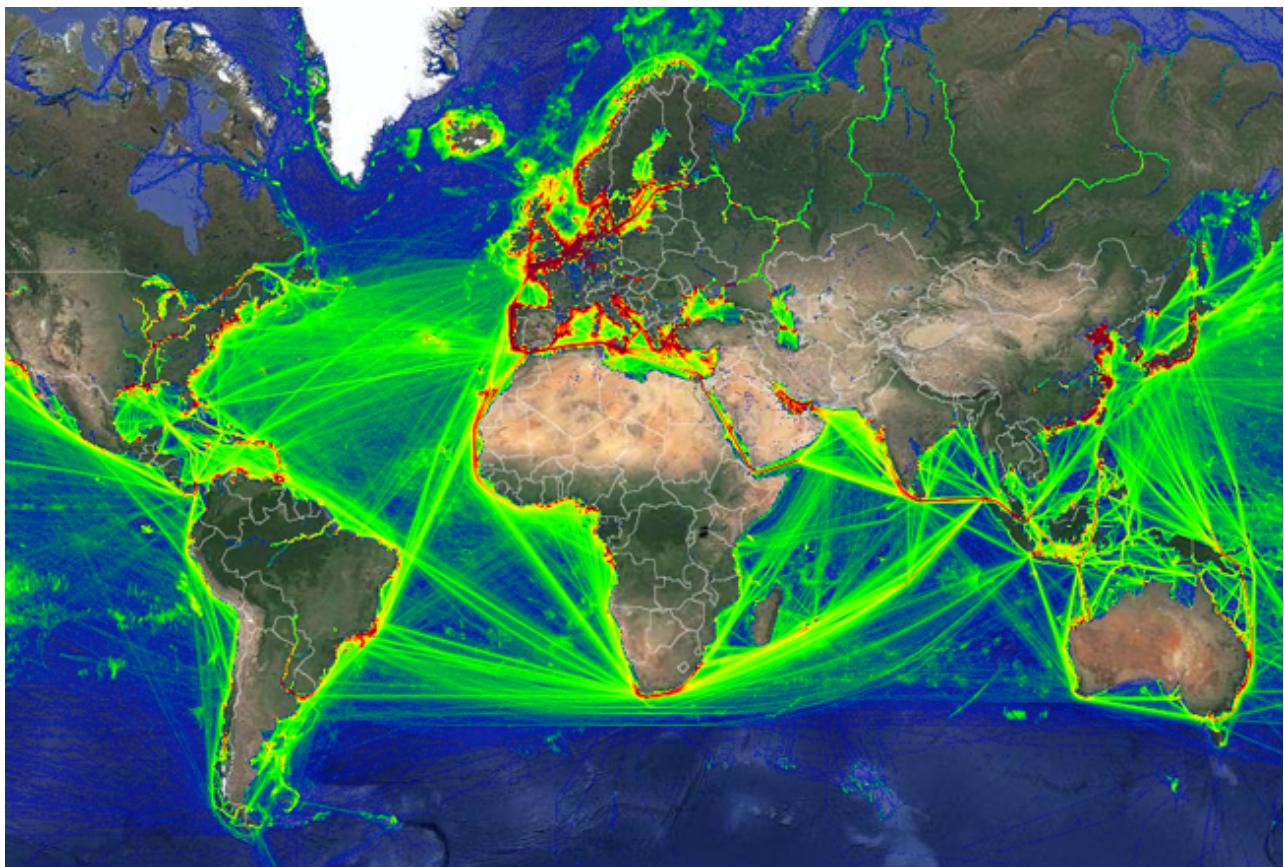


Rapport PRDW

Traitements de données géo-localisées en haute volumétrie

Compiègne - 11 January 2017



Client : Ministère de la Défense

Étudiants : MARECHAL Anaig, GOUJON Paul, PAUL Valentin, WACHALSKI Pierre

Semestre : A16

Présentation Générale du projet	5
1. Contexte	5
2. Définition du projet	5
3. Objectifs visés	5
4. Plannification	6
5. Compétences nécessaires	6
6. Produit du projet	6
7. Risques	7
Introduction	8
Veille technologique	9
1. Moteur d'analyse	9
Hadoop	9
Spark	12
Base de données distribuée	14
Scaper	18
1. Etudes d'accessibilité des données	18
2. Etudes des données reçues par l'appel	20
3. Etudes des headers de l'appel	21
4. Réduction des headers	21
5. Mise en place d'une box visuelle	22
6. Etude du niveau de zoom	23
7. Problèmes rencontrés	24
8. Premier scraper : Le scraper de positions	26
9. Deuxième scraper : Le scraper des destinations	27
10. Scraper - Conclusion & perspectives	29
L'architecture Big Data du projet	30

1. Un système Big Data adapté aux scénarios de recherche	30
2. Conception des tables Cassandra	31
3. Alimentation des tables Cassandra	34
4. Envoi des données dans Spark	35
5. Conclusion / Améliorations possible	35
Analyse des données	37
1. Le machine learning sur Spark	37
Problématique	37
Librairies de Machine Learning	38
Manipulation de données géolocalisées	39
Choix de librairie :	40
2. Algorithmes de machine learning pour notre problématique	40
Les méthodes à noyaux	40
One-classe SVM	41
L'algorithme DBSCAN	43
3. Génération d'une carte de densité	44
4. Génération d'une carte de densité des vaisseaux pour une destination donnée	47
5. Algorithme annexe : suivi d'un vaisseau en particulier	48
6. Algorithmique : Perspectives	48
Court terme : Routes récurrentes / anormales	49
Long terme : propositions	50
Conclusion	51
Bibliographie	52

Présentation Générale du projet

1. Contexte

Le projet présenté au sein de ce rapport s'est effectué dans le cadre d'une Unité de Valeur de l'Université de Technologie de Compiègne, associant l'organisme partenaire et le département de Génie Informatique. Il a été réalisé par quatre étudiants en Génie Informatique, spécialisés en Fouille de Données et Décisionnel.

2. Définition du projet

Le projet consiste au traitement de données géolocalisées dans un contexte de Big Data. Les données représentent des véhicules mobiles, dans notre cas des navires, et sont ainsi des données spatio-temporelles.

Du fait de la haute volumétrie des données, un premier challenge se situe dans le stockage et le traitement des données au sein d'un entrepôt adapté à la problématique Big Data, apportant une solution scalable et des données rapidement accessibles.

La suite du projet consiste à adopter une démarche de data scientist, c'est à dire déterminer et appliquer des algorithmes optimaux de caractérisation des données, tels que des algorithmes de pattern recognition, afin de tirer un maximum d'information du dataset.

Le choix des technologies à adopter dans le cadre de cette étude est laissé libre, avec pour seule contrainte leur origine open-source : une veille technologique est donc à effectuer en amont.

3. Objectifs visés

- Déterminer les technologies open-source les plus adaptées à la réussite du projet
- Récupérer les données AIS à partir d'une source internet
- Concevoir une architecture distribuée optimisant au maximum la performance des calculs
- Mettre en place un moteur de stockage des données adapté à la géolocalisation

- Implémenter des scénarios de recherche
- Analyser les données grâce à des algorithmes de Data Mining

4. Plannification

Le projet a débuté le 30 septembre 2016, suite à une réunion de clarification des objectifs en présence de l'organisme client.

Les différents livrables devront être rendus début janvier 2017 et une soutenance aura lieu fin janvier afin de présenter notre travail.

Tâches	2016													2017			
	Sept.		Octobre			Novembre			Décembre			Janvier					
	39	40	41	42	43	44	45	46	47	48	49	50	51	52	1	2	3
Rencontre n°1 (Lancement du projet)																	
Veille technologique																	
Proof of concept																	
Implémentation du jeu de données																	
Rencontre n°2																	
Implémentation des scénarios																	
Rendu rapport écrit							Vacances				Médians						
Soutenance																	

5. Compétences nécessaires

Des compétences seront tout d'abord nécessaires dans la conception d'une architecture distribuée.

Des compétences sont également nécessaires dans le stockage des données. Toute l'équipe étudiante a des connaissances à ce sujet.

Enfin, des compétences en Data Mining et Machine Learning sont indispensables. Toute l'équipe maîtrise ce dernier point.

Suite à une veille technologique, nous nous concentrerons sur des outils tels que Hadoop et Apache Spark.

6. Produit du projet

Le rendu se présente sous la forme d'un "Proof of Concept" du projet élaboré dans une première partie, puis de notre travail final sur les données au niveau du stockage et de l'analyse, accompagné d'un rapport final.

Une soutenance aura également lieu afin de présenter notre avancée.

Les livrables se présentent comme suit :

- Rapport de projet : à destination du partenaire et de l'UTC, expliquant la démarche générale et algorithmique
- Code du projet : le travail de développement doit être rendu
- Documentation technique : une documentation technique permettant de comprendre l'architecture du module développé doit être rendue

Note : en fonction de leurs disponibilités, les étudiants sont également invités à venir présenter leur projet au cours de la journée des métiers à laquelle participera l'organisme partenaire.

7. Risques

Un premier risque se situe au niveau de la récupération des données brutes. En effet, si celles-ci sont corrompues, incomplètes ou présentent un effort de nettoyage trop important, les délais du projet pourraient être allongés. De plus, n'ayant encore aucune connaissances du niveau de sécurité des sites que nous voudrions scraper, le temps d'implémentation du scraper reste vague.

Un autre risque se situe au niveau du choix des technologies. En effet, la majorité des technologies utilisées dans le Big Data ne sont malheureusement pas encore enseigné à l'université et le temps de formation et de veille technologique pourraient alors être plus long que prévu. De plus, le traitement de ces données sous la problématique Big Data étant faites dans des langages non étudiés à l'université, chaque étudiant devra alors se former auparavant et ne pourra donc avoir des connaissances profondes dans la théorie comme dans la pratique de ce type de langage ce qui pourrait compliquer et ralentir le déroulement du projet.

Introduction

La présentation générale du projet étant faite, dans la suite de ce rapport, vous retrouverez les 5 différentes étapes qui ont composées notre PR lors de ce semestre. Ainsi, nous commencerons tout d'abord par la veille technologique d'un point de vue général au niveau du contexte Big Data et des choix technologiques vers lesquels nous voudrions partir pour résoudre notre problématique de base. Ensuite, nous nous pencherons sur l'implémentation du scraper, qui a été une tâche plus difficile qu'elle n'y a d'abord paru et dont l'implémentation était primordiale afin de récupérer des jeux de données. Dans un troisième temps, nous reviendrons sur l'architecture finale mise en place sur le serveur ainsi que nos constatations pour une évolution future de cette architecture. Puis, nous reviendrons sur les différentes idées que nous avons eues et recherches que nous avons faites pour certains scénarios, ainsi que les résultats que nous avons obtenus. Nous conclurons ce rapport par une prise de recul sur les différents résultats obtenus ainsi que les choix qui ont été faits lors de ce projet.

Veille technologique

La veille technologique a constitué une des premières étapes de notre projet et a été bien sûr déterminante sur le reste du déroulement. La seule contrainte à respecter est le choix de technologies open-source.

1. Moteur d'analyse

Les architectures distribuées sont de bonnes solutions pour faire face à une volumétrie importante. Les deux principales existantes sont Hadoop et Spark que nous avons principalement étudiées. De nos jours, il est important de prendre du recul sur le choix du moteur de recherche en fonction du problème que l'on cherche à résoudre. Ainsi, certains problèmes nécessitent une combinaison d'Hadoop et de Spark, alors que d'autres problèmes pourront n'utiliser qu'un seul écosystème.

Hadoop

Hadoop a été inspiré par la publication de MapReduce, GoogleFS et BigTable de Google. Hadoop a été créé par Doug Cutting et fait partie des projets de la fondation logicielle Apache depuis 2009. Il est donc en open-source, ce qui répond à notre contrainte de départ.



Le noyau d'Hadoop est constitué d'une partie de stockage: HDFS (Hadoop Distributed File System), et d'une partie de traitement appelée MapReduce. Nous allons ainsi expliciter ces deux composants afin de mieux comprendre ce framework et ainsi justifier son utilisation pour notre problème.

Hadoop Distributed File System (HDFS)

HDFS a été conçu pour stocker de très gros volumes de données sur un grand nombre de machines équipées de disques durs banalisés. Il se situe donc au dessus des systèmes de fichiers de chaque machine présente dans le cluster et permet ainsi l'abstraction de l'architecture physique de stockage. HDFS peut-être considéré comme le système de fichiers distribués car il fonctionne comme si l'on avait qu'un seul et unique disque dur.

Bénéfices d'utilisation de HDFS :

- Supporte le traitement distribué : Afin de gagner en rapidité, chaque fichier qui sera amené à être traité sera alors divisé en plusieurs blocs répartis sur différentes machines afin de gagner en rapidité grâce à la parallélisation des tâches.
- Gère les erreurs : En informatique, une erreur peut arriver à n'importe quel moment et les erreurs réseaux sont courantes. HDFS permet de gérer ces erreurs grâce à la mise en place de réPLICATIONS de blocs (par défaut le facteur réPLICATION est de 3). Ainsi, même si un noeud échoue sa tâche, deux autres noeuds auront tout de même fini celle-ci et le résultat ne sera donc pas perdu.
- Scalabilité : HDFS est capable de supporter une future expansion de son cluster. Ainsi, il peut être très intéressant de le mettre en place lorsque nous pensons avoir des données qui seront de plus en plus importantes dessus. Il n'est pas rare de l'utiliser avec des pétadonnées et afin de gagner en rapidité, il suffira d'agrandir son cluster en ajoutant ainsi d'autres machines pour le traitement des données.
- Rentabilité : HDFS n'a pas besoin de machines très performantes pour pouvoir fonctionner mais a justement été pensé afin de fonctionner sur des machines ordinaires. Ainsi, il est plus performant pour des calculs sur des données massives de mettre en place HDFS sur un cluster de plusieurs machines ordinaires plutôt que d'utiliser un ordinaire ayant des composants hardwares très puissants et donc un matériel beaucoup plus cher.

MapReduce

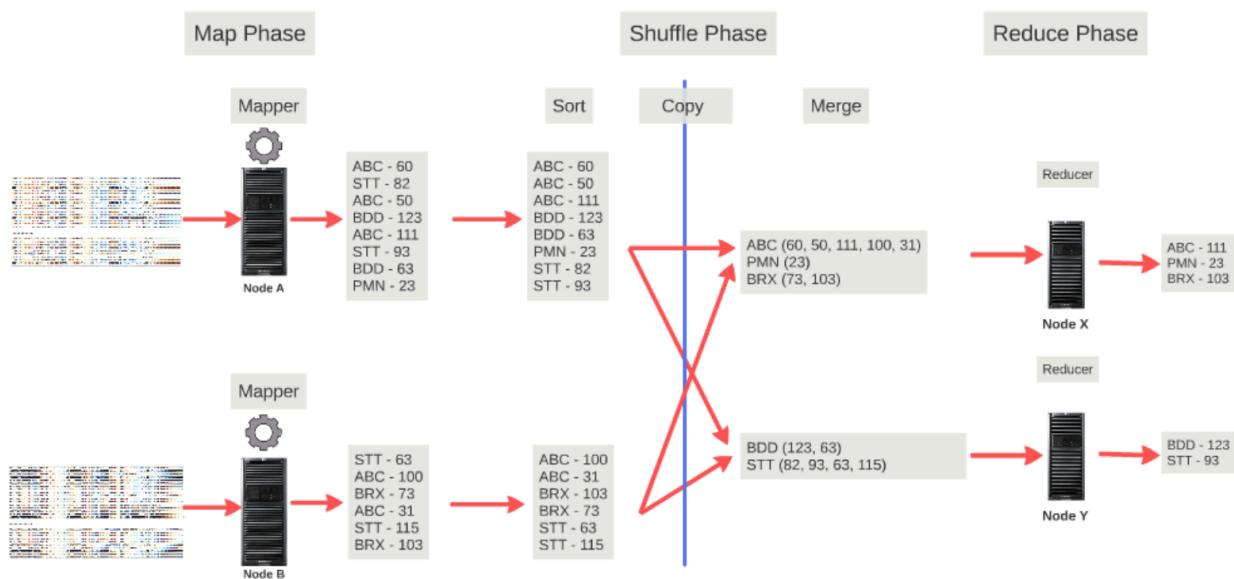
Map Reduce n'est pas un langage de programmation mais plutôt un modèle de programmation distribuée pour le traitement de données massives. Ce modèle a été conçu par Google et peut donc être implémenté dans n'importe quel langage de programmation. Hadoop a ainsi implémenté MapReduce, permettant ainsi le transfert de données ainsi que la parallélisation des exécutions sur des serveurs distribués.



Fonctionnement de MapReduce :

- Map Phase : Chaque set de données est divisé en plusieurs parties appelées (Input Split). Pour chaque input split, un mapper lui sera alloué et traitera ses données. Une fois le traitement fini, le mappeur donnera en sortie des données en format (clé - valeur). Lors d'un lancement de Map Reduce, il y aura toujours un ou plusieurs mappers. Ce nombre dépendra notamment du nombre d'input split que nous allons traiter et donc du nombre de données initiales que nous avons sur notre node. Ainsi, comme son nom l'indique, la "Map Phase" se chargera de répartir le traitement des données sur différents mappers.

- Reduce Phase : Une fois la "Map Phase" terminée, nous nous retrouvons alors avec des données traitées et au format (clé - valeur), il faut dorénavant les récupérer et les "réduire" en un seul output : c'est la "Reduce Phase". Cette phase va donc récupérer les réponses de plusieurs mappers comme valeur de départ et les réduire pour en donner un seul et unique fichier. Les clés sont groupées par valeur et la fonction reduce n'est appelée qu'une seule fois par clé. Ainsi, chaque mappeur contenant la clé donnera son output au Reducer qui a appelé cette clé. Afin d'être performant, même lors de très grands volumes de données, il est possible d'avoir plusieurs reducers afin de diviser le temps d'exécution de la phase de réduction.



Limitations d'Hadoop

Hadoop est une technologie très utilisée en batch processing, cependant elle montre ses limites pour le streaming. En effet, les données sont stockées sur le disque. Rien n'est prévu dans MapReduce pour maintenir un cache en mémoire. Ainsi, les temps de réponses sont alors élevés, ce qui ne répond pas aux contraintes du streaming que nous recherchons. Du fait que Hadoop ne puisse être utilisé pour le traitement en temps réel, de nombreux industriels ont commencé à développer un processus permettant d'utiliser la force de HDFS pour stocker les données et utiliser Spark pour le processing en temps-réel. Nous allons donc dorénavant nous intéresser à Spark dans la prochaine partie.

Spark

Spark est également un framework big data permettant de travailler avec des données distribuées. À la différence de Hadoop, il ne propose pas de stockage distribué. Il a donc besoin d'un système de stockage distribué et peut donc ainsi s'employer en surcouche de HDFS par exemple.

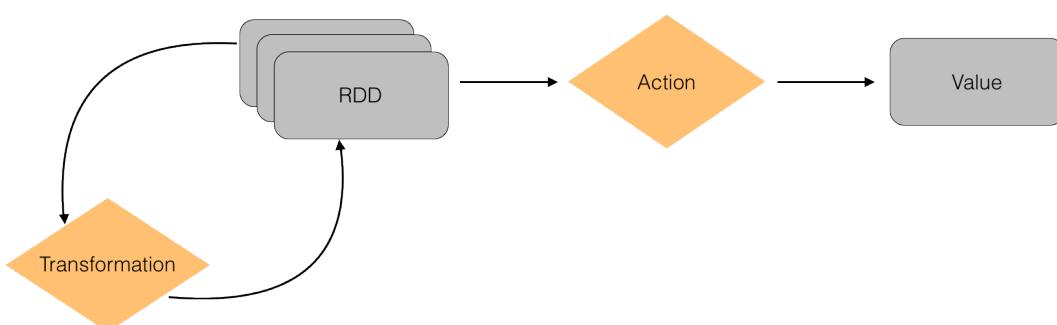
Avantages de Spark

La méthode utilisée par Spark pour traiter les données fait qu'il est beaucoup plus rapide que MapReduce. Alors que MapReduce fonctionne en étapes, Spark peut travailler sur la totalité des données en une seule fois. La séquence de travail de MapReduce ressemble à ceci : il lit les données au niveau du cluster, il exécute une opération, il écrit les résultats au niveau du cluster, il lit à nouveau les données mises à jour au niveau du cluster, etc. Au contraire, Spark exécute la totalité des opérations d'analyse de données en mémoire et en temps quasi réel. Ainsi, Spark est jusqu'à 10 fois plus rapide que MapReduce pour le traitement en lots et jusqu'à 100 fois plus rapide pour effectuer l'analyse en mémoire.

Les Resilient Distributed Datasets (RDD)

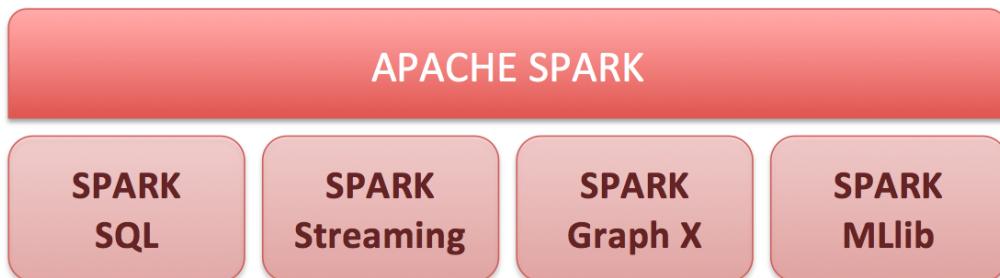
C'est sous ce format que sont gérées les données en Spark. Le Resilient Distributed Dataset (RDD) est un concept créé par les fondateurs de Spark et correspond à une collection immuable. Par défaut, lors de la lecture d'un fichier, les données sont transformées sous forme d'un RDD de String où chaque élément correspond à une ligne de notre fichier initial. Il existe alors deux opérations possibles sur nos RDDs :

- Les transformations : Comme leur nom l'indique, elles transforment un RDD en un autre RDD via différentes méthodes (map, filter, reduceByKey, ...)
- Les actions : Elles transforment un RDD en une valeur (count, collect, ...). Il est intéressant de remarquer qu'une action ne sera lancée que si elle est appliquée à un RDD. On dit alors que les actions sont de type "lazy" sous Spark.



Outils de Spark

Apache Spark peut s'écrire avec les langages Scala, Java et Python, cependant, il utilise au mieux ses capacités avec son langage natif Scala. De plus, il propose de nombreux outils dont deux nous intéressent plus particulièrement : Spark Streaming et Spark MLlib.



Spark Streaming

Spark ne permet de traiter que des données qui sont figées à un instant T. Grâce au module de Spark Streaming, il est possible de traiter des flux de données qui arrivent en continu, et donc de traiter ces données au fur et à mesure de leur arrivée. Spark Streaming va initialiser un contexte avec une durée et ajoutera toujours un délai entre l'arrivée d'un message et son traitement. En effet, le framework va accumuler des données pendant cette durée puis produire un petit RDD. On parle alors ici de micro-batches par opposition à un traitement des événements un par un.

Spark MLlib

C'est la librairie de Machine Learning de Spark. Elle est optimisée pour le calcul parallélisé. Ainsi, tous les algorithmes de cette librairie sont conçus de manière à être optimisés pour le calcul en parallèle sur un cluster. MLlib a été conçu pour une utilisation très simple des algorithmes en les appelant sur des RDD dans un format spécifique, quel que soit l'algorithme choisi.

Attention cependant à son utilisation qui n'est pas idéale pour chaque problème. En effet, une des conséquences directes des algorithmes en mode distribués est que, pour de petits datasets qui tiennent en mémoire, Spark mettra beaucoup plus de temps à s'exécuter que le même algorithme lancé depuis Python ou R. En revanche, les performances de MLlib deviennent extrêmement intéressantes lorsque les volumétries sont importantes.

Choix du moteur d'analyses

Après nous être renseigné en détails sur Hadoop et Spark, il s'est avéré que Spark convenait mieux à notre problème et ce pour plusieurs raisons :

- Tout d'abord, Hadoop MapReduce n'est pas fait pour du temps réels et ne convenait donc pas à nos exigences. Au contraire, Spark avait déjà un module (Spark Streaming) qui nous permettait de pouvoir avoir cet aspect temps-réel.
- De plus, Spark permet un accès rapide aux données et à un module de Machine Learning déjà prêt à l'usage et marchant pour un grand nombre de datas. Ce module pourrait notamment nous être indispensable lors de l'implémentation de certains scénarios.

Il aurait tout de même été concevable d'allier Hadoop avec Spark en utilisant ainsi le système HDFS Hadoop et le traitement de données sous Spark. Cependant, ce système bien que très performant aurait demandé une mise en place complexe et aurait été trop coûteuse en temps pour un projet de cette taille et de cette durée. C'est pourquoi, nous allons plutôt utiliser une base de données de type NoSQL avec Spark. Le choix de cette base de donnée est expliquée dans la prochaine partie de ce rapport.

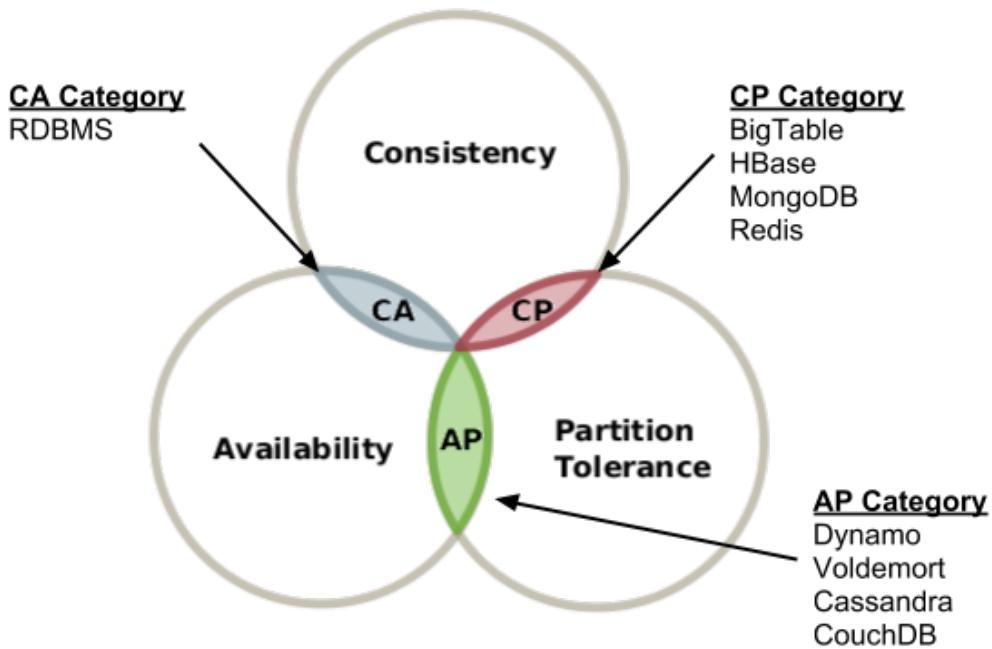
Base de données distribuée

Qu'est ce que le NoSQL ?

Le NoSQL ou “not only SQL” désigne les bases de données qui ne sont pas fondées sur le principe classique des bases de données relationnelles. Développé à l'origine pour gérer du Big Data, l'utilisation de base de données NoSQL a explosé depuis quelques années. Ne plus répondre au modèle relationnel est la caractéristique principale des bases de données NoSQL, cependant, elles répondent aussi au “théorème” du CAP d'Eric Brewer qui est plus adapté aux systèmes distribués. En effet, sur un plan intuitif, un système distribué devrait idéalement posséder les trois caractéristiques suivantes :

- Cohérence : Chaque donnée est présente dans la même version sur tous les noeuds du réseau. Il n'y a pas de retard dans la propagation des mises à jour.
- Haute disponibilité : L'application est accessible à tout instant et chaque requête reçoit une réponse qui lui confirme si elle a été traitée avec succès ou non.
- Résistance au morcellement : Le système doit pouvoir continuer à fonctionner lorsque différents noeuds sont isolés consécutivement à une rupture du réseau.

Cependant, le "théorème" CAP dans sa formulation originale suggère que dans un système distribué seules deux des trois caractéristiques souhaitables mentionnées peuvent être réalisées simultanément.



D'autres caractéristiques communes aux différentes bases de données NoSQL peuvent être citées tel que le partitionnement horizontal sur plusieurs noeuds mais aussi la réplication des données (permettant ainsi de gérer les erreurs réseaux tout HDFS).

Plusieurs types de base de données NoSQL

Nous pouvons catégoriser les bases de données NoSQL en deux catégories :

- les bases de données orientées agrégats (BDOA)
- les bases de données orientées graphes (BDOG)

Dans cette partie, nous nous intéresserons essentiellement aux BDOA, les BDOG n'étant pas utiles pour notre projet. Parmi ces BDOA, nous distinguerons trois sous-catégories, selon la structure des agrégats qu'elles manipulent :

- les entrepôts clé-valeur (ECV)
- les bases de données orientées documents (BDOD)
- les bases de données orientées colonnes (BDOC)

Les entrepôts clé-valeur (ECV)

Concept :

Un ECV peut être envisagé comme une collection de tables de hachages persistantes, c'est-à-dire, comme une collection de couples clé-valeur persistées sur disque. La valeur en question peut être un morceau d'information sans aucune structure particulière. Les opérations que l'on peut effectuer sur une telle BDD sont : la récupération de la valeur pour une clé donnée, la mise à jour, la création ou la suppression d'une valeur pour une certaine clé.

Les implémentations les plus répandues des entrepôts clé-valeur sont Riak, Redis, Memcached DB et DynamoDB chez Amazon.

Usages :

Le stockage de données de session utilisateur est le cas typique d'utilisation d'un ECV. De manière similaire, le stockage des préférences ou des profils utilisateurs ainsi que les associations entre clients et paniers d'achats des sites e-commerce conviennent bien aux ECV.

Les bases de données orientées documents (BDOD)

Concept :

Sur un plan conceptuel, les BDOD diffèrent peu des ECV. Une BDOD peut schématiquement se décrire comme un ECV dont les valeurs sont des documents semi-structurés (= documents auto-descriptifs avec un format type JSON ou XML). Par contraste avec les SGBDR qui exigent que chaque enregistrement d'une table ait les mêmes colonnes, spécifiées par un schéma, rien n'exige que les documents stockés dans une BDOD aient tous le même format.

De ce point de vue, une BDOD est donc beaucoup plus flexible qu'un SGBDR dans la mesure où il est possible de modifier coup par coup la structure d'un document sans avoir à respecter un schéma préétabli.

Les implémentations les plus répandues de BDOD à ce jour sont : MongoDB, CouchDB et RavenDB.

Usage :

Nous pouvons tout d'abord mentionner les systèmes de logs qui, par définition, ont à stocker des associations entre événements et contenus sans structure prédéfinie. A cet usage, nous pouvons ajouter celui des applications qui manipulent naturellement des documents comme les systèmes de gestion de contenu ou les plateformes de blogs.

Les bases de données orientées colonnes (BDOC)

Concept :

La première manière de définir, sur le plan conceptuel, un BDOC est qu'il s'agit d'un ECV dont les valeurs possèdent une structure bien particulière. Cette structure en l'occurrence est celle de colonnes dont les noms peuvent être soit statiques, ils sont alors partagés par tous les enregistrements d'une collection de colonnes, soit dynamiques, c'est-à-dire qu'ils peuvent varier d'un enregistrement à l'autre au sein d'une même collection.

La possibilité d'avoir des colonnes dynamiques permet d'avoir des enregistrements avec colonnes différentes en type et en nombre, chose impossible avec un SGBDR. Un autre élément des structures, commun à beaucoup de BDOC, est la possibilité de regrouper entre elles des colonnes contenant des données sémantiquement liées. L'enregistrement d'un client contiendra toutes ses commandes par exemple. Ces regroupements s'appellent des super-colonnes. Elles aussi peuvent à leur tour être statiques ou dynamiques, on peut alors construire quatre types de structures, que l'on appellera des familles de colonnes, appropriées à des situations différentes.

Les BDOC les plus populaires à l'heure actuelle sont : Cassandra, HBase, Hypertable, Amazon SimpleDB.

Usage :

La flexibilité des BDOC en fait un choix idéal, par exemple, pour stocker des rapports d'erreur issus de plusieurs applications dont chacune pourra utiliser le groupe de colonnes qui lui convient. Les blogs et leur publications, leur liens ainsi que les commentaires associés sont particulièrement bien adaptés à l'utilisation d'une BDOC. Pour finir, le stockage de timeseries venant de données de capteurs ou autres objets connectés est possible.

Choix de la base de données

Nous avons donc choisi une base de donnée orientées colonnes car celles-ci correspondent mieux aux usages dont nous avons besoin afin de pouvoir réaliser ce POC. De plus, l'une des grandes forces de Cassandra est le stockage de timeseries, ce que nous allons essentiellement avoir car nous rappelons que nos données seront des données de géo-localisations de bateaux.

Pour finir, Datastax (société assurant la distribution et le support d'une version pour l'entreprise de Cassandra) a mis en open-source un connecteur entre Spark et Cassandra, rendant ainsi leur communication beaucoup plus simple à mettre en place.

Scraper

L'une des principales difficultés de notre projet résidait dans le fait qu'aucun dataset ne nous était fourni. Nous devions donc nous procurer un jeu de données d'étude par nos propres moyens.

Notre client nous ayant recommandé de consulter plusieurs websites de tracking de vaisseaux via position AIS, nous avons décidé, après étude, de construire ce que nous appellerons par la suite "scraper". Cet outil nous permet de récupérer, par voie détournée, les données alimentant l'interface graphique du website "vesselfinder.com".

Ainsi, cette partie du rapport se divisera en trois parties afin de comprendre toute l'implémentation mise en place. Dans un premier temps, les différentes études d'accessibilité des données seront mises en avant. Puis, nous détaillerons en deux parties, les deux scrapers développés afin de récupérer l'ensemble des données indispensables au bon fonctionnement de chacun des scénarii attendus

1. Etudes d'accessibilité des données

Fonctionnement du site vesselfinder

Plusieurs websites nous avaient initialement été recommandés par notre client. Citons notamment :

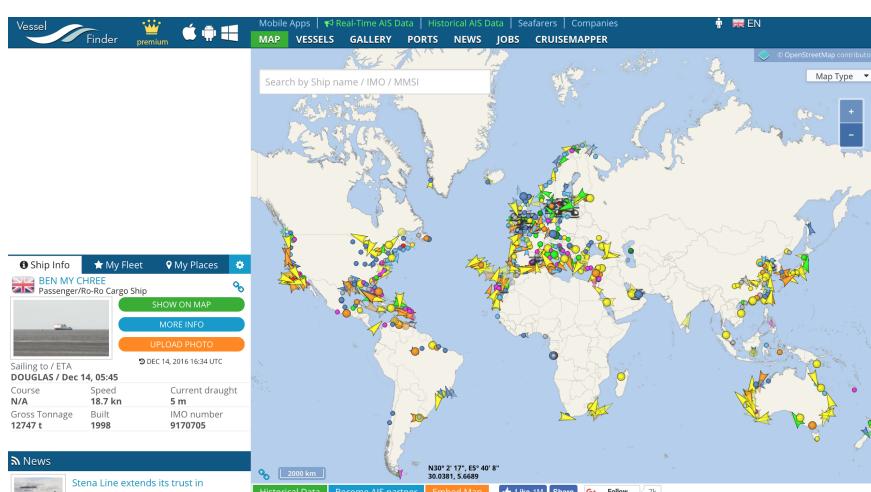
<http://www.aishub.net/>

<http://www.marinetraffic.com/>

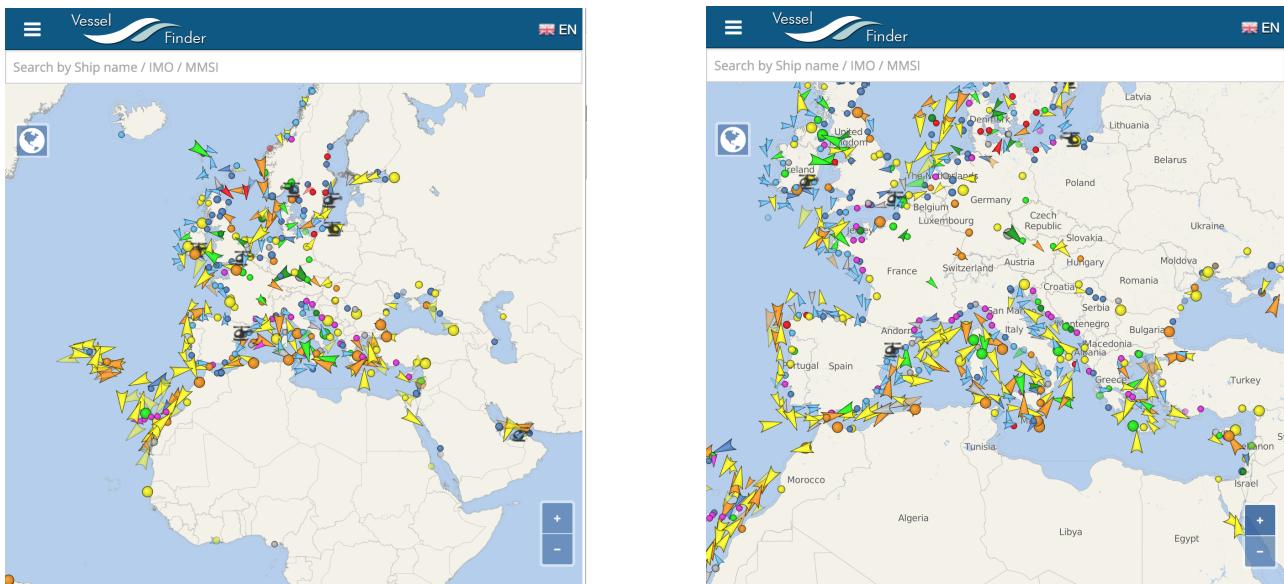
<https://www.vesselfinder.com/>

... et n'importe quel autre website que nous aurions repéré entre temps.

Une première étude du fonctionnement de ces différents websites nous a permis de retenir la piste de l'exploitation du fonctionnement de Vesselfinder afin de constituer nos datasets. La page d'accueil de Vesselfinder présente en effet une map contenant la position de nombreux vaisseaux.

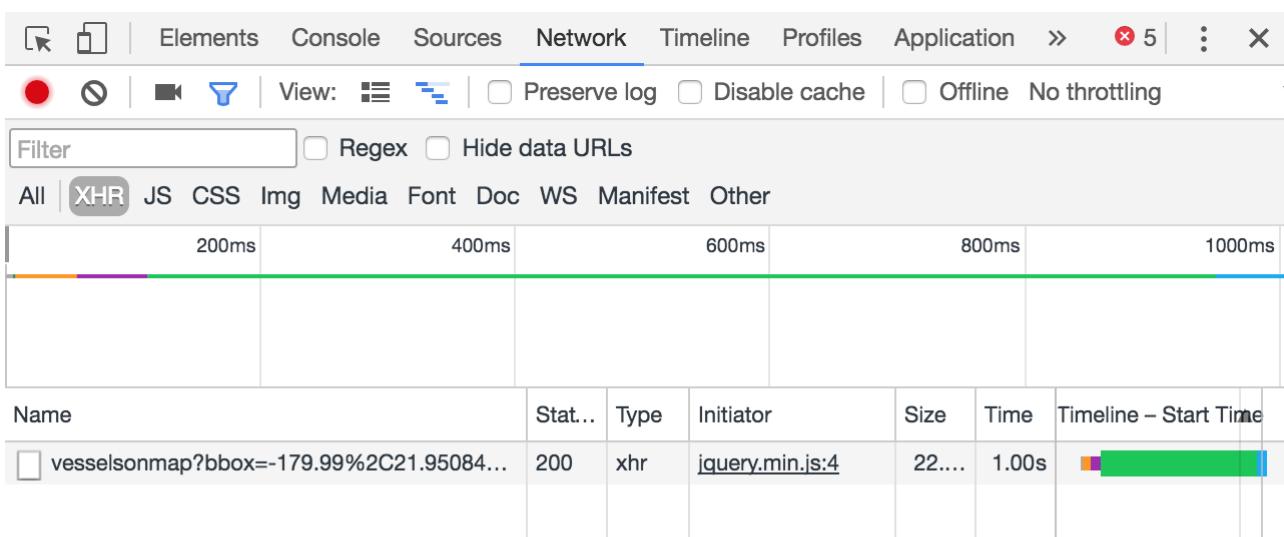


L'étude du fonctionnement de cette map nous a permis de voir que cette page était rafraîchie de manière régulière (toutes les minutes pour être exact), ainsi qu'à chaque fois qu'une transformation y était appliquée (zoom, déplacement, ...).



Cette mise à jour des données la composant se fait sans rechargement de la page, il s'agit d'appels AJAX que nous avons su repérer. Par exemple, la transformation suivante (zoom avant) :

Résulte en un appel AJAX (visualisé ici grâce à la console de google chrome) :



Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
vesselsonmap?bbox=-179.99%2C21.95084...	200	xhr	jquery.min.js:4	22....	1.00s	<div style="width: 22%; background-color: #007bff; height: 10px;"></div>

2. Etudes des données reçues par l'appel

Ayant repéré cet appel, nous nous sommes intéressés à la forme de la réponse obtenue afin de regarder si les données nous intéressent. La réponse de cet appel récurrent se présente comme suit :

	x Headers	Preview	Response	Cookies	Timing
1					
2	33591680	-1226220	3557	86 181 235007413 0	
3	18878134	-70077344	1677	142 36 367421390 1	
4	17568471	-53306801	3004	142 60 367477280 0	
5	17997034	-54254293	555 76	30 367189630 0	
6	19431770	-54539030	2060	0 48 366936660 0	
7	12632370	-40791930	3022	187 260 311000396 0	
8	21904156	-2924287	1212	1 36 205881910 0	
9	17515319	73169279	136 0	30 412448791 0	
10	22873865	8022182 2716	0	37 247279200 0	
11	22873865	AE110030	2600	0 0 3669000010 0	

Après quelques recherches sur internet et sur le site vesselfinder, nous avons réussi à définir la nature de chacune des colonnes, étant respectivement :

LATITUDE LONGITUDE COG SOG HEADING MMSI PAC

Soit, plus précisément :

- LATITUDE : la latitude (convention AIS) du vaisseau
- LONGITUDE : la longitude (convention AIS) du vaisseau
- COG : Celerity Over Ground, la célérité (convention AIS) du vaisseau
- SOG : Speed Over Ground, la vitesse (convention AIS) du vaisseau
- HEADING : le cap du vaisseau
- MMSI : l'identifiant MMSI du vaisseau
- PAC : Accuracy, un booléen dépendant de la méthode d'acquisition des données GPS (et donc leur précision)

Ces informations formant une excellente première base pour notre étude, nous avons alors décidé d'essayer de les extraire.

Nous avons également déterminé les coefficients nécessaires pour obtenir les données suivant la convention WGS car c'était une des consignes demandées dans le cahier des charges. Les coefficients de passage sont les suivants :

- Latitude réelle = latitude brute / 600000
- Longitude réelle = longitude brute / 600000
- SOG réelle = SOG brute / 10
- COG réelle = COG brute / 10

3. Etudes des headers de l'appel

Ces informations nous intéressent, nous nous sommes alors penchés sur les headers de l'appel permettant d'obtenir ces informations afin de réussir à les utiliser pour récupérer les données qui nous intéressent et ainsi remplir notre base de données de toutes ces données géo-localisées de bateaux. Voici les headers d'un appel :

```
GET /vesselsonmap?  
bbox=-179.99%2C21.950847180683354%2C179.99%2C59.31506060664509&zoom=4&mmsi  
=0&show_names=0&ref=99285.11229778039&pv=6 HTTP/1.1  
Host: www.vesselfinder.com  
Connection: keep-alive  
Accept: */*  
X-Requested-With: XMLHttpRequest  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36  
    (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36  
DNT: 1  
Referer: https://www.vesselfinder.com/  
Accept-Encoding: gzip, deflate, sdch, br  
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4  
Cookie:PHPSESSID=e017b111e70f8f7ab7bd576eae4d3d49;mapType=full;tf=0;mapFilter2=  
511; _ga=GA1.2.1983121371.1481733396;gat=1;ls=;shipNames=auto;myships=0;  
vesselfinder-mapPosition=4,10.899983209841611,43.42711777034532
```

L'étude de ces headers nous permet de supposer que l'URL contient un certain nombre de paramètres, notamment ceux de la zone géographique pour laquelle on souhaite obtenir les vaisseaux, ainsi que le niveau de zoom.

4. Réduction des headers

Une fois ces paramètres étudiés, nous avons donc immédiatement tenté de reproduire ces appels AJAX depuis notre machine, et avons eu la joie de constater que cela fonctionnait.

Nous avons par la suite entrepris de réduire les headers des requêtes au strict minimum, espérant que ces dernières n'étaient pas soumises à des mesures de sécurité drastiques. Il s'est avéré que non, seul le header "X-Requested-With" était indispensable.

5. Mise en place d'une box visuelle

Obtention de l'URL optimale

Ayant pour objectif d'obtenir les données des bateaux géo-localisées autour de la mer Méditerranée, nous avons décidé dans un second temps de nous focaliser sur les paramètres à fournir afin de pouvoir récupérer les données d'une zone précise.

Suite à quelques essais, nous avons pu déterminer que l'url prenait le format suivant :
https://www.vesselfinder.com/vesselsonmap?bbox=BOTTOM_LEFT_EAST,BOTTOM_LEFT_NORTH,TOP_RIGHT_EAST,TOP_RIGHT_NORTH&...

Avec BOTTOM_LEFT_EAST, BOTTOM_LEFT_NORTH, TOP_RIGHT_EAST, TOP_RIGHT_NORTH les latitudes et longitudes des coins bas-gauche et haut-droit de la box.

En choisissant les paramètres suivants :

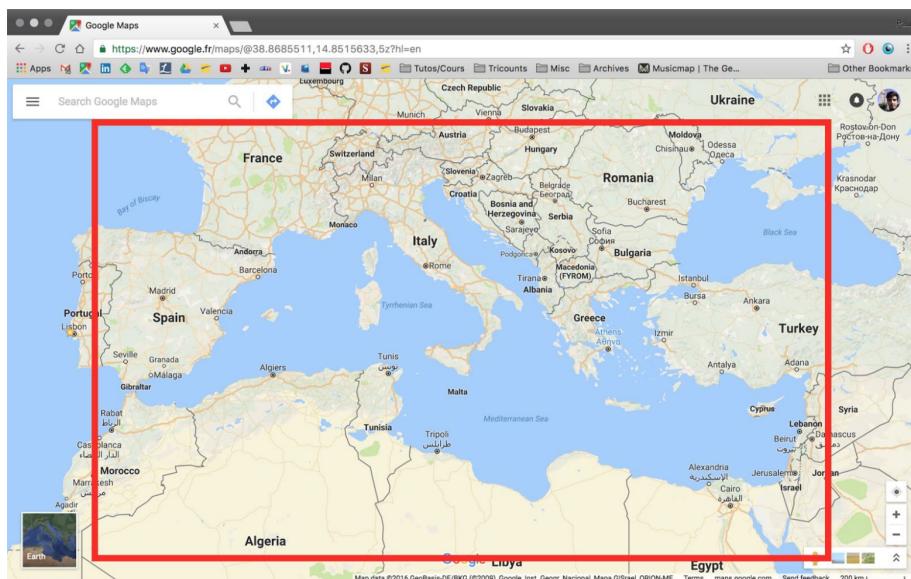
BOTTOM_LEFT_EAST = -4

BOTTOM_LEFT_NORTH = 26

TOP_RIGHT_EAST = 37

TOP_RIGHT_NORTH = 48

Nous obtenons la box :



Cette région est la meilleure représentation que nous pouvons avoir de la mer Méditerranée et c'est pourquoi nous utiliserons celle-ci afin de récupérer le maximum de bateaux possibles.

6. Etude du niveau de zoom

L'étude des niveaux de zoom au niveau de la carte nous a permis de comprendre une information très utile : lorsque les niveaux de zoom sont bas (c'est-à-dire que notre champs de vision est large), tous les vaisseaux présents dans la zone ne sont pas affichés, tandis que lorsque l'on restreint notre champs de vision en se rapprochant (niveaux de zoom élevés), la liste de vaisseaux affichés devient exhaustive

Ainsi, nous avons étudié l'influence du niveau de zoom sur l'exhaustivité des données qui nous étaient retournées, et avons décidé de fixer notre niveau de zoom à 8, avec lequel nous obtenons environ 5000 vaisseaux par appel pour la zone étudiée.

7. Problèmes rencontrés

Impossibilité de récupérer toutes les données via un seul appel

Parmi les objectifs de notre projet, figure l'implémentation d'algorithme utilisant la notion de "route récurrente". Une route récurrente est une route empruntée de manière récurrente entre deux ports : le port de départ, et le port de destination.

Une des limites de la première partie de notre scraper réside dans le fait que ces informations ne nous sont pas fournies par ce dernier. Nous avons donc dû trouver une solution pour obtenir ces informations alternatives.

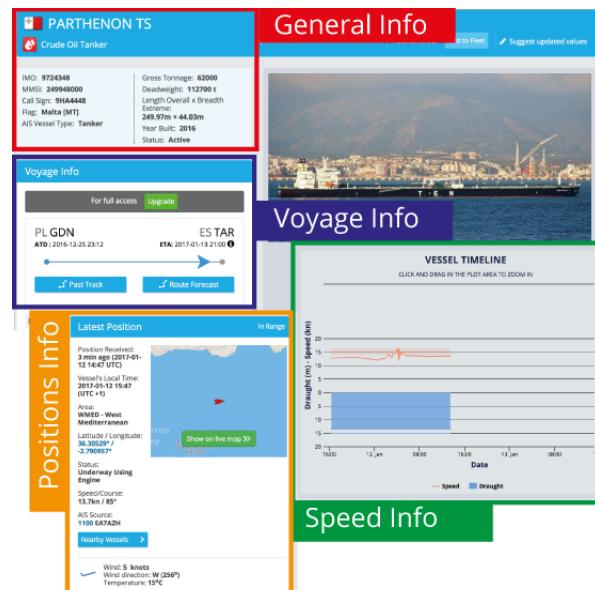
Fausses pistes successives

C'est ainsi que nous nous sommes intéressés dans un premier temps au website "MarineTraffic" : <http://www.marinetraffic.com/> proposant un très important panel d'informations additionnelles.

Nous avons donc développé un script nous permettant de scraper et parser les informations de ce website là. Le développement et le test de ce script nous a pris environ deux semaines.

Malheureusement, nous avons rapidement constaté qu'après un certain nombre d'appel, la première heure, nous ne recevions plus de réponse AJAX.

Nous avons perdu beaucoup de temps à faire différents tests pour comprendre le problème cherchant d'abord l'erreur dans notre code, puis ayant compris que le problème ne venait pas de notre code mais du fait que nous étions considérés comme des "robots" par le website, afin d'outrepasser ces mesures de sécurité.



Nous avons finalement décidé d'abandonner la piste marinetrack au bout de deux semaines, décidant de nous tourner vers la base de données de vaisseaux du website aishub : <http://www.aishub.net/vessels-database.php>

Cette piste a elle aussi été très vite abandonnée, lorsque nous avons constaté que les bateaux que nous détections via notre première partie de scraper étaient en majorité absents de aishub.

Nous avons finalement décidé de nous rabattre sur l'option vesselfinder, décrite ci-dessous, dont le principal défaut est de nous permettre de récupérer les ports d'arrivé, mais pas de départ.

Nouvelle page à scraper pour les informations supplémentaires

Après étude de Vesselfinder, nous nous sommes aperçus que l'URL :

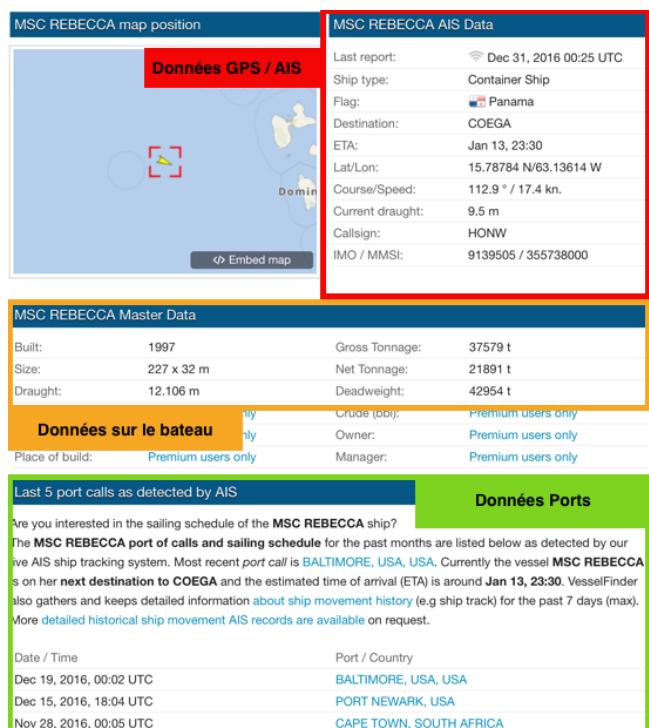
<https://www.vesselfinder.com/vessels/MSC-REBECCA-IMO-9139505-MMSI-355738000>

Fonctionnait également tronquée :

<https://www.vesselfinder.com/vessels/randomstring-MMSI-355738000>

C'est à dire qu'elle est interprétée au moment de la requête et l'utilisateur peut donc faire l'objet d'une redirection en cas de faute de frappe, nous amenant alors sur la page que nous souhaitons tout de même.

Dans notre cas, cela correspondait très exactement à notre besoin, ayant uniquement à notre disposition l'attribut "MMSI" des vaisseaux détectés. Nous avons donc décidé d'étendre notre scraper grâce à un second module, nous permettant de récupérer les informations contenues dans ces pages de "détails" dont nous pouvons voir un exemple ci-joint:



MSC REBECCA Master Data			
Built:	1997	Gross Tonnage:	37579 t
Size:	227 x 32 m	Net Tonnage:	21891 t
Draught:	12.106 m	Deadweight:	42954 t

Données sur le bateau			
Place of build:	Premium users only	Crude (bbl):	Premium users only
		Owner:	Premium users only
		Manager:	Premium users only

Last 5 port calls as detected by AIS	
Date / Time	Port / Country
Dec 19, 2016, 00:02 UTC	BALTIMORE, USA, USA
Dec 15, 2016, 18:04 UTC	PORT NEWARK, USA
Nov 28, 2016, 00:05 UTC	CAPE TOWN, SOUTH AFRICA

Utilisation du second scraper

Notre but, lors de l'établissement de ce second scraper, était donc de récupérer le maximum d'informations additionnelles possible sur les vaisseaux détectés dans la zone considérée.

Il n'était cependant pas possible d'effectuer des requêtes à intervalle aussi régulier qu'au sein de notre première partie de scraper : en effet, envoyer toutes les minutes plusieurs milliers de requêtes, demandant en réponse les plusieurs milliers de pages contenant les informations des vaisseaux correspondants n'était pas envisageable, sans quoi nous aurions immédiatement été détectés et potentiellement bannis, car considérés comme des "bots" par Vesselfinder.

Nous nous sommes donc résolus à n'envoyer qu'une requête par jour pour chacun de nos vaisseaux, en étalant l'envoi de nos requêtes sur plusieurs heures.

8. Premier scraper : Le scraper de positions

script:

https://github.com/prBigData/aisdata/blob/master/get_data.py

Ce script a pour vocation de récupérer les positions successives des vaisseaux dans la zone étudiée tout au long de la journée, en appelant l'API de Vesselfinder à intervalle régulier.

Pseudo algorithme :

```
WHILE (vrai), DO :
    read daily MMSI list;
    request;
    parse response;
    FOR vessels IN response, DO :
        append vessel to vessels dict;
        IF MMSI first time detected today, DO:
            append MMSI the daily MMSI list;
    append vessel dict to the export json;
    update the json MMSI list export;
    sleep(TIME_INTERVAL);
```

Explications du fonctionnement général

Grâce au pseudo-algorithme ci-dessus, nous pouvons remarquer que nous effectuons une requête toutes les x secondes (nous avons choisi une requête toutes les 60 secondes dans notre cas afin de reproduire le comportement du rafraîchissement de la page Web et ainsi ne pas se faire détecter par le site vesselfinder.)

La réponse à cet appel est par la suite parsée, afin de récupérer les informations sur les vaisseaux contenues à chaque ligne de cette réponse. Ces informations sont par la suite exportées en les ajoutant à la suite d'un fichier d'export qui est sauvegarder toutes les heures (un nouveau fichier est donc créé pour chaque heure afin de ne pas perdre de données en cas de panne de leur serveur notamment).

Il y a, de plus, une autre information très importante à remarquer dans ce pseudo-algorithme car cette ligne permet en effet de faire ensuite fonctionner notre second scraper. Ainsi, la dernière ligne de notre algorithme va mettre à jour la liste des bateaux MMSI obtenus dans la journée. Nous obtenons donc chaque jour une liste de bateaux qui ont émis dans données géolocalisées lors de la journée afin de pouvoir ensuite dans un second temps rechercher les différentes informations manquantes, que nous avons explicité dans la partie précédente, telles que les ports d'arrivées et de destinations ou encore leur informations de poids, taille, etc ...

9. Deuxième scraper : Le scraper des destinations

La seconde partie de notre scraper se décompose en deux scripts. Le principal d'entre eux est le suivant.

Script :

<https://github.com/prBigData/aisdata/blob/master/mmsi2/spider3.py>

Ce script code une classe helper nommée "Spider" (inspiration scrapy), qu'il suffit d'instancier avec le bon jeu de paramètres pour obtenir les informations escomptées, scrapées à partir d'un jeu d'URLs de Vesselfinder.

Cette classe propose une méthode "scrap()" fonctionnant comme suit.

Pseudo algorithme:

```
FOR provided url, DO:  
    request;  
    parse responses;  
    potential post-treatment;  
    append treated info to export json;  
    sleep(TIME_INTERVAL);
```

L'instanciation et utilisation de cette classe sont assurées par l'exécution du second script suivant.

Script :

https://github.com/prBigData/aisdata/blob/master/cron_get_data2.py

Ce script est déclenché tous les matins à minuit et 5 minutes, et se charge de récupérer à j+1 le maximum d'informations additionnelles sur les vaisseaux détectés le jour j. Pour cela, ce script récupère la liste des MMSI détectés au cours de la journée de la veille, puis étale les requêtes sur la journée j en cours afin de ne pas spammer le serveur cible avec plusieurs milliers de requêtes quasi simultanées.

Nous détaillons ci-dessous son fonctionnement :

Pseudo algorithme:

```
get yesterday's MMSI list;  
generate the set of url from MMSI list;  
calculate the time delay between spider calls;  
spider instantiation and scraping launch;
```

La génération du set d'urls se fait en concaténant le MMSI du bateau désiré au radical présenté précédemment (dans notre cas nous avons choisi d'utiliser <https://www.vesselfinder.com/vessels/PRDW-MMSI->)

Pour le calcul l'attente de sleep laissé par le spider, nous avons choisi de procéder comme suit. Nous avons estimé que pour qu'un délai raisonnable (> 10s, arbitraire) soit laissé entre deux requêtes, en se basant sur une détection d'environ 5000 vaisseaux différents par jour (observation tirée de nos mesures), il était intéressant d'étaler nos requêtes sur 22 heures, au cours desquelles elles seraient faites à intervalle régulier.

10. Scraper - Conclusion & perspectives

Afin de clôturer cette partie, il est tout d'abord important de revenir sur la complexité de cette tâche. En effet, l'implémentation de ce scraper a mis beaucoup plus de temps qu'escompté et a donc mis le groupe en retard sur les attentes de la PR. Cependant, il est à noter que le travail de scrapage s'est révélé beaucoup plus complexe que ce que l'organisme pensait.

Notre scraper n'est donc complètement fonctionnel que depuis le début du mois de décembre environ, mais cela ne s'est pas fait sans difficultés. Nous en avons rencontré un certain nombre, à la fois liées à notre expérience dans certains domaines, notamment du point de vue des standards http et de la sécurité web.

Aujourd'hui, le scraper est totalement fonctionnel et permet de répondre à la problématique du Big Data demandée au tout début du projet car ce scraper est directement relié à l'architecture de base de données et rempli donc en continu chaque jour le serveur de nouvelles données afin ensuite de pouvoir les étudier.

A l'heure actuelle, nos jeux de données font la taille suivante :

- Environ 95 000 records de destinations journalières
- Environ 120 000 000 de records de position

Il est tout de même à noter au niveau des améliorations que ce scraper n'est malheureusement pas flexible et est donc sensible aux modifications futures faites sur le site Vesselfinder, ainsi afin de l'améliorer, il y aurait tout d'abord un travail de flexibilité des requêtes à faire ainsi qu'une possible optimisation du nombre de requêtes.

L'architecture Big Data du projet

Dans la partie précédente, nous avons parlé de l'implémentation et de l'utilisation du système de scraping : à raison d'un appel par minute, sur 24h, nous recueillons 35 MB de données à l'échelle de la méditerranée. On peut rajouter à cela les données sur les bateaux obtenues depuis vesselfinder...

Dans l'optique d'appliquer des algorithmes de machine learning, il est intéressant de stocker ces données, de manière optimisée. En effet, l'accès à des données bien mises en forme permettra d'obtenir les résultats voulus beaucoup plus facilement.

Nous allons donc aborder en premier lieu la conception souhaitée de l'architecture. Après cela, nous expliciterons le résultat obtenu, et nous terminerons sur les améliorations possibles.

1. Un système Big Data adapté aux scénarii de recherche

Comme postulat de départ, nous nous étions fixé l'objectif d'avoir une architecture permettant de répondre aux différents scénarii. Cela allait de la carte de densité à l'analyse des routes récurrentes en passant par le repérage de positions suspectes de bateaux. Pour cela, les données récupérées nous suffisent, mais il faut pouvoir accéder à toutes ces données suivant plusieurs angles.

1. Pour la carte des densités, nous avons besoin seulement des positions des bateaux.
2. Pour les routes récurrentes, il est préférable de faire une recherche des positions par route (ex : Marseille-Tunis)
3. Pour repérer la trajectoire d'un bateau, on peut chercher par l'ID du bateau. Afin d'affiner la recherche, on pourrait aussi chercher par ID de bateau ainsi que par "route". Et finalement, si ce bateau a déjà fait plusieurs fois le voyage, on pourrait ajouter un compteur, dans l'optique de différencier les différents voyages.

De ce fait, si l'on creuse un peu la façon de stocker les données, on constate qu'il faut faire vraiment attention à bien pouvoir obtenir les données de plusieurs façons.

Pour rappel, nous avions choisi de partir sur un système de stockage de données de type clé/valeur (Cassandra). L'avantage de tels système est la lecture rapide des données et la tolérance aux pannes machines.

Ces données ont pour but final d'être chargées dans un moteur d'analyse. Le moteur retenu est Spark : en effet, Spark est l'une des rares librairies permettant de faire de l'analyse de données efficace en réparti. De plus, il existe un composant, développé par l'entreprise Datastax, permettant de charger des données stockées dans des tables cassandra directement dans Spark. Une fois chargées dans Spark, les données peuvent être analysées via les librairies de machine learning de Spark.

2. Conception des tables Cassandra

Le scraping de données nous retourne des fichiers contenant les positions de tous les bateaux (par minute) ainsi que les informations sur les bateaux qui ont été recensés dans la journée de scraping. Les informations à disposition sont donc (de façon non-exhaustive) :

MMSI - POSITION (LAT / LONG) - DÉPART - ARRIVÉE

Pour répondre à la problématique mise en avant dans la partie d'avant, il faut penser la façon d'écrire et de lire nos données dans notre architecture. Dans le cas d'une base de donnée de type clef/valeur, cela revient à penser la clef.

Cependant, nous souhaitons soit faire des recherches par MMSI, soit par Route, soit avec ces deux informations réunies... Il semble donc dur d'avoir un pattern de clef répondant à tous ces problèmes. Nous nous sommes penchés sur la création de 3 tables différentes, avec 3 clefs.

Voici un aperçu de la définition des ces tables :

```
AIS-POSITION (DATA TABLE)
PRIMARY KEY((MMSI, TripKey, TripNumber), Timestamp)
Table des faits permettant de trouver les datas
{
    "MMSI": "STRING",
    "TripKey": "STRING",
    "TripNumber" : "INT",
    "Timestamp" : "TIMESTAMP",
    "Longitude": "STRING",
    "Latitude": "STRING",
}
```

SHIPS TABLE

PRIMARY KEY(MMSI, TripStart)

Table répertoriant les bateaux avec les routes empruntées par ces bateaux

{

```
"MMSI": "STRING",
"TripKey": "STRING",
"TripNumber": "INT",
"TripStart" : "DATETIME",
```

}

TRIP TABLE

PRIMARY KEY(TripKey, MMSI)

Table destinée à récupérer les bateaux empruntant un itinéraire, analyse de routes récurrentes.

{

```
"TripKey": "STRING",
"MMSI": "STRING",
"TripNumber": "INT",
```

}

Avant de d'expliquer notre choix, une petite explication s'impose sur les clés de Cassandra:

PRIMARY KEY((MMSI, TripKey, TripNumber), Timestamp)

PRIMARY KEY(MMSI, TripStart)

PRIMARY KEY(TripKey, MMSI)

On voit qu'une Primary Key Cassandra peut être définie via deux paramètres, c'est ce qu'on va appeler une Composite Key.

Le premier paramètre est la Partition Key. Cette clef vise à répartir les données dans le dataset. Elle est obligatoire pour toute lecture écriture. De plus, cette clef peut être elle-même constitué de plusieurs colonnes (comme pour le premier exemple).

Le deuxième paramètre de la Composite Key est la Clustering Key. Elle permet d'ordonner les lignes de la base de données.

Pour illustrer cet exemple, imaginons cette pseudo requête SQL :

```
SELECT * FROM AIS-POSITION WHERE MMSI = "1234" AND TripKey = "Tunis-Marseille"
    AND TripNumber = 2
```

Cette requête est correcte car elle comporte les 3 paramètre de la Partition Key dans les conditions. Le résultat de cette requête sera obtenue extrêmement rapidement, et le résultat comportera toutes les positions de ce voyage, dans l'ordre chronologique.

MMSI	TripKey	TripNumber	Timestamp	Longitude	Latitude
1234	Tunis-Marseille	2	31-12-16 23:59	12	14
1234	Tunis-Marseille	2	31-12-16 23:58	12	15

Cette rapidité est lié à la définition de la table. Si l'on souhaite faire une recherche sur seulement les "TripKey", la requête échouera...

Revenons à la conception de nos tables. La première table (ais-position) est le coeur de notre base de données. On y retrouve les positions (latitude / longitude) de tous les bateaux, organisées par chaque voyage de chaque bateau existant.

Maintenant, si l'on souhaite obtenir les données pour un bateau en particulier, il nous faut obligatoirement savoir quels voyages il a effectué, dans le but de reconstituer la clef. De même, si l'on s'intéresse aux routes récurrentes, il nous faut la liste des bateaux qui ont parcouru ce parcours...

C'est dans ce but que les deux tables ont été créées. La table (Ships) permet de retrouver les voyages effectués par le bateau correspondant au MMSI soumis. La table (Trip) renvoie les MMSI ayant effectué le voyage souhaité.

Maintenant que le système est pensé, il faut réussir à remplir les tables des bonnes informations, ce que nous allons voir par la suite.

3. Alimentation des tables Cassandra

Les données sont rapportées par notre scraper. Nous avons 2 scrapers, qui écrivent dans des fichiers. Cependant, comme nous n'avons pas choisi un système de fichier distribué (de type HDFS), nous avons préféré penser l'alimentation de la table en "streaming". En effet, imaginons que le disque arrive à saturation, les fichiers ne peuvent plus être créés avec les données, et le batch processing n'apporterait donc pas les nouvelles données.

Nous nous sommes donc basés dans le code des crawlers pour qu'ils effectuent les insertions des données qu'ils ont récupérées. Mais le système est assez complexe. Afin d'alimenter la table principale, nous avons besoin du MMSI (présent dans les données du scraper), du voyage du bateau (le "TripKey" constitué du port de départ et d'arrivée, rapporté par le deuxième scraper), ainsi que du nombre de fois que le voyage est effectué par ce bateau (une information fabriqué de toute pièce par notre système).

De par ce problème, nous avons très vite vu les limites de notre système : malgré la conception pensée autour de la lecture des données, il nous était impossible d'ajouter les données dans ces trois tables...

Afin de continuer dans notre projet, nous avons donc créé des tables Cassandra permettant de stocker les données rapportées par les scrapers. La première stocke la position d'un MMSI en fonction d'un timestamp. La deuxième stocke la destination d'un MMSI en fonction d'un timestamp.

Même si cette décision a été prise tardivement, nous avons quand même réussi à insérer les données scrapées depuis début novembre pour les données de positions et début décembre pour les données de destination dans ces nouvelles tables, grâce à l'écriture des données dans des fichiers.

Les données récupérées nous permettent de faire une carte des densités, scénario sur lequel nous sommes partis. Il serait aussi possible d'identifier les trajets des bateaux avec un ensemble de requêtes bien choisies, mais nous avons assez peu de données sur les destinations (récupération des données depuis fin décembre...)

4. Envoi des données dans Spark

Afin de pouvoir analyser ces données présentes dans Cassandra, nous avons choisi d'utiliser Apache Spark. Au préalable, lors de notre veille technologique, nous avons vu qu'il était simple de transférer les données présentes dans Cassandra, grâce à une librairie développée par Datastax.

Après une rapide prise en main de Spark, nous nous sommes très vite retrouvés avec un Data Frame constitué de nos données. L'avantage même de l'utilisation de cette librairie permet de charger l'intégralité de la table dans Spark, sans même faire de requête sur la table...

5. Conclusion / Améliorations possible

Pour conclure, nous pouvons voir que l'architecture est rapide, elle traite facilement un gros volume de données (+100 Millions d'entrées dans la table principale). Mais elle n'est pas optimale. Aujourd'hui nous sommes capables de récupérer les données, les stocker dans une base de données distribuée, et de les charger dans le moteur d'analyse, ce qui nous permet de réaliser certains scénarios. Mais il n'est pas possible de réaliser le reste des scénarios...

Pour cela, il serait préférable de se baser sur un système de fichier distribué dans lequel les scrapers viendraient écrire les informations qu'ils récupèrent. Puis à postériori, faire du batch processing afin d'alimenter des vues, chaque vue viendrait répondre à un scénario en particulier.

En effet, les limites de notre système se sont vues lorsqu'il a fallu croiser les données venant de plusieurs scrapers. Ce regroupement d'informations est difficile à faire dans un système où l'on souhaite mettre directement la bonne information dans la bonne table (typiquement, le cas de la construction de la clef pour les tables Cassandra).

Au niveau des technologies, il serait sûrement plus judicieux de monter un cluster Hadoop pour profiter de HDFS, puis faire tourner du map/reduce afin de croiser les informations récupérées, dans le but d'alimenter des vues. Ces vues peuvent être représentées par des tables Cassandra ou autre.

Il est vrai que Cassandra est très performant, mais les API Java / Scala de Datastax sont assez mal documentées, il est assez difficile de les utiliser. Il serait peut être intéressant

de regarder comment HBase peut s'intégrer au système, surtout si le projet passe sous Hadoop.

Puis, depuis ces vues, on récupérerait les données voulues afin de les charger dans Spark, afin de faire les analyses souhaitées.

Malgré tout, nous avons réussi à mener quelques unes de ces analyses à bien, répondant à des scénarios. Ces analyses sont explicitées dans la partie qui suit.

Analyse des données

1. Le machine learning sur Spark

Problématique

Afin d'analyser les données recueillies et stockées, nous avons décidé d'effectuer, comme précédemment pour l'architecture, une veille technologique, en particulier sur les librairies de machine learning disponibles sur Spark. Deux problématiques ont motivé cette décision : la potentielle complexité des modèles à mettre en place au vu des différents scénarios à implémenter et la spécificité de nos données qui sont géolocalisées.

Dans un premier temps, notre veille a porté sur les librairies de machine learning. Spark propose déjà une librairie associée nommée MLLib. Celle-ci a été conçue de manière à être optimisée pour le calcul parallélisé et permet d'appeler les algorithmes directement sur les RDDs. Les algorithmes sont développés en Scala et sont principalement basés sur le package Scala d'algèbre linéaire "Breeze". Mllib propose les fonctionnalités de base et des modèles principalement linéaires, comme la régression logistique, les SVMs linéaires, le classifieur bayésien naïf ou l'ACP. Cependant, pour des données comme des trajectoires de bateaux, nous aurons certainement besoin d'utiliser des modèles relativement complexes et des algorithmes de base ne seraient alors pas suffisants. Nous avons donc décidé d'explorer les librairies supplémentaires pouvant être utilisées. Cela nous permettrait d'avoir accès à un nombre de modèles plus important que ceux proposés par MLLib, qui sont les plus classiques.



Dans un second temps, nous nous sommes penchés sur la spécificité même de nos données : en effet, Spark et MLLib ne permettent pas de traiter directement de données géo-spatiales. Il est donc intéressant pour notre étude de connaître les librairies permettant d'utiliser des données sous forme de coordonnées.

Librairies de Machine Learning

Il y a plusieurs librairies de machine learning proposées sur Spark. Nous avons ici étudié les principales.

Spark ML Context

Spark ML Context est une librairie permettant d'exécuter Apache SystemML depuis Spark, avec Scala, Python ou Java. SystemML un système de machine learning développé par des chercheurs d'IBM en 2010. Il est devenu open-source en 2015. L'objectif premier de ce système est de mettre automatiquement à l'échelle des algorithmes écrits grâce à un langage similaire à R ou Python, pour des grands jeux de données. Il fonctionne aussi bien sur Spark en mode batch qu'en mode standalone.



La librairie Spark ML Context ajoute peu de librairies supplémentaires par rapport à MLLib, les SVM-multiclasses par exemple. Son principal atout semble être plutôt dans sa facilité d'utilisation, son efficacité et sa scalabilité.

Mahout

Mahout est un framework permettant de construire simplement des algorithmes scalables. Il a l'avantage de proposer un panel d'algorithme plus important que MLLib, cependant il se situe sur la couche au dessus de MapReduce et pas Spark comme MLLib. On perd donc l'avantage de l'accès à la mémoire vive qui rend les calculs plus rapides et qui est une des raisons pour laquelle nous avons choisi d'utiliser Spark.



Spark-sklearn

Spark-sklearn est un outil permettant d'intégrer la librairie Scikit-learn avec Spark. C'est une librairie Python open-source, proposant de nombreux algorithmes différents, dont certains permettent de travailler directement avec des données géolocalisées. Elle est développée par de nombreux contributeurs comme l'Inria et Telecom ParisTech.



Scikit-learn n'est cependant disponible que sur la console Python et malgré l'outil d'intégration Spark-sklearn, il n'a pas été conçu pour Spark en premier lieu, comme MLLib.

SparkDBSCAN et DBSCAN

SparkDBSCAN et DBSCAN sur Spark proposent des implémentations de l'algorithme DBSCAN (voir partie 2) pour Spark.

Manipulation de données géolocalisées

Deux librairies sont principalement proposées avec Spark pour répondre à la problématique des données géospatiales: GeoSpark et Magellan. Relativement similaires, elles sont toutes deux disponibles sur GitHub, en open-source, et sont régulièrement mises à jour. On peut également trouver d'autres librairies, comme SpatialSpark par exemple, mais leur développement par la communauté GitHub semble moins important.

Geospark

Geospark propose une extension au RDD Spark avec les SRDD, Spatial Resilient Distributed Datasets, qui permettent de charger puis analyser de grands volumes de données spatiales grâce à des actions parallélisées. Une fois les objets géométriques stockés dans les SRDD, il est possible d'effectuer des opérations géométriques comme des combinaisons de polygones, ou des opérations à caractère spatial, comme l'algorithme des KNN adapté. Geospark permet ainsi de créer des cartes de densité par exemple. Les SRDD peuvent être construits à partir de CSV, TSV, WKT et GeoJSON, ainsi que depuis un RDD Spark depuis une version plus récente. Ils peuvent être sous des formes telles que des points, des rectangles, des polygones et des lignes.

Geospark est disponible pour Java et Scala.



Magellan

Magellan est proposé sur Scala et Python. Il permet de travailler avec des structures de données géométriques, plus complètes qu'avec GeoSpark : point, multipoint, polygone, multipolygon... Elles sont créées à partir de données de dataframe. Quant aux opérations proposées, on y trouve les unions, les distances, les intersections, les différences symétriques...

Choix de librairie :

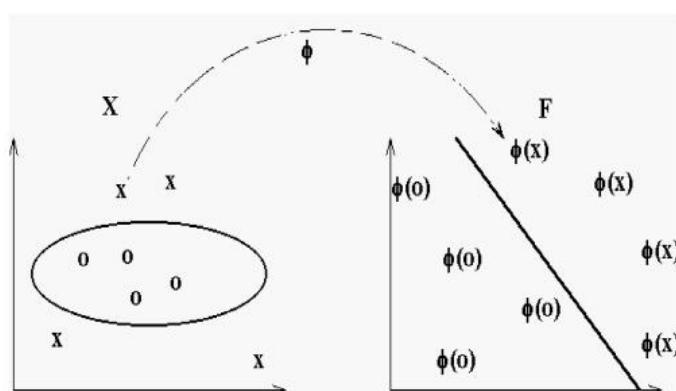
Pour nos premières implémentation de scénario, notre choix s'est porté sur la librairie Skicit-Learn. En effet, elle propose un panel d'algorithmes très varié et en permet une grande liberté d'utilisation, notamment au niveau des paramétrages. Elle est de plus très documentée. Enfin, nous avons réfléchi à des algorithmes précis pour l'implémentation de nos scénarios (voir partie suivante), et Skicit-Learn propose tous les outils nécessaires pour leur réalisation, ce qui n'est pas le cas de toutes les autres librairies. Elle permet d'utiliser directement des coordonnées géographiques, ce qui nous permet de ne pas avoir à passer par une librairie supplémentaire pour répondre à cette problématique.

2. Algorithmes de machine learning pour notre problématique

Les méthodes à noyaux

Les méthodes à noyaux sont des algorithmes populaires en apprentissage statistique pour la reconnaissance de motifs. En effet, elles présentent l'avantage de ne pas faire d'hypothèse sur la forme que prennent les données. De plus, elles sont généralement très efficaces sur les frontières non-linéaires.

Les méthodes à noyaux permettent de transposer les données de leur espace d'origine à un espace à plusieurs dimensions, où les différentes classes seront plus facilement séparables par un classifieur linéaire.



Le produit scalaire des observations est généralisé par une fonction de la forme $K(x_i, x'_i)$, qui assigne un poids à x_i en fonction de sa distance à x'_i . Cette fonction K est le noyau, qui quantifie donc la similarité entre deux observations.

Un des noyaux fréquemment utilisé est par exemple le noyau gaussien :

$$k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x} - \mathbf{y}\|^2 / c}.$$

$$K_\lambda(x_0, x) = \frac{1}{\lambda} \exp \left[-\frac{\|x - x_0\|^2}{2\lambda} \right]$$

L'affectation de poids est alors basée sur la fonction de densité gaussienne ci-dessous.

On observe que plus la distance euclidienne entre x et x_0 est grande, plus le poids donné sera faible, avec λ supérieur à zéro. Ce dernier paramètre correspond à la variance de la densité gaussienne et permet de contrôler le périmètre accordé au voisinage.

A partir d'ici, on peut avoir l'intuition qu'une méthode à noyau pourrait ici nous permettre d'effectuer une carte de densité à partir de nos données. En effet, cette méthode est adaptée à la non-linéarité de nos données et se base sur un poids accordé selon une distance par rapport à un point.

Parmi les méthodes à noyaux, on trouve en particulier les Support Vector Machines (SVMs), une méthode utilisée pour la classification mais également la régression et très efficace en grande dimension. Les SVMs ont pour but de maximiser l'espace entre les différentes classes, autour de l'hyperplan séparateur dans l'espace du noyau. Cet espace est appelé "marge". Plus celle-ci est importante, plus faible seront les erreurs de classification.

One-classe SVM

Le one-class SVM est un SVM avec pour particularité de se baser sur un ensemble d'apprentissage ne contenant qu'une seule classe.

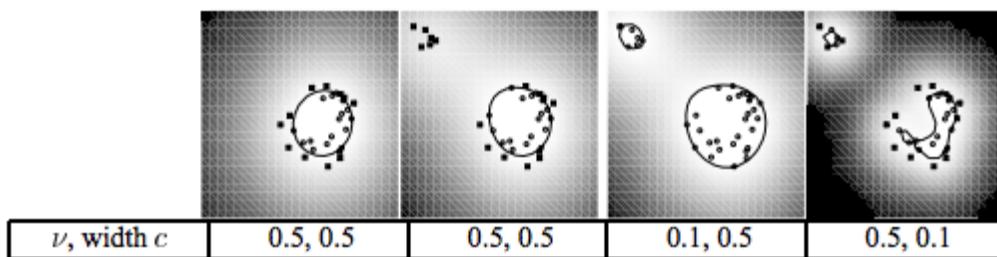
L'algorithme retourne une fonction f qui retourne +1 dans la région englobant le plus de points, -1 sinon. L'apprentissage se fait donc sur une classe seulement, qui est la classe "normale". Toutes les données dont les propriétés ne respectent pas l'exemple normal sont considérées comme des anomalies. Ainsi, les données sont projetées dans l'espace

du noyau et séparées par une marge la plus grande possible: la valeur de $f(x)$ pour chaque point x est déterminée selon de quel côté de la frontière de décision celui-ci se trouve. Ce qui est considéré comme les "margin errors" en classification binaire devient des "outliers" quand on ne considère plus qu'une classe. Les points qui "soutiennent" la marge qui permet de classer les points comme des outliers ou non se nomment les "support vector", c'est-à-dire qu'une modification d'un de ces points va modifier la marge. Parmi les noyaux utilisés, on trouve le noyau Gaussien ainsi que d'autres noyaux permettant des frontières non linéaires comme le noyau radial ou le noyau polynomial au d -ème degré.

Le one-class SVM est donc un algorithme utile pour la détection d'anomalies. Quand toutes les observations sont positives, c'est-à-dire qu'il n'y a pas d'erreur de classification, le one-class SVM peut se généraliser en estimateur de densité. En effet, plus les points considérés comme des anomalies sont loin de la frontière de décision, plus la densité est faible.

Un premier paramètre important, ν , permet d'indiquer le compromis entre le nombre de points outliers et le nombre de support vectors. Ce chiffre, compris entre 0 et 1, représente la borne supérieure de la fraction d'outliers et la borne inférieure de la fraction de vecteurs de support.

Un autre paramètre utilisé dans ce modèle est celui qui régule la taille du noyau, appelons-le c . Il permet de prendre en compte plus ou moins de clusters différents. Ainsi, en diminuant la taille du noyau, on augmente le nombre de contours déconnectés et on obtient un nombre de clusters plus important.



Influence des paramètres ν et c

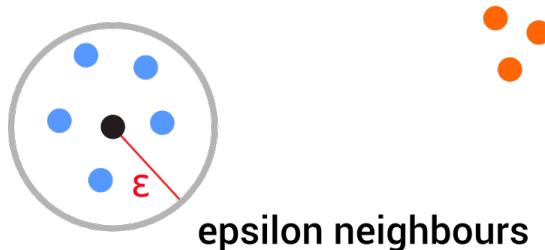
Malgré la difficulté pour trouver les bonnes valeurs pour les paramètres et la complexité relative de l'algorithme, l'avantage des SVC est qu'on peut utiliser des espaces de dimension infinie. On peut donc détecter toutes formes de clusters, même les plus

complexes comme un cluster en forme d'anneau par exemple, qui aurait posé problème avec les k-plus-proches-voisins ou les k-means.

L'algorithme DBSCAN

DBSCAN est un algorithme de clustering des données basés sur la densité spatiale. Deux paramètres sont nécessaires : une distance epsilon, ainsi qu'un nombre de voisins minimum qui doivent être présents dans ce périmètre afin de le considérer comme un cluster. Le premier point à comparer est choisi aléatoirement. Si son nombre de voisins est suffisant pour créer un cluster, ce point se nomme le "core point". Les autres points du cluster qui ne réunissent pas un nombre de voisins suffisant se nomment les "border points".

Les paramètres jouent un rôle important : plus le périmètre est resserré et plus le nombre minimal de points est grand, plus la zone recherchée est dense.



Ainsi, DBSCAN permet de repérer des régions à fortes densités, séparées par d'autres régions moins denses. Les points n'appartenant à aucun cluster sont les "noise points".

DBSCAN est un algorithme simple qui de plus ne nécessite aucune hypothèse sur le nombre de clusters. Il est ici moins adapté que l'algorithme de one-class SVM pour créer une carte de densité, car il n'y a ici qu'un niveau noise points, ils ne sont pas hiérarchisés en fonction de leur distance au cluster à partir du moment où celle-ci est supérieure à epsilon. Il peut néanmoins se révéler intéressant dans la recherche de routes récurrentes en mettant en évidence les chemins dont la densité est la plus forte, c'est-à-dire les routes les plus empruntées pour un parcours donné.

3. Génération d'une carte de densité

L'algorithme de one-class SVM est proposé par la librairie Skicit-Learn. Son implémentation est basée sur la librairie open-source libSVM, développée par la National Taïwan University.

Sa complexité est de $O(\text{nombre_de_paramètres} * (\text{nombre_d'individus})^2)$ ou $O(\text{nombre_de_paramètres} * (\text{nombre_d'individus})^3)$ selon l'utilisation du cache.

Nous avons ici décidé de l'utiliser dans un premier temps pour générer une carte de densité des navires.

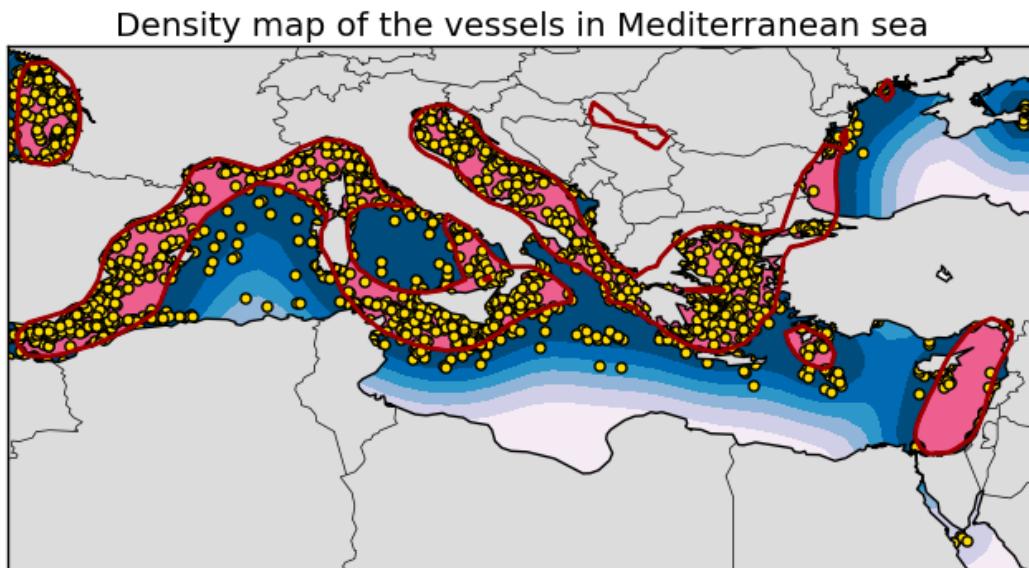
Pour cela, nous avons utilisé la fonction `svm.OneClassSVM`.

Afin d'améliorer notre modèle, nous avons joué sur plusieurs paramètres proposés par la fonctions :

- **Kernel** : Ce paramètre permet de spécifier la forme du noyau voulue. Celui-ci peut être linéaire, polynomial, radial, sigmoïde, ou être défini de manière "manuelle"
- **nu** : Ce paramètre permet de définir le rapport entre le nombre de vecteur support et le nombre d'outliers, comme expliqué dans la section précédente. Il est compris entre 0 et 1 et est de 0.5 par défaut.
- **Degree** : Degree permet de donner une valeur au degré du polynôme dans le cas où le noyau est polynomial.
- **Gamma** : Gamma est le coefficient du noyau, comme vu précédemment. C'est ce coefficient qui va jouer sur le nombre de clusters.

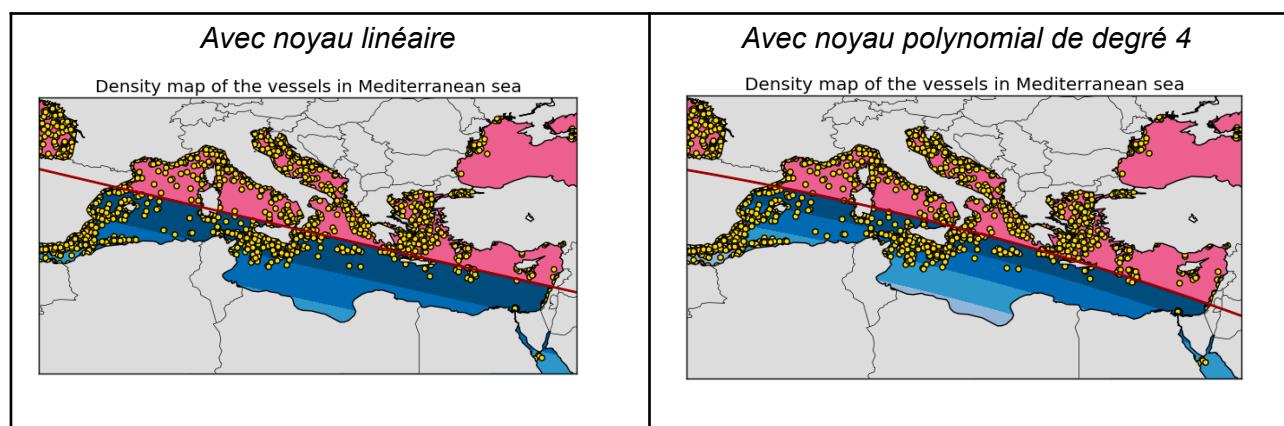
En affinant les paramètres au fur et à mesure, nous sommes finalement parvenus au meilleur résultat avec un noyau radial (voir formule ci-dessous), pour $\nu=0.3$ et $\gamma=0.2$.

$$\exp(-\gamma|x - x'|^2)$$



Noyau radial, $\nu=0.3$, $\gamma=0.2$

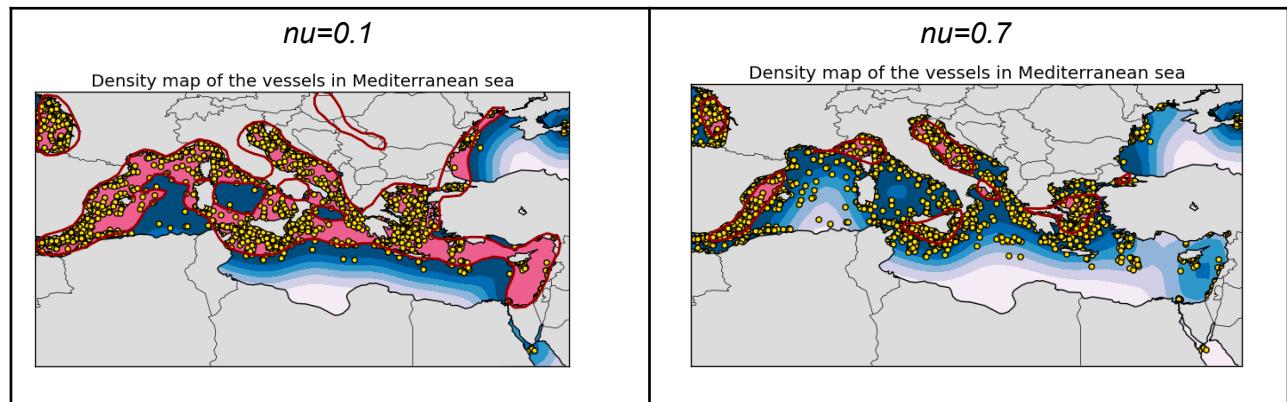
Le noyau radial est ici essentiel car la densité n'est représentable qu'avec une forme particulière et de forme très arrondie. Ci-dessous, des exemples de frontières plus linéaires avec un noyau linéaire et un noyau polynomial de degré 4, pour les mêmes paramètres $\nu = 0.3$ et $\gamma=0.2$.



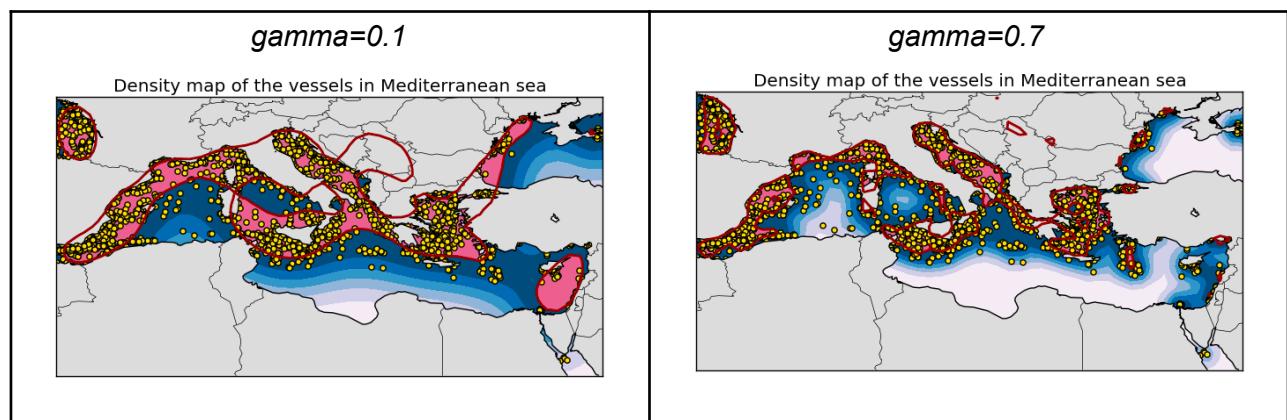
Nous pouvons également remarquer l'importance du choix des paramètres.

Un ν faible va ainsi considérer plus de points comme des vecteurs supports et donnera une carte de densité moins précise, considérant plus facilement une zone comme dense. À l'inverse, un ν très élevé sera très sélectif sur le nombre de points nécessaires pour constituer une densité forte. C'est alors un paramétrage adapté si l'on veut faire apparaître seulement quelques zones qui ont vraiment la densité la plus forte.

Ci-dessous, deux exemples de one-class SVM avec un noyau radial et un $\gamma=0.3$.



Un γ large va entraîner une plus grande sensibilité par rapport aux changements entre les support vectors. Cependant, un γ trop grand peut donc entraîner un surapprentissage, en créant des clusters avec un faible nombre de points.



4. Génération d'une carte de densité des vaisseaux pour une destination donnée

Une fois effectuée la génération de cette première carte de densité. Nous avons estimé intéressant d'essayer de générer une carte de densité pour les bateaux allant à un port en particulier. Cette démarche est selon nous intéressante pour plusieurs raisons :

- Lier un vaisseau à un port de destination pour un trajet en particulier est une première étape à la détermination des "routes récurrentes".
- Une zone à forte densité est une zone très empruntée. Ne pas ne pas être dans une zone récurrente lorsque l'on prétend se diriger vers le port considéré est anormal. Cette étude de densité pour un port de destination fixé est ainsi une première introduction à la notion de "route anormale".

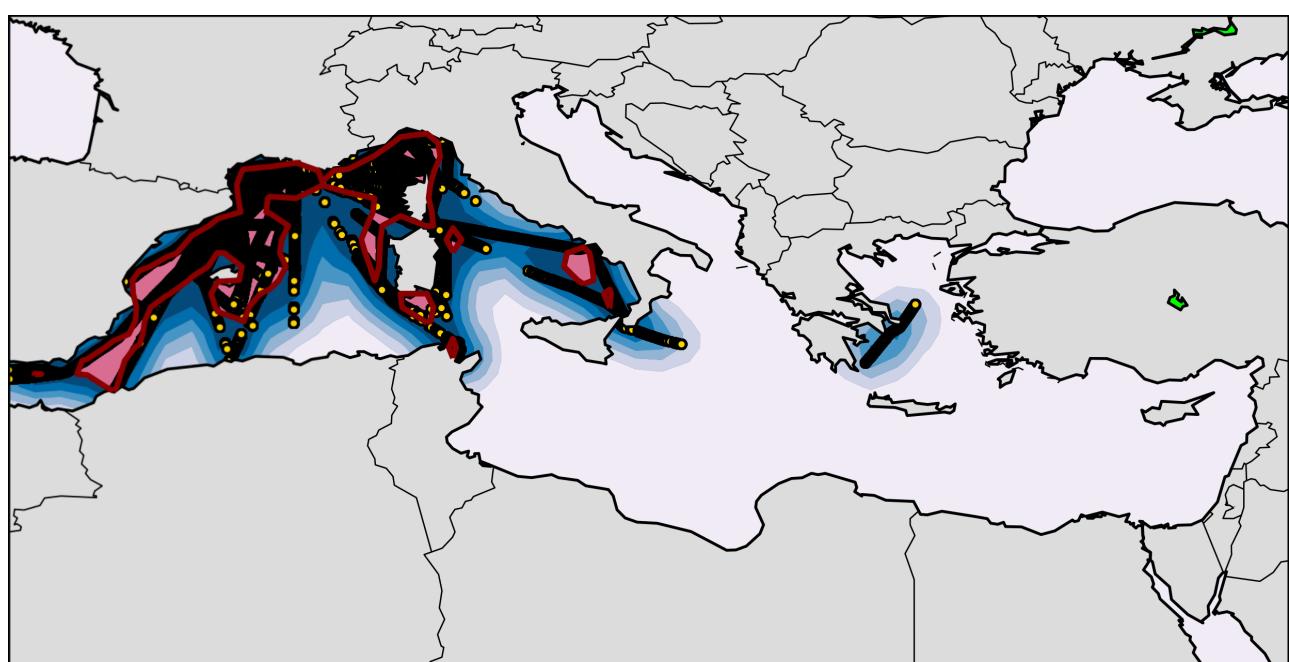
Intéressons nous à la réalisation de cette map. La principale difficulté consistait à lier les positions géographiques à une destination. Pour cela nous avons procédé comme suit :

Pseudo Algorithme :

```

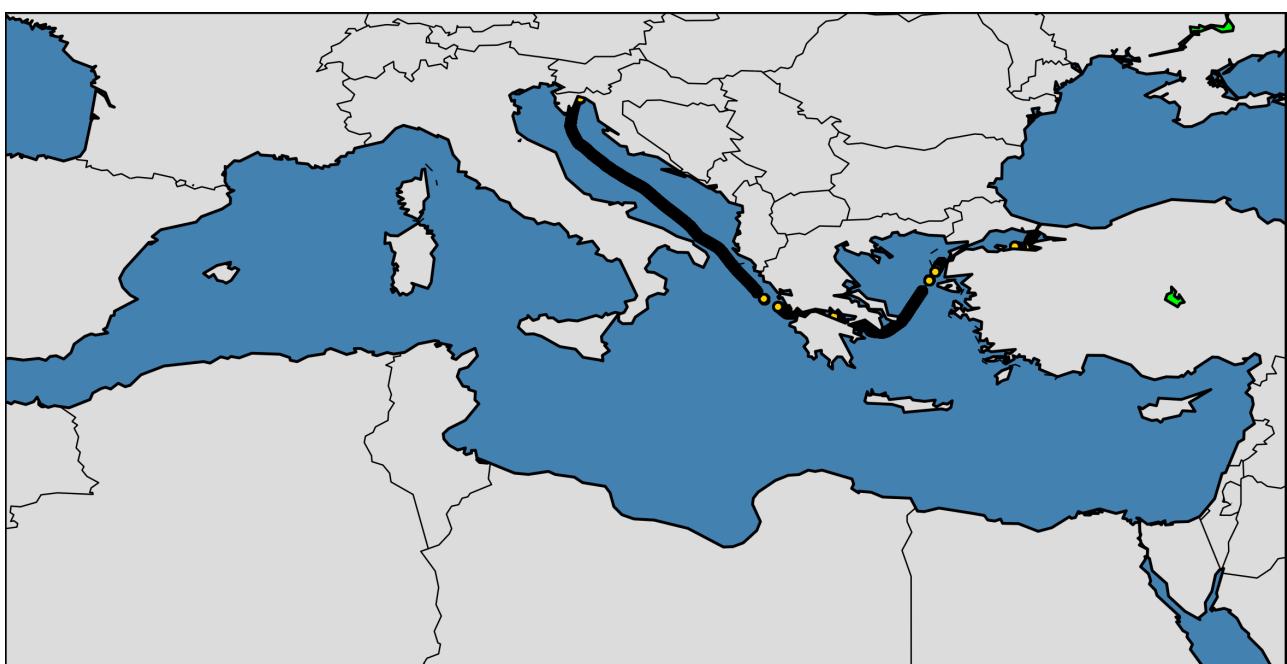
select every (mmsi, date) couple in the daily destination log table
matching the considered destination;
for each (mmsi, date) selected:
    get every (long, lat) couple for the boat with matching mmsi, at
    the matching date;
```

Nous récupérons ainsi tous les points appartenant à un trajet ayant pour destination la destination considérée. Nous utilisons par la suite la SVM sur ces points et pouvons obtenir une figure similaire à la figure suivante, obtenue en appliquant l'algorithme à la destination "MARSEILLE" :



5. Algorithme annexe : suivi d'un vaisseau en particulier

Parmi les modules rendus, nous avons développé un module permettant de retracer la trajectoire d'un bateau en particulier en fonction de son MMSI, grâce à ses positions successives. Le rendu, par exemple, du tracking du vaisseau avec pour MMSI 256510000 est le suivant :



6. Algorithmique : Perspectives

Ces six mois ont été l'occasion pour nous de réellement "dégrossir" les objectifs fixés par notre client. Nous avons eu l'occasion de travailler sur l'ensemble des phases de la constitution du dataset, à son stockage puis son analyse.

Cela accompli, nous avons proposé, "forts de notre expérience", des perspectives d'amélioration de chacune de ces phases. Voici une ébauche des idées que nous avons pour le futur de la partie algorithmique.

Court terme : Routes récurrentes / anormales

Le prochain chantier majeur pour ce projet va être l'implémentation de la détection des routes récurrentes / anormales. Selon nous, deux composantes seront nécessaires à cette réalisation :

- Étendre la notion de destination vers une notion de trajet (couple départ - destination)
- Choisir le bon algorithme de machine learning

Extension de la notion de destination vers la notion de trajet :

Cela nous semble réalisable via deux méthodes. La première d'entre elle consiste à trouver une nouvelle source de données permettant d'obtenir le port de départ et d'arrivée des vaisseaux (pour rappel, à partir de vesselfinder, nous ne disposons que du port d'arrivée).

La solution alternative serait de post-traiter les données actuelles selon l'algorithme suivant :

Pseudo algorithme :

```
For each vessel :  
    get every position since the beginning of time;  
    For each position:  
        match position to date from destination table;  
        departure point is previous destination port (i.e. last  
        destination port different from the current one, if exists);  
        destination point is current destination port;
```

Choix du bon algorithme :

Nous avons explicité précédemment nos résultats de veille concernant les algorithmes nous paraissant intéressants pour les scenarii envisagés.

Dans le cas des routes récurrentes / anormales, nous pensons que l'algorithme DBSCAN est une piste intéressante.

Long terme : propositions

A plus long terme, un certain nombre d'algorithmes et améliorations sont envisageables.

Voici quelques propositions :

- Intégration de la dimension temps : à l'heure actuelle, les algorithmes que nous proposons sont principalement basés sur des densités de points sur une zone de l'espace. Le paramètre temps n'entre pas du tout en considération dans nos algorithmes. Il pourrait être intéressant d'intégrer le paramètre temps dans les analyses, et réellement appliquer un traitement orienté "spatio-temporel" à nos données : analyse de la vitesse des vaisseaux, de leurs arrêts, détection d'arrêts anormaux, optimisation des temps de navigation, etc...
- Extension à une plus grande zone géographique / un plus grand jeu de données pour faire des tests de scalabilité des algorithmes
- Prédiction : ajouter un aspect prédictif à l'outil. L'outil serait alors en mesure de prédire avec un certain taux de certitude les trajectoires des vaisseaux / leur position au cours du temps
- ... les pistes d'exploration sont encore vastes et variées.

Conclusion

Ce projet a été l'occasion pour notre groupe d'effectuer un premier pas dans l'univers des technologies du Big Data & Data Warehousing, dont l'aspect technique est encore peu enseigné à l'UTC.

Nous avons ainsi pu revêtir les casquettes de l'ingénieur Big Data, et du Data Scientist, au cours des différentes phases, en accompagnant notre Dataset de sa naissance à son exploitation, c'est à dire de sa construction à son stockage puis son analyse.

Ce projet a été un réel challenge dans la mesure où nous le commençons avec des connaissances d'étudiants (c'est à dire très restreintes), et avons dû assimiler sur une période de six mois, différentes technologies et méthodes afin d'arriver à nos fins.

Nous avons ainsi buté plusieurs fois sur des difficultés liées à notre manque de connaissances dans divers domaines, mais avons su prendre notre mal en patience, et les surmonter par la persévérance.

Nous sommes finalement fiers d'avoir su livrer ce projet, composé de ses quatre parties principales (veille, scraper, architecture Big Data et algorithmique), et satisfaits des connaissances engrangées au cours de ce semestre grâce à PRDW.

Bibliographie

Breeze

<https://github.com/scalanlp/breeze>

Apache System ML

<https://apache.github.io/incubator-systemml/index.html>

Mahout

<https://mahout.apache.org/users/basics/algorithms.html>

Spark SKlearn

<https://github.com/databricks/spark-sklearn>

SparkDBSCAN

https://github.com/alitouka/spark_dbSCAN

dbscan-on-spark

<https://github.com/irvingc/dbscan-on-spark>

GeoSpark

<http://geospark.datasyslab.org/>

Magellan

<https://github.com/harsha2010/magellan>

Spark sk-learn

<https://github.com/databricks/spark-sklearn>

Scikit-learn

<http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM>

Méthodes à noyau

[The Elements of Statistical Learning](#), Livre de Jerome H. Friedman, Robert Tibshirani et Trevor Hastie

Estimating the Support of a High-Dimensional Distribution, Article de Bernhard Schölkopf, John C. Plattz, John Shawe-Taylor, Alex J. Smolax et Robert C. Williamsonx

DBSCAN

A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Article de Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu