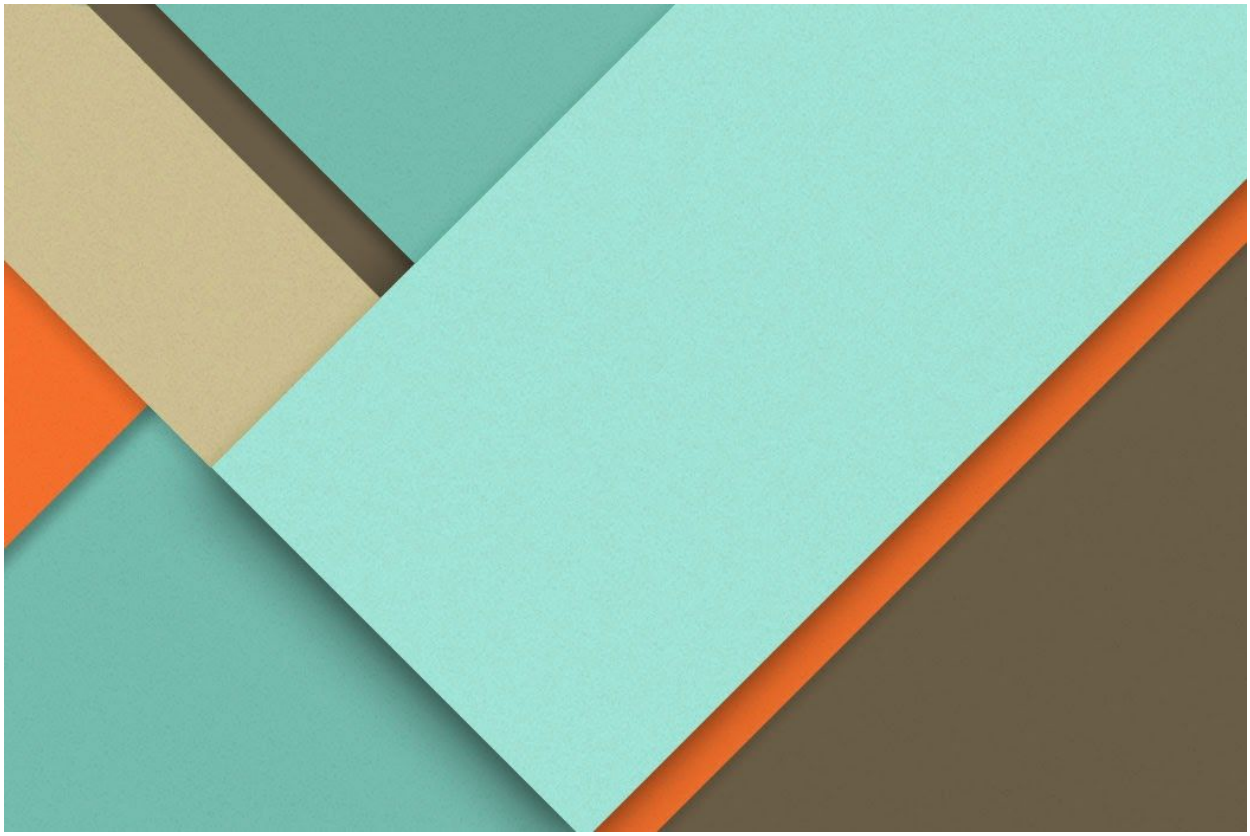


Outil de calcul des risques à l'ouverture d'un contrat d'assurance vie

14.02.2020

Théo Dupuis - Clément Frade - Yann Moulaire
Télécom Saint-Etienne - Projet Big Data

Objectifs

L'objectif global de ce projet est de déterminer, pour chaque potentiel client d'une compagnie d'assurance, le niveau de risque associé à la prise en charge de ce client. Ce niveau sera évalué par un facteur de risque allant de 1 à 8 calculé à partir des données informatives sur le client (données médicales, sexe, ascendance ethnique, etc). Pour ce faire nous avons découpé ce projet en 5 étapes.

1. Importer les données des clients sur une VM Hadoop
2. Faire un script pour récupérer les données en local depuis la VM Hadoop.
3. Chiffrer puis pousser les données sur une machine AWS.
4. Développer en Python un algorithme de machine Learning pour analyser les données. Le modèle développé permettra d'obtenir le risque d'un contrat client. Cet algorithme devra s'exécuter sur une machine AWS.
5. Prédire les risques du fichier predict.csv à l'aide de notre modèle. Le format de retour, sera un fichier CSV contenant pour chaque ligne, la ligne correspondante de predict.csv ainsi que l'indice de risque prédit par notre modèle. Ce fichier sera conservé dans la machine AWS.
6. Écrire un script permettant de récupérer Le fichier créé à l'étape 4 et le charger dans une base de données NoSQL locale.

Grandes étapes

I. Gestion des données au travers d'hadoop

Pour ce projet nous avons choisi d'installer les données sur HDFS d'hadoop, pour ce faire nous avons installé une sandbox hadoop (bitnami-hadoop-3.2.0-8 avec un Os linux debian 9) et l'avons utilisé par le biais d'Oracle VirtualBox. Une fois ceci effectué, nous pouvons accéder à un tableau de bord Ambari qui nous communique tous les indicateurs de la VM (quelles service fonctionne et leur état ainsi que les métriques liées) (voir Figure 1).

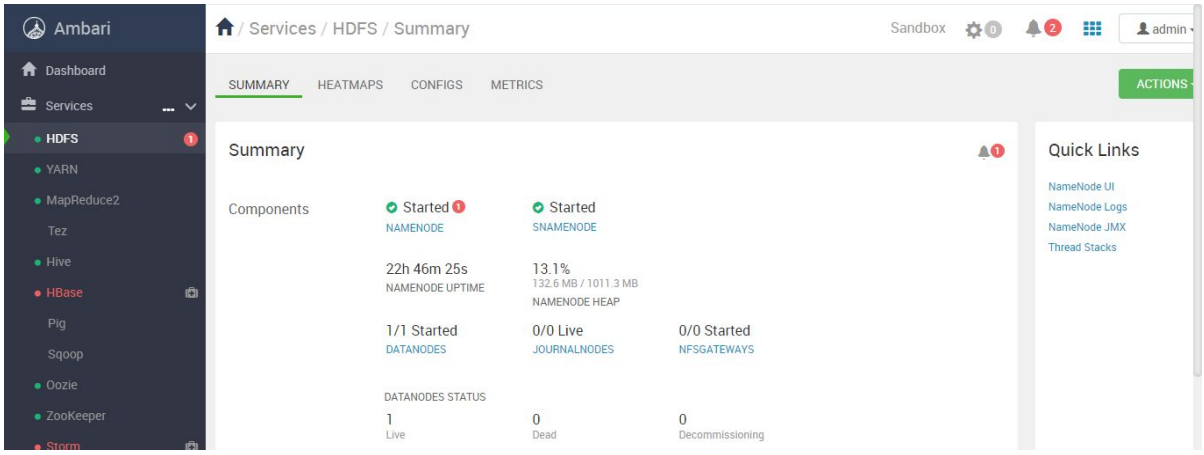


Figure 1 : tableau de bord Ambari

L'utilisateur pourra ensuite, grâce à cette interface, accéder au système de fichier HDFS pour y déposer ou télécharger des fichiers. On peut voir dans notre exemple que nous avons déposé deux fichiers dans un dossier Projet BigData créé par nos soins (voir Figure 2).

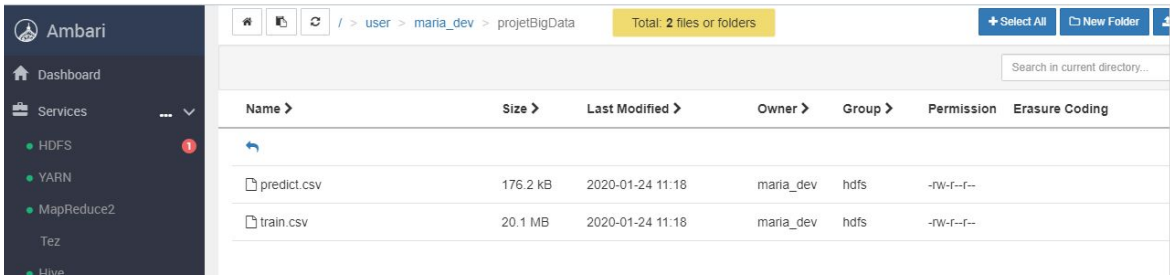


Figure 2 : System de fichier HDFS (Interface graphique)

Ces fichiers pourront être déposés ou téléchargés également par le biais du terminal disponible en connection SSH. Pour accéder au terminal en SSH deux possibilité s'offrent à nous: via la page du localhost 4200 ou en créant une connexion SSH dans notre terminal (machine physique Windows) sur le port 2222, en nous identifiant (commande : ssh root@localhost -p 2222 avec root représentant le nom d'utilisateur).

Une fois les fichiers importés sur la VM Hadoop nous devons ensuite trouver un moyen de transférer ces données vers un bucket AWS (bucket S3) après les avoir récupérées en local sur la VM hadoop. Nous avons donc, dans un premier temps, installé tous les outils nécessaire (Python et Boto) sur la VM hadoop grâce au terminal disponible en SSH. Nous avons ensuite réalisé un script (que vous retrouverez en annexe dans le répertoire convenu). Ce script va d'abord copier les données en local depuis hadoop dans un dossier que nous avons choisi. Ensuite notre client va envoyer chacun des fichier vers notre bucket. Le résultat nous convient et les données sont effectivement transféré sur notre bucket.

II. Le cloud AWS

A. EC2 - Instance

Le but principal de l'utilisation du cloud d'Amazon AWS est de pouvoir réaliser les calculs sur les données directement depuis une machine virtuelle se trouvant sur ce cloud. Pour cela, on utilise une instance EC2 à laquelle on accède en SSH depuis nos propres machines. On a alors mis en place, sur cette machine, un notebook Jupyter après avoir installé Python 3 et Boto 3 pour échanger avec le module S3 de AWS.

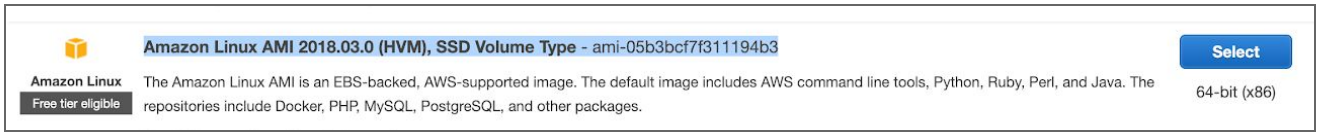


Figure 3 : Type d'instance sélectionnée dans AWS

B. S3 - Storage

On utilise S3 pour stocker les deux fichiers importants pour l'analyse, c'est à dire predict.csv et train.csv. Pour cela, on a créé un bucket "pbd_cty" dont les données contenues sont automatiquement chiffrées côté serveur en utilisant des clés gérées par Amazon S3. Les données sur ce bucket vont pouvoir être récupérées en Python grâce à Boto 3 qui nécessite, dans un dossier ".aws" sur la machine virtuelle, un fichier "credentials" qui contient les autorisations d'accès au bucket.

Figure 4 : Interface d'un bucket S3 dans AWS

III. Traitement des données

A. Prétraitement

Les données étant initialement contenues dans des fichiers ".csv", on commence par les transformer en des Dataframes avec Panda. Avant d'utiliser les classifieurs, on conditionne les données. On encode les différents labels (les chaînes de caractères deviennent des nombres). On utilise ensuite un scaler pour normaliser les dimensions. On utilise ensuite une PCA pour réduire le nombre de dimensions.

B. Tests de solutions

1. K-Nearest Neighbors

Le premier algorithme que nous avons testé est celui des K-plus proches voisins. Chacune des dimensions prises en compte par l'algorithme correspond à une colonne dans le tableau des données prétraitées. Nous avons testé avec un nombre K variant entre les valeurs 5, 25, 50, 100 et 190.

2. Logistic Regression

L'algorithme suivant est la régression logistique pour laquelle on fait varier le paramètre C entre 0.2, 1 et 5 qui correspondent à des forces de régularisation plus (0.2) ou moins (5) grandes.

3. Decision Tree

Nous avons ensuite testé d'analyser les données avec des arbres de décision (en régression) où le paramètre variant est le nombre minimum d'échantillons requis pour diviser un noeud interne. Il va varier entre 2, 50, 100, 725, 800 et 900.

4. Support Vector Classifier

On utilise ensuite un Classifieur à support de vecteur où le paramètre changeant est C qui correspond, comme pour la régression logistique, à l'inverse de la puissance de régularisation. C va varier entre 0.1, 1 et 10.

5. Multi-Layer Perceptron

La dernière solution que nous avons testé est l'utilisation d'un réseau de neurone pour la classification des données. Pour cette solution, on teste trois solveurs différents prenant comme valeurs "adam", "sgd" et "lbfgs". "lbfgs" est un optimisateur de la famille des quasi-Newton, "sgd" correspond à une descente de gradient stochastique et "adam" est un optimisateur se servant de cette même méthode.

C. Résultats

algorithme	paramètre	test	train
KNN	5	0,1639	0,4594
	25	0,2688	0,3355
	50	0,2788	0,3145
	100	0,2765	0,299
	190	0,2719	0,2855
Regression Logistique	C=1	-0,1933	-0,1754
	0,2	-0,1935	-0,1755
	5	-0,1933	-0,1755
Arbre de décision	2	-0,4224	1
	50	0,0776	0,5106
	100	0,1612	0,4185
	725	0,2399	0,279
	800	0,2378	0,276
	900	0,2342	0,2706
Machine à support de vecteur	1	-0,2307	-0,217
	0,1	-0,2302	-0,2167
	10	-0,23	-0,2165
Perceptron Multi-couches	Adam	-0,0801	
	SGD	-0,0817	
	LBFGS	-0,064	

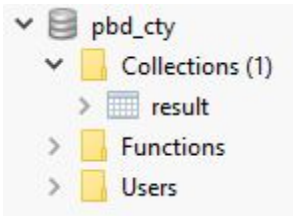
Figure 5 : Résultats des différents algorithmes testés

Les résultats présentés sont le coefficient de détermination des différentes méthodes utilisées. Les meilleurs résultats sont pour KNN et les arbres de décision. Les autres méthodes ne donnent pas de résultats satisfaisant.

IV. Stockage des données résultats

A. Importation dans MongoDB

Sur une machine en local (Windows), nous avons réalisé un script en Batch qui récupère les données depuis la VM sur le cloud AWS. Le script importe ensuite les données, initialement dans un fichier ".csv", dans une base de données qu'il crée sur MongoDB. On obtient alors dans la collection, pour chaque ligne du fichier, un objet contenant chacune des valeurs des colonnes qui lui correspondent ainsi que son résultat.



B. Scripts de recherche des données

Étant donné la façon dont sont importées les données, il est très simple d'accéder uniquement aux entités pour lesquelles le résultat correspond à un nombre en particulier. Par exemple :

```
db.getCollection('result').find({ result : 1})
```

Cette expression retourne la liste des potentiels clients pour lesquels le risque à la prise en charge est le moins élevé.