**Synthèse Chapitre 2 : IA**

Analyser et regarder le problème :

En premier, il faut savoir exactement ce que la société veut donc son objectif business et ensuite ce que la société souhaite retirer comme bénéfice et l’utilité du modèle. Cela permettra de vraiment pouvoir déterminer et savoir dans quel sens partir. Ensuite il faut voir si qqch existe déjà s’il y a déjà des modèles existant ou une solution existante. Pipeline est une séquence de données qu’on appelle component. Les components travaillent de manière asynchrone, chaque component prend un grand nombre de données et ensuite travaille dessus pour ensuite mettre le tout dans un data store. Et ensuite ce data store peut être utilisé par un autre component. Il faut aussi choisir et voir pour le problème s’il est supervisé, non-supervisé ou qu’il faut mettre du Reinforcement Learning. Si on doit utiliser une solution en ligne ou alors sur une machine en batch. Il faut ensuite sélectionner la mesure de performance. Soit RMSE (Root Mean Square Error) ou alors MAE (Mean Absolute Error). Le RMSE est plus souvent utilisé.

Récupérer les données :

La plupart du temps, les données sont en CSV donc il faut les télécharger et ensuite les décompresser avec « tar xzf \*.tgz ». Lorsqu’on a les données il faut regarder les données pour voir ce dont on a accès. Ex : info(), value\_count(), describe(), mean, min, max, hist() Ensuite il faut créer un ensemble de test pour pouvoir ensuite faire nos tests. Pour créer cet ensemble, on prend une partie des données qu’on a télécharger avant. Et on split avec split\_train\_test ou train\_test\_split.

Découvrir et visualiser les données :

Pour découvrir et visualiser, on va faire des graphiques pour voir ce qu’on possède. On utilise donc le .plot(kind= «scatter»,x=«longitude»,y=«latitude»,alpha=0.1). Kind => sorte de schéma que l’on veut, x et y le nom sur les axes et alpha c’est pour l’opacité pour mieux voir les endroits denses. Il y a aussi s => pour la taille des cercles, c => pour la couleur, cmap pour les couleurs prédéfinies. Et plt.legend() crée une légende avec les infos. On va alors ensuite chercher les corrélations dans le schéma. On utilise alors .corr(). Cela varie entre -1 et 1. On peut aussi utiliser les outils panda en faisant from pandas.tools.plotting import scatter\_matrix. Cette fonction permet de faire tous les tableaux possibles. C’est très bien mais il y a certaines données qui doivent aller ensemble et que l’on peut combiner pour avoir d’autres infos intéressantes. Par exemple, le nombre de personne et le nombre de maison pour savoir en moyenne le nombre de personnes par maison.

Préparer les données pour les algorithmes du Machine Learning :

Maintenant, on va préparer les données. C’est important d’écrire des fonctions pour le faire car quand on aura de nouvelles données ce sera plus rapide de réutiliser ce qui a été fait. Pour nettoyer, les données on va mettre des valeurs comme par exemple zéro ou alors la moyenne, … pour cela on utilise dropna(), drop(), fillna(). Bien sûr de temps en temps on a des données qui manquent et pour cela on va utiliser l’Imputer qui se trouve dans from sklearn.preprocessing import Imputer. Dans son constructeur on a besoin de rajouter sa stratégie (zéro, moyenne, …) Imputer crée lui-même une instance statistics\_ dans laquelle il conserve toutes les statistiques qu’il a calculé. Les algorithmes de Machine Learning préfèrent les chiffres et du coup si on a des lettres comme des string par exemple on peut utiliser le transformer de Scikit-Learn qui est le LabelEncoder qui change les éléments en chiffres et les mêmes strings on les mêmes chiffres. Pour avoir le nom de chaque élément on fait print(encoder.classes\_). Si on souhaite créer nos propres transformers il faut alors qu’on crée une classe et qu’on implémente 3 méthodes => fit(), transform() et fit\_transform() . Plus on automatise des préparations de données, plus on va pouvoir essayer de nouvelles combinaisons plus facilement. Les algorithmes de Machine Learning ne travaillent pas bien avec des grandes différences de chiffres du coup on va tout mettre à la même échelle. Soit on utilise le min-max scaling (ou normalisation) soit on fait de la standardisation. Pour le min-max on doit arriver à des valeurs entre 0 et 1. Et il existe le transformer MinMaxScaler. Pour la standardisation, on n’a pas de bornes. On peut utiliser le StandardScaler. Pour faire plusieurs transformations en 1 coup on peut utiliser un Pipeline qui va contenir chaque transformers en tant que component.

Choisir le modèle et l’entrainer :

Pour l’entrainer on a plusieurs modèles (Linear Regression, voir autres précédemment). Ensuite on fait un .fit pour pouvoir entrainer le modèle avec les features et les labels. Pour avoir de meilleures prédictions on peut entrainer l’algorithme avec de meilleures features ainsi que réduire les contraintes avec le modèle. Un autre modèle qui existe aussi est le DecisionTreeRegressor, c’est un modèle puissant et capable de trouver des relations complexes dans les données. Pour pouvoir retester de façon plus pointilleuses le modèle on va justement encore spliter l’ensemble d’entrainement en deux ensembles un d’entrainement et l’autre de validations. Et ainsi de suite mais heureusement il y a la fonction from sklearb.model\_selection import cross\_val\_score qui permet justement de séparer de façon aléatoire et de diviser en un certain nombre, passer en paramètre, de fois. RandomForestRegressor, il fonctionne en entrainant plein de DecisionTree.

Affiner le modèle :

On doit alors affiner le modèle pour que l’on perçoive vraiment ce que l’on souhaite. On utilise dans ScikitLearn GridSearchCV. Il va donc évaluer plein de combinaisons différentes. Il existe aussi une constante qui contient le grid\_search.best\_estimator\_. Il existe aussi le Randomized Search qui permet d’évaluer un certain nombre aléatoire de combinaisons en sélectionnant une valeur aléatoire pour chaque paramètre à chaque itération. Pour tester le modèle on va prendre les set de test et on va les faire passer dans nos modelés déjà créer. On aura donc une prédiction et sur celle-ci on peut alors calculer le Rmse. Pour cela, il faut d’abord faire le mse. Final\_mse = mean\_squared\_error(y\_test, final\_predictions) ; Final\_rmse = np.sqrt(final\_mse)

Présenter sa solution :

Pour présenter sa solution, on doit préciser ce que l’on a appris, ce qui a fonctionné et ce qui n’a pas fonctionner, quel choix ont été fait, et quel sont les limites du système qui a été créé. Il faut tout documenter et créer une belle présentation du tout avec des belles représentations.

Lancer, monitorer et maintenir son système :

Maintenant on est prêt à lancer le système. Pour cela il faut qu’elle soit prête à lancer en production mais il faut d’abord écrire des tests et mettre des sources de données dans le système. On a aussi besoin d’écrire du code qui va faire du monitoring pour pouvoir vérifier en direct ce que le système est en train de faire en live mais aussi mettre des triggers afin que l’on soit directement notifié en cas de disfonctionnement. Il va falloir aussi quand même inspecter les données entrantes pour vérifier que tout est bien et que la qualité des données y est. Il faut aussi pour pouvoir maintenir le système entrainer le système avec des données récentes. Pour cela, le mieux est d’avoir des fonctions pour automatiser tout cela. Si le système est en ligne alors il faut avoir un système pour pouvoir sauvegarder au cas ou on aurait besoin de faire un rollback.