Complément partie **Machine Learning (1/2)**

- ❑ **Machine Learning (1/2) :**
- ❑ Objectif : Prédire qu'un repository devrait devenir massivement adopté

Question 1)

Disons que les repository massivement adoptés (dits MA dans la suite) sont ceux qui sont dans les X% les plus starrés (cf event WatchEvent).

Réponse 1)

Étapes de Création de la variable MA:

Variables sélectionnées et crées à partir de la Table Sample_repo (Mais rigoureusement voir tables githubarchive(year, month, day))

- **WatchEvent :** is related to starring a repository, not watching (https://developer.github.com/v3/activity/events/types/#watchevent)
- **Watch_count**

Codes ayant permis de generer **les Tables sample_files and sample_contents.**

Top 400k GitHub repositories with more than 2 stars received during Jan-May 2016.
This set is used to select entries for **sample_files and sample_contents.**

```
SELECT   repo_name,   COUNT(*) watch_count
FROM
  [githubarchive:month.201605],
  [githubarchive:month.201604],
  [githubarchive:month.201603],
  [githubarchive:month.201602],
  [githubarchive:month.201601]
WHERE type="WatchEvent"
GROUP BY 1
HAVING watch_count >= 2
ORDER BY watch_count DESC
LIMIT 400000
```

Ces codes ont été adaptés selon le l'année=year, le mois et le jour dans d'autres requêtes

```
/* Requête de calcul du total des watch */
SELECT   SUM(watch_count) AS total_watch
FROM [bigquery-public-data:github_repos.sample_repos]
LIMIT 1000

/* Requête calcul de watch_proportion par Repository  */
SELECT  * , (100*watch_count/10006346) as watch_proportion
 FROM [bigquery-public-data:github_repos.sample_repos]
LIMIT 1000
/ * total_watch=10006346 */
```

```
/* Requête de  calcul de MA : repository massivement  adoptés, sont ceux qui sont dans les X% les
plus starrés (cf event WatchEvent). */

SELECT  watch_proportion  ,
          CASE
          WHEN   watch_proportion  > = seuil X %  THEN  1
          WHEN  watch_proportion   < seuil X %   THEN  0
           END    AS        MA
FROM  IN
SELECT  * , (100*watch_count/10006346) as watch_proportion
 FROM [bigquery-public-data:github_repos.sample_repos]
```

Question 2) :

Créer un modèle qui permette à partir d'un repository donné de savoir si il va être MA ou non. On pourra et devra sans doute créer un dataset complètement from scratch.

Qu'est-ce qui influence le fait de devenir un repository MA ?

Créez vos propres training data set et test data set. Evidemment, un même repository on peut se demander à plusieurs moments si il va devenir MA.

Deux mois après sa création, que disait notre modèle ? Trois mois après, que disait-il ? Un an après ?

Évidemment il faut s'y intéresser du moment que le nombre de Star ne dépasse pas le seuil choisi X...

- **Réponse 2) situation de Supervised Learning**

Étapes :

1. **Sélection et création des variables à partir des Tables (Commits, Languages, Licences, Sample_repo, githubarchive(year, month, day))**

- repo.id
- repo_name
- count_commit
- repository_Age (=date courante-date created_at)
- Subject (Faire Analyse des données textuelles, Analyse Sémantique Latente, voir Décomposition en valeur singulière, *Le but premier est d'explorer les mots et termes du la colonne Subject en déterminant quels sont les « concepts » (ou classes sémantiques) qui expliquent le mieux un Repository et par la suite un Repo_MA.)*
- count_contributors
- Age_repository (date courante – date created_at)
- nom_LangageProgrammation (A recoder en variables qualitatives, ou bien prendre chaque langage de programmation comme une variable booléenne)
- nombre_LangageProgrammation
- Nom_Licences
- nombre_ licences.

- WatchEvent is related to starring a repository, not watching (https://developer.github.com/v3/activity/events/types/#watchevent)
- Watch_count
- MA

2. Créer un dataset contenant toutes les variables ci-dessus après nettoyage

3. Partition le dataset obtenu en un training dataset et un test dataset (enregistrés en CSV)

4. Importer les librairies nécessaires à la Modélisation et Prévision + Continuer avec notebook dans l'environnement ci-dessous :

```
# data analysis
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

Acquire data

The Python Pandas packages helps us work with our datasets. We start by acquiring the training and testing datasets into Pandas DataFrames. We also combine these datasets to run certain operations on both datasets together.

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
combine = [train_df, test_df]
```

Analyze by describing data

Pandas also helps describe the datasets answering following questions early in our project.

Which features are categorical?

These values classify the samples into sets of similar samples. Within categorical features are the values nominal, ordinal, ratio, or interval based? Among other things this helps us select the appropriate plots for visualization.

- Categorical:.   Ordinal:.

Which features are numerical?

Which features are numerical? These values change from sample to sample. Within numerical features are the values discrete, continuous, or timeseries based? Among other things this helps us select the appropriate plots for visualization.

- Continous:……………………. Discrete: ………………………..

Assumtions based on data analysis

We arrive at following assumptions based on data analysis done so far. We may validate these assumptions further before taking appropriate actions.

Correlating.

We want to know how well does each feature correlate with Survival. We want to do this early in our project and match these quick correlations with modelled correlations later in the project.

Analyze by visualizing data

Now we can continue confirming some of our assumptions using visualizations for analyzing the data.

Correlating numerical features

Let us start by understanding correlations between numerical features and our solution goal (MA= Repository Massivement Adopté).

Correlating numerical and ordinal features

We can combine multiple features for identifying correlations using a single plot. This can be done with numerical and categorical features which have numeric values.

Correlating categorical features

Now we can correlate categorical features with our solution goal.

Wrangle data

We have collected several assumptions and decisions regarding our datasets and solution requirements. So far we did not have to change a single feature or value to arrive at these. Let us now execute our decisions and assumptions for correcting, creating, and completing goals.

Correcting by dropping features

This is a good starting goal to execute. By dropping features we are dealing with fewer data points. Speeds up our notebook and eases the analysis.

Converting a categorical feature

Now we can convert features which contain strings to numerical values. This is required by most model algorithms. Doing so will also help us in achieving the feature completing goal.

Model, predict and solve

Now we are ready to train a model and predict the required solution. There are 60+ predictive modelling algorithms to choose from. We must understand the type of problem and solution requirement to narrow down to a select few models which we can evaluate. Our problem is a classification and regression problem. We want to identify relationship between output (Survived or not) with other variables or features (Gender, Age, Port...). We are also perfoming a category of machine learning which is called supervised learning as we are training our model with a given dataset. With these two criteria - Supervised Learning plus Classification and Regression, we can narrow down our choice of models to a few. These include:

- Logistic Regression
- KNN or k-Nearest Neighbors
- Support Vector Machines
- Naive Bayes classifier
- Decision Tree
- Random Forrest
- Perceptron

```
X_train = train_df.drop("MA", axis=1)
Y_train = train_df["MA"]
X_test  = test_df.drop("repo.id", axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
 ((…..,  …..), (….,…), (…….,  …..))
```

```
# Logistic Regression
```
Logistic Regression is a useful model to run early in the workflow. Logistic regression measures the relationship between the categorical dependent variable (feature) and one or more independent variables (features) by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Reference Wikipedia.

Note the confidence score generated by the model based on our training dataset.

```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_log
Sortie = Taux de bon classement des MA
```

We can use Logistic Regression to validate our assumptions and decisions for feature creating and completing goals. This can be done by calculating the coefficient of the features in the decision function.

Positive coefficients increase the log-odds of the response (and thus increase the probability), and negative coefficients decrease the log-odds of the response (and thus decrease the probability).

# Support Vector Machines
Next we model using Support Vector Machines which are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training samples, each marked as belonging to one or the other of **two categories**, an SVM training algorithm builds a model that assigns new test samples to one category or the other, making it a non-probabilistic binary linear classifier. Reference Wikipedia.

Note that the model generates a confidence score which is higher than Logistics Regression model.

```
# Support Vector Machines
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
acc_svc
```

Sortie = Taux de bon classement des MA

# k-Nearest Neighbors In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. A sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Reference Wikipedia.

KNN confidence score is better than Logistics Regression and SVM.

```
# k-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn
```

Sortie = Taux de bon classement des MA

# Gaussian Naive Bayes
In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features) in a learning problem. Reference Wikipedia.

The model generated confidence score is the lowest among the models evaluated so far.

```
# Gaussian Naive Bayes
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
```

```
acc_gaussian
```
Sortie = Taux de bon classement des MA

# Perceptron

The perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not). It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time. Reference Wikipedia.

```
# Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
acc_perceptron
```
Sortie = Taux de bon classement des MA

# Decision Tree

This model uses a decision tree as a predictive model which maps features (tree branches) to conclusions about the target value (tree leaves). Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Reference Wikipedia.
The model confidence score is the highest among models evaluated so far.

```
# Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
acc_decision_tree
```
Sortie = Taux de bon classement des MA

# Random Forest

The next model Random Forests is one of the most popular. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees (n_estimators=100) at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Reference Wikipedia.

The model confidence score is the highest among models evaluated so far. We decide to use this model's output (Y_pred) for creating our competition submission of results.

```
# Random Forest


random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
```

```
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
acc_random_forest
```

Sortie = Taux de bon classement des MA

6) Observations et ouverture