Projet Science des Données

Bizeul Solal - Wawczak Lisa

I. Description du projet:

Ce projet en science des données a pour objectif d'aboutir à une prédiction météorologique : pleuvra-t-il ou non le lendemain ?

Nous avons à notre disposition une base de données contenant des observations météorologiques journalières mesurées par plusieurs stations météo australiennes. Ces données correspondent aux variables :

- Date de l'observation
- Localisation de la station météo
- Température minimale observée (en degrés celsius)
- Température maximale observée (en degrés celsius)
- Quantité de précipitation relevée pour la journée (en mm)
- Évaporation relevée en 24 h (en mm)
- Nombre d'heures d'ensoleillement dans la journée
- Direction du vent le plus fort
- Vitesse du vent le plus fort (en km/h)
- Direction du vent à 9 h
- Direction du vent à 15 h
- Vitesse du vent à 9 h
- Vitesse du vent à 15 h
- Pourcentage d'humidité à 9 h
- Pourcentage d'humidité à 15 h
- Pression atmosphérique réduite au niveau de la mer à 9 h (en hpa)
- Pression atmosphérique réduite au niveau de la mer à 15 h (en hpa)
- Portion du ciel obscurcie par les nuages à 9 h (en oktas)
- Portion du ciel obscurcie par les nuages à 15 h (en oktas)
- Température observée à 9 h
- Température observée à 15 h
- A-t-il plu dans les 24 h ?
- Quantité de précipitation du lendemain (en mm)
- A-t-il plu le lendemain ?

Pour parvenir à la prédiction, nous avons élaboré un code en langage python pour réaliser l'importation, l'examen et la préparation des données, la recherche de corrélations, l'extraction des jeux d'apprentissage et de test, l'entraînement d'un modèle ainsi que son évaluation et l'amélioration de l'évaluation. Ces étapes sont détaillées dans les prochaines parties.

II. L'importation des données:

La première étape à réaliser est l'importation des données. Le jeu de données se trouve sous le format CSV (Comma Separated Values). Il est importé en mémoire grâce à la fonction *read_csv()* de la bibliothèque Pandas via l'instruction suivante :

```
df=pd.read_csv("weather.csv",index_col=0)
```

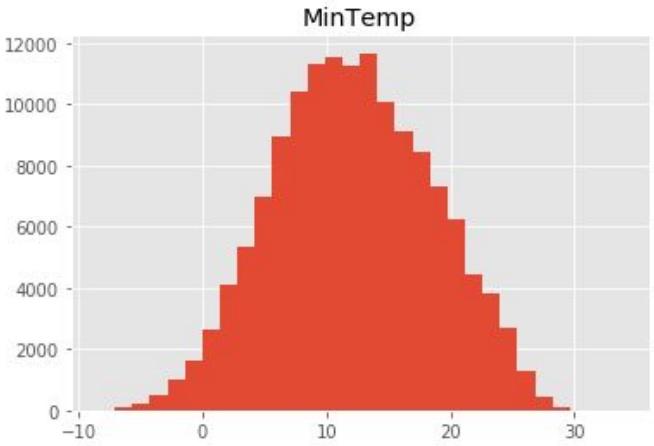
Pour forcer Pandas à utiliser la première colonne pour indexer les lignes (cette colonne contient le nom des différentes variables décrite dans la première partie), le paramètre index_col est fourni à la fonction. Nous lui attribuons la valeur 0 pour qu'elle utilise la première colonne (indexée 0) comme colonne des index. Cette fonction renvoie un objet de la classe DataFrame, classe utilisée en langage Python pour la manipulation de données.

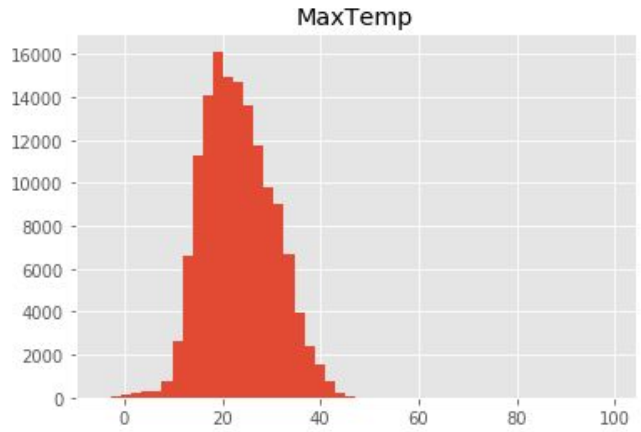
III. L'examen des données :

Après avoir importé les données, il est nécessaire de les examiner. Il est à noter que les données du jeu sont de plusieurs types : numérique (flottants sur 64 bits) pour les températures par exemple, et objet pour la localisation ou la direction du vent. Certaines données ne sont pas nécessaires à l'obtention de la prévision. Il s'agit de la variable du risque de précipitation le lendemain. D'autres données sont aberrantes. C'est le cas pour des valeurs de température maximale atteignant 99°C. Concernant les valeurs manquantes, la fonction *isna()* permet de les visualiser en renvoyant un DataFrame dans lequel les valeurs sont remplacées par False si la valeur est présente et True si la valeur est manquante. Nous regroupons ces données avec la fonction sum() pour visualiser le nombre de valeurs manquantes par colonne. Les résultats sont regroupés dans le tableau suivant :

Location	0
MinTemp	637
MaxTemp	322
Rainfall	1406
Evaporation	60843
Sunshine	67816
WindGustDir	9330
WindGustSpeed	9270
WindDir9am	10013
WindDir3pm	3778
WindSpeed9am	1348
WindSpeed3pm	2630
Humidity9am	1774
Humidity3pm	3610
Pressure9am	14014
Pressure3pm	13981
Cloud9am	53657
Cloud3pm	57094
Temp9am	904
Temp3pm	2726
RainToday	1406
RISK_MM	0
RainTomorrow	0

Nombre de données manquantes par variable





La fonction hist() a permis de tracer les histogrammes ci-dessus et de montrer l'existence de valeurs aberrantes concernant la température maximale qui peut atteindre 99°C.

IV. La préparation des données :

Les fonctions qui seront utilisées par la suite nécessitent des données complètes et numériques. Le constat réalisé dans la partie précédente montre le besoin de les préparer avant leur utilisation.

Tout d'abord, pour les valeurs manquantes, plusieurs solutions sont possibles. La première est de supprimer la variable si un nombre trop important de données sont manquantes. On estimera ce manque à ⅓ des données. Cette suppression est réalisée par l'instruction suivante :

```
for i in range(len(tab)):
    if tab[i]>142193/3:
        df=pd.DataFrame.drop(df,nom_colonnes[i],axis=1)
```

Cette étape est l'occasion de supprimer la variable RISK_MM, qui est en corrélation directe avec la variable à prédire (RainTomorrow): la garder rendrait le travail de prédiction beaucoup trop simple. Voilà pourquoi nous avons choisi de l'éliminer grâce à la fonction *drop()* :

```
df=pd.DataFrame.drop(df,'RISK_MM',axis=1)
```

Le reste des valeurs manquantes est remplacé par la moyenne de la variable (pour les variables quantitatives) grâce à la fonction *fillna()*. Un exemple d'instruction réalisant cette étape pour les variables de pression est donné ci-dessous :

```
df["Pressure9am"].fillna(df["Pressure9am"].mean(),inplace=True)
df["Pressure3pm"].fillna(df["Pressure3pm"].mean(),inplace=True)
```

Enfin, les variables qualitatives sont changées en variable quantitatives grâce à la classe *LabelEncoder.* Cette classe assigne un nombre entier à chaque classe différente d'une colonne. Par exemple, dans l'exemple de la direction du vent, les classes dans la colonne sont SE, SWS, NNE etc. Aléatoirement, le LabelEncoder va associer 0 à SE, 1 à SWS, 2 à NNE etc. La procédure pour remplacer les variables qualitatives par leur équivalent quantitatif tout en remplaçant les données manquantes est assez complexe. Il

semble alors pertinent de la présenter en étapes successives pour WindDir9am par exemple.

1. Faire une copie de df et enlever dans dfcopy les lignes avec des valeurs manquantes (nous sommes obligés de faire une copie puisque LabelEncoder ne fonctionne pas lorsque des données sont manquantes)
2. Trouver les classes avec la méthode .fit en mettant en argument le tableau numpy de la colonne voulue
3. Transformer le tableau numpy contenant les valeurs qualitatives en tableau numpy contenant les valeurs qualitatives avec la méthode .transform
4. Trouver le mode (ou valeur dominante) du tableau numpy avec la méthode .mode (et non pas la moyenne : puisque le numéro associé à chaque classe est aléatoire, une moyenne ne donnera pas la moyenne de la direction du vent, mais la moyenne des numéros rattachés aux classes, ce qui est inutile)
5. Faire la transformation inverse pour obtenir le nom de la direction du vent la plus fréquente avec la méthode .inverse_transform
6. Changer l'array en string pour pouvoir remplacer les valeurs manquantes par ce mode dans le df avec la methode fillna()
7. Désormais la colonne WindDir9am de df n'a plus de valeurs manquantes, mais cette colonne est toujours remplie de valeurs qualitatives (je vous rappelle que nous avons appliqué le LabelEncoder à dfcopy et non df)
8. Appliquer .fit et .transform à la colonne WindDir9am de df, puis remplacer cette colonne dans le df par le tableau numpy donné par la méthode .transform

```
dfcopy=df
dfcopy=pd.DataFrame.dropna(dfcopy)
```

```
le = sk.preprocessing.LabelEncoder()
le.fit(dfcopy['WindDir9am'])
list(le.classes_)
RT3=le.transform(dfcopy['WindDir9am'])
print(RT3)
freq=st.mode(RT3)
freq2=le.inverse_transform([freq])
freq3=' '.join(map(str,freq2))
df["WindDir9am"].fillna(freq3,inplace=True)

le = sk.preprocessing.LabelEncoder()
le.fit(df['WindDir9am'])
list(le.classes_)
RT3=le.transform(df['WindDir9am'])
df.iloc[:,6]=RT3[:]
```

Pour le cas des données aberrantes, nous commençons par transformer le DataFrame en tableau numpy. Ensuite, nous itérons sur la colonne MaxTemp pour trouver les valeurs de température supérieures à 50°C et nous les remplaçons par NaN. Nous remplaçons la colonne correspondante dans df par ce tableau numpy. Ensuite, nous remplaçons tous les NaN de cette colonne par la moyenne de cette variable comme indiqué précédemment.

```
129 nump=df.to_numpy()
130 print(nump)
131 for i in range(0,142193):
132     if nump[i][2]>50:
133         nump[i][2]='NaN'
134 df.iloc[:,2]=nump[:,2]
```

Les algorithmes d'apprentissage fonctionnent mieux lorsque les variables numériques ont la même échelle. La classe *StandardScaler* réalise cette uniformisation d'échelle. Cette classe normalise les données sur une Gaussienne d'espérance nulle et de variance unitaire. Puisque cette classe ne s'applique qu'aux tableaux Numpy, le DataFrame est d'abord converti en tableau Numpy auquel on applique la fonction *fit()* qui calcule la moyenne et la variance à utiliser pour une mise à l'échelle ultérieure, puis la fonction *transform()* qui réalise la normalisation en centrant et en mettant à l'échelle. On ne standardise pas la sortie (RainTomorrow).

```
nump=df.to_numpy()
scaler=StandardScaler().fit(nump)
std_nump=scaler.transform(nump)
```

V. La recherche de corrélation :

Pour poursuivre l'analyse des données, il est intéressant d'étudier les relations existant entre les différentes variables du problème. Ces relations sont mises en valeur grâce au coefficient de corrélation, calculé par la méthode *corr()* de la classe *DataFrame*. Ce coefficient est le rapport de la covariance de deux variables par le produit de leurs écarts types (voir formule ci-contre). Plus ce coefficient est proche de 1 en valeur absolue, plus la corrélation linéaire entre les deux variables est forte. C'est-à-dire qu'il existe une fonction affine liant ces variables. Ce coefficient a été calculé pour chaque couple de variable grâce à l'instruction suivante :

$$r = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

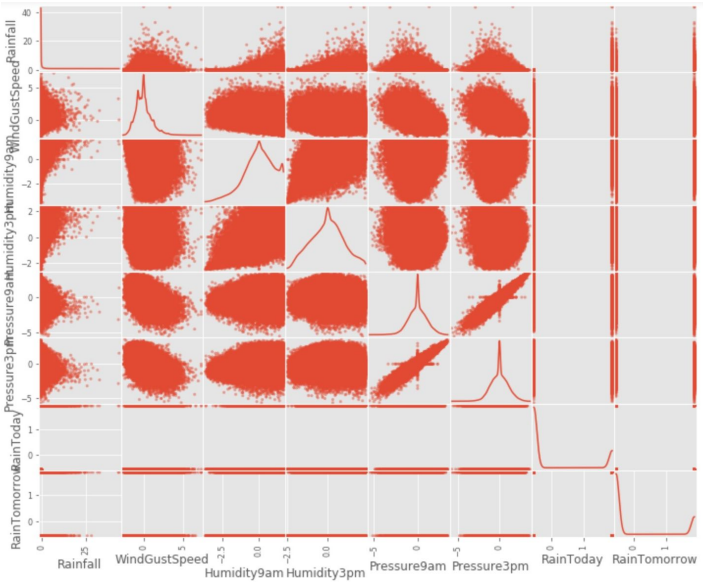
```
correlation=df.corr(method='pearson')
for i in range(0,18):
    print(correlation.ix[:,i])
```

```
Location        -0.020440
MinTemp          0.733858
MaxTemp          1.000000
Rainfall        -0.074239
WindGustDir     -0.212223
WindGustSpeed    0.066332
WindDir9am      -0.212522
WindDir3pm      -0.181303
WindSpeed9am     0.014628
WindSpeed3pm     0.050388
Humidity9am     -0.499731
Humidity3pm     -0.499653
Pressure9am     -0.309084
Pressure3pm     -0.397411
Temp9am          0.879992
Temp3pm          0.969633
RainToday       -0.226481
RainTomorrow    -0.159051
Name: MaxTemp, dtype: float64
```

Cette instruction retourne un tableau par variable dont un exemple pour la température maximale est donné ci-dessus. Chaque nombre correspond au coefficient de corrélation de cette variable avec la variable en correspondance. Ainsi, nous pouvons y lire que la température minimale, la température relevée à 9 h et la température relevée à 15 h, sont fortement corrélées avec la température maximale. Le sont aussi la pression relevée à 9h et la pression relevée à 15 h. Les variables les plus corrélées à la variable de sortie sont RainToday et l'humidité relevée à 15 h. Ces données sont donc les plus pertinentes pour la classification puisqu'elles ont un coefficient de corrélation avec RainTomorrow supérieur à 0.3 en valeur absolue. 0.3 correspond à une corrélation faible, puisqu'une corrélation doit être supérieure à 0.7 pour être considérée forte. Malheureusement, RainTomorrow n'a pas de corrélations fortes donc nous allons essayer d'entraîner le modèle à partir de ces corrélations faibles.

```
Location        -0.003579
MinTemp          0.083717
MaxTemp         -0.159051
Rainfall         0.236874
WindGustDir      0.053447
WindGustSpeed    0.225264
WindDir9am       0.035776
WindDir3pm       0.030488
WindSpeed9am     0.090524
WindSpeed3pm     0.086909
Humidity9am      0.255158
Humidity3pm      0.439678
Pressure9am     -0.234027
Pressure3pm     -0.214688
Temp9am         -0.025582
Temp3pm         -0.190286
RainToday        0.306555
RainTomorrow     1.000000
Name: RainTomorrow, dtype: float64
```

En utilisant la fonction *scatter_matrix()* de Pandas, il est possible de croiser deux à deux les variables et d'afficher le nuage de point correspondant. L'application de cette fonction à certaines données du problème (les variables avec une corrélation supérieure à 0.2 avec RainTomorrow) nous donne le tableau suivant :



```
183 pd.plotting.scatter_matrix(df,figsize=(12,10),diagonal='kde')
184 plt.show()
```

On observe que, sur la diagonale, les points forment une courbe car il s'agit des deux mêmes variables. La droite formée par le nuage de point révèle la forte corrélation entre les variables de pression entre elles qui avait déjà été révélé précédemment par le calcul du coefficient de corrélation.

VI. L'extraction des jeux d'apprentissage et de test :

Pour entraîner le modèle et l'évaluer, l'ensemble des données est divisé en deux parties. D'un côté, se trouvent les données qui serviront à entraîner le modèle, et de l'autre, celles qui serviront à le tester. Le tableau X correspond aux colonnes sauf RainTomorrow tandis que y correspond à la colonne RainTomorrow. Habituellement la proportion de données servant à l'entraînement est de l'ordre de 70 % à 80 %. La fonction utilisée pour créer cette division est *train_test_split()*. Elle choisit aléatoirement 75 % des données qui serviront à l'entraînement, les 25 % servant au test du modèle.

```
192 X_train, X_test, y_train, y_test=sk.model_selection.train_test_split(X,y,random_state=0)
```

VII. L'entraînement du modèle :

L'entraînement du modèle utilise un algorithme basé sur la régression logistique. Il s'agit d'un modèle de régression binomial, cas particulier de modèle linéaire. Cet algorithme est implémenté par la classe *LogisticRegression().* Il est utilisé lorsque la variable est binaire. En notant y=f(x) cette variable et p (y = 1|x) = μ(x), alors p (y = 0|x)= 1-μ(x) et l'expression du logarithme du rapport des vraisemblances log[p(y=1|x)/p(y=0|x)]= log(μ(x))-log(1-μ(x)) . On émet l'hypothèse que μ(x)=w(f(x))=1/(1+exp(f(x))) avec f(x)=W0+WTx donc que log[p(y=1|x)/p(y=0|x)]=f(x).

L'algorithme cherche à maximiser la vraisemblance des exemples d'apprentissage, ce qui revient à maximiser la log-vraisemblance ou minimiser la fonction coût suivante :

$$J(w,w_0) = -\sum_{j=1}^{n} y^{j}\left[(w_0+w^T x^j) - \log\left(1+e^{(w_0+w^T x^j)}\right)\right]$$

Cette fonction est minimisée grâce à la technique de descente du gradient. Pendant la phase d'apprentissage de l'algorithme, les paramètres W0 et WT sont calculés.

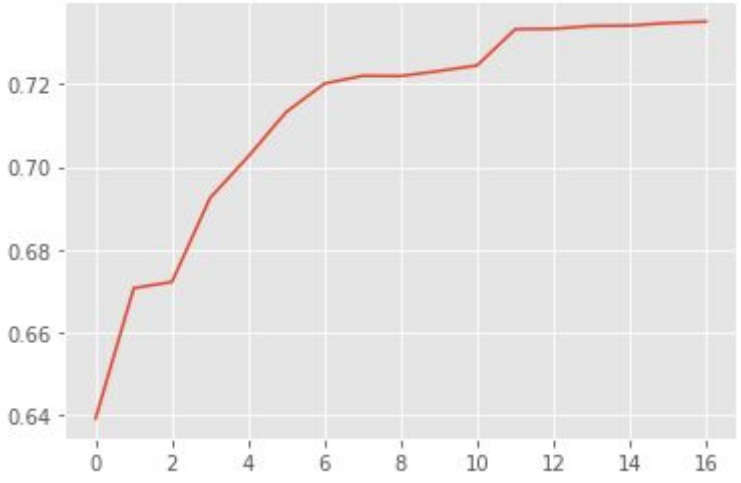
En pratique, certains problèmes se sont posés. Par exemple, l'erreur "Unknown label type: 'continuous' " lors de l'application de LogisticRegression.fit à X_train et y_train. En cherchant sur Internet, nous avons réalisé que la méthode fit s'attend à recevoir des arrays de int et pas de float. Or dans la partie 5 du projet, on nous demande d'utiliser la classe StandardScaler qui normalise les données et les transforme en float64 au passage (avec un grand nombre de chiffres après la virgule). Voilà d'où vient cette erreur. Pour résoudre ce

problème, nous avons envisagé de multiplier toutes les données par 10^6 par exemple, puis d'arrondir ces chiffres à l'entier le plus proche pour essayer de garder le plus de précision possible (et faire en sorte que l'erreur 'continuous' soit résolue).

VIII. L'évaluation du modèle:

L'étape suivante consiste à tester le modèle en comparant ses prédictions aux données du jeu. Plusieurs métriques permettent d'évaluer le modèle.

Nous avons fait face à un problème au niveau du nombre de variables à garder pour l'étape de la prédiction. Nous avions décidé de garder uniquement Humidity3pm (corrélation de 0.439678 avec RainTomorrow) et RainToday (0.306555). Cela nous donnait un accuracy_score de 0.6706517764212777. Nous avons testé l'inclusion de la troisième variable ayant le coefficient de corrélation le plus haut avec RainTomorrow (Humidity9am avec 0.255158) dans les données utilisées pour la prédiction et cela nous a donné un accuracy_score de 0.6721426763059439, ce qui est supérieur au score obtenu précédemment. Il n'est donc pas optimal de garder seulement les deux variables ayant le coefficient de corrélation le plus haut avec RainTomorrow. Nous avons décidé de tester l'algorithme de prédiction avec uniquement la variable la plus pertinente, puis avec les deux variables les plus pertinentes, puis les trois variables les plus pertinentes, etc. Voilà nos résultats :



Ce graphe représente le accuracy_score par rapport au nombre de variables rajouté pour le calcul. La valeur en 0 correspond donc au accuracy_score avec uniquement la variable la plus pertinente, la valeur en 1 correspond au accuracy_score avec les deux variables les plus pertinentes, etc. Nous pouvons remarquer que cette fonction est strictement croissant. Nous en concluons que l'algorithme pourra prédire les valeurs de RainTomorrow de manière optimale si nous ne supprimons aucune colonne (ce qui nous donne un accuracy_score de 0.7351542940729697).

L'étape suivante consiste à tester le modèle en comparant ses prédictions aux données du jeu. Plusieurs métriques permettent d'évaluer le modèle.

La première est accuracy_score() qui calcule la précision : l'ensemble d'étiquettes prévu pour un échantillon doit correspondre exactement à l'ensemble d'étiquettes correspondant. La valeur retournée est une précision de 0.7351542940729697.

La seconde est *confusion_matrix()* qui calcule la matrice de confusion. C0,0 représente le nombre de vrais négatifs; C1,0 , de faux négatifs ; C0,1, de faux positifs et C1,1, de vrais positifs. La fonction retourne la matrice : [[19557 7986] [1429 6577]].

La suivante est *precision_score()* qui calcule le rapport tp/(tp+fp) où tp est le nombre de vrais positifs et fp celui de faux positifs. La valeur retournée est une précision de 0.6917654820926484.

La fonction *recall_score()* calcule le rapport tp/(tp+fn) où fn est le nombre de faux négatifs. Cette fonction mesure la capacité du modèle à trouver tous les échantillons positifs. La valeur retournée est une précision de 0.765781119720606

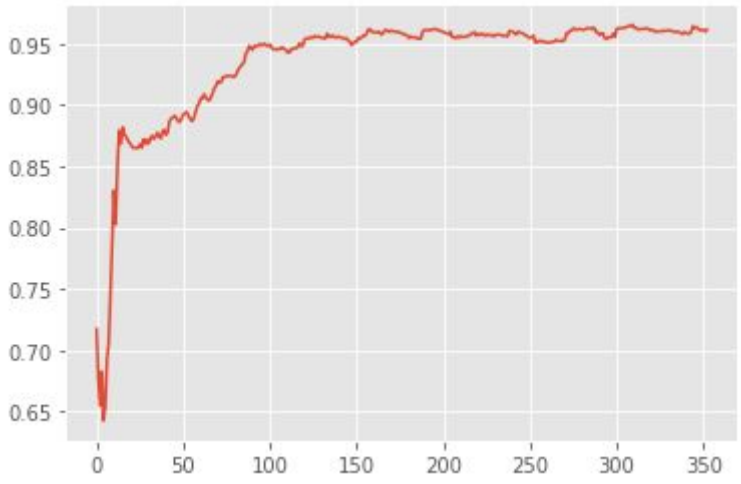
La fonction *f1_score()* est interprété comme le calcul de la moyenne pondéré de la précision et du rappel: F1= 2*précision*rappel/(précision+rappel). Le score retourné est 0.6944135776233816

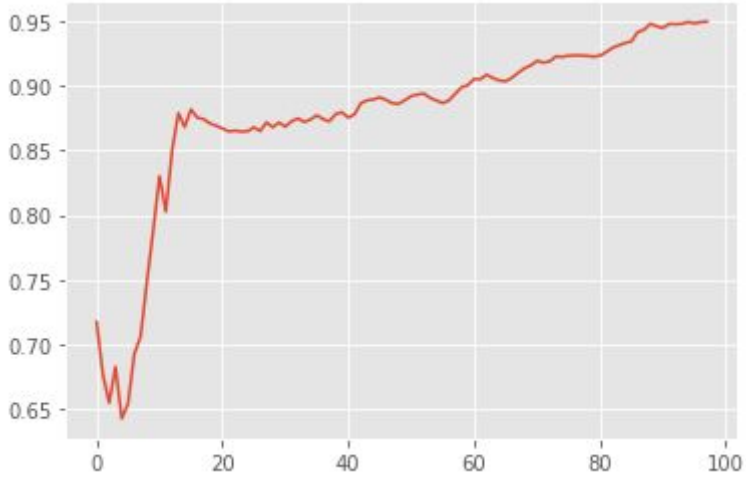
IX. Amélioration de l'évaluation

Enfin, il est possible d'améliorer le modèle en ayant recours à la méthode de validation croisée. Cette méthode consiste à diviser le jeu d'entraînement de manière aléatoire et à entraîner et tester le modèle en passes successives. Chaque sous-ensemble constitue un nouveau jeu de donnée qui est divisé en données d'entraînement et en données de test comme il a été fait précédemment. La fonction *cross_val_score()* de la classe KFold permet d'obtenir les résultats d'une validation croisée.

Nous avons donc testé cette nouvelle méthode en faisant varier le nombre de divisions du jeu d'entraînement (folds) pour optimiser le cross_val_score. Vu la quantité de données avec laquelle on travaille, nous ne pouvions pas réaliser plus de 355 folds, mais, en même temps, nous n'avons pas eu de problèmes de temps de calcul trop long (20 minutes maximum pour tester tous les nombres de folds de 2 à 355). Nous obtenons donc la courbe suivante (première courbe ci -dessous). Nous pouvons remarquer plusieurs choses. D'abord, cette fonction est croissante dans son ensemble, ce qui est logique puisque plus de folds sont utilisés et plus l'algorithme est précis. Ensuite, nous pouvons voir que ce type d'évaluation est plus performant que l'utilisation d'un train_test_split puisque à partir de 8 folds, le cross_val_score du KFold est supérieur à celui de train_test_split. Finalement, nous pouvons voir que le cross_val_score reste à peu près constant à partir de 100 folds. Voilà pourquoi nous avons fait un zoom sur les 100 premières valeurs (deuxième courbe ci-dessous).

Lorsque nous faisons tourner l'algorithme de KFold avec 100 folds par exemple, nous obtenons 100 cross_val_score différents. Nous avons donc choisi de faire une moyenne de ces 100 scores pour obtenir un chiffre représentatif de la performance de l'algorithme. Nous obtenons un cross_val_score de 0.9482678571428572 pour 100 folds, chiffre qui monte à 0.964516129032258 au maximum pour 310 folds. Ces scores sont tout à fait acceptables et montrent la supériorité de cette méthode par rapport au train_test_split.


0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
50
100
150
200
250
300
350


0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
20
40
60
80
100