Challenge Kaggle : Rapport de la team LC

laris kokou EDJINEDJA
ayoko AYIKA

Décembre 2021

1 Traitement des données

Avant de prédire les labels du Train5slices, explorons nos données. Dans un premier temps redimensionnons chaque ligne du dataframe qui constitue les valeurs de pixel en une liste de matrice 2×2. Cela nous permet d'avoir après en utilisant les méthodes de la bibliothèque matplotlib.pyplot un aperçu des images contenues dans les dataframes. En affichant 3 images de notre dataframe on obtient les figures ci dessous :

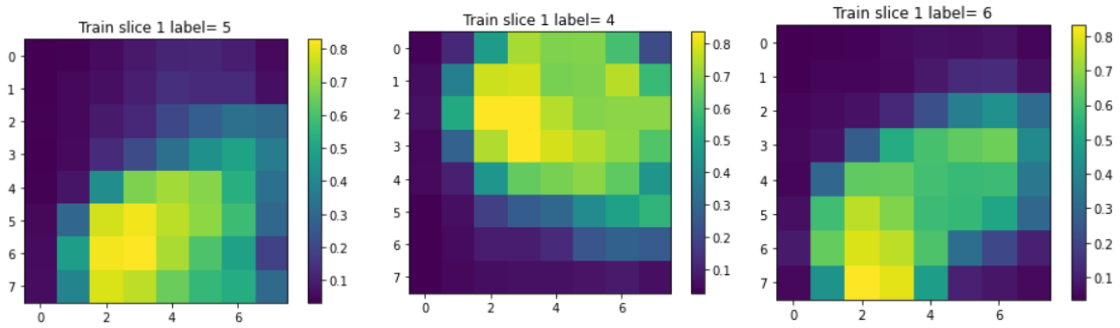


FIGURE 1 – 3 image formé par un vecteur de pixel

une petite analyse statistique nous montre une corrélation entre les données des 10 slices et une indépendances entre les colonnes d'un ensemble de données (ensemble train1slice ou ensemble test du premier slice).

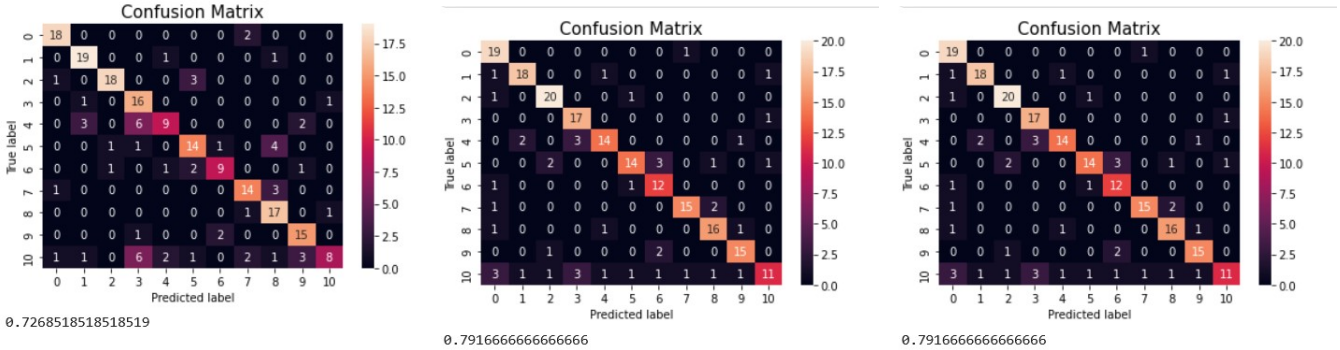
2 Première partie : Apprentissage du train5slices et prédiction des labels du test5slices

2.1 Models sklearn

2.1.1 Apprentissage et inference sur train5

Nous avons coupé les données du train5 en ensembles d'apprentissage et en ensemble de validation qui representent en proportions respetifs 75% et 25% des données.Nous avons ensuite testé plusieurs classifieurs avec leurs paramètres par défaut et avons sélectionné ceux qui donnaient une accuracy supérieure à 72%. Les classifieurs retenus sont : RandomForest, Extratrees, SVC, NuSVC, MLPClassifier et KNN. Nous avons ensuite optimisé ces classifieurs grâce à un gridsearch. Après entraînement et test, nous obtenons les matrices de confusion suivantes :

Random Forest :Accuracy score = 0.7268 SVC : Accuracy score = 0.7916 MLP :Accuracy score =0.72



Après ces études,nous avons retenu SVC avec comme paramètres C=10, gamma=0.1 et randomstate=0. Après l'inférence sur le test5 nous avons réalisé à ce stade un accuracy sur le leaderboard de 69%

Insatisfaits du résultat, nous avons pensé à travailler avec tout le train5 comme ensemble de train et le test5 comme notre ensemble de test.

2.1.2 Apprentissage sur train5

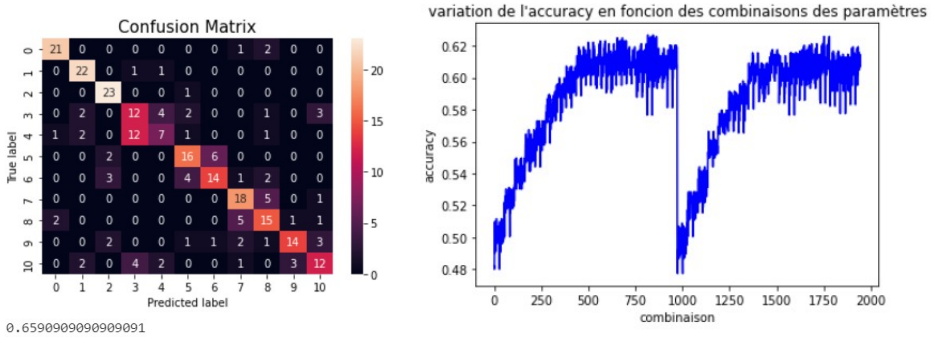
Afin d'optimiser l'accuracy score, nous travaillons maintenant sur la totalité des données du train5slices pour prédire les labels du test5slices. Après avoir testé plusieurs modèles, celui qui nous donne la meilleure accuracy sur le leaderboard est MPLClassifier.Etant donné que l'acuracy score donné par le gridsearch score était 70% nous étions convaincus que c'est le model qu'il faut pour avoisiner une accuracy de 80% sur le leaderbard.Dans ce sens, pour l'optimiser, nous avons combiné un gridsearch à l'analyse en composantes principales (ACP).L'utilisation d'un pipeline nous as permis de bien utiliser ces deux méthodes pour l'entraînement.
L'ACP est une analyse factorielle, en ce sens qu'elle produit des facteurs (ou axes principaux) qui sont des combinaisons linéaires des variables initiales, hiérarchisées et indépendantes les unes des autres. On appelle parfois ces facteurs des «dimensions latentes», du fait qu'ils sont l'«expression de processus généraux dirigeant la répartition de plusieurs phénomènes qui se retrouvent ainsi corrélés entre eux» (Béguin Pumain, 2000) Source : (https ://journals.openedition.org/cal/7364)

Cette méthode nous as permis d'obtenir un score de 76% sur le leaderboard

Toujours insatisfaits de notre résultat nous avons pensé à construire notre propre réseaux de neurones pour la prédiction.

2.1.3 Recherche de l'influence des autres données d'enprentissage sur le test5

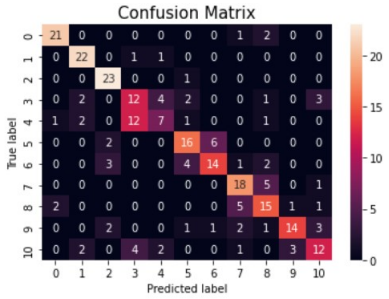
Afin de raffiner nos résultats, nous voulions voir si le fait d'apprendre sur un dataset et de prédire les labels d'un autre pouvait apporter une valeur ajoutée à notre analyse. Pour ce faire, nous avons donc travaillé avec plusieurs ensembles d'apprentissage. Pour chaque trainset, nous avons cherché le meilleur estimateur que nous avons appliqué à d'autres trainsets afin de prédire les labels de leurs ensembles de test. Pour le Train1slice, le meilleur estimateur était : ExtraTreesClassifier(max depth=18, nestimators=110, randomstate=0). Nous l'avons d'abord appliqué dur le Test1slice. Nous obtenons un score assez faible (=0.659).



0.6590909090909091

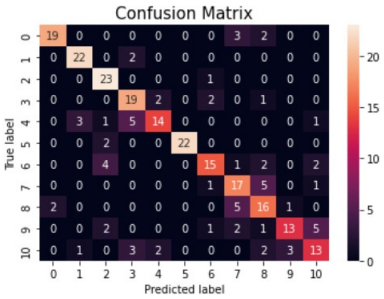
Après l'avoir appliqué à d'autres ensembles, nous obtenons les résultats suivants :

Sur train test3slices :



0.6590909090909091

Sur train test4slices :



0.7310606060606061

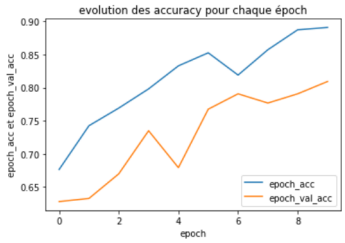
On remarque que l'accuracy score est plus élevée lorsque l'on applique ce modèle sur d'autres ensembles. Nous avons également entraîné le classifieur SCV sur train10slices. Nous avons ensuite appliqué le modèle à d'autres datasets dont train5slices. Nous avons constaté que tous nos scores étaient supérieurs à 90%. En particulier, le score de train4slices était de 0.928 et celui de train6slices, 0.931. Nous constatons également que plus il y a de slices, plus le score est élevé. Par conséquent, pour le train5slices, nous nous attendions à un score d'environ 90%. Pour autant, après soumission, nous avons obtenu une précision de 0.71717 sur le leaderboard. Ce qui est moins que celui obtenu en travaillant uniquement avec les données du dataset 5slices (On avait obtenu un score de 0,72).

Remarque : Nous avons aussi essayé de voir comment ce modèle prédisait les labels des différents trains. Tandis que tous les autres scores dépassent les 70%, celui du train1slice atteint à peine les 62%. Cela est dû au fait que lorsqu'on entraîne le classifieur sur plusieurs slices, il n'arrive pas à s'adapter à un seul slice. Il a emmagasiné trop d'informations et ne connaît plus les informations nécessaires à utiliser. A l'issue de nos analyses dans cette partie, on conclut que lorsqu'un classifieur est entraîné sur un dataset, il prédit rarement bien les labels d'autres datasets. On assiste rapidement à un surapprentissage. Il faut donc éviter de faire des prédictions d'un dataset avec le modèle d'un autre dataset.

2.2 Models pytorch

Dans cette partie nous avons construit un réseaux de neurones de 4 couches dont la fonction d'activation est **Relu** qu'on dropout avec une probabilité de 0.2 après la quatrième couche pour éviter qu'on surapprenne.Notre input est $8 \times 8 \times nbrslices$ et notre output est 11.

En entraînant notre model sur 10 épochs et en choisissant comme criterion=nn.CrossEntropyLoss() ,optimizer=Adam avec learning rate 0.005 nous avons réalisé une accuracy de 71%. Les images ci-dessous montrent comment le loss et l'accuracy évoluent en fonction des épochs.



accuracy en fonction des épochs : **Le Loss en fonction des épochs :**

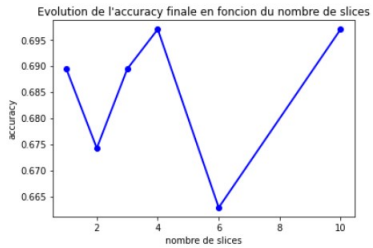
Cette méthode pouvait nous aider à atteindre les 100% sur le leaderbord mais malheuresement le temps nous as surpris.

3 Deuxième partie : Etude statistique sur l'influence du nombres de slices sur l'accuracy final

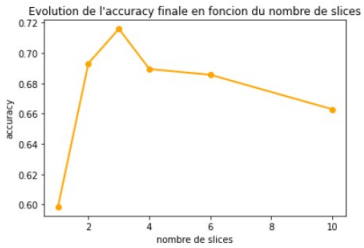
Cas simple

Dans cette partie, nous allons voir l'influence du nombre de slices sur l'Accuracy finale. Remarquons d'abord que pour les ensembles d'apprentissages utilisés, les classifieurs qui reviennent souvent sont les suivants : RandomForest,MLPClassifier. Nous avons donc optimisé ces modèles grâce à un gridSearch et nous obtenons pour chaque classifieur, les graphes suivants qui traduisent l'évolution de l'accuracy en fonction du nombre de slices :

RandomForest

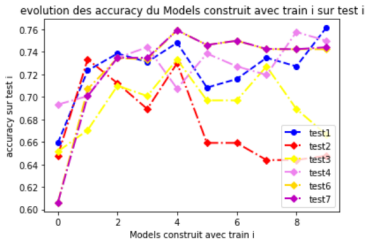


MLPClassifier



Pour RandomForest, L'accuracy obtenune avec 6 slices est la plus faible. On peut donc dire que l'accuracy finale n'augmente pas systématiquement avec plus de slices. Pour MLP aussi on observe que l'accuracy n'augmente pas systématiquement lorsque le nombre de slices augmente. On voit que l'accuracy augmente et atteint sa valeur max pour nslices = 3 puis diminue.

Cas complexe



En rapport à la dernière interpretation, ce graphe explique aussi l'influence du nombre de slices pour la prédiction.Grâce à ces courbe, on conclut que l'accuracy n'augmente forcément pas quand le nombre de slices augmente.