Machine Learning

HEPIA Laroche David

Objectif

Ce projet a pour but de faire pratiquer les méthodologies caractéristiques de l'approche "Data Science" pour la classification de données. Il se déroulera en trois parties spécifiques associées aux modèles suivants :

1. modèles de base vus au cours ;
2. ensembles ;
3. réseaux convolutifs.

1. Modèles de base

Fonctionnement du script

1. L'exécution du script prend en paramètres optionnels les données à traiter wisc pour `wisconsin.dat` et dj pour `dow_jones_index.data`, pour les données à choix.
2. Il prend aussi les méthode de classification PPV pour le plus proche voisin, AD pour les arbres de décision, PMC pour le perceptron multi-couches et finalement SVM pour la machine à vecteurs de support.
3. Le script va donc calculer pour chaque ensemble de données et de classificateurs indiqués avec ses paramètres (définis en durs dans le programme pour des raisons de praticité) puis produire le fichier .csv avec tous les modèles ainsi calculés.

Ceci permet de comparer les résultats des différents classificateurs mais aussi quels paramètres influencent le mieux le taux de classification (TCC) au sein de la même méthode.

Exemple de commande :

```
$ python3 machine_learning.py dj wisc PMC AD
```

Produit les 4 fichier: `output/dj_pmc.csv`, `output/dj_ad.csv`, `output/wisc_pmc.csv`, `output/wisc_ad.csv` contenant les résultats de wisconsin et du dow jones pour les méthodes du perceptron et des arbres de décision.

Choix des données

J'ai choisi les données de la bourse `dow_jones_index.data`. Presque tous les attributs sont continus. Afin de travailler avec des classificateurs, la classe à prédire a été définie à partir de l'attribut `percent_return_next_dividend`. Ainsi une valeur inférieure à 1 indique une perte et donc appartenant à la classe 0 si elle est supérieure à 1 ceci indique un gain et donc appartenant à la classe 1. Les données manquantes ont été comblées avec la valeur moyenne des exemples. Finalement tous les attributs continus ont été normalisés.

Mesure des performances

Temps de calcul

Les temps d'entraînement par modèle sont plutôt négligeables. Par ailleurs, plus un estimateur possède de paramètres, plus il sera coûteux de trouver le meilleur ajustement, car cela implique de calculer un modèle supplémentaire pour chaque combinaison de paramètres. Il est donc judicieux de les limiter et de les choisir de manière intelligente.

Table 1. Temps de calcul sur les données wisconsin.dat

Classificateur	Nombre de modèles entraînés	Temps médian pour l'entraînement d'un modèle [msec]	Temps total [secondes]
AD	32076	1.2	247
PPV	50	0.9	5.11
PMC	432	731	774

Table 2. Temps de calcul sur les données `dow_jones_index.data`

Classificateur	Nombre de modèles entraînés	Temps médian pour l'entraînement d'un modèle [msec]	Temps total [secondes]
AD	32076	3.2	525
PPV	50	0.9	5.8
PMC	432	837	868

Nous pouvons observer que l'AD et le PPV sont peu coûteux en calculs (temps médian par modèle) contrairement au PMC qui est assez lourd en comparaison.

Validation croisée

Le nombre de sous échantillons choisis pour la validation croisée augmente grandement les temps de calculs car le nombre de résultats est multiplié par le nombre d'échantillons à entraîner.

Pour cela il a été plus pratique de chercher les meilleurs paramètres avec 3 sous-échantillons, ensuite les calculs finaux on été exécutés sur 10 sous-échantillons afin d'obtenir des résultats plus réalistes (en les moyennant).

Taux de classification

Voici le meilleur taux de classification correct (TCC) moyen en calculant la moyenne des résultats obtenus de la validation croisée avec dix sous-échantillons (méthode : k-fold cross-validation).

Table 3. Meilleur TCC par classificateur sur les données wisconsin.dat

Classificateur	TCC moyen test	TCC moyen entraînement
AD	0.88	0.88
PPV	0.97	0.98
PMC	0.97	0.98

Table 4. Meilleur TCC par classificateur sur les données `dow_jones_index.data`

Classificateur	TCC moyen test	TCC moyen entraînement
AD	0.92	0.94
PPV	0.88	1
PMC	0.97	0.98

Les performances entre classificateurs sont assez proches. Les TCC des sets d'entraînement sont légèrement supérieurs aux TCC des tests ce qui est attendu car le modèle est entraîné avec le set d'entraînement par définition. Mais la différence de valeurs entre TCC test et TCC entraînement n'est pas assez grande pour indiquer avec assurance un overfitting. Seule la valeur du TCC au-delà de 95 % indiquerai une performance trop grande et donc un overfitting sur nos données malgré l'utilisation de la validation croisée.

Plus Proche Voisin

Paramètres modifiés:

1. Nombre de vosins pour la requête [`n_neighbours`]: nombre impair de 1 à 50
2. Poids pour la requête: [uniform, distance]

Ce classificateur ne possédant que deux paramètres intéressants dont un binaire, il est possible de représenter les résultats avec deux courbes sur un graphe.

Données du wisconsin

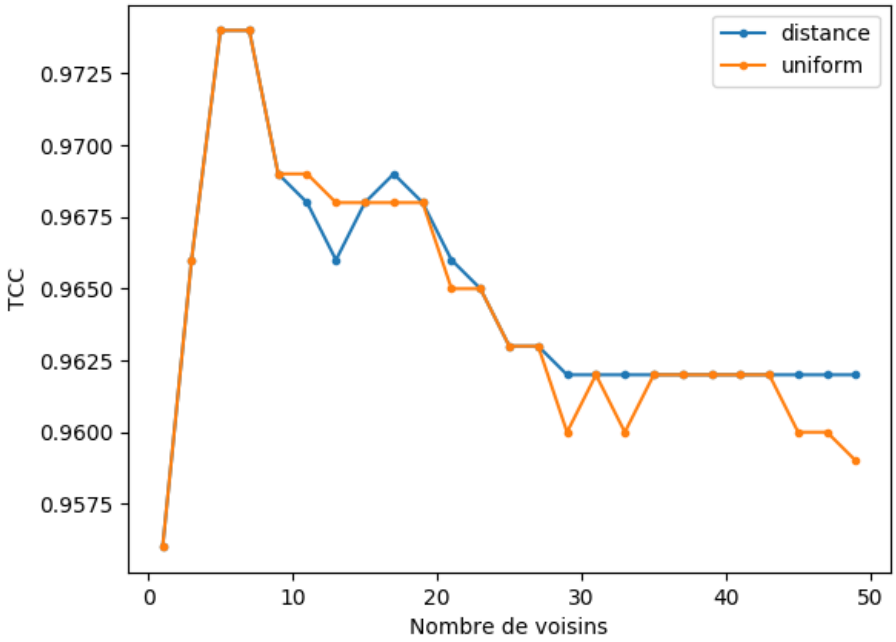

Fig 1. Plot on wisconsin.data

Il y a un maximum à 5 et 7 voisins, ensuite l'augmentation de ce nombre n'améliore pas la performance. Avec ces données, le changement des paramètres influence peu la performance (différence de 1,8 point entre le maximum et le minimum).

Données du Dow Jones

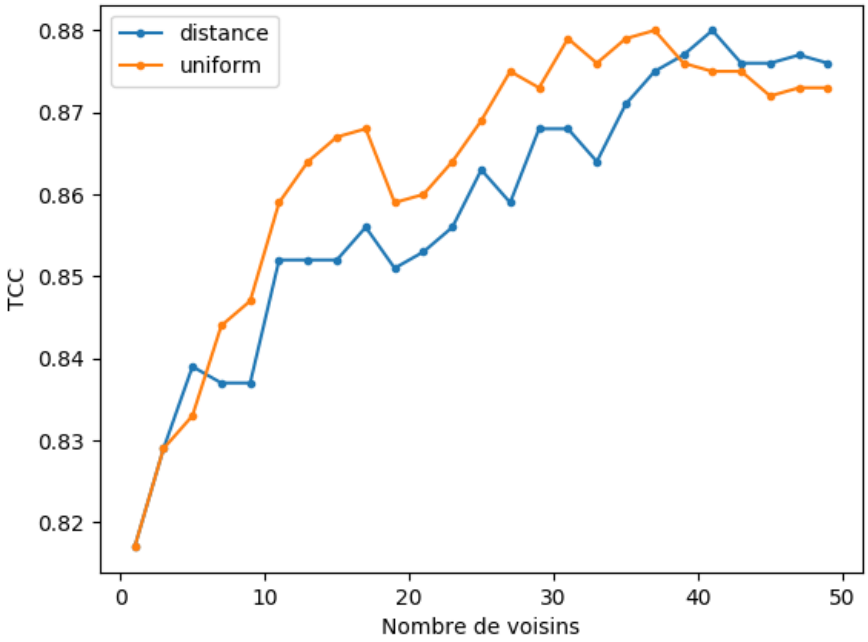

Fig 2. Plot on `dow_jones_index.data`

Cette courbe fait penser à la fonction logarithme, le TCC s'améliore avec l'augmentation de [`n_neighbours`] pour définir la classe mais devient de plus en plus plate.

Arbre de Décision

Paramètres modifiés:

1. Profondeur maximale (`max_depth`) : 1 à 99
2. Nombre d'exemples minimum par feuille (`min_samples_leaf`) : 2 à 20
3. Nombre minimal pour le découpage d'une node (`min_samples_split`) : 2 à 20

Le découpage d'une node se fait avec le meilleur gini par défaut.

Données du wisconsin

Table 5. Meilleurs scores sur wisconsin.dat

TCC Test Moyen	STD	TCC Train Moyen	Fit Time Moyen	Max Depth	Min Samples Leaf	Min Samples Split
0.875	0.03	0.88	0.002	1	2	2
0.875	0.03	0.88	0.002	99	19	19

Par soucis de représentation j'ai omis les 100 modèles (ou lignes) possédant le même TCC Test Moyen (0.875), [`max_depth`] augmentant de 1 à chaque ligne

Nous pouvons observer (ref. Table 1.5) que [`max_depth`] n'influence pas le TCC test moyen (`mean_test_score`) pour une certaine combinaison des paramètres [`min_samples_leaf`] et [`min_samples_split`].

Il est tout a fait possible que certains paramètres comme [`min_samples_leaf`] arrêtent la croissance de l'arbre avant d'arriver à la profondeur maximale, ainsi [`max_depth`] ne produit aucune amélioration du TCC à partir d'un certain seuil. Le même phénomène est possible avec le paramètre du nombre d'exemple minimal pour découper une node. Ces paramètres s'influence donc entre eux mais sont utiles pour se prémunir de l'overfitting, le pire des cas étant d'avoir une feuille pour chaque exemple.

Données du Dow Jones

Table 6. Meilleurs scores sur `dow_jones_index.data`

TCC Test Moyen	STD	TCC Train Moyen	Fit Time Moyen	Max Depth	Min Samples Leaf	Min Samples Split
0.923	0.09	0.94	0.002	3	2	7
0.923	0.09	0.94	0.002	3	19	19

Par soucis de représentation j'ai omis les 170 modèles (ou lignes) possédant le même TCC Test Moyen (0.923), tous ces modèles possèdent le même [`max_depth`] de 3.

Sur ces données Table 1.6 c'est la profondeur de l'arbre qui influence le plus les résultats, la profondeur restant à 3 malgré le changement des deux derniers paramètres pour les meilleurs TCC Test Moyen. On peut supposer que peu d'attributs suffisent à trouver la meilleure prédiction, le reste étant du bruit, il serait utile des les négliger dans nos modèles.

Perceptron Multi Couches

Paramètres modifiés:

1. Fonction d'activation (activation) : Identité, Tangente hyperbolique (tanh), Logistique, Unité de rectification linéaire (relu)
2. Alpha : 0.1, 1, 10
3. couches cachées et nombre de neurones y figurant (`hidden_layer_sizes`) : (5, 50), (5, 100), (10, 50), (10,100)
4. Taux d'apprentissage (`learning_rate`) : constant, invscaling, adaptif
5. solveur : lbfgs, sgd, adam

Données du wisconsin

mean_test_score	std_test_score	mean_train_score	mean_fit_time	activation	alpha	hidden_layer_sizes	learning_rate	solver
0.974	0.038	0.982	0.7918	logistic	1	(5, 100)	constant	lbfgs
0.972	0.038	0.978	0.9364	tanh	10	(10, 100)	invscaling	lbfgs
0.972	0.038	0.978	0.8705	tanh	10	(10, 100)	adaptive	lbfgs
0.972	0.038	0.979	0.8291	tanh	10	(5, 100)	invscaling	lbfgs
0.972	0.038	0.979	0.8074	tanh	10	(5, 100)	constant	lbfgs
0.972	0.038	0.979	0.8057	logistic	1	(10, 100)	adaptive	lbfgs
0.972	0.038	0.978	0.5517	tanh	10	(10, 50)	constant	lbfgs
0.972	0.038	0.979	0.4857	tanh	10	(5, 50)	invscaling	lbfgs
0.972	0.038	0.979	0.4714	tanh	10	(5, 50)	constant	lbfgs
0.972	0.036	0.978	0.5675	tanh	10	(10, 50)	invscaling	lbfgs
0.972	0.036	0.982	0.5308	logistic	1	(10, 50)	invscaling	lbfgs

Fig 3. Meilleurs scores sur wisconsin.dat

Pour le PMC les meilleurs résultats sont obtenus avec un alpha de 1 ou 10 et la fonction d'activation tangente hyperbolique le plus souvent et lbfgs comme solveur. Dans notre cas, nombre de de couches cachées et le nombres de neurones y figurant influencent peu les résultats. Je suppose que le alpha de 0.1 systématiquement plus mauvais dans notre cas est causé par la tendance à rester dans des minimas locaux moins bons lors de la descente de gradient.

Le TCC minimum obtenu est de 0.439 ce qui est mauvais. Nous pouvons déduire que la configuration des paramètres est d'autant plus important pour ce classificateur. Cependant, à part les trois paramètres cités avant, il est difficile de déterminer quelle est l'influence des autres paramètres car c'est sans doute leur combinaison qui explique cette variation de résultats.

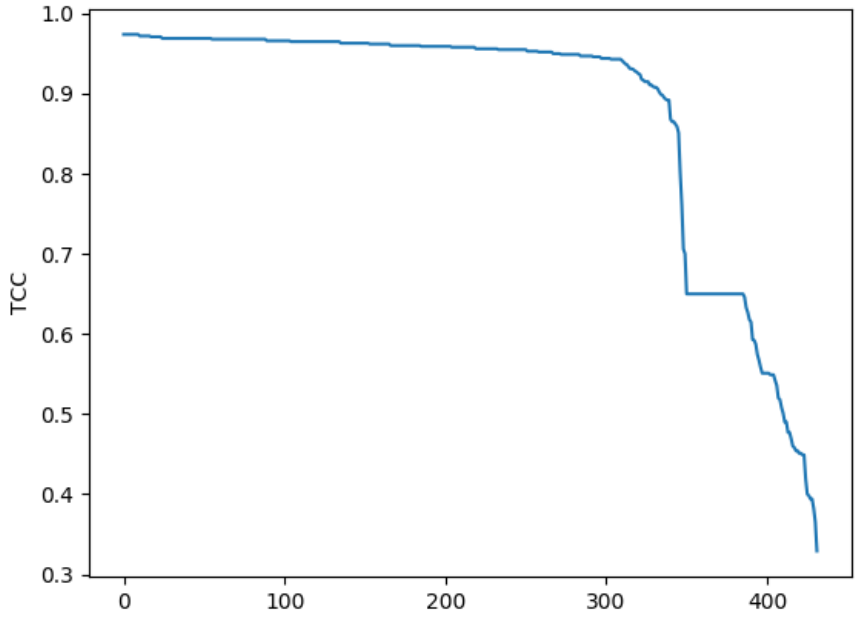

Fig 4. Décroissance du TCC sur wisconsin.data

Variation observable Fig 1.4 en classant les modèles par ordre de TCC décroissant. Nous pouvons remarquer un saut de 0,9 à 0,65 environ du TCC, ce saut serait expliqué la fonction logistique et une combinaison des autres paramètres et sinon le solveur invscaling et une combinaison avec les autres paramètres.

Données du Dow Jones

mean_test_score	std_test_score	mean_train_score	mean_fit_time	activation	alpha	hidden_layer_sizes	learning_rate	solver
0.974	0.038	0.982	0.8254	logistic	1	(5, 100)	invscaling	lbfgs
0.974	0.034	0.98	0.5005	tanh	10	(5, 50)	constant	lbfgs
0.972	0.044	0.981	0.8786	logistic	1	(10, 100)	constant	lbfgs
0.972	0.042	0.982	0.9145	logistic	1	(10, 100)	invscaling	lbfgs
0.972	0.04	0.982	0.898	logistic	1	(10, 100)	adaptive	lbfgs
0.972	0.038	0.979	0.5043	tanh	10	(5, 50)	invscaling	lbfgs
0.972	0.038	0.979	0.8432	tanh	10	(5, 100)	invscaling	lbfgs
0.972	0.038	0.979	0.8692	tanh	10	(5, 100)	adaptive	lbfgs
0.972	0.038	0.978	0.9345	tanh	10	(10, 100)	constant	lbfgs
0.972	0.033	0.971	1.3111	relu	1	(5, 100)	adaptive	adam

Fig 5. Meilleurs scores sur `dow_jones_index.data`

Il intéressant de relever la similarité des scores et des meilleurs paramètres malgré deux ensembles de données très différents.

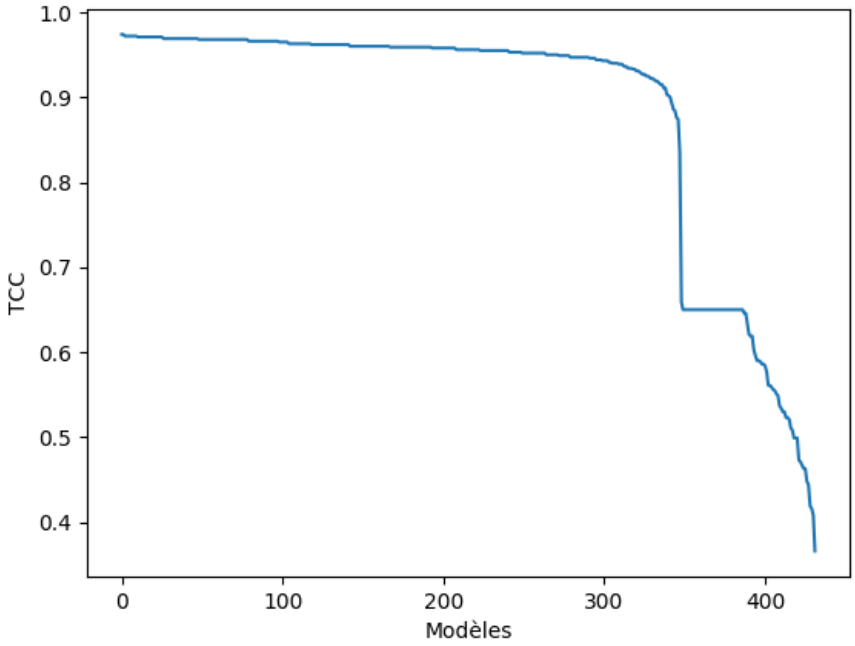

Fig 6. Décroissance du TCC sur `dow_jones_index.data`

On observe le même phénomène si l'on classe les modèles par TCC.

2. Ensembles

Bagging

Commande:

```
$ python3 machine_learning.py PMC bagging
```

Cette commande créer le fichier `output/wisc_pmc_bagging.csv` avec tous les modèles calculés selon les hyper-paramètres modifiés suivants:

1. Nombre de PMC `[n_estimators]`: 10, 15, 20
2. Nombre de neurones cachés `[hidden_layer_sizes]`: 50, 100, 200
3. Taux d'apprentissage `[learning_rate]`: invscaling, constant, adaptive
4. Nombre d'itérations `[Max samples]`: [0.5, 0.6321, 0.9]

Le Bagging consiste en un échantillonnage aléatoire avec remise des données d'entraînement afin d'entraîner un modèle (le nombre d'échantillons est défini par le paramètre 4). Le nombre de modèles entraîné ainsi est définit par le paramètre 1. Le score est moyenné sur les résultats de ces modèles.

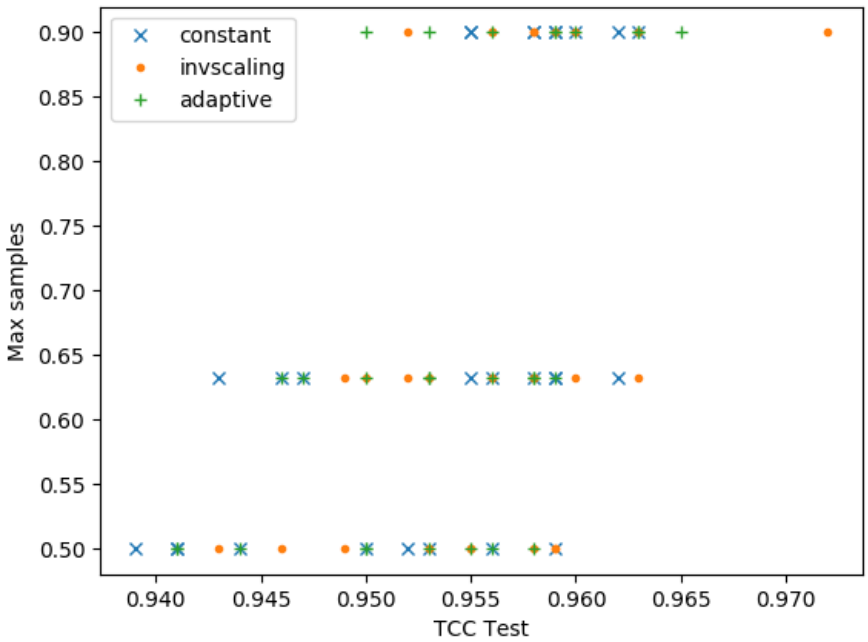

Fig 7. Résultats sur les données du wisconsin

Le nombre de paramètres étant important il est difficile de représenter les résultats par des courbes. Cependant nous pouvons voir que le pourcentage d'échantillonnage (Max samples) pour le bagging déplacent les résultats vers la droite (amélioration générale du TCC). Le taux d'apprentissage invscaling permet d'avoir les meilleurs TCC. Cependant il a été difficile de remarquer une corrélation entre le TCC et le nombre de neurones dans la couche cachée.

Arcing

Commande:

```
$ python3 machine_learning.py AD arcing
```

Cette commande créer le fichier `output/dj_ad_arcing.csv` avec tous les modèles calculés selon les hyper-paramètres modifiés suivants:

1. Nombre de d'arbres de décision `[n_estimators]`: 1 à 101 par pas de 10

Afin de créer des "stumps" `[max_depth]` est réglé à 1. Le choix de la coupure est déterminé par défaut par le meilleur gini.

Table 7. Résultats des TCC sur les données `dj_ad_arcing.csv`

TCC moyen test	STD	TCC Train Moyen	Fit Time Moyen	Max Depth	N Estimators
0.875	0.003	0.875	0.0036	1	1
0.875	0.003	0.875	0.0233	1	11
0.875	0.003	0.875	0.0488	1	21
0.875	0.003	0.875	0.0619	1	31
0.875	0.003	0.875	0.0796	1	41
0.875	0.003	0.875	0.0989	1	51
0.875	0.003	0.875	0.1179	1	61
0.875	0.003	0.875	0.1386	1	71
0.875	0.003	0.875	0.1546	1	81
0.875	0.003	0.875	0.1775	1	91
0.875	0.003	0.875	0.1695	1	101

Selon les résultats ci dessus la méthode d'ensemble arcing n'a pas aboutit à une amélioration du TCC en faisant varier le nombre de modèles (N estimators). Cependant les résultats plus mauvais obtenus dans la partie 1 ne sont pas présents.

3. Réseaux convolutifs

Voici les résultats obtenus selon la démarche proposée par le document "Deep Learning with Keras" p.229.

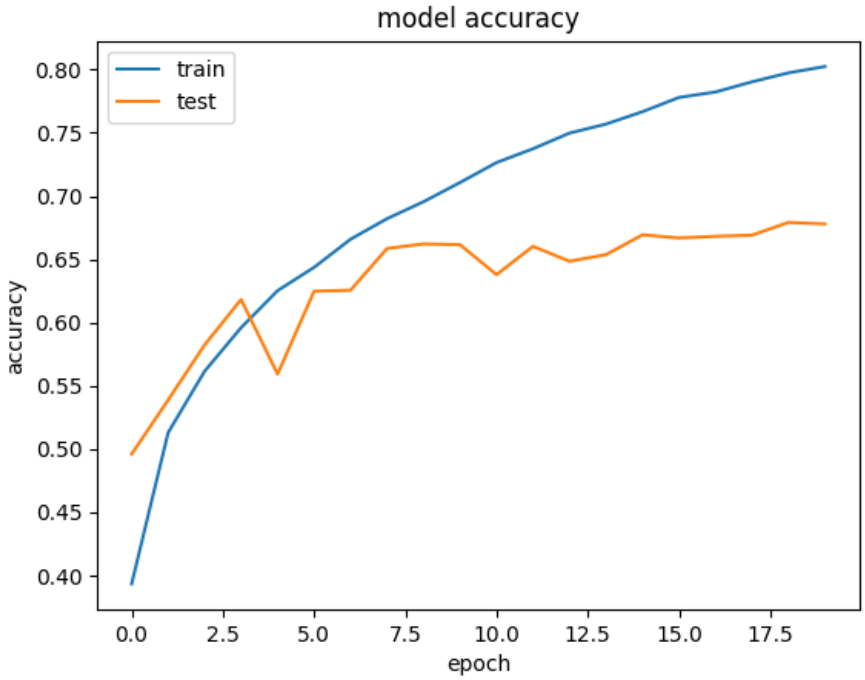

Fig 8. Précision du modèle sur les données CIFAR10

Le taux de classification correct des données test est de 0.68 à epoch (nombre d'itérations) 20. Le TCC (accuracy) d'entraînement (en bleu) diverge du TCC test (en orange) vers epoch 5. Il est donc raisonnable d'arrêter d'entraîner le modèle à cet instant. Toute itération supplémentaire n'aboutit qu'à une amélioration du TCC d'entraînement et donc à un overfitting sur ces données.

Amélioration du modèle

Voici les résultats obtenus avec les améliorations proposées p.234

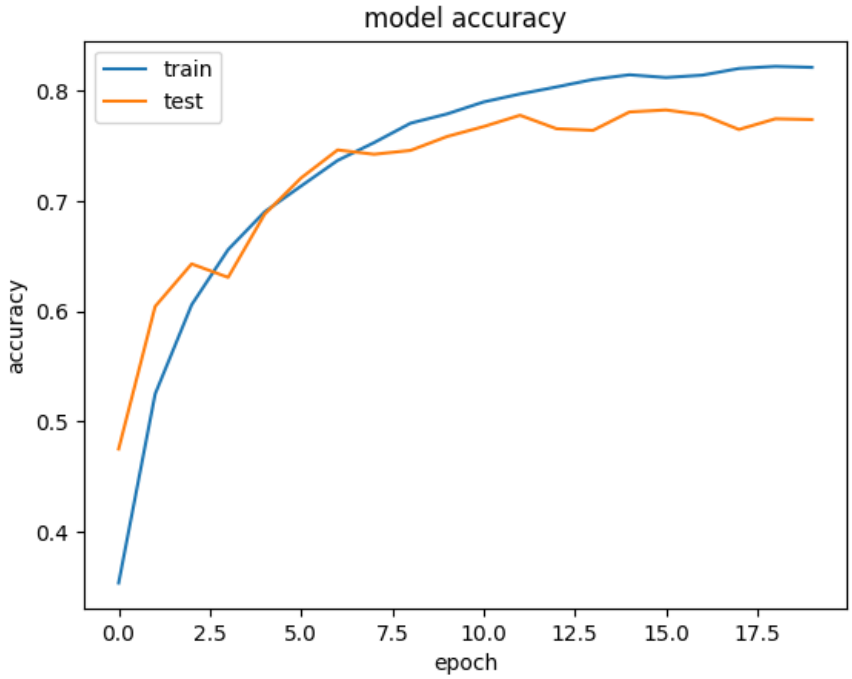

Fig 9. Précision du modèle amélioré sur les données CIFAR10

Une nette amélioration des résultats avec les données de test peut être observée. Le taux de classification correct des données test est de 0.76 à epoch 20. Les résultats TCC test et TCC entraînement sont assez proches. L'ajout de réseau profond avec plusieurs opérations de convolutions a donc été bénéfique pour ce modèle.

Conclusion

Nous avons pu voir que certains classificateurs se prêtent mieux à certaines données, leur complexité n'étant pas synonyme de meilleure performance. Nous avons vu aussi certaines techniques comme la validation croisée, le bagging et le arcing qui permettent de limiter le surentraînement afin d'obtenir des modèles plus robustes et généralisables. Tous ces choix sont à considérer lors de la création de nos modèles prédictifs, car certaines améliorations présentent une contrepartie en terme de temps calculs.

Bibliography

1. https://scikit-learn.org

2. Antonio Gulli, Sujit Pal. Deep Learning with Keras: Implement neural networks with Keras on Theano and TensorFlow. 2017 April; p.229-234.