Apprentissage automatique (classification)

Il faut installer les packages suivants : pandas, matplotlib, sklearn, numpy, torch et torchsummary.

Conseils :
— La plupart du temps, on traite des données simulées. Vous pouvez largement faciliter votre travail si vous regardez le code qui a permis de générer les données.
— Il est fortement recommandé de lire en ligne la documentation le site de sklearn, pytorch ...

Dans ce fichier, il y a 10 exercices (on peut considérer que les 9 premiers forment un méta-exercice et le dernier est un exercice à part entière). Ces exercices devraient prendre légèrement plus de 4 séances. Il y aura ensuite 3 exercices indépendants qui sont dans des fichiers à part.

Exercice 1 – SVM (cas simple)

Question 1 : Ouvrir le fichier exo1.py. Lancer le code et le comprendre dans ses grandes lignes. A quoi correspondent `x` et `y`. A quoi correspondent `x.shape` et `y.shape` ? Peut-on séparer linéairement les 2 classes ?

Question 2 : Sachant comment les deux classes ont été définies (les 2 classes ont été codées avec 0 et 1, on aurait pu faire autrement), quelle devrait être la classe des 3 échantillons suivants : (15,16), (-15,14) et (3,5) ? Compléter le code du fichier `exo1.py` de manière à vérifier vos prédictions.

Exercice 2 – SVM – influence de C et matrice de confusion

Question 1 : La fonction `generate_random_dataset_linear_separable_noise` (`generateData.py`) est un copié collé de la fonction `generate_random_dataset_linear_separable`. Modifier cette fonction de manière à ce qu'elle fasse comme `generate_random_dataset_linear_separable` à la différence que, quand `(y1-2x1)` est entre -1 et 6, cette échantillon est gardée et sa classe est choisie au hasard (2 chances sur 3 d'être dans la classe 0).

Question 2 : Lancer et analyser le code du fichier `exo2.py`. Que représente la matrice de confusion ? Vous pouvez noter la grande influence de C sur les résultats. Les résultats sont relativement difficiles à analyser, sauf pour le cas `C` très petit. On rappelle que l'on minimise le terme $\min \frac{1}{2}||w||^2 + C\sum_i \delta_i$, avec $\delta_i = max(0, 1 - y_i.(w^T x_i + b))$. Si C est beaucoup trop petit, l'objectif devient de minimiser $\min \frac{1}{2}||w||^2$! ! L'analyse sera plus aisée en utilisant un noyau non linéaire.

Question 3 : Modifier le fichier `exo2.py` de manière à utilier un SVM non linéaire (noyau RBF). Lancer. Commenter l'influence de C. A votre avis, peut-on conclure qu'il est préférable d'utiliser le SVM non-linéaire avec une grande valeure de C ? Pour répondre à la question, vous pouvez également penser à la façon dont sont générés les échantillons des différentes classes.

Exercice 3 – SVM non linéaire

Question 1 : Lancer et commenter les résultas obtenus en lançant le script `exo3.py`. Commenter notamment les tailles de `X`, de `Y` et de `a` qui sont affichées dans la fonction `my_kernel`.
Remarque : on aurait pu s'attendre à ce que le produit scalaire ne soit calculé que pour les vecteurs support lors des tests. Est-ce le cas ?

Question 2 : Changer la ligne
`dataset = generate_random_dataset_circle(size)` par la ligne
`dataset = generate_random_dataset_circle(size,1,1)` et la ligne
`dataset = generate_random_dataset_circle(10)` par
`dataset = generate_random_dataset_circle(10,1,1)`.
Remarquer que la transformation ne permet plus d'obtenir un problème linéairement séparable et donc que le noyau défini dans la fonction `my_kernel` n'est pas adapté. Changer la transformation et le noyau de manière adéquat. Tester et vérifier. On pourra conclure que le noyau adapté permet d'obtenir de meilleurs résultats mais il est moins général .

Exercice 4 – Normalisation des données

Question 1 : Lancer et commenter les résultas obtenus en lançant le script `exo4.py`. Pour la plupart des méthodes de classification et de régression, il est important de normaliser les données. Quels sont les paramètres estimés par la ligne `Scaler = StandardScaler().fit(X=x)`. Afficher les (on affiche juste -1 pour le moment). Que fait la ligne `x = Scaler.transform(x)` ?

Question 2 : Lorsque l'on veut prédire la classe d'un nouvel échantillon, il faut effectuer la même normalisation sur l'échantillon que la normalisation effectuée sur les données d'apprentissage. C'est pourquoi, les résultats obtenus après le code `print("testing new data")` ne sont pas satisfaisants. Appliquer la normalisation de deux manières différentes. Vérifier et tester.

Exercice 5 – Données déséquilibrées.

Question 1 : On parle de données déséquilibrées (unbalanced data) lorsque les deux classes ne sont pas représentées de manière égale. Lancer et commenter les résultas obtenus en lançant le script `exo5.py`. Pour vous aider, vous pouvez réfléchir aux points suivants : Existe-il une différence entre les caractéristiques des deux classes ? Pourquoi le classifieur semble-t-il si bon lorsque l'on normalise pas la matrice de confusion, et pourquoi semble-t-il si mauvais lorsque l'on normalise cette dernière ? Que fait réellement le classifieur ? Remarque : l'analyse des résultats doit donc toujours être faite avec soin !

Question 2 : Si la proportion des différentes classes dans la base de tests (pour les données testées dans l'avenir) est différente des proportions dans la base d'apprentissage, il faut empêcher que le classifieur soit biaisé par le fait que les données soient déséquilibrées. On peut utiliser deux grandes stratégies : des stratégies d'échantillonage pour que les classes soient représentées de manière équivalente dans la base d'apprentisssage ou attribuer des poids aux différents échantillons de manière à ré-équilibrer les données. Dans le second cas, le classifieur doit être modifié en conséquence pour prendre en compte les poids. Ajouter une option à la ligne `model = svm.SVC(kernel='linear')` pour ré-équilibrer les données avec des poids et vérifier que le classifieur ainsi obtenu ne fait guère mieux que le hasard (ce qui est normal à la vue des données). Théoriquement, le classifieur ne fait pas mieux que le hasard mais on peut parfois avoir cette impression. Ceci est lié au phénomène de

sur-apprentissage.

Exercice 6 – Sur-apprentissage.

Question 1 : Lancer le script `exo6.py`. Plus g est grand et plus on a l'impression que la frontière de décision est complexe. Par conséquent, plus les résultats sont bons, d'où la conclusion, plus g est grand, meilleurs sont les résultats. Qu'en pensez-vous ? Est-ce qu'il y a une faille dans ce raisonnement ? La question suivante peut vous aider...

Question 2 : Générer de nouvelles données (suivant le même modèle que les données d'apprentissage) qui ne seront pas utilisées pour l'apprentissage mais uniquement pour évaluer les performances du modèle une fois ce dernier appris. On pourra utiliser le même critère calculé à partir de la matrice de confusion. Commenter les résultats. A votre avis, quelle valeur de g faut-il prendre ? A retenir : on parle de sur-apprentissage (overfitting) quand les résulats sont plus satisfaisants sur la base d'apprentissage que sur la base de tests. Il est important d'avoir deux jeux de données différents pour évaluer les performances du modèle.

Exercice 7 – Estimation des hyperparamètres.

Question 1 : Lire le code. Comprendre ce que fait la fonction `GridSearchCV`. A votre avis, combien de classieurs SVM vont-ils être appris ? Vérifier en lançant le programme. Il faudra ajouter 1 au résultat affiché car un dernier modèle est appris à partir des hyper-paramètres préalablement estimés.

Question 2 : Faire passer la variable `Normalisation`. Qu'observez-vous ? Comme dans l'exercice 4, il est donc nécessaire de réaliser la normalisation des données. On pourrait directement écrire le code suivant :

```
Scaler = StandardScaler().fit(X=x)
x = Scaler.transform(x)
x2 = Scaler.transform(x2)
```

Cependant, ce n'est pas très cohérent à cause de la méthode `GridSearchCV`. Il est plus logique que pour chaque modèle appris, on apprenne à la fois l'étape de normalisation et l'étape de classification. Cela peut être réalisé par la classe `Pipeline`.

```
model = svm.SVC(kernel='rbf',class_weight='balanced')
new_model = Pipeline([('scaler', StandardScaler()), ('model', model)])
```

Modifier la variable `p_grid` de manière adéquat (rechercher sur internet comment il faut faire). Tester.

Remarque : on pourrait imaginer que le pipeline comporte plusieurs algorithmes (réduction de dimension, normalisation, classification...) et chacun des algorithmes pourrait avoir des hyperparamètres et on pourrait les estimer de la même manière avec `GridSearchCV`.

Exercice 8 – Mesure de la confiance des résultats couplée avec l'estimation des hyper-paramètres

Question 1 : Lire et comprendre le code. Que fait la fonction `StratifiedKFold` ?

Exercice 9 – Forêts aléatoires

Question 1 : Comprendre et lire le code `exo9a.py`. Les forêts aléatoires sont souvent adaptés à des problèmes de classification ou de régression lorsque le nombre de caractéristiques est important (il n'est pas utile de normaliser les données avec les forêts aléatoires). De plus, il permet également de mesurer l'importance de chaque caractéristique et d'estimer l'erreur de généralisation sans utiliser de bases de test ou de validation. Enfin, on aurait pu également utiliser la fonction `GridSearchCV` pour estimer différents hyperparamètres. Que peut-on dire des différents résultats obtenus ? En particulier, le résultat concernant l'importance des caractéristiques est-il cohérent avec la manière dont les données ont été générées.

Question 2 : Comprendre et lire le code `exo9b.py`. On sélectionne certaines caractéristiques en utilisant différentes stratégies. Lire rapidement comment fonctionnent les fonctions `SelectFromModel`, `RFE` et `RFECV`. Comment sont sélectionnés le nombre de caractéristiques dans chacun des algorithmes ? Est-ce que les résultats obtenus sont satisfaisants ? Commenter les temps d'exécution.

Question 3 : Comprendre et lire le code `exo9c.py`. Que fait ce code ? A noter que l'on estime la précision des résultats de classification en utilisant la même approche que l'exercice 8 (validation croisée) sauf que l'on n'estime pas d'hyperparamètres. On obient des résultats bien plus satisfaisants que le hasard. Est-ce cohérent avec la manière dont les données ont été générées ? Où est le problème ? Corriger le problème. Dans le corrigé, dans la seconde solution, on vous montre comment on peut aussi estimer des hyper-paramètres. On obtient alors dans les deux cas un résultat de l'ordre de 0.5. Pouvait-on s'attendre à ce résultat ?

Remarque : on réduit ici la dimension du problème en sélectionnant des caractéristiques. Il existe aussi des méthodes de réduction de dimension qui « mélangent » les caractéristiques (vous les verre plus tard).

Exercice 10 – Réseaux de neurones

Question 1 : Exécuter le code `exo10.py`. Le code peut être relativement long à exécuter la première fois à cause du téléchargement des données. Commenter le résultat des lignes suivantes :

```
print(len(train_data))
print(len(val_data))
print(len(train_loader))
print(len(test_loader))
print(len(val_loader))
```

Quelle est la taille des images et comment ces dernières ont-elles été normalisées ? Que représentent les images ? Quel est le problème de classification que l'on essaie ici de résoudre ?

Question 2 : Regarder le fichier `model.py` pour regarder comment a été créé la classe `Net`. Dans le constructeur, on crée les couches dont on a besoin. Dans la fonction `forward`, on définit la structure du réseau de neurone. Dans la fonction `def forward(self, x)`, `x` représente l'entrée. Attention,

`x` est un objet de type `Tensor` et non un tableau Numpy. On modifie ensuite `x` en le passant dans les différentes couches. La valeur retournée correspond à la sortie du réseau. A noter que l'on n'a pas utilisé une couche `softmax`. La fonction de coût utilisée la modélisera et en pratique, on n'en a pas forcément besoin pour les tests. Représenter sur papier les différentes couches ? Quel est le nombre de paramètres à estimer par couches ? Vous pouvez vérifier en regardant le résultat rendu par `summary(model,(`,28,28),20)1 (cf fonction lancée dans `exo10`). A quoi correspond à votre avis la sortie du réseau de neurone (quelle est sa dimension) ?

Question 3 : Changer la ligne `optimization= False` en `optimization= True` dans le fichier `exo10.py`. Vous pouvez regarder l'aide de la fonction de coût utilisée : `CrossEntropyLoss` pour vous assurer que vous avez bien répondu à la question 2. Que fait-on à chaque époch ? Avec la ligne suivante `for data, target in train_loader:`, combien de fois on va itérer sur cette boucle ? Quelle est la taille de `data` ? Pourquoi utilise-t-on un ensemble de validation ? A quoi peut-il servir à votre avis dans le cas présent ?

Question 4 : Exécuter le code. Que peut-on dire de l'optimisation ? A votre avis, est-ce que l'on réalise du sur-apprentissage ?

Question 5 : A la fin du fichier `exo10.py`, on a sauvé le modèle obtenu. On va l'utiliser dans le fichier `exo11.py` pour classer de nouvelles données. Lire le fichier et tester. Comment une nouvelle donnée est-elle classée ? Commenter les résultats ?

Question 6 : Les images sont mieux modélisées avec des réseaux convolutionnels. Vous pouvez répondre aux mêmes questions (de la 3 à la 6) en modifiant la ligne `model = Net()` en `model = LeNet()`. Vous pouvez utiliser 5 itérations à la place de 15.

Question 7 : Les courbes d'apprentissage (learning curves) consistent à calculer un critère soit sur une base de tests, soit sur la base d'apprentissage, en faisant varier la taille de la base d'apprentissage. Cela permet de diagnostiquer certains problèmes (modèle trop ou pas assez complexe) et de déterminer éventuellement s'il peut y avoir un gain à augmenter la taille de la base d'apprentissage. Plus de détails peuvent être trouvés sur :
`https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/`
A votre avis, en gardant le modèle `LeNet`, est-ce qu'il y aurait possiblement un intérêt à augmenter la taille de la base de données ?

Remarque : Il y a un vrai savoir-faire pour construire des architectures adaptés, pour optimiser convenablement les modèles. La puissance de modélisation est très importante : il y a de nombreuses types de couches avec des rôles différents... Certaines ont des effets régularisant (dropout), d'autres permettent de rendre l'optimsation plus facile (batchnorm), ... Vous trouverez quelques conseils ici :

`http://karpathy.github.io/2019/04/25/recipe/`