JUSTINE Jean

Rapport d'Analyse de Données

Git hub repository : https://github.com/Justine-IA/projet-starcraft/tree/master

I) Objectif :

L'objectif de ce projet est : utiliser une base de donnée de partie du jeu vidéo Starcraft contenant les actions de différents joueurs au cours de la partie et de prédire quels joueurs c'est en fonction de leurs actions au cours d'une partie.

II) Base de Données :

Pour cela, nous avons utilisé une base de données qui contient : un fichier TEST.CSV un fichier TRAIN.CSV et leur version longue avec donc dans chaque fichier, la première colonne, l'adresse des noms des joueurs, la deuxième, quelles classes ils ont choisi et tout le reste sont leurs actions au cours de la partie. (Choisir leurs classes est aussi une action)

III) Prétraitement des Données :

Les données nécessitaient un prétraitement car, premièrement les noms des joueurs été des URL j'ai donc extrait leur nom, ensuite, toutes les actions des joueurs, appuyer sur des touches et autre, ont dû être encoder car sinon inutilisable pour le modèle (il ne pouvait pas lire des actions comme s1 ou autre mais il peut lire des chiffres) l'encodage a donc attribué à chaque touche un chiffre pour pouvoir entrainer le modèle, ensuite les joueurs ont chacun un nombre d'actions différentes en fonction de leur partie, il y avait donc des lignes plus grande que d'autres pour chaque joueur, pour résoudre ce problème nous avons appliqué un masque, un masque de donnée c'est une technique utilisée dans le traitement des séquences (comme les séries temporelles ou le traitement du langage naturel) pour indiquer à un modèle quelles parties des données doivent être ignorées pendant l'entraînement.

IV) Choix et Entraînement des Modèles :

Une fois tous les problèmes résolus concernant les données, nous avons choisi d'entraîner différents modèles mais le model LSTM est le plus performant : basé sur un réseau de neurones récurrents (RNN) avec des couches LSTM (Long Short-Term Memory), est particulièrement adapté pour les séquences de données. En effet nous avons de très longues séquence de longueurs différentes, qui sont temporelle (elle « raconte » une partie de jeu), et ce modèle est robuste aux séquences de longueur variable, il arrive bien à capturer les dépendances temporelles dans des données séquentielles, ceci ainsi que son efficacité pour les tâches de classifications et de prédictions de série temporelle nous permets d'obtenir des résultats plus précis et fiables.

V) Evaluation de l'entrainement :

L'entrainement s'est très mal passé au débuts, je n'arrivais pas à dépasser une précision de 5%, ensuite j'ai changé mon modèle et l'ai amélioré (voir la partie VI) Optimisation) Mais il a été très longs à entrainer (100 epochs avec 4 sec par étapes pour 56 étapes et cela m'a pris 10h). J'ai donc eu une précision de : 70%

VI) Optimisation :

L'entrainement s'étant mal passé, pour changer cela j'ai essayé d'ajouter du poids de classes, mais cela a complètement bloqué mon modèle et il ne voulait plus s'entrainer, j'ai donc laissé tomber cette idée (des vestiges de cela sont toujours au seins de mon code), j'ai donc utiliser une approche différente, j'ai rajouté des couches à mon modèle, augmenté le nombre d'epoch (je suis passée de 10 à 100), diminuer le nombre de batch_size (de 128 à 64)au seins de mes couches mais également de l'entrainement de mon modèle(de 64 à 32) ,et cela m'a permis d'avoir des résultats beaucoup plus concluant.

Ici j'ai un dataset de 1792 échantillons et un batch_size de 32, les epochs sont de 56 itérations (je n'ai pas utilisé les versions longues des datasets).

(Voir définitions dans Annexe)

VII) Conclusion :

Pour conclure, les méthodes de machine learning se sont révélées assez efficaces pour automatiser la reconnaissance des joueurs le modèle LSTM est bon pour s'entrainer sur des datasets qui évoluent dans le temps, la méthode du masque est également très efficace pour traiter des données de différentes longueurs. Cependant, il est important de noter que le modèle est loin d'être parfait, beaucoup d'optimisation peuvent être faite, je n'ai pas utilisé les versions longues des datasets, donc mes résultats sont moins précis.

JUSTINE Jean

Annexe

Définition :

Le batch_size est le nombre d'exemples de données utilisés pour calculer une mise à jour des paramètres du modèle à chaque itération.

Une epoch correspond à un passage complet à travers l'ensemble du dataset par le modèle.

Les class weights sont des coefficients utilisés pour accorder plus d'importance à certaines classes lors de l'entraînement d'un modèle de classification, généralement utilisé pour traiter des problèmes de classes déséquilibrées.

```
Epoch 1/10
56/56 [==============================] - 212s 4s/step - loss: 5.5177 - accuracy: 0.0073 - val_loss: 5.1937 - val_accuracy: 0.0000e+00
Epoch 2/10
56/56 [==============================] - 210s 4s/step - loss: 5.2371 - accuracy: 0.0134 - val_loss: 5.1341 - val_accuracy: 0.0000e+00
Epoch 3/10
56/56 [==============================] - 237s 4s/step - loss: 5.0397 - accuracy: 0.0263 - val_loss: 5.0363 - val_accuracy: 0.0179
Epoch 4/10
56/56 [==============================] - 255s 5s/step - loss: 4.8561 - accuracy: 0.0364 - val_loss: 4.9664 - val_accuracy: 0.0067
Epoch 5/10
56/56 [==============================] - 247s 4s/step - loss: 4.7504 - accuracy: 0.0347 - val_loss: 5.1009 - val_accuracy: 0.0000e+00
Epoch 6/10
56/56 [==============================] - 250s 4s/step - loss: 4.6089 - accuracy: 0.0565 - val_loss: 5.2553 - val_accuracy: 0.0089
Epoch 7/10
56/56 [==============================] - 235s 4s/step - loss: 4.4890 - accuracy: 0.0621 - val_loss: 4.9764 - val_accuracy: 0.0179
Epoch 8/10
56/56 [==============================] - 252s 4s/step - loss: 4.4247 - accuracy: 0.0705 - val_loss: 4.8848 - val_accuracy: 0.0134
Epoch 9/10
56/56 [==============================] - 250s 4s/step - loss: 4.2774 - accuracy: 0.0872 - val_loss: 5.0221 - val_accuracy: 0.0201
Epoch 10/10
56/56 [==============================] - 244s 4s/step - loss: 4.1574 - accuracy: 0.0856 - val_loss: 5.0233 - val_accuracy: 0.0268
```



```
Epoch 86/100
56/56 [==============================] - 235s 4s/step - loss: 1.2328 - accuracy: 0.6298 - val_loss: 5.4754 - val_accuracy: 0.1518
Epoch 87/100
56/56 [==============================] - 235s 4s/step - loss: 1.2960 - accuracy: 0.6079 - val_loss: 4.3526 - val_accuracy: 0.2946
Epoch 88/100
56/56 [==============================] - 235s 4s/step - loss: 1.2205 - accuracy: 0.6303 - val_loss: 4.2179 - val_accuracy: 0.3058
Epoch 89/100
56/56 [==============================] - 235s 4s/step - loss: 1.1466 - accuracy: 0.6633 - val_loss: 4.2120 - val_accuracy: 0.3393
Epoch 90/100
56/56 [==============================] - 1346s 24s/step - loss: 1.0733 - accuracy: 0.6672 - val_loss: 4.2344 - val_accuracy: 0.3170
Epoch 91/100
56/56 [==============================] - 238s 4s/step - loss: 1.0402 - accuracy: 0.6812 - val_loss: 4.0143 - val_accuracy: 0.3616
Epoch 92/100
56/56 [==============================] - 237s 4s/step - loss: 1.1368 - accuracy: 0.6616 - val_loss: 4.2906 - val_accuracy: 0.3214
Epoch 93/100
56/56 [==============================] - 236s 4s/step - loss: 1.0876 - accuracy: 0.6628 - val_loss: 3.9189 - val_accuracy: 0.3750
Epoch 94/100
56/56 [==============================] - 233s 4s/step - loss: 1.2027 - accuracy: 0.6437 - val_loss: 4.2773 - val_accuracy: 0.3058
Epoch 95/100
56/56 [==============================] - 233s 4s/step - loss: 1.0846 - accuracy: 0.6790 - val_loss: 4.0905 - val_accuracy: 0.3438
Epoch 96/100
56/56 [==============================] - 232s 4s/step - loss: 1.0262 - accuracy: 0.6812 - val_loss: 4.0883 - val_accuracy: 0.3281
Epoch 97/100
56/56 [==============================] - 232s 4s/step - loss: 1.0467 - accuracy: 0.6890 - val_loss: 4.2776 - val_accuracy: 0.3192
Epoch 98/100
56/56 [==============================] - 23922s 435s/step - loss: 1.0289 - accuracy: 0.6840 - val_loss: 3.9005 - val_accuracy: 0.3996
Epoch 99/100
56/56 [==============================] - 314s 6s/step - loss: 0.9957 - accuracy: 0.6857 - val_loss: 4.0993 - val_accuracy: 0.3795
Epoch 100/100
56/56 [==============================] - 249s 4s/step - loss: 1.0292 - accuracy: 0.6806 - val_loss: 4.3370 - val_accuracy: 0.3013
```

JUSTINE Jean

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 2564)]            0

 reshape_1 (Reshape)         (None, 2564, 1)           0

 masking_1 (Masking)         (None, 2564, 1)           0

 lstm_2 (LSTM)               (None, 2564, 64)          16896

 batch_normalization_2 (Bat  (None, 2564, 64)          256
 chNormalization)

 dropout_2 (Dropout)         (None, 2564, 64)          0

 lstm_3 (LSTM)               (None, 64)                33024

 batch_normalization_3 (Bat  (None, 64)                256
 chNormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 179)               11635
...
Total params: 62067 (242.45 KB)
Trainable params: 61811 (241.45 KB)
Non-trainable params: 256 (1.00 KB)
```