TP DL & NLP Keras

Objectif : Création d'un modèle d'apprentissage pour prédire la rétention des employés dans une entreprise

Conditions préalables

Avant de commencer, vous aurez besoin des éléments suivants:

- Un environnement de développement Anaconda sur votre machine.
- Une installation de Jupyter Notebook . Anaconda installera Jupyter Notebook pour vous lors de son installation. Vous pouvez également suivre ce didacticiel pour un guide sur la navigation et l'utilisation de Jupyter Notebook.

Étape 1 - Prétraitement des données

Le prétraitement des données est nécessaire pour préparer vos données d'une manière qu'un modèle d'apprentissage en profondeur peut accepter. S'il existe des *variables catégorielles* dans vos données, vous devez les convertir en nombres car l'algorithme n'accepte que des chiffres. Une *variable catégorielle* représente des données quantitatives représentées par des noms. Dans cette étape, vous allez charger votre ensemble de données à l'aide `pandas`, qui est une bibliothèque Python de manipulation de données.

Après avoir accédé à Jupyter Notebook, cliquez sur le fichier **anaconda3** , puis cliquez sur **Nouveau** en haut de l'écran et sélectionnez **Python 3**.

Vous allez maintenant importer les modules requis pour le projet, puis charger l'ensemble de données dans une cellule de bloc-notes. Vous allez charger dans le `pandas` module pour manipuler vos données et `numpy` pour convertir les données en `numpy` tableaux. Vous convertirez également toutes les colonnes au format chaîne en valeurs numériques pour que votre ordinateur les traite. Pour ce faire, inserer le code suivant :

```
import pandas as pd
import numpy as np

df=pd.read_csv("https://raw.githubusercontent.com/mwitiderrick/kerasDO/master/HR_comma_sep.csv")
```

Vous avez importé `numpy` et `pandas`. Vous avez ensuite utilisé `pandas` pour charger dans l'ensemble de données pour le modèle.

Vous pouvez avoir un aperçu de l'ensemble de données avec lequel vous travaillez en utilisant head(). Il s'agit d'une fonction utile pandas qui vous permet de visualiser les cinq premiers enregistrements de votre trame de données. Ajoutez le code suivant à une cellule de bloc-notes, puis exécutez-le: `df.head()`

Vous allez maintenant procéder à la conversion des colonnes catégorielles en nombres. Pour ce faire, vous les convertissez en *variables factices* . Les variables fictives sont généralement des uns et des zéros qui indiquent la présence ou l'absence d'une caractéristique catégorielle. Dans ce type de situation, vous évitez également le *piège variable factice* en supprimant le premier factice. Pour ce faire, Insérez ce code :

```
feats = ['department','salary']
df_final = pd.get_dummies(df,columns=feats,drop_first=True)
```

`feats = ['department','salary']` définit les deux colonnes pour lesquelles vous

Étape 2 - Séparation de vos ensembles de données de formation et de test

Vous utiliserez `scikit-learn` pour diviser votre ensemble de données en une formation et un ensemble de tests. Cela est nécessaire pour que vous puissiez utiliser une partie des données des employés pour former le modèle et une partie pour tester ses performances. Le fractionnement d'un ensemble de données de cette manière est une pratique courante lors de la création de modèles d'apprentissage en profondeur.

Vous commencerez par importer le `train_test_split` module à partir du `scikit-learn` package. Il s'agit du module qui fournira la fonctionnalité de fractionnement. Insérez ce code :

```
from sklearn.model_selection import train_test_split
```

Une fois le `train_test_split` module importé, vous utiliserez la `left` colonne de votre ensemble de données pour prédire si un employé quittera l'entreprise. Par conséquent, il est essentiel que votre modèle d'apprentissage en profondeur n'entre pas en contact avec cette colonne. Insérez ce qui suit dans une cellule pour supprimer la `left` colonne:

```
X = df_final.drop(['left'],axis=1).values
```

```
y = df_final['left'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
```

Étape 3 - Transformer les données

Lors de la création de modèles d'apprentissage en profondeur, il est généralement recommandé de mettre à l' *échelle* votre ensemble de données afin de rendre les calculs plus efficaces. Dans cette étape, vous allez mettre à l'échelle les données à l'aide de `StandardScaler`; cela garantira que vos valeurs de jeu de données ont une moyenne de zéro et une variable d'unité. Cela transforme l'ensemble de données pour qu'il soit distribué normalement. Vous utiliserez le `scikit-learn` `StandardScaler` pour mettre à l'échelle les entités pour qu'elles soient dans la même plage. Cela transformera les valeurs pour avoir une moyenne de 0 et un écart-type de 1. Cette étape est importante car vous comparez des entités qui ont des mesures différentes; il est donc généralement requis dans l'apprentissage automatique.

Pour mettre à l'échelle l'ensemble de formation et l'ensemble de test, ajoutez ce code

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Ici, vous commencez par importer le `StandardScaler`et en appelant une instance de celui-ci. Vous utilisez ensuite sa `fit_transform` méthode pour mettre à l'échelle l'ensemble de formation et de test.

Étape 4 - Construire le réseau neuronal artificiel

Vous allez maintenant utiliser `keras` pour construire le modèle d'apprentissage en profondeur. Pour ce faire, vous allez importer `keras`, qui sera utilisé `tensorflow` comme backend par défaut. À partir de `keras`, vous importerez ensuite le `Sequential` module pour initialiser le *réseau neuronal artificiel* . Un *réseau de neurones artificiels* est un modèle de calcul qui est construit en s'inspirant du fonctionnement du cerveau humain. Vous importerez également le `Dense` module, ce qui ajoutera des couches à votre modèle d'apprentissage en profondeur.

Lors de la création d'un modèle d'apprentissage en profondeur, vous spécifiez généralement trois types de couches:

- La *couche d'entrée* est la couche à laquelle vous transmettrez les fonctionnalités de votre jeu de données. Aucun calcul ne se produit dans cette couche. Il sert à transmettre des fonctionnalités aux couches cachées.
- Les *calques masqués* sont généralement les calques entre le calque d'entrée et le calque de sortie, et il peut y en avoir plusieurs. Ces couches effectuent les calculs et transmettent les informations à la couche de sortie.
- La *couche de sortie* représente la couche de votre réseau neuronal qui vous donnera les résultats après la formation de votre modèle. Il est responsable de la production des variables de sortie.

Pour importer les `Keras`, `Sequential` et les `Dense` modules, exécutez le code suivant dans votre cellule de portable:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

Vous allez utiliser `Sequential` pour initialiser une pile linéaire de couches. Comme il s'agit d'un *problème de classification* , vous allez créer une variable de classificateur. Un *problème de classification* est une tâche dans laquelle vous avez étiqueté des données et souhaitez faire des prédictions en fonction des données étiquetées. Ajoutez ce code à votre bloc-notes pour créer une variable de classificateur:

```
classifier = Sequential()
```

Vous pouvez maintenant commencer à ajouter des couches à votre réseau

```
classifier.add(Dense(9, kernel_initializer = "uniform",activation =
"relu", input_dim=18))
classifier.add(Dense(1, kernel_initializer = "uniform",activation =
"sigmoid"))
```

on compile avec un optimizer et une fonction d'erreur :

```
classifier.compile(optimizer= "adam",loss =
"binary_crossentropy",metrics = ["accuracy"])
```

puis on execute l'entrainement :

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 1)
```

Étape 5 - Exécution de prévisions sur l'ensemble de test

Pour commencer à faire des prédictions, vous allez utiliser l'ensemble de données de test dans le modèle que vous avez créé. Keras vous permet de faire des prédictions en utilisant la `.predict()` fonction.

Insérez le code suivant suivante pour commencer à faire des prédictions:

```
y_pred = classifier.predict(X_test)
```

Puisque vous avez déjà formé le classificateur avec l'ensemble de formation, ce code utilisera l'apprentissage du processus de formation pour faire des prédictions sur l'ensemble de test. Cela vous donnera les probabilités du départ d'un employé. Vous travaillerez avec une probabilité de 50% et plus pour indiquer une forte probabilité que l'employé quitte l'entreprise. Entrez la ligne de code suivante dans votre cellule afin de définir ce seuil:

```
y_pred = (y_pred > 0.5)
```

Étape 7 - Faire une seule prédiction

Dans cette étape, vous ferez une seule prédiction compte tenu des détails d'un employé avec votre modèle. Vous y parviendrez en prédisant la probabilité qu'un seul employé quitte l'entreprise. Vous passerez les fonctionnalités de cet employé à la `predict` méthode. Comme vous l'avez fait précédemment, vous allez également mettre à l'échelle les fonctionnalités et les convertir en `numpy` tableau.

Pour transmettre les fonctionnalités de l'employé, exécutez le code suivant dans une cellule:

```
new_pred = classifier.predict(sc.transform(np.array([[0.26,0.7 ,3.,
238., 6., 0.,0.,0.,0., 0.,0.,0.,0.,0.,1.,0., 0.,1.]])))
```

Ces fonctionnalités représentent les fonctionnalités d'un seul employé. Comme indiqué dans l'ensemble de données à l'étape 1, ces caractéristiques représentent: le niveau de satisfaction, la dernière évaluation, le nombre de projets, etc. Comme vous l'avez fait à l'étape 3, vous devez transformer les fonctionnalités d'une manière que le modèle d'apprentissage en profondeur peut accepter.

Ajoutez un seuil de 50% avec le code suivant:

```
new_pred = (new_pred > 0.5)
new_pred
```

Ce seuil indique que lorsque la probabilité est supérieure à 50%, un employé quittera l'entreprise.

Vous pouvez voir dans votre sortie que l'employé ne quittera pas l'entreprise:

```
Output
array([[False]])
```