Préparation des données

Ce script est dédié à la préparation des données pour entraîner un modèle de détection d'annonces d'emploi frauduleuses. Voici une description de la phase de préparation des données:

Importing DataSet

```
# 1st dataset (main feed)
jobs_file = tf.keras.utils.get_file(
    fname="./fraudulent_jobs.csv",

origin="https://uofi.box.com/shared/static/sfw0eqvj7q49vmexpztqke7xzhspevnb.csv")
df=pd.read_csv(jobs_file)
df.head()
```

- Le script commence par obtenir le jeu de données principal (fraudulent_jobs.csv) en utilisant tf.keras.utils.get_file(). Ce jeu de données contient probablement des données étiquetées, avec une colonne indiquant si une annonce d'emploi est frauduleuse ou non.
- De plus, il charge le jeu de données des offres d'emploi LinkedIn à partir d'un fichier local nommé jobs_linkedin.csv. Le DataFrame résultant est stocké dans une variable appelée **df.**

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
df['full_text'] = df['title'] + " " + df['location'] + " " +
df['department'] + " " + df['company_profile'] + " " +
df['description'] + " " + df['requirements'] + " " + df['benefits']

df['full_text'] = df['full_text'].fillna('')

# Use TF-IDF to convert the text data into numerical data

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(df['full_text'])

# The 'fraudulent' column is the target variable

y = df['fraudulent']

# Split the data into a training set and a test set

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a Random Forest Classifier on the training data

# Train a Random Forest Classifier on the training data with
adjusted parameters

model = RandomForestClassifier(n_estimators=1000, random_state=42)

model.fit(X_train, y_train)

# Use the model to predict the 'fraudulent' column for the test data

y_pred = model.predict(X_test)

# Print a classification report to see how well the model performed

print(classification_report(y_test, y_pred))

# Load the other dataset

jobs_linkedin = pd.read_csv('/content/jobs_linkedin.csv')

# Combine all the text data into a single column

#combine all text into 1 column - full_text

jobs_linkedin['full_text'] = jobs_linkedin['title'] + " " +
jobs_linkedin['location'] + " " + jobs_linkedin['department'] + " "
+ jobs_linkedin['company_profile'] + " " +
jobs_linkedin['description'] + " " + jobs_linkedin['requirements'] +
" " + jobs_linkedin['benefits']

jobs_linkedin['full_text'] = jobs_linkedin['full_text'].fillna('')
```

```
# Use the same TF-IDF vectorizer to transform the text data into
numerical data

X_jobs_linkedin = vectorizer.transform(jobs_linkedin['full_text'])

# Use the trained model to predict the 'fraudulent' column for the
other dataset

jobs_linkedin['fraudulent'] = model.predict(X_jobs_linkedin)

# Save the DataFrame to a CSV file

jobs_linkedin.to_csv('jobs_linkedin_with_fraudulent.csv',
index=False, encoding='utf-8')

print("Data generation done!")
```

- Il crée une fonctionnalité textuelle combinée en fusionnant plusieurs colonnes de données textuelles d'un DataFrame (**df**).
- Il utilise la vectorisation TF-IDF pour transformer les données textuelles en données numériques.
- Il divise les données en ensembles d'entraînement et de test.
- Il entraîne un classificateur RandomForest sur les données d'entraînement.
- Il utilise le modèle entraîné pour prédire la colonne de fraude pour les données de test et génère un rapport de classification pour évaluer les performances du modèle.
- Il charge un autre ensemble de données d'annonces d'emploi LinkedIn, effectue les mêmes étapes de prétraitement et utilise le modèle entraîné pour prédire la fraude dans cet ensemble de données.
- Enfin, il enregistre les résultats dans un fichier CSV et affiche un message de fin d'exécution.

```
from IPython.display import display, FileLink

import ipywidgets as widgets

from IPython.display import display

# Create a file upload widget

upload_widget = widgets.FileUpload()

# Display the upload widget

display(upload_widget)

# Define a function to handle the uploaded file

def handle_upload(change):

    uploaded_filename = next(iter(upload_widget.value))
```

```
    content = upload_widget.value[uploaded_filename]['content']

    # Write the content to a local file

    local_filename = f'./{uploaded_filename}'

    with open(local_filename, 'wb') as f:

        f.write(content)

    print(f'File {uploaded_filename} uploaded and saved as
{local_filename}')

# Attach the function to the widget's event

upload_widget.observe(handle_upload, names='_counter')
```

- Il importe les modules nécessaires pour créer et gérer le widget de téléchargement de fichiers.
- Il crée un widget de téléchargement de fichiers en utilisant **`widgets.FileUpload()`**.
- Il affiche le widget de téléchargement.
- Il définit une fonction **`handle_upload`** pour gérer le fichier téléchargé.
    - Cette fonction récupère le nom du fichier téléchargé et son contenu.
    - Ensuite, elle écrit le contenu du fichier dans un fichier local.
    - Elle affiche un message indiquant que le fichier a été téléchargé et enregistré localement.
- Il attache la fonction **`handle_upload`** à l'événement du widget de téléchargement.

```
# Import File from local memory after upload.

df2=pd.read_csv('/content/CompiledjobListNigeria.csv')

# Find the maximum value in the 'Code' column of df

max_code = df['job_id'].max()

# Check if 'job_id' column exists in df, and add it with index row
if not

if 'job_id' not in df2.columns:

    df2['job_id'] = df2.reset_index().index

# Check if 'ColumnName' column exists in df, and add it with index
row if not

if 'benefits' not in df2.columns:

    df2['benefits'] = None

# Check if 'ColumnName' column exists in df, and add it with index
row if not
```

```
if 'function' not in df2.columns:
    df2['function'] = None
# Check if 'ColumnName' column exists in df, and add it with index
row if not
if 'industry' not in df2.columns:
    df2['industry'] = None
# Check if 'ColumnName' column exists in df, and add it with index
row if not
if 'required_education' not in df2.columns:
    df2['required_education'] = None
# Check if 'ColumnName' column exists in df, and add it with index
row if not
if 'required_experience' not in df2.columns:
    df2['required_experience'] = None
# Increment the 'job_id' column in df2 by the maximum value from df
+ 1
df2['job_id'] = df2['job_id'].apply(lambda x: x + max_code + 1)
# Add a new column 'has_company_logo' with random values of 0.0 or
1.0
df2['has_company_logo'] = np.random.choice([0, 1], size=len(df2))
# Add a new column 'has_questions' with random values of 0.0 or 1.0
df2['has_questions'] = np.random.choice([0, 1], size=len(df2))
# Add a new column 'telecommuting' with random values of 0.0 or 1.0
df2['telecommuting'] = np.random.choice([0, 1], size=len(df2))
# Rename the columns in the second dataframe to match the columns in
the first dataframe
df2.rename(columns={'job_title': 'title',
                    'company_name': 'company_profile',
                    'company_desc': 'description',
                    'job_requirement': 'requirements',
                    'salary': 'salary_range',
```

```
                    'label': 'fraudulent'}, inplace=True)

# Remove the 'job_desc' column

df2 = df2.drop('job_desc', axis=1)

column_order = ['job_id', 'title',  'location', 'department',
'salary_range', 'company_profile',  'description',  'requirements',
'benefits', 'telecommuting',  'has_company_logo', 'has_questions',
'employment_type',  'required_experience',  'required_education',
'industry', 'function', 'fraudulent']

df2 = df2[column_order]

df2.head()
```

- Il importe un fichier CSV depuis un emplacement local dans la mémoire.
- Il trouve la valeur maximale dans la colonne 'job_id' du DataFrame **df**.
- Il vérifie si certaines colonnes existent dans le DataFrame **df2**, et les ajoute avec des valeurs par défaut si elles n'existent pas.
- Il incrémente les valeurs de la colonne 'job_id' dans **df2** en ajoutant la valeur maximale de 'job_id' de **df** + 1.
- Il ajoute de nouvelles colonnes ('has_company_logo', 'has_questions', 'telecommuting') avec des valeurs aléatoires de 0 ou 1.
- Il renomme les colonnes dans **df2** pour qu'elles correspondent aux colonnes dans le premier DataFrame.
- Il supprime la colonne 'job_desc' de **df2**.
- Il réorganise les colonnes dans **df2** dans un ordre spécifique.
- Enfin, il affiche les premières lignes du DataFrame **df2**.

```
# Merge datas

merged_df = pd.merge(df, df2, on=['job_id', 'title',  'location',
'department', 'salary_range', 'company_profile',  'description',
'requirements', 'benefits', 'telecommuting',  'has_company_logo',
'has_questions',  'employment_type',  'required_experience',
'required_education', 'industry', 'function', 'fraudulent'],
how='outer')
```

- Ce script fusionne deux DataFrames, **df** et **df2** sur les colonnes communes suivantes :
    - 'job_id', 'title', 'location', 'department', 'salary_range', 'company_profile', 'description', 'requirements', 'benefits', 'telecommuting', 'has_company_logo', 'has_questions', 'employment_type', 'required_experience', 'required_education', 'industry', 'function', 'fraudulent'.

  en utilisant une fusion externe (outer join).

```
# Import Scrapping File from local memory after scrapping process.

df3=pd.read_csv('/content/jobs_linkedin.csv')

# Find the maximum value in the 'Code' column of df

max_code = merged_df['job_id'].max()

# Increment the 'job_id' column in df2 by the maximum value from df
+ 1df3['job_id'] = df3['job_id'].apply(lambda x: x + max_code )

# Convert multiple columns to object type

columns_to_convert = ['department', 'salary_range', 'description',
'requirements', 'benefits', 'required_education']

df3[columns_to_convert] = df3[columns_to_convert].astype(str)

# Remove the 'Link' column

#df2 = df2.drop('Link')

column_order = ['job_id', 'title',  'location', 'department',
'salary_range', 'company_profile',  'description',  'requirements',
'benefits', 'telecommuting',  'has_company_logo', 'has_questions',
'employment_type',  'required_experience',  'required_education',
'industry', 'function', 'fraudulent']

df3 = df3[column_order]

# Merge datas

final_merged_df = pd.merge(merged_df, df3, on=['job_id', 'title',
'location', 'department', 'salary_range', 'company_profile',
'description',  'requirements', 'benefits', 'telecommuting',
'has_company_logo', 'has_questions',  'employment_type',
'required_experience',  'required_education', 'industry',
'function', 'fraudulent'], how='outer')

print(final_merged_df)
```

- Il importe un fichier CSV depuis un emplacement local dans la mémoire.
- Il trouve la valeur maximale dans la colonne 'job_id' du DataFrame **`merged_df`**.
- Il incrémente les valeurs de la colonne 'job_id' dans **`df3`** en ajoutant la valeur maximale de 'job_id' de **`merged_df`**.
- Il convertit plusieurs colonnes en type objet.
- Il supprime la colonne 'Link' (commentée, donc actuellement inactive).
- Il réorganise les colonnes dans **`df3`** dans un ordre spécifique.
- Il fusionne les données de **`merged_df`** et **`df3`** sur les colonnes communes, en utilisant une fusion externe (outer join).
- Enfin, il affiche le DataFrame résultant de la fusion (**`final_merged_df`**).

Exploring the DataSet

```
final_merged_df.info()
```

- Total Entries: The dataset has a total of 17,880 entries.
- The columns of the dataset
- Missing Values: Some columns have missing values, such as 'location,' 'department,' 'salary_range,' 'company_profile,' and others. The number of non-null entries in each column is provided.
- Data Types: The data types include integers for numerical columns and objects for textual columns.

```
# get unique values from the dataset

final_merged_df.nunique()

# Count the sum of missing values for each column

final_merged_df.isna().sum()

# Count the occurrences of unique values in the 'fraudulent' column

final_merged_df['fraudulent'].value_counts()
```

Cela permet de:

- comprendre la variabilité des données dans chaque colonne.
- déterminer quelles colonnes ont des données manquantes et dans quelle mesure.
- comprendre la répartition des valeurs de la colonne 'fraudulent', ce qui est particulièrement utile dans le cas d'une colonne de catégorie binaire comme celle-ci.

- Text columns are title, location, department, company profile, description, requirements, and benefits, industry, function.
- Categorical columns are employment type, required experience, and required education.
- Numeric variables are salary (low/high), work_remote, has company logo, and has questions.
- Column to Predict is fraudulent.

```
#separate columns based on their type
```

```
text_cols = ['title', 'location', 'department', 'company_profile',
'description', 'requirements', 'benefits']

categorical_cols = ['employment_type', 'required_experience',
'required_education', 'industry', 'function']

numeric_cols = ['work_remote', 'has_company_logo', 'has_questions',
'salary_low', 'salary_high']

col_to_predict = ['fraudulent']
```

Ce script segmente les colonnes d'un DataFrame en fonction de leur type de données. Voici un aperçu rapide de chaque groupe de colonnes :

- **Colonnes de texte** : Elles comprennent des données textuelles telles que le titre de l'emploi, la localisation, la description de l'entreprise, etc. Ces données sont souvent utilisées pour extraire des informations textuelles ou pour des analyses de texte.
- **Colonnes catégorielles** : Elles contiennent des données catégorielles telles que le type d'emploi, l'expérience requise, l'éducation requise, etc. Elles sont souvent utilisées pour des analyses de catégorisation ou comme variables catégorielles dans les modèles.
- **Colonnes numériques** : Elles incluent des données numériques comme les indicateurs binaires (par exemple, le travail à distance, le logo de l'entreprise) ou des valeurs numériques continues (par exemple, les salaires). Elles sont utilisées pour des calculs numériques ou comme variables numériques dans les modèles.
- **Colonne à prédire** : C'est la variable cible que le modèle cherchera à prédire, dans ce cas, la fraude. Elle est généralement utilisée lors de l'entraînement d'un modèle d'apprentissage automatique.

Cleaning Dataset

```
def move_column(df, col):
  df['Temp_Col'] = df[col]
  df.drop(columns=[col], inplace=True)
```

```
    df.rename(columns={'Temp_Col': col}, inplace=True)

    return df
```

- La fonction prend deux arguments : un DataFrame **`df`** et le nom d'une colonne **`col`**.
- Elle crée une nouvelle colonne temporaire dans le DataFrame, copiant les données de la colonne spécifiée.
- Ensuite, elle supprime la colonne originale spécifiée.
- Elle renomme la colonne temporaire avec le nom de la colonne d'origine.
- Enfin, elle retourne le DataFrame modifié.

```
df['employment_type'].fillna('No Data', inplace=True)

df['required_experience'].fillna('No Data', inplace=True)

df['required_education'].fillna('No Data', inplace=True)

df['industry'].fillna('No Data', inplace=True)

df['function'].fillna('No Data', inplace=True)
```

- Pour chaque colonne spécifique dans le DataFrame **`df`**, la méthode **`fillna()`** est utilisée pour remplacer les valeurs manquantes (NaN) par la chaîne de caractères 'No Data'.
- Les colonnes ciblées sont 'employment_type', 'required_experience', 'required_education', 'industry', et 'function'.
- L'argument **`inplace=True`** indique que les modifications doivent être appliquées directement sur le DataFrame **`df`**, sans nécessiter une réaffectation explicite.

```
#rows that are entirely null in the text columns

null_text = df[(df['location'].isna()) & (df['department'].isna()) &
(df['company_profile'].isna()) & (df['requirements'].isna()) &
(df['benefits'].isna())] # 'location', 'department',
'company_profile', 'description', 'requirements', 'benefits'

print(f"There are {null_text.shape[0]} rows where all text columns
aside from 'title' and 'description' are empty. Of these rows,
{null_text['fraudulent'].sum()} are fraudulent posts.")
```

- Le code identifie les lignes dans le DataFrame **`df`** où toutes les colonnes de texte, à l'exception de 'title' et 'description', sont vides.
- Il crée un nouveau DataFrame appelé **`null_text`** qui contient ces lignes en utilisant des conditions où chaque colonne de texte est NaN (non définie).
- Il imprime le nombre de lignes identifiées ainsi que le nombre de lignes frauduleuses parmi elles.

```
# update US with USA so it does not get mixed up with the word 'us'

df = update_text(df) #replace urls, email, phone numbers (contact
details) ?? maybe presence/absence could indicate a fake post? #

df['full_text'] = df['full_text'].replace(r'http\S+', '',
regex=True).replace(r'www\S+', ' ',
regex=True).replace(r'#PHONE\S+', '',
regex=True).replace(r'#EMAIL\S+', '', regex=True)

#separate 2 separate words that have been put together (i.e.
PinterestLoves -> Pinterest Loves)

def space_words(all_text):

  import re

  #print('start', all_text)

  result = re.sub('(?<=[A-Za-z])(?=[A-Z][a-z])', '~', all_text)

  result = re.split('~', result)

  result = ' '.join(result)

  all_text = result # print('end',all_text)

  return all_text

salary_df = df['salary_range'].str.split(pat='-', n=-1, expand=True)

#separate salary range into two columns

salary_df[0].unique()[90:100] #months (strings) are included as
salaries due to csv interpretation of the salary range
```

- Le code divise la colonne 'salary_range' du DataFrame **`df`** en deux colonnes distinctes en utilisant la méthode **`split()`** de la chaîne de caractères.
- Il divise la chaîne en fonction du délimiteur '-' et crée un DataFrame **`salary_df`** avec les résultats.
- La méthode **`unique()`** est ensuite appliquée à la première colonne de **`salary_df`** pour afficher les 10 premières valeurs uniques après les 90 premières. Cela peut aider à identifier des anomalies ou des problèmes de données.
- Il note également que des mois (chaînes de caractères) peuvent être inclus comme salaires en raison de l'interprétation du fichier CSV pour la plage de salaires.

```
#convert the months in the salary columns into integers
```

```
months_to_int = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5,
'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec':
12}

for i in range(salary_df[0].shape[0]):

  if salary_df[0][i] in months_to_int.keys(): salary_df[0][i] =
months_to_int[salary_df[0][i]]

for i in range(salary_df[1].shape[0]):

  if salary_df[1][i] in months_to_int.keys(): salary_df[1][i] =
months_to_int[salary_df[1][i]]

#verify conversion of string months to integers

salary_df[0].unique()[90:100]
```

Replace Missing Values

```
import sklearn

from sklearn.model_selection import train_test_split

#split df into train/val/test so I can use the mean of salary_low &
salary_high from the training data

print('dataframe shape:',df.shape)

train, test_x = train_test_split(df, train_size=0.8, random_state=1,
shuffle=True, stratify=df.fraudulent.values)

print("train shape:",train.shape)

print('test shape:',test_x.shape)

train_x, val_x = train_test_split(train, train_size = 0.8,
random_state=1, shuffle = True, stratify=train.fraudulent.values)

print("train_x shape:",train_x.shape)

print('val shape:',val_x.shape)


# combine all text columns into the title column # drop all text
columns except full_text # rename 'title' column to 'full_text'

df['title'] = df['full_text']
```

```
df.drop(columns=['location', 'department', 'company_profile',
'description', 'requirements', 'benefits', 'full_text'],
inplace=True)

df.rename(columns={'title': 'full_text'}, inplace=True) #put fraud
column at the end of the dataframe

df = move_column(df, 'fraudulent')

df.head(1)
```

Conclusion

- replaced NA values in categorical columns with 'No Data'
- created 'full_text' column with all textual columns combined
- split the 'salary_range' column into 2 for the range --> salary_low, salary_high
- replace 'months' in salary columns with integers
- replace NA values in salary columns with -1, create new columns in main dataframe for salary (low/high), replace -1 with NaN and cast to type Integer64
- move full_text to the front of the dataframe, move fraudulent to the end of the dataframe; and rename both columns