Universidad ORT

Uruguay Facultad de Ingeniería

Machine Learning para Sistemas Inteligentes

Fake Covid-19 News

Obligatorio I

Clavijo Vitale, Tomás Andrés (235426)

Meljem Cabrera, Sebastián (251557)

Docentes: Sergio Yovine, Matías Carrasco

2022

Tecnología utilizada

Con el objetivo de elaborar y probar el modelo realizado que buscaba predecir si dada una noticia (tweet) acerca del Covid-19, esta era verdadera (“real”) o falsa (“fake”), se utilizaron tres grandes tecnologías que se describen a continuación:

Python

Es un lenguaje de alto nivel de programación que surgió a comienzos de los años 90s. Soporta parcialmente la orientación a objetos, programación imperativa, y en menor medida programación funcional. Se lo define como un lenguaje interpretado, dinámico y multiplataforma; destacando entre los más populares a nivel mundial. Aplicado a este ámbito, es muy útil para crear modelos de Machine Learning a través de la librería sklearn, la cual cuenta con una amplia variedad de algoritmos a utilizar.

Kaggle

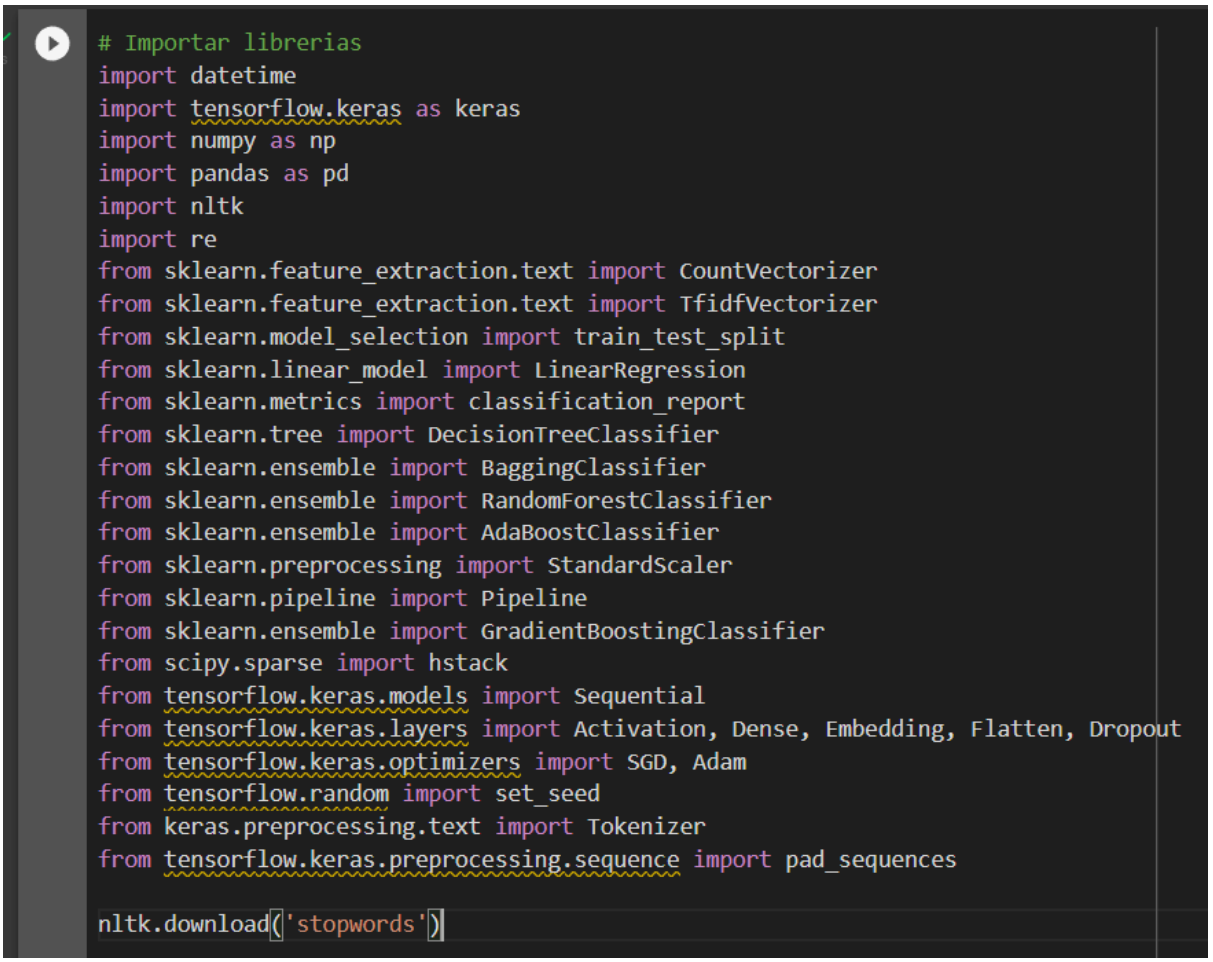
Es una comunidad en línea de científicos de datos y profesionales del aprendizaje automático. Permite a los usuarios encontrar y publicar conjuntos de datos, explorar y crear modelos en torno a la ciencia de datos, trabajando con otros usuarios y participando en concursos. En este caso, utilizamos dicho sitio para probar nuestro modelo en Test, midiendo a través del score y la competencia planteada en una tabla de posiciones, qué tan bien predecía los tweets.

Colab

Colaboraty, o “Colab” denominado comúnmente, es un producto de Google Research. Permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación. Desde un punto de vista más técnico, Colab es un servicio de cuaderno alojado de Jupyter que no requiere configuración y que ofrece acceso a recursos informáticos como GPUs.

Importaciones

Previo a comenzar a desarrollar nuestro modelo, lo primero a realizar fue importar librerías/bibliotecas/funcionalidades que serían de utilidad en el transcurso de todo el trabajo. En base al progreso obtenido, se iban agregando nuevas librerías que serían de utilidad para algunas funciones específicas. Las importaciones realizadas fueron las siguientes:



```
# Importar librerias
import datetime
import tensorflow.keras as keras
import numpy as np
import pandas as pd
import nltk
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier
from scipy.sparse import hstack
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Embedding, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.random import set_seed
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

nltk.download('stopwords')
```

En esta primera imagen obtenida del notebook, se puede observar que muchas importaciones fueron realizadas a partir de **sklearn**. Sin embargo, no fue la única biblioteca utilizada, por lo que procederemos a describir a grandes rasgos cada una de ellas.

Sklearn

Es una biblioteca para aprendizaje automático de software libre para el lenguaje de Python. Incluye algoritmos de clasificación, regresión y análisis de grupos.

Numpy

Es una biblioteca de Python que da soporte para crear vectores y matrices multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel que nos permiten operar con ellas.

Pandas

Es una librería de Python especializada en la manipulación y el análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales; como si de Excel se tratase.

Nltk

Es un conjunto de bibliotecas y programas que permiten el procesamiento del lenguaje natural simbólico y estadísticos para Python. Incluye demostraciones gráficas y datos de muestra.

Re

Comúnmente conocida como “expresión regular” o “regex”, es una secuencia de caracteres que conforman un patrón de búsqueda. Se utilizan principalmente para la búsqueda de patrones de cadena de caracteres u operaciones de sustituciones.

SciPy

Es una biblioteca libre y de código abierto que posee herramientas y algoritmos matemáticos. Contiene especialmente módulos para optimización, álgebra lineal, interpolación, integración, procesamiento de señales e imágenes, entre otras tareas.

```
import tensorflow.keras as keras
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.random import set_seed
import datetime
```

TensorFlow

Es una biblioteca dedicada al aprendizaje automático a través de un rango de tareas, desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar **redes neuronales** para detectar y descifrar patrones y correlaciones.

Archivos utilizados

Una vez realizadas las importaciones que nos serían de utilidad a futuro, continuamos con los archivos. Contábamos con dos archivos: obligatorio.csv, el cual contaba con tres columnas: id, tweet, label. La segunda columna mencionada tenía el mensaje y la tercer columna la respuesta a si era fake o real. Y el otro archivo utilizado fue test_without_label.csv, el cual solo contaba con id y tweet, siendo nosotros quienes debíamos predecir cada una de las noticias.

Al momento de trabajar con ellos, teníamos dos opciones:

1) Importarlos desde Google Drive
2) Subirlos en cada ejecución

Por razones de simplicidad en el momento, decidimos que sería mejor agregarlos en cada ejecución. Por lo que al primer archivo lo definimos como “corpus”, y al segundo como “test”.

```
[ ]  corpus = pd.read_csv('/content/obligatorio.csv', index_col = 0)

[ ]  test = pd.read_csv('/content/test_without_label.csv', index_col = 0)
```

Después de subir ambos archivos, utilizamos funciones tales como: **corpus.head(), test.head()**, con el objetivo de verificar que habían quedado bien.

Previo a continuar trabajando, habíamos decidido evaluar qué tan bien funcionaba nuestro modelo en base a la **“accuracy”,** pero debíamos verificar que los datos brindados tuvieran aproximadamente la misma cantidad de noticias falsas que verdaderas. Por lo que utilizamos la función **value_counts()**, con el objetivo de conocer más al respecto. Obteniendo lo siguiente:

```
corpus['label'].value_counts()

real    4480
fake    4080
Name: label, dtype: int64
```

Estos datos nos permitieron establecer que podríamos tomar la accuracy como referencia sin inconveniente alguno.

Bag of Words

Con el objetivo de realizar la bag of words con todo el contenido brindado por el archivo, sin utilizar técnicas como stop word removal o reduce to a root, entre otras, decidimos aplicar la solución a un problema NLP básico, sin mayor detalle.

Comenzamos particionando los datos como parte del procesamiento. Ignoramos en ambos archivos la columna “id”, ya que solo cumplía la función de identificar al elemento, no aportando información significativa.

```
[ ]  X = corpus['tweet']
     Y = corpus['label']

[ ]  X_test = test["tweet"]

[ ]  X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size = 0.2, random_state = 1)
```

A su vez, realizamos la separación correspondiente a train y validation; con el objetivo de dedicar parte de nuestros datos a entrenar al modelo y otros a validarlo.
A continuación, creamos el vectorizer, utilizando **CountVectorizer()** sin parámetro alguno, y aplicándole posteriormente **.fit()**, el cual se entera de todas las palabras y genera un set. Decidimos realizar el fit y el transform por separado, ya que si bien se podría realizar todo en la misma línea, no es lo recomendado.
Por último, realizamos el tablero, obteniendo en las columnas todas las palabras, en las filas los tweets, y en cada celda las ocurrencias.

```
vectorizer = CountVectorizer()

vectorizer.fit(X_train)
X_train = vectorizer.transform(X_train)
X_val = vectorizer.transform(X_val)

X_test = vectorizer.transform(X_test)

tablero = pd.DataFrame(X_train.toarray(), columns=vectorizer.get_feature_names_out())
tablero
```

Classifiers

Como se mencionó anteriormente, previo a continuar utilizando diversas estrategias al momento de aplicar CountVectorizer() que tuvieran como objetivo aumentar la exactitud del modelo, realizamos los distintos Classifiers sobre el vectorizer definido anteriormente, buscando conocer cuál presenta mayor exactitud.

Classifier	Accuracy Validation	Accuracy Train
Decision Tree	0,9007	1,0000
Bagging	0,9095	0,9936
Random Forest	0,9235	1,000
Ada	0,9112	0,9134
Gradient	0,8995	0,9071

Se presenta un mejor valor de la exactitud en Random Forest en cuanto a Validation, pero también tiene como defecto que está sobreajustado a Train. El menor sobreajuste lo presenta Gradient.
Se espera que Random Forest, debido a la cantidad de datos, obtendrá más árboles que le permitirá combatir el sobreajuste y brindar una buena predicción.

Deep Learning

Es un subconjunto de machine learning donde las redes neuronales, algoritmos inspirados en cómo funciona el cerebro humano, aprenden de grandes cantidades de datos.

Definición del modelo y decisiones tomadas:

```
maxlen = 900

tokenizer = Tokenizer()

tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_train_padded_seq = pad_sequences(X_train_sequences, truncating = 'post', padding ='post', maxlen = maxlen)

y_train_dl = np.where(y_train == "real", 1, 0)
```

```
[342] maxlen = 900

      tokenizer = Tokenizer()

      tokenizer.fit_on_texts(X_val)

      X_val_sequences = tokenizer.texts_to_sequences(X_val)
      X_val_padded_seq = pad_sequences(X_val_sequences, truncating = 'post', padding ='post', maxlen = maxlen)

      y_val_dl = np.where(y_val == "real", 1, 0)
```

Se hace uso del Tokenizer de keras, para convertir los documentos a secuencias, las cuales después son introducidas en la primera capa, la cual es un Embedding, la cual convertirá cada palabra en un vector denso, de esta manera las palabras se encuentran mejor representadas

```
[343] model = Sequential()
      model.add(Embedding(5000, 8, input_length=maxlen))
      model.add(Flatten())
      model.add(Dense(4))
      model.add(Dropout(0.2))
      model.add(Dense(2))
      model.add(Dropout(0.3))
      model.add(Activation("relu"))
      model.add(Dense(1))
      model.add(Activation("sigmoid"))

      model.summary()
```

Debido a que se trata de un problema de clasificación binaria, en la última capa utilizamos la función de activación “sigmoid”. Se observa que comenzamos definiendo la capa de Embedding previamente mencionada, luego una de Flatten para poder pasar el output tridimensional de Embedding a la siguiente densa, aplicamos Dropout para reducir sobreajuste y función de activación “relu”.

```
[344] criterion = SGD(learning_rate = 0.01) #Empleamos learning rate
      model.compile(optimizer=criterion, loss='binary_crossentropy', metrics=['accuracy']) #Es un problema de clasificación, se utiliza la entropía

callbacks = [keras.callbacks.TensorBoard(),
             keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True)]

model.fit(X_train_padded_seq, y_train_dl, validation_data=(X_val_padded_seq, y_val_dl), batch_size=20, epochs=100, callbacks=callbacks)

y_pred_model_val = model.predict(X_val_padded_seq)
y_pred_model_train = model.predict(X_train_padded_seq)
```

Como se mencionó anteriormente, se trata de un problema de clasificación binaria, por lo que se tomó como métrica la exactitud y como función de loss la binary cross entropy. Por otra parte, también optamos por utilizar learning rate.

```
[321] y_pred_model_val = np.where(y_pred_model_val >= 0.5, 1, 0)
      y_pred_model_train = np.where(y_pred_model_train >= 0.5, 1, 0)
```

Se prosigue convirtiendo el resultado de la sigmoidea a 0 o 1, dependiendo de si es menor o mayor-igual a 0.5

```
report = classification_report(y_val_dl, y_pred_model_val, digits=4)
print(report)
```

	precision	recall	f1-score	support
0	0.8074	0.7606	0.7833	827
1	0.7878	0.8305	0.8086	885
accuracy			0.7967	1712
macro avg	0.7976	0.7955	0.7959	1712
weighted avg	0.7973	0.7967	0.7964	1712

La exactitud siempre se mantuvo en el rango de 0,80 sin importar las alteraciones realizadas a la red neuronal, por lo tanto se optó elegir el clasificador Random Forest para seguir adelante.

Reduce to a Root

Llevamos a cabo las dos formas Stemmer y Lemmatizer con el objetivo de reducir la cantidad de columnas y evaluar el funcionamiento de la predicción posterior.

```
# Stemmer

from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer(language = 'english')
analyzer = TfidfVectorizer().build_analyzer()

def stemmed_words(doc):
  return (stemmer.stem(w) for w in analyzer(doc))

vectorizer = TfidfVectorizer(analyzer = stemmed_words, stop_words = nltk.corpus.stopwords.words('english'), min_df = 0.0025)
```

```
# Lemmatizer

from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()

def lemmatized_words(doc):
    return (lemmatizer.lemmatize(w.lower()) for w in doc.split())

vectorizer = TfidfVectorizer(analyzer = lemmatized_words, stop_words = nltk.corpus.stopwords.words('english'), min_df = 0.0025)
```

En este caso particular, concluímos que no sería necesario utilizarlo, ya que no se observaban variaciones en la exactitud de las predicciones, y en diversas ocasiones se mostraba un poco por debajo de lo ya obtenido anteriormente.

Min_df, Max_df

Observamos que el max_df puesto en 0,9 o 0,7 en este caso es lo mismo; y que el min_df genera un impacto muy grande en la cantidad de columnas al utilizarlo en 0,1. Es por esto, que descartamos el uso de max_df ya que recién se veía su uso si se usaba un valor muy pequeño, y al mismo tiempo aplicamos min_df pero dentro de valores más pequeños como puede ser: min_df = 0,0025. Se obtuvieron algunas ventajas en cuanto a la exactitud. Sin embargo se consideró que al reducir los datos aumentaba el sobreajuste por lo que se descartó el uso de ambos al final.

Funciones

Se implementaron diversas funciones con el objetivo de agregar nuevas columnas y así poder evaluar si la exactitud aumentaba o disminuía utilizándolas. Buscábamos conocer si las características programadas estaban relacionadas con el hecho de que una noticia sea verdadera o falsa.

Realizadas:

- Caracteres
- Palabras
- Hashtags
- Arrobas
- Símbolos de porcentaje
- Links
- Letras

Se aplicaron distintas combinaciones de las mismas utilizando random forest con el objetivo de subir la exactitud de la predicción.

Predicciones Realizadas

Luego de analizar los distintos classifiers que fueron evaluados, se decidió usar Random Forest Classifier, ya que era el que mejor accuracy conseguía en general. Fueron llevados a cabo múltiples predicciones con distintas técnicas usando dicho clasificador. En la siguiente tabla se detallan las predicciones realizadas.

Técnicas aplicadas	Accuracy en Validation
CountVectorizer	0.9235
TfidfVectorizer, stop word removal ("english"), max_df = 0.95	0.9381
TfidfVectorizer, stop word removal ("english"), max_df = 0.95, count = (words, characters)	0.9328
TfidfVectorizer, stop word removal ("english"), max_df = 0.9, count = ("#")	0.9387
TfidfVectorizer, stop word removal ("english"), count = ("#", "@")	0.9381
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@")	0.9404
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@")	0.9398
TfidfVectorizer, stop word removal ("english"), min_df = 0.0021, count = ("#", "@")	0.9416
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = (characters, "#", "@")	0.9451
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@", links)	0.9416
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@", "%", links, letters)	0.9404
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@", "%", links, letters)	0.9422
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@", links, letters)	0.9439
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = ("#", "@", links, letters)	0.9422
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, max_df = 0.1, count = (characters, "#", "@", "%", links)	0.9422
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = (characters, "#", "@", "%", links)	0.9404
TfidfVectorizer, stop word removal ("english"), min_df = 0.0025, count = (characters, "#", "@", "%", links)	0.9410
TfidfVectorizer, stop word removal ("english"), count = (characters, "#", "@", "%", links)	0.9346

Se concluyó que el último clasificador realizado con una accuracy de 0.9346 era el mejor, ya que no sobreajusta tanto como el resto. Al ser subido a kaggle logró una accuracy de 0.9501