UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2020 - 1er Cuatrimestre

ORGANIZACIÓN DE DATOS (75.06)

TRABAJO PRÁCTICO Nº 2
TEMA: Machine Learning
GRUPO: 23 – Barbijo De Uranio

ALUMNO	PADRON	MAIL
CASTILLO, JULIO EDGARDO	80039	jecastillo@fi.uba.ar
LA PENNA, MARIANO	98432	mlapenna79@hotmail.com
RIBORATI, FRANCO	99411	francoriboratig@gmail.com

Repositorio en GitHub: https://github.com/francoriboratig/TP2-Datos

INTRODUCCIÓN

El trabajo práctico consta de una competencia de Machine Learning donde intentamos determinar, para cada tweet brindado, si el mismo se basa en un hecho real o no.
Para ello tenemos un set con 7613 tweets de ejemplo que ya indica si el hecho es real o no mediante un atributo binario para cada tweet.
Luego tenemos otro set de 3263 tweets donde no está especificado si el cada uno es real o no, es este conjunto el que intentamos predecir.

Trabajamos en lenguaje Python con diversos algoritmos y técnicas de machine learning, con la ayuda de una variedad de librerías.

INICIO DE LA TAREA

Comenzamos con varios notebooks de Python, programando por primera vez redes neuronales, árboles de decisión y haciendo feature engineering con los atributos descubiertos en el primer trabajo práctico. Cada modelo probado quedó implementado en un notebook diferente, para facilitar la división del trabajo y organizar mejor modificaciones y optimizaciones.

FEATURE ENGINEERING

Agregamos los siguientes atributos

- character_count
- word_count
- contains_word_fire
- contains_word_storm
- contains_word_flood
- contains_word_death
- contains_word_love
- has_mentions
- amount_of_exclamation_marks
- amount_of_interrogation_marks
- has_link
- stopwords_count
- contains_countries
- contains_cities

Esto lo realizamos primero leyendo los CSV originales y grabándolos de vuelta con los nuevos atributos, para que se puedan usar en cualquier notebook.

MODELOS

KNN

Lo primero que se probó fue algo bien fácil, **KNN** sólo con los features descritos, obteniendo un puntaje pobre (0.66) pero comprobando que algo de validez tiene el criterio para un problema de NLP. Esto sirvió como base para aprender a desarrollar modelos para resolver problemas de esta índole.

Perceptrón Multicapa

Con el objetivo de probar una variedad de cosas se usó también **Perceptron Multicapa**, el del nombre retro, con la ayuda del módulo Sequential de la librería Keras, sin feature engineering y sólo empleando un escalador de datos. Obtuvo un resultado de 68% que es más triste que vergonzoso, así que no lo optimizamos porque ya sabíamos que por ese camino no venía el éxito.

Deep learning

Luego creamos una **red neuronal** usando un embedding de TensorFlow Hub, o sea que no configuramos filtros convolucionales nosotros sino que una herramienta externa lo hace y nos devuelve los vectores. Estos son un vector de 20 dimensiones por cada tweet (no por cada palabra). Luego se emplea una red multicapa, mediante el módulo Sequential. Con esto ya llegamos a oscilar entre 0.76 y 0,79, fue un claro avance. Mediante la función train_test_split se divide el set en una parte que es usada para entrenamiento, y otra para validación de los resultados, obteniendo una aproximación para el score. Con respecto a los hiperparámetros, en principio se realizaron pruebas manuales con distintos valores.

Sospechando que la gloria exigía unir tanto una red que tomara text embedding más los features, hicimos primero una red neuronal con TensorFlow Keras que lee sólo los parámetros, obteniendo también un valor menor a 0.7, luego creamos otro notebook que lee tanto el embedding del texto de **TensorFlor Hub más los features** y tampoco pudimos llegar al 0.8. Pero nada nos detuvo porque somos así.

Naive-Bayes

Mediante el módulo MultinomialNB se probó la variante con el clasificador basado en este algoritmo, otra vez con resultados cercanos a 0.79. Resultado similar se logró al reemplazarlo por TfIdfVectorizer, basado en Tf-Idf.

Regresión logística

Se agregó la variante de **regresión logística** a la paleta de algoritmos probados, el cual trabaja con un clasificador binario a través de la librería LogisticRegression de Sklearn, consiguiendo resultados similares a los logrados con CountVectorizer.

TF-IDF

Luego se probó reemplazar el CountVectorizer por TfidfVectorizer, basado en Tf-Idf, aunque los resultados no fueron mejores.

Hyperparameter tuning

Aquí es cuando una mente brillante del equipo pensó en **optimizar los hiperparámetros** ya no de una forma manual sino con un procedimiento automatizado, con ayuda de la librería Talos, obteniendo finalmente el 0.8 soñado.

Los hiperparámetros buscados fueron los siguientes:

- Cantidad de neuronas
- Función de activación
- Cantidad de rondas
- Tamaño de cada bloque

Luego de obtener los hiperparámetros se vectorizaron los textos con CountVectorizer, sin procesar ni concatenar los features, para aplicarlo al modelo que también incluye un par de capas de neuronas. El proceso de búsqueda llevó un par de horas. Cuando se probó aplicar la optimización en los diferentes modelos aplicando rangos de valores amplios fueron surgiendo interrupciones en la ejecución, teniendo en cuenta que el proceso puede tardar varias horas o incluso días.

Árboles de decisión

Quedaba claro por dónde había que seguir avanzando, probablemente puliendo el texto y mejorando los features.
Sin embargo, los árboles de decisión clamaban por su presencia, y nuestra piedad nos llevó a incorporar al set de notebooks **RandomForest, XGBoost, LightGBM y CatBoost**. Cegados en el afan de la red neuronal, no les pusimos muchas fichas originalmente, hasta que luego de una segunda visita a los métodos, estos empezaron a amenazar los score de las redes neuronales alcanzando 0.78.
RandomForest y XGBoost pudieron competir con valores entre 0.77 y 0.78 pero no hemos sido capaces de aplicar un LightGBM competente.
CatBoost sin ningun tipo de tuneo de hiperparámetros alcanza naturalmente un 0.79 de precisión (*accuracy score*)

Para tunear hiperprámetros de estos modelos. Se utilizó una auto proclamada función de random search universal que no utiliza ninguna librería. Esta funciona de

manera muy apropiada y, si bien es simple, nos da un poco de felicidad la independencia que nos brinda.

Esta función llego tarde al trabajo, por lo que solo se pudieron tunear los hiperparámetros de RandomForest y XGBoost, con los siguientes espacios de hiperparámetros:

```
random_forest_hyperparam_space = {
    'n_estimators' : [10,100,1000,5000,500,20,50],
    'criterion' : ['gini','entropy'],
    'max_depth' : [None,10,100]
}


xgb_hyperparam_space = {
    'eta' : [0.1,0.2,0.3,0.4,0.5],
    'max_depth' : [1,2,3,4,5,6,10],
    'objective' :
['binary:logistic','reg:squarederror','multi:softmax','multi:softprob']
,
    'num_class' : [2,3,5,7,9,10,15],
    'gamma' : [0,0.5,0.7,1,1.5,2,3],
    'min_child_weight' : [0.5,1,1.5,2,2.5,3],
    'subsample' : [0.5,0.7,0.9,1],
    'colsample_bytree' : [0.5,0.7,0.9,1]
}
```

Faltante es el tuneo de hiperparámetros de catboost. Pero para el 13/8, dia que termina la competencia, le tenemos confianza, basados en su excelente primer desempeño.

Red convolucional

Por último se implementó el modelo de red convolucional, con ayuda de la librería conv1d, al set de datos se le aplicó un tokenizador. Los resultados obtenidos también cayeron en torno al 0.79.

Todo en uno

En un mismo modelo nos aventuramos a juntar text embedding, features y optimización automatizada de parámetros. “Red neuronal con text embeddings, features e hyperparameter automatic tunning.ipynb”

SHOW MUST GO ON

Sufrimos el abandono de un integrante del grupo los últimos días de julio, a quien veníamos esperando para el inicio de su participación. Esto nos dejó un poco cerca de la fecha de entrega con bastante por realizar. Tomamos un par de bebidas energizantes y avanzamos a destino con el nuevo cuarteto reducido.

KAGGLE

El notebook que produjo el mejor resultado en Kaggle es “Keras_hyperparameter_tuning.ipynb” con el que obtuvimos 0.80294 pero que luego no pudimos superar, al menos con nuestra estrategia de explorar varios modelos.

CONCLUSIONES

Este trabajo práctico, a diferencia del primero, nos pareció que no requería tanta intuición, olfato “artístico” o imaginación para aplicar los conocimientos técnicos sino que se imponía más la constancia, las horas silla y el conocimiento de los algoritmos. Se basa mas bien en pruebas, investigación y conclusiones que definieron los pasos siguientes.

Es interesante notar que cuando uno llega al 0,79, cada centésima siguiente se vuelve exponencialmente más difícil de lograr que la anterior. El modelo más efectivo resultó ser el más simple en cuanto a cantidad de capas y número de neuronas empleadas, cuya red está constituida con una sóla capa con 2 neuronas, además de la de salida, la cual recibe el texto procesado por el Countvectorizer. Por otra parte, dicho modelo no emplea los features extraídos anteriormente.

Link al notebook con el modelo que obtuvo el mejor puntaje:

https://github.com/francoriboratig/TP2-Datos/blob/master/Keras_hyperparameter_tuning.ipynb