mgmtfm Documentation

Versión 1.0.0

Manuel Gómez

15 de diciembre de 2019

Índice general

CAPÍTULO 1

Índice

- genindex

CAPÍTULO 2

Clean

class mgmtfm.clean.**Clean**

Clase para el preprocesamiento de documentos que se destinaran al PLN.

balancing(*x=None*, *class_binary=None*, *balancing_types=None*)

Balancear los datos para que todas las clases que se usen en el entrenamiento tengan el mismo número de elementos. Y realizar un shuffle entre ellos.

Args: x (DataFrame): Dataframe que se va a proceder a balancear. Se balanceará en función del campo «tipo». Si fuera None usará los tokens actuales.

class_binary (str): Establecer si estamos en un caso de clasificación binaria y queremos balancear entre esa clase y el resto.

balancing_types (list<str>): Realiza lo mismo que en el caso anterior, pero con una lista de clases realizando el balanceado entre esta lista y el resto.

Returns: DataFrame: Dataframe balanceado.

get_combined_and_distribution(*calls*, *types*, *ivr=False*)

Combinar el dataframe de llamadas con el dataframe de tipos correspondiente.

Args: verint (dataframe): Dataframe de Verint con las transcripciones de las llamadas.

tipos (dataframe): Dataframe con los tipos.

ivr (boolean): Si es True el dataframe de tipos será el de IVR, de no ser así será el de monitorizaciones.

Returns: dataframe: Join de ambos dataframes.

get_set_data(*test_size=0.33*, *seed=2019*, *class_binary=None*, *sequence=True*, *shuffle=True*, *max_seq_len=None*, *balancing_types=None*)

Obtener conjuntos de datos de test y de entrenamiento, realizar el etiquetado de los conjuntos en el caso que sea binario o haya balanceo. Realizar el padding si tratamos con secuencias.

Args: test_size (float): Porcentaje de datos de entrenamiento. Puede ser 0 para obtener los datos en un solo conjunto y en el mismo orden (sin *shuffle*).

seed (int): Semilla para reproducir los resultados aleatorios.

class_binary (str): Válido para el etiquetado. Asigna un 1 a la clase binaria y un 0 al resto.

sequence (bool): Si se esta trabajando con secuencias o no. Se realiza padding en el caso de que se active. *True* por defecto.

shuffle (bool): Coger los datos en orden aleatorio(True) o secuencial (False). No afecta con test_size=0.

max_seq_len (int): En el caso de usar secuencias la longitud máxima para el *padding*.

balancing_types (list): Tipos que se van a usar para clasificar. El resto de tipos se etiquetaran como «Resto».

Returns: (array, array, array, array): Conjunto de entrenamiento, etiquetas de entrenamiento, conjunto de test y etiquetas de test.

load_tokens(*file_name*)
Cargar los tokens desde un fichero. Además carga las variables «combined», «distribucion_tipos» y «word_index».

one_hot_ys()
Realizar la conversión de las etiquetas de entrenamiento y test al formato one_hot.

Returns: (array, array): Tupla de arrays en formato one_hot.

pad_Xs(*maxlen*)
Realizar el padding de los conjuntos de entrenamiento y test.

Args: maxlen (int): Longitud máxima para el padding.

Returns: tuple (<Dataframe, Dataframe>): conjunto de entrenamiento y de test tras el padding.

quit_class(*delete*)
Eliminar una clase de los tokens actuales.

Args: delete (str): Clase que se eliminará.

save_tokens(*file_name*)
Guardar los tokens en un fichero (pickle).

Args: file_name (str): nombre del fichero en el que se almacenaran los tokens.

tokenize(*quit_commons=True*, *limit=None*)
Tokenizar el «plaintext» del *dataframe* «combined» utilizando el método privado *_tokenize*.

Args: quit_commons (bool): Eliminar palabras comunes al tokenizar.

limit (int): Si se establece tokeniza unicamente este número de llamadas.

Returns: dataframe: Dataframe con los tokens. Tambien se almacenan en self.tokens.

CAPÍTULO 3

Embedding

class mgmtfm.embedding.**Embedding**(*train_data*, *word_index*, *num_words=20000*, *train_from_verint=True*)

Clase encargada de realizar y gestionar los embedding. Actualmente existen las opciones de Word2Vec (CBOW y Skip-Gram) o Doc2VEC.

doc2vec_infer(*doc*)

Inferir el vector Doc2Vec de un documento.

Args: doc (str): Documento de texto a inferir.

Returns: Array: Vector Doc2Vec.

get_embedding_matrix()

Obtener la matriz de embeddings.

Returns: array<array>: Matriz de embeddings.

load_embedding(*path*, *type=1*)

Cargar embedding desde un fichero.

Args: path (str): Ruta de la que cargar el embedding. type (int): 0-> CBOW 1-> SKIP-GRAM (default) 2-> Doc2Vec

save_embedding(*path*)

Almacenar el embedding (tras el entrenamiento).

Args: path (str): Ruta del fichero.

train_embedding(*min_count=1*, *size=100*, *workers=16*, *window=5*, *type=1*)

Entrenar el embedding para word2vec o doc2vec.

Args: min_count (int): Número mínimo de veces que debe aparecer una palabra para ser tenida en cuenta. size (int): Dimension del vector de embedding. workers (int): Número de workers usados en el entrenamiento. window (int): Tamaño de la ventana de palabras para el entrenamiento. type(int): 0-> CBOW 1-> SKIP-GRAM (default) 2-> Doc2Vec

CAPÍTULO 4

Models

class mgmtfm.models.**Models**(*nclasses=1*, *sequence_length=None*, *embedding_dim=None*, *vocabulary_size=None*, *embedding_matrix=None*, *load=False*, *path=None*)

Clase para implementar diferentes modelos orientados a PLN.

compile_and_train(*X_train, y_train, batch_size=50, epochs=10, verbose=1, lr=0.001, decay=1e-06, validation_data=None, metrics=['acc'], callbacks=None, loss='categorical_crossentropy'*)

Compilar y entrenar el modelo definido.

Args: X_train (array): Conjunto de entrenamiento.

y_train (array): Etiquetas de entrenamiento.

batch_size (int): Tamaño de batch para el entrenamiento.

epochs (int): Número de epochs para el entrenamiento.

verbose (int): Tipo de verbose usado en el entrenamiento.

lr (float): Tasa de aprendizaje del entrenamiento.

decay (float): Pesos decayentes para las redes neuronales.

get_best_model(*name*, *params*)

Devolver un modelo a partir del nombre y los parámetros.

Args: name (str): Nombre del modelo.

params (dict): Parámetros del modelo.

get_confusion_matrix(*X_test*, *y_test*, *binary=False*)

Obtener la matriz de confusión.

Args: X_test (array): Conjunto de test.

y_test (array): Etiquetas de test.

binary (bool): Si se trata de una clasificación binaria.

load_model (*path*)
Cargar un modelo.

Args: path (str): Ruta desde donde cargar el modelo.

load_weights (*path*)
Cargar pesos de un modelo.

Args: path (str): Ruta desde donde cargar los pesos del modelo.

model_cnn_1 (*filter_sizes=[3, 4, 5], drop=0.3, num_filters=100, regl2=0.01*)
Crear un modelo de redes neuronales convolucional con diferentes capas.

Args: filter_sizes (list): Lista con los tamaños de los kernel de las capas convolucionales. La longitud de la lista determinará las capas convolucionales de la red.

drop (float): Dropout.

num_filters (int): Número de filtros a usar en cada capa.

regl2 (float): Regularizador de pesos nivel 2.

model_cnn_1_bin (*filter_sizes=[3, 4, 5], drop=0.3, num_filters=100, regl2=0.01*)
Crear un modelo de redes neuronales convolucional con diferentes capas para clasificación binaria.

Args: filter_sizes (list): Lista con los tamaños de los kernel de las capas convolucionales. La longitud de la lista determinará las capas convolucionales de la red.

drop (float): Dropout.

num_filters (int): Número de filtros a usar en cada capa.

regl2 (float): Regularizador de pesos nivel 2.

model_dense_1 (*sizes=[500, 300, 200, 150, 100, 50, 20, 1], drop=0.2*)
Crear un modelo de redes neuronales con capas totalmente conectadas.

Args: sizes (list): Lista con los tamaños de las capas. La longitud de la lista determinará las capas de la red. La primera capa debe coincidir con el tamaño del input y la última con la salida (si es 1 será una clasificación binaria).

drop (float): Dropout.

model_rnn_1 (*memory_units=200*, *cell=1*, *drop=0.2*)
Crear un modelo de redes neuronales recurrentes.

Args: memory_units (int): Número de celdas.

cell (int): Tipo de celda. 0-> LSTM 1->GRU

drop (float): Dropout

model_rnn_2 (*memory_units=200*, *cell=1*, *drop=0.2*, *filters=32*, *kernel_size=3*, *pool_size=2*, *padding='same'*)
Crear un modelo combinando redes neuronales recurrentes con una capa convolucional.

Args: memory_units (int): Número de celdas.

cell (int): Tipo de celda. 0-> LSTM 1->GRU

drop (float): Dropout

filters (int): Número de filtros de la capa convolucional.

kernel_size (int): Tamaño de kernel de la capa convolucional.

pool_size (int): Tamaño del poolin.

padding (str): Tipo de padding a realizar.

predict (*X*)
Predecir usando un modelo.

Args: X (array): Conjunto de datos del que realizar la predicción.

Returns: array: Array de predicciones.

save_model (*path*)
Guardar un modelo.

Args: path (str): Ruta donde guardar el modelo.

summary_model ()
Presentar un resumen del modelo actual.

CAPÍTULO 5

Optimize

class mgmtfm.optimize.**Optimize**(*project='dummy'*, *project_db='mgm_optuna'*, *user_db='postgres'*, *pass_db='postgres'*, *host_db='10.148.172.147'*, *direction='maximize'*)

Clase creada para utilizar optkeras (basado en Optuna) para bucar los mejores hiperparámetros de un modelo.

delete_study(*study*)

Eliminar un estudio y sus trials de la base de datos.

Args: study (str): Nombre del estudio a eliminar.

get_best_acc()

Obtener la mejor precisión de un estudio.

get_best_params()

Obtener los mejores parámetros de un estudio.

get_best_trial()

Obtener el id del mejor trial de un estudio.

get_studies()

Obtener los estudios realizados.

Returns: dataframe: Dataframe con los datos de todos los estudios.

rename_study(*study*, *new_name*)

Renombrar estudio.

Args: study (str): Nombre original del estudio.

new_name (str): Nuevo nombre del estudio.

run_optuna(*function*, *timeout=None*)

Correr el proceso de optuna.

Args:

function (function): Función a maximizar o minimizar.

timeout(int): Número de segundos de ejecución.

CAPÍTULO 6

Plot Utils

Modulo con diferentes funciones de utilidad

mgmtfm.plot_utils.**plot_confusion_matrix**(*cm*, *target_names*, *title='Confusion matrix'*, *cmap=None*, *normalize=True*)

Realizar una visualización dada una matriz de confusión.

Args:

cm (matrix): Matriz de confusión de sklearn.metrics.confusion_matrix

target_names (list): Etiquetas de clasificación.

title (str): Texto para mostrar en la parte superior.

cmap (cmap): El gradiente para mostrar en la matriz. Ver http://matplotlib.org/examples/color/colormaps_reference.html

normalize (bool): Si utiliza o no medidas normalizadas.

Citations:

http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Índice de Módulos Python

m

Índice

T