Text Classification: usage of bag-of-words or embedding layer

Practicing DatScy · Follow
16 min read · Feb 21, 2024

Recently I ran into a 'difficult' text classification dataset on Kaggle, news-category-dataset by the Huffington Post [1], because I could not accurately classify the sentences using an Embedding layer. This text classification dataset contains 10 classes: 'politics', 'food & drink', 'travel', 'business', 'sports', 'style & beauty', 'world news', 'entertainment', 'parenting', 'wellness'.

Since the popularity of Transformers, using an Embedding layer had been shown to be more useful than bag-of-words (ie: word count), in terms of text classification because it allowed for text to be clustered in a feature-space based on similar lexical meaning. Sentences on different topics can be classified because each topic is likely to use specific unique words that are likely to be clustered in embedding space. Thus, if similar words are used to several classes it may be more difficult to distinguish between classes because the embedding space will share the same location. One way to distinguish each class separately in embedding space via words, would be to intelligently clean or pad specific class data without or with keywords for specific classes.

In this post, I classify sentences using two datasets: news-category-dataset mentioned above and a 'less challenging' text classification dataset (bbc-new-dataset), that was a Kaggle competition [2].

```
import numpy as np
import pandas as pd

import tensorflow as tf

from collections import Counter

from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

import regex

import csv
```

Subfunctions

```
def clean_dataset(df, list_of_columns_2_drop):

    # Remove columns that are not needed
    for i in list_of_columns_2_drop:
        df = df.drop(i, axis=1)

    # Rename X and Y
    if len(df.iloc[0,0]) > len(df.iloc[0,1]):
        df.columns = ['X', 'Y']
    else:
        df.columns = ['Y', 'X']
```

```
    # Make every column lowercase
    df = df.applymap(str.lower)

    print("class count:", df['Y'].value_counts())

    n_classes = len(df['Y'].value_counts())
    print('number of classes: ', n_classes)

    return df
```

```
def count_the_number_of_times_a_char_appears(text, char2find):
    c = 0
    for char in text:
        if char == char2find:
            c = c + 1
    # print('c: ', c)
    return c
```

```
def remove_text_from_start_end_marker_for_a_string(sentence):

    start_marker = ['(', '{', '[']
    end_marker = [')', '}', ']']

    clean_sen = sentence

    for ind in range(len(start_marker)):
        # print('start_marker[ind]: ', start_marker[ind])

        # Count the number of times the marker appears
        loops = count_the_number_of_times_a_char_appears(clean_sen, start_marker

        for x in range(loops):
            if start_marker[ind] in clean_sen and end_marker[ind] in clean_sen:
                start_ind = clean_sen.find(start_marker[ind])
                # print('start_ind: ', start_ind)

                end_ind = clean_sen.find(end_marker[ind])
                # print('end_ind: ', end_ind)

                if start_ind == 0:
                    clean_sen = clean_sen[end_ind+1::]
                else:
                    clean_sen = clean_sen[0:start_ind-1] + clean_sen[end_ind+1::
```

```
    # print('clean_sen: ', clean_sen)

    return clean_sen
```

```
def remove_characters_from_string(sen_str):

    # Remove undesireable long characters repeatively, matching characters in th
    # These words should be unique words, such that parts of [the string "word"]
    to_replace = ['</p>', '<a', 'id=', "href=", 'title=', 'class=', '</a>', '</s
                  '?']
    replace_with = ''

    word_array = sen_str.split()
    # print('word_array: ', word_array)

    word_array_new = []
    for wind, word in enumerate(word_array):
        # print('word: ', word)

        # I do the same thing as regex sub, I search across the word string and
        # If there is nothing to replace, out just stays the same.
        out = word # initialization

        for ind, to_replace_val in enumerate(to_replace):
            # print('to_replace_val: ', to_replace_val)
            out_b4 = out
            out = word.replace(to_replace_val, replace_with)

            # Take the shortest out to ensure previous changes are stored
            if len(out_b4) < len(out):
                out = out_b4
            # print('out: ', out)

        # Stores the last changed word
        word_array_new.append(out)

    sen_str_clean = ' '.join(word_array_new)

    return sen_str_clean
```

```
def clean_procedure_per_string(sen_str):
```

```
    # [Step 0] Make sentences lowercase
    sen_str = sen_str.lower()

    # [Step 1] Remove parentheses and text in between parentheses, so that phras
    sen_str = remove_text_from_start_end_marker_for_a_string(sen_str)

    # [Step 3] Remove long undesireable characters repeatively, matching charact
    sen_str = remove_characters_from_string(sen_str)

    return sen_str
```

```
def sequential_padder(X, Y):

    # Train class count
    from collections import Counter
    c = Counter(Y)

    # 1 = apple, 0 = tomatoe
    class_key = list(c.keys())
    print('class_key: ', class_key)

    class_value = list(c.values())
    print('class_value: ', class_value)

    max_class = np.argmax(class_value)
    print('max_class: ', max_class)

    samples_to_add_per_class = [class_value[max_class] - i for i in class_value]
    print('samples_to_add_per_class: ', samples_to_add_per_class)

    # -------------------------------------------

    # Need to account for when a class does not need padding: assign directly
    X_pad = X
    Y_pad = Y

    for ind, samples2pad in enumerate(samples_to_add_per_class):

            print('Number of values to pad:', samples2pad)

            # Identify class number
            class_num = class_key[ind]   # class_key:  [1, 0]
            print('Class number that needs padding:', class_num)

            # Need to find the index of Y_train_filtered/X_train_filtered that b
            class_num_index = [index for index, val in enumerate(Y) if val == cl
```

```
        print('class_num_index:', class_num_index)

        # Select X and Y for the specified class number index
        X_class_num_select = [X[i] for i in class_num_index]
        Y_class_num_select = [Y[i] for i in class_num_index]

        # Find the number of samples for the specified class number
        curSamples = len(X_class_num_select)
        print('Number of samples to repeat for this specific class:', curSam

        # If the number of class samples are greater than the amount to pad,
        if curSamples > samples2pad:
            # Loop over every element in a because it is 3d matrix, and add
            for i in range(samples2pad):
                X_pad.append(X_class_num_select[i])
                Y_pad.append(Y_class_num_select[i])
        else:
            # If the number of class samples are less than the amount to pad
            num_of_full_loops = int(samples2pad/curSamples)
            print('num_of_full_loops:', num_of_full_loops)

            for i in range(num_of_full_loops):
                for j in range(curSamples):
                    X_pad.append(X_class_num_select[j])
                    Y_pad.append(Y_class_num_select[j])

            remaining_vals = samples2pad - num_of_full_loops*curSamples
            print('remaining_vals:', remaining_vals)
            for i in range(remaining_vals):
                X_pad.append(X_class_num_select[i])
                Y_pad.append(Y_class_num_select[i])

# --------------------------------------------

print('Length of Y matrix after padding:', len(Y_pad))

# --------------------------------------------

# Confirm that classes are even after padding
c = Counter(Y_pad)

# 1 = apple, 0 = tomatoe
class_key = list(c.keys())
print('class_key: ', class_key)

class_value = list(c.values())
print('class_value: ', class_value)

max_class = np.argmax(c)
print('max_class: ', max_class)
```

```
    samples_to_add_per_class = [class_value[max_class] - i for i in class_value]
    print('samples_to_add_per_class: ', samples_to_add_per_class)

    return X_pad, Y_pad
```

```
def create_tokenizer0(sentences):

    vocabulary = list(set(' '.join(list(set(sentences))).split(' ')))
    NUM_WORDS = len(vocabulary)
    #NUM_WORDS = 2000

    # Instantiate the Tokenizer class, passing in the correct values for num_wor
    tokenizer = Tokenizer(num_words=NUM_WORDS, oov_token="<OOV>")

    # Fit the tokenizer to the training sentences
    tokenizer.fit_on_texts(sentences)

    return tokenizer
```

```
def encode_labels(labels):

    unq_labels = list(set(labels))
    NUM_OF_CLASSES = len(unq_labels)

    # Assign a number to each unique label
    y_assignment = dict(zip(unq_labels, np.arange(NUM_OF_CLASSES)))
    print('y_assignment ', y_assignment)

    label_sequences = [y_assignment[i] for i in labels]

    return label_sequences, y_assignment
```

Load data

```
# Dataset 0: Original data in Coursera Natural Language Processing Tensorflow (D
# https://www.kaggle.com/competitions/learn-ai-bbc
df = pd.read_csv('/kaggle/input/bbc-new-dataset/BBC News Train.csv')

# Clean the columns of the Dataframe
list_of_columns_2_drop = ['ArticleId']
df = clean_dataset(df, list_of_columns_2_drop)
df# Dataset 1: Another news dataset (Huffington Post)
df = pd.read_csv('/kaggle/input/news-category-dataset/NewsCategorizer.csv')

# Clean the columns of the Dataframe
list_of_columns_2_drop = ['headline', 'links', 'keywords']
df = clean_dataset(df, list_of_columns_2_drop)
d
```

```
class count: Y
sport            346
business         336
politics         274
entertainment    273
tech             261
Name: count, dtype: int64
number of classes:  5
/tmp/ipykernel_33/871282564.py:14: FutureWarning: DataFrame.applymap has been deprecated. Use D
ataFrame.map instead.
  df = df.applymap(str.lower)
```

	X	Y
0	worldcom ex-boss launches defence lawyers defe...	business
1	german business confidence slides german busin...	business

1485	double eviction from big brother model caprice...	entertainment
1486	dj double act revamp chart show dj duo jk and ...	entertainment
1487	weak dollar hits reuters revenues at media gro...	business
1488	apple ipod family expands market apple has exp...	tech
1489	santy worm makes unwelcome visit thousands of ...	tech

1490 rows × 2 columns

```
# Dataset 1: Another news dataset (Huffington Post)
df = pd.read_csv('/kaggle/input/news-category-dataset/NewsCategorizer.csv')

# Clean the columns of the Dataframe
list_of_columns_2_drop = ['headline', 'links', 'keywords']
df = clean_dataset(df, list_of_columns_2_drop)
df
```

```
class count: Y
wellness          5000
politics          5000
entertainment     5000
travel            5000
style & beauty    5000
parenting         5000
food & drink      5000
world news        5000
business          5000
sports            5000
Name: count, dtype: int64
number of classes:  10
/tmp/ipykernel_33/871282564.py:14: FutureWarning: DataFrame.applymap has been deprecated. Use D
ataFrame.map instead.
  df = df.applymap(str.lower)
```

	Y	X
0	wellness	resting is part of training. i've confirmed wh...
1	wellness	think of talking to yourself as a tool to coac...
2	wellness	the clock is ticking for the united states to ...
3	wellness	if you want to be busy, keep trying to be perf...
4	wellness	first, the bad news: soda bread, corned beef a...
...	...	...
49995	sports	many fans were pissed after seeing the minor l...
49996	sports	never change, young man. never change.
49997	sports	wallace was hit with a first technical for a h...
49998	sports	they believe cbd could be an alternative to po...
49999	sports	the gymnast is in a league of her own.

50000 rows × 2 columns

Step 0 : Rigorously clean the data and determine correctly associated label

```
X = []
Y = []

for i in range(len(df)):

    sen_str = df["X"].iloc[i] # per row of the DataFrame is a string
    y_str = df["Y"].iloc[i]

    # ----------------------------
    # Clean/pre-process the sentence
    # ----------------------------
    # [0] Perform two types of a string cleaning procedure: handmade and regex
    sen_str = clean_procedure_per_string(sen_str)

    # ----------------------------

    # [Step 0] Make sentences lowercase
    # Performed by FASTER sen_str = clean_procedure_per_string(sen_str)

    # [Step 1] Remove parentheses and text in between parentheses, so that phras
    # Performed by FASTER sen_str = clean_procedure_per_string(sen_str)

    # [Step 2] Remove a single undesireable character
    patterns_to_remove = r'[–"\.\€\$\£\%\d,\[\]\(\)\{\}\!-><\n]'
    sen_str = regex.sub(patterns_to_remove, "", sen_str)

    # [Step 3] Remove long undesireable characters repeatively, matching charact
    # Performed by FASTER sen_str = clean_procedure_per_string(sen_str)

    # [Step 4] Remove exact stopwords that are separated by spaces or [space and
    stopwords = ["a", "about", "above","after", "again", "against", "and", "anyt
                  'always', 'again', 'also',
                  "because", "become", "becomes", "been", "before", "being", "be
                  "called", "could",
                   "did", "didnt", "does", "doing", "during",
                  "each",
                  "few",  "from", "further",
                  "having", "he'd", "he'll", "he's", "here", "here's", 'her', "h
                  "herself", "him", "himself", "his", "how", "how's",
                  "i", "i'd", "i'll", "i'm", "i've", "into", "it", "it's", "its"
                  "let's",
                  "myself", "means",
                  "once", "only", "other", "ought", "ourselves",
                  'part', 'parts' "probably",
                  "she'd", "she'll", "she's", "should", "such", "seems", 'someth
                  "the", "than", "that", "that's", "thats", "their", "theirs", "
                  "there's", "theres", "these", "they", "they'd", "they'll", "th
                 "those", "through", 'things', 'thing', "truly",
```

```
                    "until", "up",
                    "very",
                    "we'd", "we'll", "we're", "we've", "were", "what", "what's", "
                    "when's", "where", "where's", "which", "while", "whoever", "wh
                    "would", "whatever",
                     "you'd", "you'll", "you're", "you've", "youve", "yourself", "
        for word in stopwords:
            sen_str = regex.sub(r'(?<=^)' + word + '(?=\s)', '', sen_str)
            sen_str = regex.sub(r'(?<=\s)' + word + '(?=\s)', '', sen_str)
            sen_str = regex.sub(r'(?<=\s)' + word + '(?=\n)', '', sen_str)

        # [Step 5] Remove text that are 1 or 2 characters long - should remove all a
        sen_str = regex.sub(r'(?<=^)[A-Za-z]{1,3}(?=\s)', '', sen_str)
        sen_str = regex.sub(r'(?<=\s)[A-Za-z]{1,3}(?=\s)', '', sen_str)
        sen_str = regex.sub(r'(?<=\s)[A-Za-z]{1,3}(?=\n)', '', sen_str)

        # [Step 6] Finally remove all multiple spaces, and replace with a single spa
        sen_str = regex.sub(r'\s+', ' ', sen_str)

        # ----------------------------

        # [1] Remove sentences with less than 10 words. Narrow the sentences down to
        if (len(sen_str.split()) > 10) & (len(y_str) > 0):
            X.append(sen_str)
            Y.append(y_str)
```

```
# General information about the dataset
print('There are ', len(X), 'sentences provided.')
unq_labels = list(set(Y))
NUM_OF_CLASSES = len(unq_labels)
print('There are ', NUM_OF_CLASSES, 'prediction categories.')
print('The prediction categories include: ', unq_labels)

# Dataset0
# There are  1490 sentences provided.
# There are  5 prediction categories.
# The prediction categories include:  ['entertainment', 'politics', 'sport', 'bu

# Dataset1 : good cleaning
# There are  16222 sentences provided.
# There are  10 prediction categories.
# The prediction categories include:  ['politics', 'food & drink', 'travel', 'bu
```

Another pre-processing step Used for Huffington Post dataset: concatenate sentences from the same class label to make the data have more descriptive features per category (MAXLEN larger) instead of making the data have more samples with respect to a class (padding samples per category)

```
# Verify that Y classes have similiar count values
c = Counter(Y)
c
```

```
class_label = list(c.keys())
# print('class_label:',class_label)

num_of_sen2cat = 2 # concatenate 2 sentences

X_longer = []
Y_longer = []
for i in class_label:
    # print('i:', i)

    # Make an index for X and Y, for each category
    index = []
    for ind, val in enumerate(Y):
        if val == i:
            index.append(ind)
    # print('index:', index)

    tot_len = len(index)
    new_tot_len = int(tot_len/num_of_sen2cat)
    # print('new_tot_len:', new_tot_len)

    for k in range(new_tot_len):

        # for each k I need to index temp  [start_ind:end_ind]
        start_ind = k*num_of_sen2cat
        # print('start_ind:', start_ind)

        if (start_ind + num_of_sen2cat) > tot_len:
            # at the end and there are not enough sentences
            end_ind = tot_len
        else:
            end_ind = start_ind + num_of_sen2cat
```

```
        # print('end_ind:', end_ind)

        indicies = index[start_ind:end_ind]
        #print('indexes:', indexes)

        X_cat = [X[i] for i in indicies]
        X_cat = np.ravel(X_cat)
        X_cat = ' '.join(X_cat)

        X_longer.append(X_cat)
        Y_longer.append(i)
```

Step 1:Balance the classes (add repeated samples of the same class (ie: pad data))

```
which_one = 0

if which_one == 0:
    X_padded, Y_padded = sequential_padder(X, Y)
else:
    X_padded, Y_padded = sequential_padder(X_longer, Y_longer)
```

Step 2a: Tokenize the sentences for [classification by token]

```
# Tokenizer 0: tensorflow Tokenizer
tokenizer0 = create_tokenizer0(X_padded)

word_index = tokenizer0.word_index  # is dict[word] = index

NUM_WORDS = len(tokenizer0.word_index) + 1  # add 1 because count starts at 0
print('NUM_WORDS: ', NUM_WORDS)

word_index
```

```
# Convert X to sequences
sequences = tokenizer0.texts_to_sequences(X_padded)

# Calculate the maximum sentence length
sen_len = [len(i.split(' ')) for i in X_padded]
max_sen_len = np.max(sen_len)
print('Maximum sentence length: ', max_sen_len)
print('Minimum sentence length: ', np.min(sen_len))

MAXLEN = max_sen_len # make each sequence this length # accuracy 0.17
# MAXLEN = int(max_sen_len/2)  # accuracy 0.15
# MAXLEN = int(max_sen_len/4)  # accuracy 0.2
print('MAXLEN: ', MAXLEN)

# Pad the sequences using the correct padding and maxlen
sequences = pad_sequences(sequences, maxlen=MAXLEN, padding='post', truncating='
```

Step 2b: Tokenize the sentences for [bag-of-words classification]

```
# Using scikit functions :
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.t
from sklearn.feature_extraction.text import CountVectorizer

# Get term-frequency matrix
vectorizer = CountVectorizer()

# Learn the vocabulary dictionary and return
# document-term matrix.
X = vectorizer.fit_transform(X_longer)

# Print keywords
# after version 1.0 = get_feature_names_out()
keywords = vectorizer.get_feature_names_out()
# print('keywords: ', keywords)
print('length of keywords: ', len(keywords))

# Term-frequency matrix OR matrix of counts
tf_mat = X.toarray()
```

```
print('size of tf_mat (sentences, keywords) : ', tf_mat.shape)
X_freq_count = tf_mat
```

Encode labels

```
# Encode labels
label_sequences, y_assignment = encode_labels(Y_longer)
```

Create Train and Test datasets

```
# Train-test split on X and Y
TRAINING_SPLIT = 0.7

which_one = 0

if which_one == 0:
    train_size = int(TRAINING_SPLIT*len(label_sequences))
    X_train = [sequences[i] for i in range(train_size)]
    Y_train = [label_sequences[i] for i in range(train_size)]

    X_test = [sequences[i] for i in range(train_size, len(label_sequences))]
    Y_test = [label_sequences[i] for i in range(train_size, len(label_sequences)

else:
    from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test = train_test_split(X_freq_count, label_sequ
                                                        train_size=TRAINING_SPLI
                                                        random_state = 0)

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)

print('X_train.shape: ', X_train.shape)
print('Y_train.shape: ', Y_train.shape)
```

```
print('X_test.shape: ', X_test.shape)
print('Y_test.shape: ', Y_test.shape)
```

Create Dataset

```
BATCH_SIZE = 1
ds_train = tf.data.Dataset.from_tensor_slices((X_train, Y_train)).batch(BATCH_SI
ds_test = tf.data.Dataset.from_tensor_slices((X_test, Y_test)).batch(BATCH_SIZE)
```

Load Model

```
EMBEDDING_DIM = 64

which_one = 0

if which_one == 0:

    # Sort of like clustering: learning which words are grouped together
    # with respect to the label [classification by token]
    kernel_regularizer=tf.keras.regularizers.l2(0.1)
    initializer = tf.keras.initializers.HeUniform()
    num_of_cols = len(X_train[1])
    print('num_of_cols: ', num_of_cols)
    inputs = tf.keras.Input(shape=(num_of_cols,))

    # Basic model for text classification and embeddings
    x = tf.keras.layers.Embedding(input_dim=NUM_WORDS, output_dim=EMBEDDING_DIM,
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    outputs = tf.keras.layers.Dense(NUM_OF_CLASSES,
                                    activation='softmax',
                                    kernel_regularizer=kernel_regularizer,
                                    kernel_initializer=initializer)(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

```
        base_learning_rate = 0.01
        # from_logits=False says to NOT calculate sigmoid/softmax, because it is alr
        optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate) # OR
        loss = tf.losses.SparseCategoricalCrossentropy(from_logits=False)  # OR loss
        metrics = ['accuracy'] # OR metrics=['acc']
        model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

    elif which_one == 1:

        # Bidirectional RNN: learning which words are grouped together with respect
        # learning sequencial ordering of likely grouped words [classification by to
        kernel_regularizer=tf.keras.regularizers.l2(0.1)
        initializer = tf.keras.initializers.HeUniform()
        num_of_cols = len(X_train[1])
        print('num_of_cols: ', num_of_cols)
        inputs = tf.keras.Input(shape=(num_of_cols,))

        # In notes: Text_classification_example14.ipynb for overfitting
        x = tf.keras.layers.Embedding(input_dim=NUM_WORDS, output_dim=EMBEDDING_DIM,
        # ndim=3 after Embedding Vector
        x = tf.keras.layers.Dropout(0.3)(x)
        # BidirectionalLSTM requires ndim=3
        n_a = 64
        x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(n_a))(x)
        x = tf.keras.layers.Dense(n_a, activation='relu', kernel_regularizer=kernel_

        outputs = tf.keras.layers.Dense(NUM_OF_CLASSES,
                                  activation='softmax',
                                  kernel_regularizer=kernel_regularizer,
                                  kernel_initializer=initializer)(x)

        model = tf.keras.Model(inputs=inputs, outputs=outputs)

        base_learning_rate = 0.0001
        # from_logits=False says to NOT calculate sigmoid/softmax, because it is alr
        optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate) # OR
        loss = tf.losses.SparseCategoricalCrossentropy(from_logits=False)  # OR loss
        metrics = ['accuracy'] # OR metrics=['acc']
        model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

    elif which_one == 2:

        # Deep layer NN for classifying frequency count [bag-of-words]
        # Learning which words repeat (or 'are important') with respect to the label
        # but it does not consider the sequencial ordering of the words in a 'projec

        kernel_regularizer=tf.keras.regularizers.l2(0.1)
        initializer = tf.keras.initializers.HeUniform()
```

```
num_of_rows, num_of_cols = X_freq_count.shape
print('num_of_cols: ', num_of_cols)
inputs = tf.keras.Input(shape=(num_of_cols,))

# Good architecture for overfitting
x = tf.keras.layers.Dense(128, input_dim=num_of_cols, activation='relu')(inp
x = tf.keras.layers.Dropout(0.4)(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)

outputs = tf.keras.layers.Dense(NUM_OF_CLASSES,
                          activation='softmax',
                          kernel_regularizer=kernel_regularizer,
                          kernel_initializer=initializer)(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

base_learning_rate = 0.0001
# from_logits=False says to NOT calculate sigmoid/softmax, because it is alr
optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate) # OR
loss = tf.losses.SparseCategoricalCrossentropy(from_logits=False)  # OR loss
metrics = ['accuracy'] # OR metrics=['acc']
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

```
model.summary()
```

```
# Embedding layer: bbc
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=60, m

EPOCHS = 30
STEPS_PER_EPOCH = 200

history = model.fit(X_train, Y_train,
                    validation_data=(X_test, Y_test),
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH,
                   callbacks=[early_stopping])
```

```
Epoch 27/30
200/200 [... val_accuracy:
0.7360
Epoch 28/30
200/200 [==============================] - 4s 19ms/step - loss: 1.1837 - accuracy: 0.8100 - val_loss: 1.1788 - val_accuracy:
0.9364
Epoch 29/30
200/200 [==============================] - 3s 16ms/step - loss: 1.1866 - accuracy: 0.8050 - val_loss: 1.2099 - val_accuracy:
0.8112
Epoch 30/30
```

Using the Embedding layer for the BBC dataset I reached roughly 0.8 accuracy for 30 epochs, it could be trained longer for 50–100 epochs for stable accuracy results.

```
# Frequency term: huffington post
EPOCHS = 10

history = model.fit(ds_train,
                    validation_data=ds_test,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS
                   )
```

```
Epoch 1/10
9142/9142 [==============================] - 256s 28ms/step - loss: 2.0279 - accuracy: 0.3019 - val_loss: 1.1376 - val_accur
acy: 0.6756
Epoch 2/10
9142/9142 [==============================] - 243s 27ms/step - loss: 0.8630 - accuracy: 0.7396 - val_loss: 0.6846 - val_accur
acy: 0.8173
Epoch 3/10
9142/9142 [==============================] - 246s 27ms/step - loss: 0.4555 - accuracy: 0.8882 - val_loss: 0.5696 - val_accur
acy: 0.8596
Epoch 4/10
9142/9142 [==============================] - 244s 27ms/step - loss: 0.2713 - accuracy: 0.9430 - val_loss: 0.5349 - val_accur
acy: 0.8719
Epoch 5/10
9142/9142 [==============================] - 237s 26ms/step - loss: 0.1749 - accuracy: 0.9708 - val_loss: 0.5016 - val_accur
acy: 0.8859
Epoch 6/10
9142/9142 [==============================] - 241s 26ms/step - loss: 0.1221 - accuracy: 0.9803 - val_loss: 0.5268 - val_accur
acy: 0.8770
Epoch 7/10
9142/9142 [==============================] - 237s 26ms/step - loss: 0.0926 - accuracy: 0.9844 - val_loss: 0.5309 - val_accur
acy: 0.8862
Epoch 8/10
9142/9142 [==============================] - 240s 26ms/step - loss: 0.0681 - accuracy: 0.9908 - val_loss: 0.5526 - val_accur
acy: 0.8849
Epoch 9/10
9142/9142 [==============================] - 237s 26ms/step - loss: 0.0508 - accuracy: 0.9947 - val_loss: 0.5484 - val_accur
acy: 0.8874
Epoch 10/10
9142/9142 [==============================] - 244s 27ms/step - loss: 0.0446 - accuracy: 0.9949 - val_loss: 0.5605 - val_accur
acy: 0.8892
```

Using the Frequency Term matrix [bag-of-words] for the Huffington Post dataset I could get 0.9 accuracy in 10 epochs, using the which_one =2 model selection.

```
# Embedding layer: huffington post
EPOCHS = 10
STEPS_PER_EPOCH = 20

# validataion_data does not work AND train accuracy is different for using X_tra
history = model.fit(X_train, Y_train,
                    validation_data=(X_test, Y_test),
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH)
```

```
Epoch 1/10
20/20 [==============================] - 125s 6s/step - loss: 12.5843 - accuracy: 0.1000 - val_loss: 12.4136 - val_accuracy: 0.0715
Epoch 2/10
20/20 [==============================] - 122s 6s/step - loss: 12.2560 - accuracy: 0.0000e+00 - val_loss: 12.1105 - val_accuracy: 0.0189
Epoch 3/10
20/20 [==============================] - 121s 6s/step - loss: 11.9426 - accuracy: 0.1500 - val_loss: 11.8350 - val_accuracy: 0.0000e+00
Epoch 4/10
20/20 [==============================] - 120s 6s/step - loss: 11.6453 - accuracy: 0.1500 - val_loss: 11.5824 - val_accuracy: 0.0000e+00
Epoch 5/10
20/20 [==============================] - 121s 6s/step - loss: 11.3672 - accuracy: 0.0500 - val_loss: 11.3208 - val_accuracy: 0.0000e+00
Epoch 6/10
20/20 [==============================] - 122s 6s/step - loss: 11.0604 - accuracy: 0.1500 - val_loss: 11.1045 - val_accuracy: 0.0000e+00
Epoch 7/10
20/20 [==============================] - 122s 6s/step - loss: 10.8002 - accuracy: 0.2000 - val_loss: 10.8677 - val_accuracy: 0.0000e+00
Epoch 8/10
20/20 [==============================] - 121s 6s/step - loss: 10.4863 - accuracy: 0.1500 - val_loss: 10.6500 - val_accuracy: 0.0000e+00
Epoch 9/10
20/20 [==============================] - 120s 6s/step - loss: 10.3522 - accuracy: 0.1000 - val_loss: 10.4188 - val_accuracy: 0.0000e+00
Epoch 10/10
20/20 [==============================] - 121s 6s/step - loss: 10.0634 - accuracy: 0.1000 - val_loss: 10.1245 - val_accuracy: 0.0222
```

Using the Embedding layer for the Huffington Post dataset I reached roughly 0.2 accuracy for 10 epochs, not a reliable accuracy trend as was seen in the BBC dataset.

Test the model

In the test, model I only evaluate the Huffington Post dataset because it had difficulty training with the Embedding layer architecture.

```
# ---------------------------
# Obtain a sentence
# ---------------------------
i = np.random.permutation(np.arange(len(X_train)))[0]
print('i:', i)


which_way = 'input_seq'  # input_sen
```

```
    if which_way == 'input_seq':
        seq_example = X_train[i]

        print('sentence:', X_longer[i])

        # Print sentence: decode the first sequence using the Tokenizer class
        # https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tok
        #sen_example = tokenizer0.sequences_to_texts([X_train[i]])
        #print('sen_example: ', sen_example)

    else:
        # Sentence example
        # ---------------------------
        # Transform the sentence into a sequence
        # ---------------------------
        # Prepare seed_text
        seq_example = tokenizer0.texts_to_sequences([sen_example])[0]

        # Pad the sequence
        seq_example = pad_sequences([seq_example], maxlen=MAXLEN, padding='post', tr
    # print('seq_example: ', seq_example)


    seq_example =  tf.constant(seq_example, dtype=tf.float32)
    seq_example = tf.reshape(seq_example, [len(seq_example), 1])
    seq_example = tf.expand_dims(seq_example, axis=0)


    # ---------------------------
    # Predict with pre-trained model
    # ---------------------------
    probabilities = model.predict(seq_example, verbose=0)
    predicted_index = np.argmax(probabilities)
    print('predicted_index: ', predicted_index)

    # ---------------------------
    # Print result
    # ---------------------------
    y_assignment_reverse = dict((v, k) for k, v in y_assignment.items())
    print('y_assignment_reverse: ', y_assignment_reverse)
    print('predicted_label:', y_assignment_reverse[predicted_index])
    print('true_label:', y_assignment_reverse[Y_train[i]])
```

Below we can see that the model predicts extremely well for the Frequency term matrix, however we can also see that the sentences do not literally correspond to the class label topic. In example 1 the sentence words are related to the topic of travel, but in example 2 the sentence words could either correspond to wellness or business.

```
i: 3344
sentence:  apocalypse approaching start crossing bucket wanted private island gamble savings treat earth weeks it california
known gorgeous weather breathtaking landscapes personal favorite yearround festivals difficult narrow festivals picks
predicted_index:  8
y_assignment_reverse:  {0: 'wellness', 1: 'business', 2: 'food & drink', 3: 'entertainment', 4: 'world news', 5: 'parenting
', 6: 'politics', 7: 'style & beauty', 8: 'travel', 9: 'sports'}
predicted_label: travel
true_label: travel
```

Example 1. A Huffington Post dataset sentence that corresponds to the class label travel.

```
i: 7046
sentence:  simmons legend better recent interview business jeffrey hayzlett opportunity legend learn business  starting busi
nesses finishing novels selling albums getting casting calls chained chairs shackled spreadsheets drowned deadend jobs
predicted_index:  0
y_assignment_reverse:  {0: 'wellness', 1: 'business', 2: 'food & drink', 3: 'entertainment', 4: 'world news', 5: 'parenting
', 6: 'politics', 7: 'style & beauty', 8: 'travel', 9: 'sports'}
predicted_label: wellness
true_label: wellness
```

Example 2. A Huffington Post dataset sentence that sort of corresponds to the class label wellness, the sentence could be similar to the topic of business.

It is likely that the Embedding layer failed to capture differences between classes because many of the sentences had mixed keywords across different class labels.

Understanding a little bit why using the embedding layer works!

I evaluated both the word embedding vectors for both datasets. The BBC sentence embedding results per class are shown below because they produced the most contrast between classes.

Get the Embedding weights

```
def normalize_nestedarrs_by_max(arr):
    # Normalize the arr = [[1, 2, 3], [4, -5, 6]] from 0 to [-1 or 1]
    rows_of_dist = len(arr)

    # Find the maximum value
    max_val = np.max([np.max(np.abs(arr[i])) for i in range(rows_of_dist)])
    # print('max_val: ', max_val)

    # Normalize
    arr_nor = [arr[i]/max_val for i in range(rows_of_dist)]

    return arr_nor
```

```
# Get the embedding layer from the model (i.e. first layer)
embedding_layer = model.layers[1] # layer embedding_1 is layer 1

# Get the weights of the embedding layer
embedding_weights = embedding_layer.get_weights()[0]
print(embedding_weights.shape)  #  (vocab_size, embedding_dim)

# Get the index-word dictionary: so
reverse_word_index = tokenizer0.index_word  # is dict[index] = word
reverse_word_index
```

```
      (24261, 64)
[73…  {1: '<OOV>',
       2: 'said',
       3: 'with',
       4: 'have',
       5: 'will',
       6: 'more',
       7: 'people',
       8: 'year',
       9: 'over',
       10: 'first',
```

We can see the assignment of words per token.

```
# Get the max embedding_weights value
max_emb = np.max(abs(embedding_weights))
```

```
print('max_emb: ', max_emb)
```

```
# Create a [sentence embedding] from word embeddings

# Loop over each sequence
num_of_seq, num_of_words = X_train.shape
avg_seqemb_per_sentence = []
for i in range(num_of_seq):

    seq_emb = np.zeros((len(embedding_weights[0]),))
    sequence = X_train[i]
    sequence_nozeros = [i for i in sequence if i > 0]

    # Per sequence, loop over each word and add up all the word embeddings to ge
    temp_seq_emb = []
    for word_num in sequence_nozeros:
        # get embedding for each word
        # Get the embedding weights associated with the current index, scale it
        word_embedding = embedding_weights[word_num]/max_emb

        # Without evaluation process of word embedding direction
        seq_emb = seq_emb + word_embedding

    # Sentence embedding: the average vector could represent the entire [sequenc
    avg_seqemb_per_sentence.append(seq_emb/len(sequence_nozeros))
```

```
# Normalize the sentence embeddings so they are from 0 to [-1 or 1]
max_val = np.max([np.max(abs(avg_seqemb_per_sentence[i])) for i in range(num_of_
print('max_val: ', max_val)

# Normalize the average sentence embedding from 0 to [-1 or 1]
seq_emb_per_sentence_nor = [avg_seqemb_per_sentence[i]/max_val for i in range(nu

# Sum the normalized [sentence embeddings] per class
class_count = Counter(Y_train)
class_num = sorted(list(class_count.keys()))
print('class_num: ', class_num)

# Initialize the [sentence embedding class dictionary]
seq_emb_avg_dict = {}
for i in class_num:
    seq_emb_avg_dict[i] = np.zeros((len(embedding_weights[0]),))
```

```
# Loop over each sequence
for i in range(num_of_seq):

    class_number = Y_train[i]

    # Sum up each sentence embedding per class
    seq_emb_avg_dict[class_number] = seq_emb_avg_dict[class_number] + seq_emb_pe

# Divid by the class count to find the [average sentence embedding per class]
for i in class_num:
    seq_emb_avg_dict[i] = seq_emb_avg_dict[i]/class_count[i]
```

```
# Verify that the embeddings are from -1 to 1
for cn in class_num:
    max_val = np.max([np.max(seq_emb_avg_dict[cn][i]) for i in range(num_of_seq)
    print(f'max_val class {cn}: ', max_val)

    min_val = np.min([np.min(seq_emb_avg_dict[cn][i]) for i in range(num_of_seq)
    print(f'min_val class {cn}: ', min_val)
```

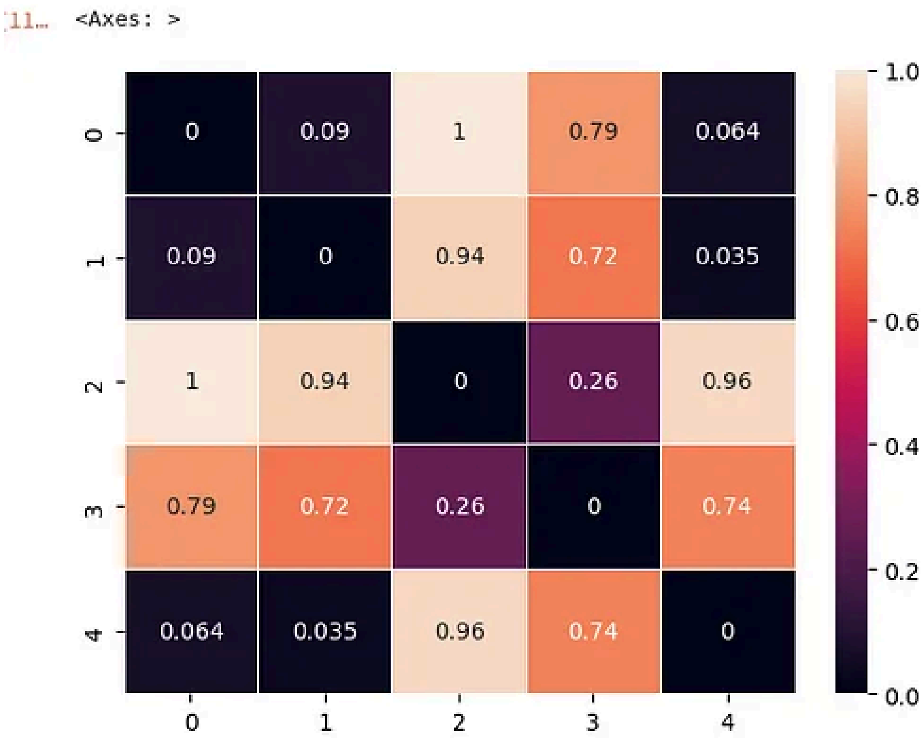
```
max_val class 0:  0.9778760760011381
min_val class 0:  -1.0
max_val class 1:  0.9778760760011381
min_val class 1:  -1.0
max_val class 2:  0.9778760760011409
min_val class 2:  -1.0
max_val class 3:  0.9778760760011392
min_val class 3:  -1.0
max_val class 4:  0.9778760760011381
min_val class 4:  -1.0
```

Indeed the average sentence embedding vector per class are each normalized from -1 to 1.

```
# Measure distance between average class embeddings
dist = []
for i in class_num:
    temp = []
    for j in class_num:
        temp.append( float(tf.norm(tf.subtract(seq_emb_avg_dict[i], seq_emb_avg_
                                ord='euclidean', axis=None, keepdims=None, name=Non
    dist.append(temp)
```

```
dist_nor = normalize_nestedarrs_by_max(dist)

import seaborn as sns
sns.heatmap(dist_nor, annot=True,  linewidths=.5)
```



L2 distance between average sentence embeddings per class. The x an y axis are the classes {'entertainment': 0, 'politics': 1, 'sport': 2, 'business': 3, 'tech': 4}.

One can see that entertainment, politics, and tech categories share similar words because their average sentence embedding vectors are pointing a similar direction. Where as sports and business class appear to have distinguishing words that cause their average sentence embedding vectors to point in different directions.

Summary

I could accurately train a model using an Embedding layer with the bbc-new-dataset because the words in the sentence often literally corresponded to the class topic label. Therefore, therefore the embedding space was more clustered per class. However, the news-category-dataset was less organized in the sense that:

1. the sentences did not always logically correspond to the class label topic
2. the sentences were shorter (100 words in comparison to 200–1000)
3. the label had 10 classes instead of 5 classes; the more the classes the more difficult it is to classify sentences.

Thus, for these reasons the news-category-dataset could not be accurately classified using an Embedding layer; the embedding space per class was too mixed. Using a simple term-frequency X-matrix allowed for accurate classification with a deep layer neural network. Transforming the sentences into a term-frequency matrix is simpler than finding outlier embeddings per class, however I think another viable algorithm solution to this problem would be to : calculate word embeddings, calculate sentence embeddings with only similar word embeddings and identify words for the non-similar word embeddings, calculate average sentence embeddings per class and compile a list of non-similar word embeddings per class.

Happy Practicing! 👋

🎁 Donate: support the blog! | 💻 GitHub | 🔔 Subscribe

References

1. Huffington Post dataset with 10 categories: https://www.kaggle.com/datasets/rmisra/news-category-dataset

2. Original data in Coursera Natural Language Processing Tensorflow (DeepLearning_AI_TensorFlow_Developer_Specialization). BBC new dataset Kaggle competition: https://www.kaggle.com/competitions/learn-ai-bbc

3. Kaggle notebook with code/workflow: https://www.kaggle.com/code/jamilahfoucher/text-classification-w-news-data

Word Embeddings | Text Classification | TensorFlow | Practicing Datscy | Embedding Layer

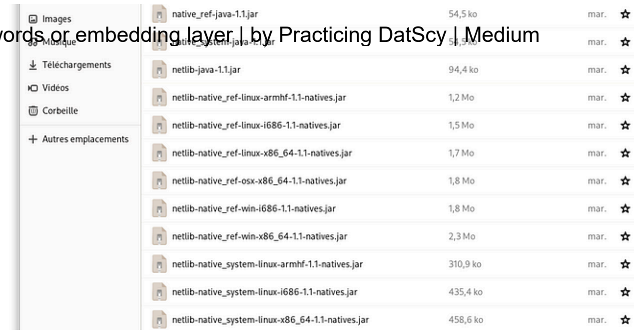
Written by Practicing DatScy

Follow

178 Followers

Practicing coding, Data Science, and research ideas. Blog brand: Use logic in a clam space, like a forest, and use reliable Data Science workflows!

More from Practicing DatScy



Practicing DatScy

Fine-tuning with OpenAI

I finally was able to test fine-tuning using OpenAI!! Fine-tuning using OpenAI's gpt...

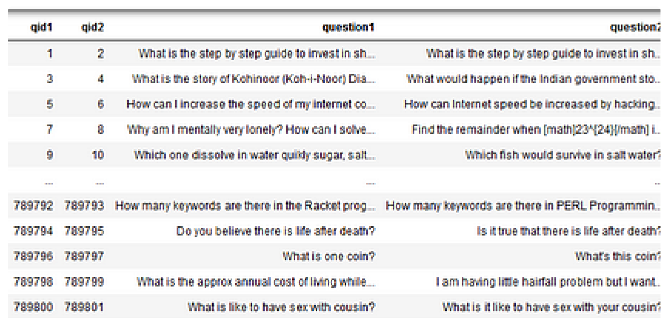
Dec 4, 2023 12

Practicing DatScy

Exploring the Java Weka Machine Learning library

For a couple of weeks I needed to use some Weka machine learning models. I knew that...

Dec 17, 2022



qid1	qid2	question1	question2
1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...
3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...
5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...
7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...
9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?
...	...	...	...
789792	789793	How many keywords are there in the Racket prog...	How many keywords are there in PERL Programmin...
789794	789795	Do you believe there is life after death?	Is it true that there is life after death?
789796	789797	What is one coin?	What's this coin?
789798	789799	What is the approx annual cost of living while...	I am having little hairfall problem but I want...
789800	789801	What is like to have sex with cousin?	What is it like to have sex with your cousin?

Practicing DatScy

HackerRank Tests: Python

I read that HackerRank tests are used by many companies as an evaluation method to...

Feb 1, 2022 14

Practicing DatScy

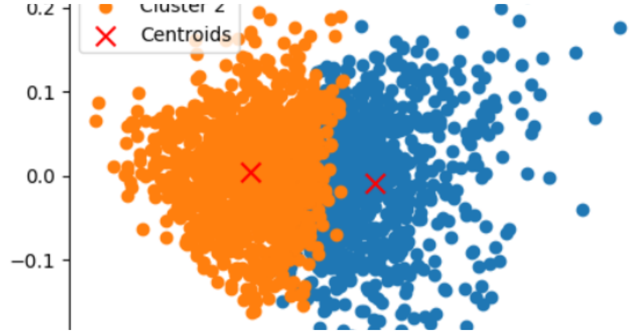
Classifying sentences: part 1 clustering sentences

After the post on chatbots, I was interested in practicing more text analysis techniques like...

Feb 24, 2022

See all from Practicing DatScy

Recommended from Medium



Mohamad Mahmood in Dev Genius

Unsupervised Text Classification Using K-Means Clustering...

Simple, Efficient, Adaptable

Mar 17 14

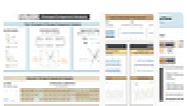
BoredGeekSociety

Improve Text Classification with OpenAI Embeddings

Can we obtain state of the art NLP classification results, with raw inputs, thanks...

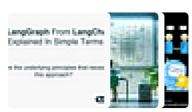
Jan 30 62

Lists



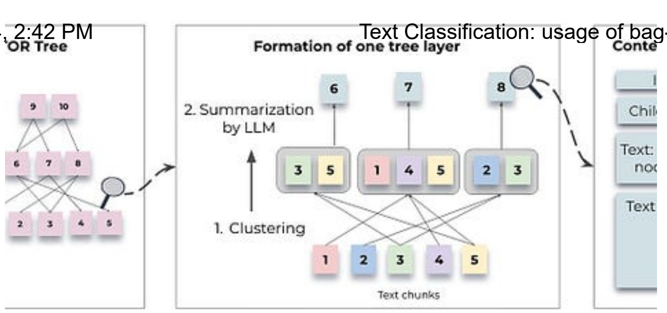
Practical Guides to Machine Learning

10 stories · 1570 saves



Natural Language Processing

1528 stories · 1061 saves

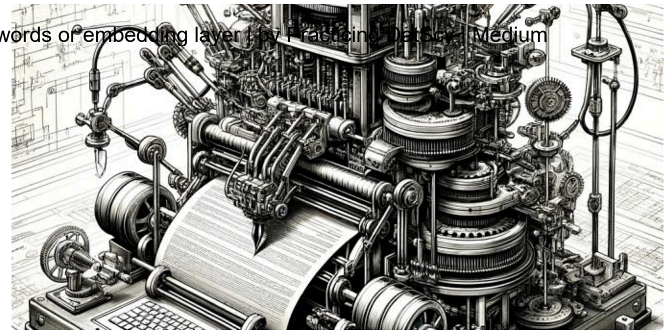


Florian June in AI Advances

Advanced RAG 12: Enhancing Global Understanding

Priciples, Code Explanation and Insights

Jun 12 · 397



Nick Hagar

LLM-generated labels for topic classification

Can synthetic data replace human annotators?

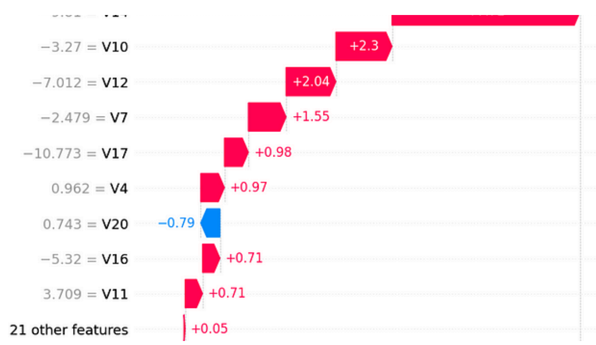
Mar 17 · 59

zhaozhiming in AI Advances

Advanced RAG Retrieval Strategies: Flow and Modular

We've discussed many advanced Retrieval Augmented Generation (RAG) retrieval...

Jun 11 · 395



Danila Morozovskii in Towards Data Science

Model Interpretability Using Credit Card Fraud Data

Why model interpretability is important

Jun 11 · 154 · 2

See more recommendations