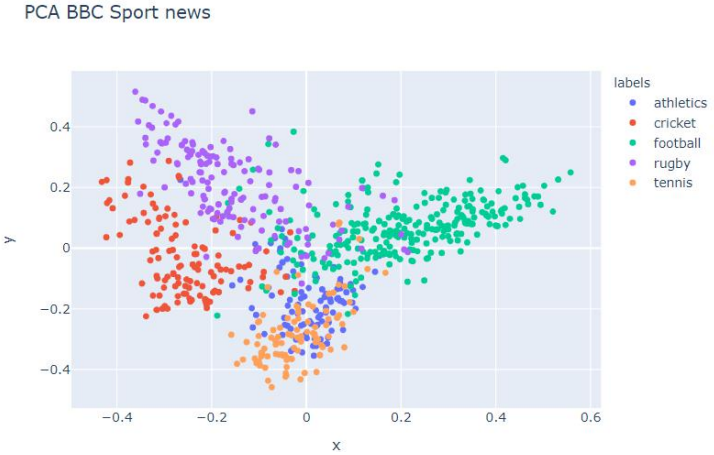
NLP Visualization

- Insights into Texts

About visualizaion

- Visualization is a very common technique used in NLP. It involves creating images or pictures for better understanding of texts.



About Texthero

- Texthero is a python toolkit to work with text-based dataset quickly. Texthero has the same expressiveness and power of Pandas and is extensively documented. Texthero is modern and conceived for programmers of the 2020 decade with little knowledge if any in linguistic.

About Texthero

- You can think of Texthero as a tool to help you *understand* and work with text-based dataset. Given a text dataset, it's harder to have quick insights into the underline data. With Texthero, preprocessing text data, mapping it into vectors, and visualizing the obtained vector space takes just a couple of lines.

Texthero include tools for…

- Preprocess text data: it offers both out-of-the-box solutions but it's also flexible for custom-solutions.
- Natural Language Processing: keyphrases and keywords extraction, and named entity recognition.
- Text representation: TF-IDF, term frequency, and custom word-embeddings (wip)
- Vector space analysis: clustering (K-means, Meanshift, DBSCAN and Hierarchical), topic modeling (wip) and interpretation.
- Text visualization: vector space visualization, place localization on maps (wip).

Installation

- Install texthero via pip:

```
pip install texthero
```

Getting started

- Given a dataset with structured data, it's easy to have a quick understanding of the underline data. Oppositely, given a dataset composed of text-only, it's harder to have a quick undertanding of the data.
- Texthero help you there, providing utility functions to quickly **clean the text data**, **map it into a vector space** and gather from it **primary insights**.

Getting started

- One of the main pillar of texthero is that is designed from the ground-up to work with **Pandas Dataframe** and **Series**.
- Most of texthero methods, simply apply transformation to Pandas Series. As a rule of thumb, the first argument and the return ouputs of almost all texthero methods are either a Pandas Series or a Pandas DataFrame.

Getting started

- The first phase of almost any natural language processing is almost the same, independently to the specific task.
- Pandas introduced Pipe function starting from version 0.16.2. Pipe enables user-defined methods in method chains

Getting started

- For a simple example we make use of the BBC Sport Dataset, it consists of 737 documents from the BBC Sport website corresponding to sports news articles in five topical areas from 2004-2005.
- The five different areas
  are *athletics*, *cricket*, *football*, *rugby* and *tennis*.
- The original dataset comes as a zip files with five different folder containing the article as text data for each topic.

Getting started

```
import texthero as hero
import pandas as pd
df=pd.read_csv
(  "https://github.com/jbesomi/texthero/raw/master/dataset/bbcsport.csv" )
```

```
>>> df.head(2)
                                                text     topic
0  "Claxton hunting first major medal\n\nBritish h...  athletics
1  "O'Sullivan could run in Worlds\n\nSonia O'Sull...  athletics
```

Preprocessing

- To clean the text data all we have to do is:

```
df['clean_text'] = hero.clean(df['text'])
```

- Recently, Pandas has introduced the pipe function. You can achieve the same results with

```
df['clean_text'] = df['text'].pipe(hero.clean)
```

Preprocessing

The default pipeline for the clean method is the following:

- fillna(s) Replace not assigned values with empty spaces.
- lowercase(s) Lowercase all text.
- remove_digits() Remove all blocks of digits.
- remove_punctuation() Remove all string.punctuation (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~).
- remove_diacritics() Remove all accents from strings.
- remove_stopwords() Remove all stop words.
- remove_whitespace() Remove all white space between words.

Custom pipeline

- We can also pass a custom pipeline as argument to clean

```
from texthero import preprocessing
custom_pipeline = [preprocessing.fillna,
                   preprocessing.lowercase,
                   preprocessing.remove_whitespace]
df['clean_text'] = hero.clean(df['text'], custom_pipeline)
```

- or alternatively

```
df['clean_text'] = df['clean_text'].pipe(hero.clean, custom_pipeline)
```

Representation

- Term_frequency representation

```
df['tf_clean_text'] = hero.term_frequency(df['clean_text'])
```

- Represent a text-based Pandas Series using term frequency.
- Return a Document Representation Series with the term frequencies of the terms for every document.

Representation

- TFIDF representation

```
df['tfidf_clean_text'] = hero.tfidf(df['clean_text'])
```

- Term Frequency - Inverse Document Frequency (TF-IDF) is a formula to calculate the _relative importance_ of the words in a document, taking into account the words' occurences in other documents. It consists of two parts:

Representation

- Term frequency (tf) tells us how frequently a term is present in a document, tf(document d, term t) = number of times t appears in d.

- The inverse document frequency (idf) measures how _important_ or_characteristic_ a term is among the whole corpus (i.e. among all documents). Thus, idf(term t) = log((1 + number of documents) / (1 + number of documents where t is present)) + 1.

- Finally, tf-idf(document d, term t) = tf(d, t) * idf(t).

Representation

- PCA representation

```
df['tfidf_clean_text'] = hero.pca(df['clean_text'])
```

- Principal Component Analysis (PCA) is a statistical method that is used to reveal where the variance in a dataset comes from. For textual data, one could for example first represent a Series of documents using tfidf to get a vector representation of each document. Then, PCA can generate new vectors from the tfidf representation that showcase the differences among the documents most strongly in fewer dimensions.

Representation

- NMF representation

```
df['tfidf_clean_text'] = hero.nmf(df['clean_text'])
```

- Non-Negative Matrix Factorization (NMF) is often used in natural language processing to find clusters of similar texts (e.g. some texts in a corpus might be about sports and some about music, so they will differ in the usage of technical terms).

Representation

- TSNE representation

```
df['tfidf_clean_text'] = hero.tsne(df['clean_text'])
```

- t-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm used to visualize high-dimensional data in fewer dimensions. In natural language processing, the high-dimensional data is usually a document-term matrix that is hard to visualize as there might be many terms. With t-SNE, every document gets a new, low-dimensional vector in such a way that the similarities between documents are preserved.

Representation

- K-means representation

```
df['tfidf_clean_text'] = hero.kmeans(df['clean_text'])
```

- K-means clustering is used in natural language processing to separate texts into k clusters (groups) (e.g. some texts in a corpus might be about sports and some about music, so they will differ in the usage of technical terms; the K-means algorithm uses this to separate them into two clusters).

Representation

- DBSCAN representation

```
df['tfidf_clean_text'] = hero.dbscan(df['clean_text'])
```

- Density-based spatial clustering of applications with noise (DBSCAN) is used in natural language processing to separate texts into clusters (groups) (e.g. some texts in a corpus might be about sports and some about music, so they will differ in the usage of technical terms; the DBSCAN algorithm uses this to separate them into clusters). It chooses the number of clusters on its own.

Representation

- Mean-shift representation

```
df['tfidf_clean_text'] = hero.meanshift(df['clean_text'])
```

- Mean shift clustering is used in natural language processing to separate texts into clusters (groups) (e.g. some texts in a corpus might be about sports and some about music, so they will differ in the usage of technical terms; the mean shift algorithm uses this to separate them into clusters). It chooses the number of clusters on its own.

All in one step

- We can achieve all the three steps show above, cleaning, tf-idf representation and dimensionality reduction in a single step. Isn't fabulous?

```
df['pca'] = (
        df['text']
        .pipe(hero.clean)
        .pipe(hero.tfidf)
        .pipe(hero.pca)
  )
```

Visualization

- texthero.visualization provide some helpers functions to visualize the transformed Dataframe. All visualization utilize under the hoods the Plotly Python Open Source Graphing Library.

hero.scatterplot(df, col='pca', color='topic', title="PCA BBC Sport news")

PCA BBC Sport news

Visualization

- we can "visualize" the most common words for each topic with top_words

NUM_TOP_WORDS = 5

df.groupby('topic')['text'].apply(lambda x: hero.top_words(x)[:NUM_TOP_WORDS])

```
topic
athletics  said       0.010068
           world      0.008900
           year       0.008844
cricket    test       0.008250
           england    0.008001
           first      0.007787
football   said       0.009515
           chelsea    0.006110
           game       0.005950
rugby      england    0.012602
           said       0.008359
           wales      0.007880
tennis     6          0.021047
           said       0.013012
           open       0.009834
```

Visualization

- hero.wordcloud(df['text_clean'])

Summary

Visualize insights and statistics of a text-based Pandas DataFrame.

scatterplot(df, col, color, hover_data[, ...])	Show scatterplot using python plotly scatter.
top_words(s[, normalize])	Return a pandas series with index the top words and as value the count.
wordcloud(s, font_path, width, height[, ...])	Plot wordcloud image using WordCloud from word_cloud package.

Summary

```
import texthero as hero
import pandas as pd

df = pd.read_csv(
   "https://github.com/jbesomi/texthero/raw/master/dataset/bbcsport.csv"
)
df['pca'] = (
   df['text']
   .pipe(hero.clean)
   .pipe(hero.tfidf)
   .pipe(hero.pca)
)
hero.scatterplot(df, col='pca', color='topic', title="PCA BBC Sport news")
```