RedditScore Documentation

Release 0.7.0

Evgenii Nikitin

Dec 16, 2018

Contents

RedditScore is a library that contains tools for building Reddit-based text classification models

RedditScore includes:

- Document tokenizer with myriads of options, including Reddit- and Twitter-specific options
- Tools to build and tune the most popular text classification models without any hassle
- Functions to easily collect Reddit comments from Google BigQuery and Twitter data (including tweets beyond 3200 tweets limit)
- Instruments to help you build more efficient Reddit-based models and to obtain RedditScores (*Nikitin2018*)
- Tools to use pre-built Reddit-based models to obtain RedditScores for your data

Note: RedditScore library and this tutorial are work-in-progress. Let me know if you experience any issues.

Usage example:

```
import os

import pandas as pd

from redditscore import tokenizer
from redditscore.models import fasttext_mod

df = pd.read_csv(os.path.join('redditscore', 'reddit_small_sample.csv'))
df = df.sample(frac=1.0, random_state=24) # shuffling data
tokenizer = CrazyTokenizer(hashtags='split') # initializing tokenizer object
X = df['body'].apply(tokenizer.tokenize) # tokenizing Reddit comments
y = df['subreddit']

fasttext_model = fasttext_mod.FastTextModel() # initializing fastText model

fasttext_model.tune_params(X, y, cv=5, scoring='accuracy') # tune hyperparameters of␣
↪the model using default grid
fasttext_model.fit(X, y) # fit model
fasttext_model.save_model('models/fasttext_model') # save model
fasttext_model = fasttext.load_model('models/fasttext_model') # load model

dendrogram_pars = {'leaf_font_size': 14}
tsne_pars = {'perplexity': 30.0}
fasttext_model.plot_analytics(dendrogram_pars=dendrogram_pars, # plot dendrogram and␣
↪T-SNE plot
                          tsne_pars=tsne_pars,
                          fig_sizes=((25, 20), (22, 22)))

probs = fasttext_model.predict_proba(X)
av_scores, max_scores = fasttext_model.similarity_scores(X)
```

References:

Contents:

CHAPTER 1

RedditScore Overview

RedditScore is a library that contains tools for building Reddit-based text classification models

RedditScore includes:

- Document tokenizer with myriads of options, including Reddit- and Twitter-specific options
- Tools to build and tune the most popular text classification models without any hassle
- Functions to easily collect Reddit comments from Google BigQuery and Twitter data (including tweets beyond 3200 tweets limit)
- Instruments to help you build more efficient Reddit-based models and to obtain RedditScores (*Nikitin2018*)
- Tools to use pre-built Reddit-based models to obtain RedditScores for your data

Note: RedditScore library and this tutorial are work-in-progress. Let me know if you experience any issues.

Usage example:

```
import os

import pandas as pd

from redditscore import tokenizer
from redditscore.models import fasttext_mod

df = pd.read_csv(os.path.join('redditscore', 'reddit_small_sample.csv'))
df = df.sample(frac=1.0, random_state=24) # shuffling data
tokenizer = CrazyTokenizer(hashtags='split') # initializing tokenizer object
X = df['body'].apply(tokenizer.tokenize) # tokenizing Reddit comments
y = df['subreddit']

fasttext_model = fasttext_mod.FastTextModel() # initializing fastText model

fasttext_model.tune_params(X, y, cv=5, scoring='accuracy') # tune hyperparameters of␣
↪the model using default grid
fasttext_model.fit(X, y) # fit model
```

(continues on next page)

(continued from previous page)

```
fasttext_model.save_model('models/fasttext_model') # save model
fasttext_model = fasttext.load_model('models/fasttext_model') # load model

dendrogram_pars = {'leaf_font_size': 14}
tsne_pars = {'perplexity': 30.0}
fasttext_model.plot_analytics(dendrogram_pars=dendrogram_pars, # plot dendrogram and␣
↪T-SNE plot
                          tsne_pars=tsne_pars,
                          fig_sizes=((25, 20), (22, 22)))

probs = fasttext_model.predict_proba(X)
av_scores, max_scores = fasttext_model.similarity_scores(X)
```

References:

CHAPTER 2

Installation

To install the package, run the following command:

```
>>> pip install git+https://github.com/crazyfrogspb/RedditScore.git
```

If you want to use all features of the RedditScore library, make sure to install these extra dependencies:

- **For training fastText models:**

```
>>> pip install Cython pybind11
>>> git clone https://github.com/crazyfrogspb/fastText.git
>>> cd fastText
>>> pip install .
```

- **For training neural networks:**

```
>>> pip install tensorflow tensorflow-gpu keras
```

- **For collecting Reddit data from BigQuery:**

```
>>> pip install pandas-gbq
```

- **For collecting Twitter data:**

```
>>> pip install selenium
>>> pip install git+https://github.com/crazyfrogspb/tweepy.git
```

- **For using stemming or NLTK stopwords lists:**

```
>>> pip install nltk
```

CHAPTER 3

Data collection

RedditScore library can be used on any kind of text data, but it was developed specifically for building Reddit-based models any applying them to Twitter data. This is why RedditScore also includes tools to easily pull data from these sources.

3.1 Reddit Data

The easiest way to obtain Reddit data is to use Google BigQuery public dataset of Reddit comments. Google BigQuery is not free, but luckily you can get 365 days trial and 300$ credit when you register for the first time.

After setting up your account, create a project and copy its Project ID. Next, create a service account as described here. Download JSON file with your private key.

After that, you're good to go! `get_comments` function in RedditScore library will help you to easily collect some Reddit comments. Let's say you want to collect all comments posted in subreddits /r/BeardAdvice and /r/beards from Jan 2016 to May 2016. This is how you do it:

```
>>> from redditscore import get_reddit_data as grd
>>> project_id = "my_first_project" # insert your Project ID here
>>> private_key = 'my_key.json' # insert path to your key file here
>>> subreddits = ['BeardAdvice', 'beards']
>>> df = grd.get_comments(('2016_01', '2016_05'),
>>>                       project_id, private_key, subreddits, verbose=True)
```

`df` is a pandas DataFrame with collected comments.

`get_comments` function also has a few additional options:

- Saving results by month into CSV files instead of returning a DataFrame:

```
>>> grd.get_comments(('2016_01', '2016_05'), project_id, private_key,␣
↪subreddits,
>>>                   verbose=True, csv_directory='reddit_data')
```

- Getting a random sample of comments from each subreddit per month:

```
>>> df = grd.get_comments(('2016_01', '2016_05'), project_id, private_key,␣
↪subreddits,
>>>                        verbose=True, comments_per_month=1000)
```

- Getting top-scoring comments from each subreddit per month:

```
>>> df = grd.get_comments(('2016_01', '2016_05'), project_id, private_key,␣
↪subreddits,
>>>                        verbose=True, comments_per_month=1000, top_
↪scores=True)
```

- Filtering out comments with low scores:

```
>>> df = grd.get_comments(('2016_01', '2016_05'), project_id, private_key,␣
↪subreddits,
>>>                        verbose=True, score_limit=3)
```

- Getting comments for a subset of users:

```
>>> df = grd.get_comments(('2016_01', '2016_05'), project_id, private_key,␣
↪subreddits,
>>>                        usernames=['crazyfrogspb', 'VladimirVladimirovich
↪'], verbose=True)
```

3.2 Twitter Data

RedditScore can help you to pull tweets from specific Twitter accounts. First, you need to create your own Twitter application and receive credentials. You can do it in Application Management. Save your consumer_key, consumer_secret, access_key, and access_secret to JSON file, and you're ready to collect the data!

```
>>> import json
>>> from redditscore import get_twitter_data as gtd
>>> cred_path = 'twitter_creds.json'
>>> with open(cred_path) as f:
>>>   twitter_creds = json.load(f)
>>> df = grab_tweets(twitter_creds, screen_name='crazyfrogspb')
>>> df = grad_tweets(twitter_creds, user_id=26281810)
```

There are a few optional arguments that `grab_tweets` function takes:

- `timeout`: time in seconds to wait between requests. Increase if you experience issues with Twitter API limits
- `fields`: additional fields to pull from tweets (e.g., favorite_count)
- `get_more`: if True, use Selenium library to pull tweets beyond infamous 3200 limit. Please note that it is quite slow and you need to have selenium package, browser (Chrome, Firefox, or Safari), and its webdriver installed.
- `browser`: which browser to use ('Chrome', 'Firefox', or 'Safari')
- `start_date`: date to start pulling additional tweets from (must be in datetime.date format). If None, pull all tweets for the user.

CHAPTER 4

CrazyTokenizer

4.1 Tokenizer description

CrazyTokenizer is a part of RedditScore project. It's a tokenizer - tool for splitting strings of text into tokens. Tokens can then be used as input for a variety of machine learning models. CrazyTokenizer was developed specifically for tokenizing Reddit comments and tweets, and it includes many features to deal with these types of documents. Of course, feel free to use for any other kind of text data as well.

CrazyTokenizer is based on the amazing spaCY NLP framework. Make sure to check it out!

4.2 Initializing

To import and to initialize an instance of CrazyTokenizer with the default preprocessing options, do the following:

```
>>> from redditscore.tokenizer import CrazyTokenizer
>>> tokenizer = CrazyTokenizer()
```

Now you can start tokenizing!

```
>>> text = ("@crazyfrogspb Hey,dude, have you heard that"
>>>         " https://github.com/crazyfrogspb/RedditScore is the best Python library?
↪")
>>> tokenizer.tokenizer(text)
['@crazyfrogspb', 'hey', 'dude', 'have', 'you', 'heard', 'that',
'https://github.com/crazyfrogspb/RedditScore', 'is', 'the', 'best', 'python', 'library
↪']
```

4.3 Features

4.3.1 Lowercasing and all caps

For many text classification problems, keeping capital letters only introduces unnecessary noise. Setting `lowercase=True` (True by default) will lowercase all words in your documents.

Sometimes you want to keep things typed in all caps (e.g., abbreviations). Setting `keepcaps=True` will do exactly that (default is False).

```
>>> tokenizer = CrazyTokenizer(lowercase=True, keepcaps=True)
>>> tokenizer.tokenize('Moscow is the capital of RUSSIA!')
['moscow', 'is', 'the', 'capital', 'of', 'RUSSIA']
```

4.3.2 Normalizing

Typing like thiiiis is amaaaaazing! However, in terms of text classification *amaaaaazing* is probably not too different from *amaaaazing*. CrazyTokenizer can normalize sequences of repeated characters for you. Just set `normalize=n`, where *n* is the number of characters you want to keep. Default value is 3.

```
>>> tokenizer = CrazyTokenizer(normalize=3)
>>> tokenizer.tokenize('GOOOOOOOOO Patriots!!!!!')
['gooo', 'patriots']
```

4.3.3 Ignoring quotes

People often quote other comments or tweets, but it doesn't mean that they endorse the original message. Removing the content of the quotes can help you to get rid of that. Just set `ignore_quotes=True` (False by deafult).

```
>>> tokenizer = CrazyTokenizer(ignore_quotes=True)
>>> tokenizer.tokenize('And then she said: "I voted for Donald Trump"')
['and', 'then', 'she', 'said']
```

4.3.4 Removing stop words

Removing stop words can sometimes significantly boost performance of your classifier. CrazyTokenizer gives you a few options to remove stop words:

- Using built-in list of the english stop words (`ignore_stopwords=True`)

```
>>> tokenizer = CrazyTokenizer(ignore_stopwords=True)
>>> tokenizer.tokenize('PhD life is great: eat, work, and sleep')
['phd', 'life', 'great', 'eat', 'work', 'sleep']
```

- Using NLTK lists of stop words. Just pass the name of the language of your documents to the `ignore_stopwords` parameter.

```
>>> tokenizer = CrazyTokenizer(ignore_stopwords='english')
# You might have to run nltk.download('stopwords') first
>>> tokenizer.tokenize('PhD life is great: eat, work, and sleep')
['phd', 'life', 'great', 'eat', 'work', 'sleep']
```

- Alternatively, you can supply your own custom list of the stop words. Letter case doesn't matter.

```
>>> tokenizer = CrazyTokenizer(ignore_stopwords=['Vladimir', "Putin"])
>>> tokenizer.tokenize("The best leader in the world is Vladimir Putin")
['the', 'best', 'leader', 'in', 'the', 'world', 'is']
```

4.3.5 Word stemming and lemmatizing

If you have NLTK installed, CrazyTokenizer can use PorterStemmer or WordNetLemmatizer for you. Just pass `stem` or `lemm` options respectively to `stem` parameter.

```
>>> tokenizer = CrazyTokenizer(stem='stem')
>>> tokenizer.tokenize("I am an unbelievably fantastic human being")
['i', 'am', 'an', 'unbeliev', 'fantast', 'human', 'be']
```

4.3.6 Removing punctuation and linebreaks

Punctuation and linebreak characters usually just introduce extra noise to your text classification problem, so you can easily remove it with `remove_punct` and `remove_breaks` options. Both default to True.

```
>>> tokenizer = CrazyTokenizer(remove_punct=True, remove_breaks=True)
>>> tokenizer.tokenize("I love my life, friends, and oxford commas. \n Amen!")
['i', 'love', 'my', 'life', 'friends', 'and', 'oxford', 'commas', 'amen']
```

4.3.7 Decontracting

CrazyTokenizer can attempt to expand some of those annoying contractions for you. **Note**: use at your own risk.

```
>>> tokenizer = CrazyTokenizer(decontract=True)
>>> tokenizer.tokenize("I'll have two number nines, a number nine large...")
['i', 'will', 'have', 'two', 'number', 'nines', 'a', 'number', 'nine', 'large']
```

4.3.8 Dealing with hashtags

Hashtags are super-popular on Twitter. CrazyTokenizer can do one of three things about them:

- Do nothing (`hashtags=False`)
- Replace all of them with a placeholder token (`hashtags='TOKEN'`)
- Split them into separate words (`hashtags='splits'`)

Splitting hashtags is especially useful for the Reddit-based models since hashtags are not used on Reddit, and you can potentially lose a lot of semantic information when you calculate RedditScores for the Twitter data.

```
>>> tokenizer = CrazyTokenizer(hashtags=False)
>>> text = "Let's #makeamericagreatagain#americafirst"
>>> tokenizer.tokenize(text)
["let's", "#makeamericagreatagain", "#americafirst"]
>>> tokenizer = CrazyTokenizer(hashtags="HASHTAG_TOKEN")
["let's", "HASHTAG_TOKEN", "HASHTAG_TOKEN"]
>>> tokenizer = CrazyTokenizer(hashtags='split')
["let's", "make", "america", "great", "again", "america", "first"]
```

4.3.9 Dealing with special tokens

CrazyTokenizer correctly handles Twitter handles, subreddits, Reddit usernames, emails, all sorts of numbers, and extracts them as separate tokens:

```
>>> tokenizer = CrazyTokenizer()
>>> text = "@crazyfrogspb recommends /r/BeardAdvice!"
>>> tokenizer.tokenize(text)
['@crazyfrogspb', 'recommends', '/r/beardadvice']
```

However, you might want to completely remove certain types of tokens (for example, it makes sense to remove subreddit names if you want to compute RedditScores for the Twitter data), or to replace them with special tokens. Well, it's your lucky day, CrazyTokenizer can do that!

```
>>> tokenizer = CrazyTokenizer(subreddits='', twitter_handles='ANOTHER_TWITTER_USER')
>>> tokenizer.tokenize(text)
['ANOTHER_TWITTER_USER', 'recommends']
```

There are two special options for Twitter handles: 'realname' and 'split'. 'realname' replaces each handle with the screen name of the user that is listed in their profile.

```
>>> tokenizer = CrazyTokenizer(hashtags='split', twitter_handles='realname')
>>> tokenizer.tokenize('@realDonaldTrump please #MakeAmericaGreatAgain')
['donald', 'j.', 'trump', 'please', 'make', 'america', 'great', 'again']
```

'split' splits handles into separate words using Viterbi algorithm.

```
>>> tokenizer = CrazyTokenizer(twitter_handles='split')
>>> tokenizer.tokenize('@realDonaldTrump loves @FoxNews')
['real', 'donald', 'trump', 'loves', 'fox', 'news']
```

4.3.10 URLs

NLP practicioners often simply remove all URL occurrences since they do not seem to contain any useful semantic information. Of course, CrazyTokenizer correctly recognizes URLs as separate tokens and can remove or replace them with a placeholder token.

```
>>> tokenizer = CrazyTokenizer(urls=False)
>>> text = "Where is my job then?https://t.co/pN2TE5HDQm"
>>> tokenizer.tokenize(text)
['where', 'is', 'my', 'job', 'then', 'https://t.co/pN2TE5HDQm']
>>> tokenizer = CrazyTokenizer(urls='URL')
>>> tokenizer.tokenize(text)
['where', 'is', 'my', 'job', 'then', 'URL']
```

CrazyTokenizer can do something even more interesting though. Let's explore all options one by one.

First, CrazyTokenizer can extract domains from your URLs.

```
>>> tokenizer = CrazyTokenizer(urls='domain')
>>> text = "http://nytimes.com or http://breitbart.com, that is the question"
>>> tokenizer.tokenize(text)
['nytimes', 'or', 'breitbart', 'that', 'is', 'the', 'question']
```

Unfortunately, links on Twitter are often shortened, so extracting domain directly doesn't make a lot of sense. Not to worry though, CrazyTokenizer can handle that for you! Setting `urls='domain_unwrap_fast'` will deal with links shortened by the following URL shorteners: t.co, bit.ly, goo.gl, tinyurl.com.

```
>>> tokenizer = CrazyTokenizer(urls='domain_unwrap_fast')
>>> text = "Where is my job then?https://t.co/pN2TE5HDQm"
>>> tokenizer.tokenize(text)
['where', 'is', 'my', 'job', 'then', 'bloomberg_domain']
```

If you want, CrazyTokenizer can attempt to unwrap ALL extracted URLs.

```
>>> tokenizer = CrazyTokenizer(urls='domain_unwrap')
>>> text = "Where is my job then?https://t.co/pN2TE5HDQm"
>>> tokenizer.tokenize(text)
['where', 'is', 'my', 'job', 'then', 'bloomberg_domain']
```

Last but not least, CrazyTokenizer can extract web page titles, tokenize them, and insert to your tokenized sentences. Note: it won't extract titles from the Twitter pages in order to avoid duplicating tweets content.

```
>>> tokenizer = CrazyTokenizer(urls='title')
>>> text = "I love Russia https://goo.gl/3ioXU4"
>>> tokenizer.tokenize(text)
['i', 'love', 'russia', 'russia', 'to', 'block', 'telegram', 'app', 'over',
↪'encryption', 'bbc', 'news']
```

Please note that CrazyTokenizer has to make requests to the websites, and it is a very time-consuming operation, so CrazyTokenizer saves all parsed domains and web page titles. If you plan to experiment with the different preprocessing options and/or models, you should consider saving extracted domains/titles and then supplying saved dictionary as an argument to `urls` parameter.

```
>>> import json
>>> with open('domains.json', 'w') as f:
      json.dump(tokenizer._domains, f)
>>> with open('realnames.json', 'w') as f:
      json.dump(tokenizer._realnames, f)
>>> with open('domains.json', 'r') as f:
      domains = json.load(f)
>>> with open('realnames.json', 'r') as f:
      realnames = json.load(f)
>>> tokenizer = CrazyTokenizer(urls=domains, twitter_handles=realnames)
```

4.3.11 Extra patterns and keeping untokenized

You can also supply your own replacement rules to CrazyTokenizer. In particular, you need to provide a tuple that contains unique name for your rule, compiled re pattern and a replacement token.

Also, it makes sense to keep some common expressions (e.g., "New York Times") untokenized. If you think that it can improve your model quality, feel free to supply a list of strings that should be kept as single tokens.

```
>>> import re
>>> rule0 = re.compile(r"[S,s]ucks")
>>> rule1 = re.compile(r"[R,r]ules")
>>> tokenizer = CrazyTokenizer(extra_patterns=[('rule0', rule0, 'rules'),
                                               ('rule1', 'rule1, "sucks')],
                               keep_untokenized=['St.Petersburg'],
                               lowercase=False)
>>> text = "Moscow rules, St.Petersburg sucks"
['Moscow', 'sucks', 'St.Petersburg', 'rules']
```

4.3.12 Converting whitespaces to underscores

Popular implementations of models (most notably, fastText) do not support custom token splitting rules and simply split on whitespaces. In order to deal with that, CrazyTokenizer can replace all whitespaces in the final tokens by underscores (enabled by deafult).

```
>>> tokenizer = CrazyTokenizer(whitespaces_to_underscores=True, keep_untokenized=[
↪"New York"])
>>> text = "New York is a great place to make a rat friend"
>>> tokenizer.tokenize(text)
['new_york', 'is', 'a', 'great', 'place', 'to', 'make', 'a', 'rat', 'friend']
```

4.3.13 Removing non-unicode characters

```
>>> tokenizer = CrazyTokenizer(remove_nonunicode=True)
>>> text = " –   ,  –   !"
>>> tokenizer.tokenize(text)
[]
```

4.3.14 Emojis

Social media users are notoriously famous for their excessive use of emojis. CrazyTokenizer correctly separates consecutive emojis.

In addition, CrazyTokenizer can replace different kind of emojis with the corresponding word tokens.

```
>>> tokenizer = CrazyTokenizer(pos_emojis=True, neg_emojis=True, neutral_emojis=True)
>>> text = '???!!!!'
>>> tokenizer.tokenize(text)
['POS_EMOJI', 'NEG_EMOJI', 'NEG_EMOJI']
```

You can supply your own lists of emojis as well.

```
>>> tokenizer = CrazyTokenizer(pos_emojis=['', ''], neutral_emojis=[''], remove_
↪punct=False)
>>> text = ' +  = '
>>> tokenizer.tokenize(text)
['POS_EMOJI', '+', 'POS_EMOJI', '=', 'NEUTRAL_EMOJI']
```

4.3.15 Unicode and hex characters

Sometimes your data gets messed up as a result of repeated save/load operations. If your data contains a lot of substrings that look like this: `\\xe2\\x80\\x99` or this: `U+1F601`, try setting `latin_chars_fix=True`.

```
>>> tokenizer = CrazyTokenizer(latin_chars_fix=True)
>>> text = "I\\xe2\\x80\\x99m so annoyed by these characters \\xF0\\x9F\\x98\\xA2"
>>> tokenizer.tokenize(text)
['i', 'm', 'so', 'annoyed', 'by', 'these', 'characters', '']
```

4.3.16 n-grams

You can add n-grams of tokens to the original list of tokens by using options `ngrams`.

```
>>> tokenizer = CrazyTokenizer(ngrams=2)
>>> text = "We need more tokens"
>>> tokenizer.tokenize(text)
['we', 'need', 'more', 'tokens', 'we_need', 'need_more', 'more_tokens']
```

CHAPTER 5

Modelling

You collected your training data, coverted it to the lists of tokens, now you can start training models!

RedditScore currently supports training the following types of models:

- Bag-of-Words models
- fastText model
- MLP, LSTM, CNN neural networks (Keras implementation) - TODO

5.1 Fitting models

All model wrappers have very similar interface:

```
from redditscore import tokenizer
from redditscore.models import fasttext_mod, bow_mod
from sklearn.naive_bayes import MultinomialNB

# reading and tokenizing data
df = pd.read_csv(os.path.join('redditscore', 'reddit_small_sample.csv'))
df = df.sample(frac=1.0, random_state=24) # shuffling data
tokenizer = CrazyTokenizer(splithashtags=True) # initializing tokenizer object
X = df['body'].apply(tokenizer.tokenize) # tokenizing Reddit comments
y = df['subreddit']

fasttext_model = fasttext_mod.FastTextModel(epochs=5, minCount=5)
multi_model = sklearn_mod.BoWModel(estimator=MultinomialNB(alpha=0.1), ngrams=2,␣
↪tfidf=False)

fasttext_model.fit(X, y)
multi_model.fit(X, y)
```

5.2 Model persistence

To save the model, use `save_model` method:

```
>>> fasttext_model.save_model('models/fasttext')
>>> multi_model.save_model('models/multi.pkl')
```

Each module has its own `load_model` function:

```
>>> fasttext_model = fasttext_mod.load_model('models/fasttext')
>>> multi_model = sklearn_mod.load_model('models/multi.pkl')
```

Note: fastText and Keras models are saved into two files with '.pkl' and '.bin' extensions.read

5.3 Predictions and similarity scores

Using models for prediction is very straightforward:

```
>>> pred_labels = fasttext_model.predict(X)
>>> pred_probs = fasttext_model.predict_proba(X)
```

Both `predict` and `predict_proba` return pandas DataFrames with class labels as column names.

5.4 Model tuning and validation

Each model class has `cv_score` and `tune_params` methods. You can use these methods to assess the quality of your model and to perform tuning of hyperparameters.

`cv_score` method has two optional arguments - `cv` and `scoring`. `cv` argument can be:

- float: score will be calculated using a randomly sampled holdout set containing corresponding proportion of `X`
- None: use 3-fold cross-validation
- integer: use n-fold cross-validation
- an object to be used as a cross-validation generator: for example, KFold from sklearn
- an iterable yielding train and test split: for example, list of tuples with train and test indices

`scoring` can be either a string or a scorer callable object.

Examples:

```
>>> fasttext_model.cv_score(X, y, cv=0.2, scoring='accuracy')
>>> fasttext_model.cv_score(X, y, cv=5, scoring='neg_log_loss')
```

`tune_params` can help you to perform grid search over the grid of hyperparameters. In addition to `cv` and `scoring` parameters it also has three additional optional parameters: `param_grid`, `verbose`, and `refit`.

`param_grid` can have a few different structures:

- None: if None, use default step grid for the corresponding model
- dictionary with parameters names as keys and lists of parameter settings as values.

- dictionary with enumerated steps of grid search. Each step has to be a dictionary with parameters names as keys and lists of parameter settings as values.

Examples:

```
>>> fasttext_model.tune_params(X, y)
>>> param_grid = {'epochs': [1,5,10], 'dim': [50,100,300]}
>>> fasttext_model.tune_params(X, y, param_grid=param_grid)
>>> param_grid = {'step0': param_grid, 'step1': {'t': [1e-4, 1e-3, 1e-3]}}
```

If `verbose` is True, messages with grid process results will be printed. If `refit` is True, the model will be refit with the full data after grid search is over.

`tune_params` returns a tuple: dictionary with the best found parameters and the best value of the chosen metric. In addition, original model parameters will be replaced inplace with the best found parameters.

5.5 Visualization of the class embeddings

For fastText and neural network models, you can visualize resulting class embeddings. This might be useful for a couple of reas

- It can be used as an informal way to confirm that the model was able to learn meaningful semantic differences between classes. In particular, classes that one expects to be more semantically similar should have similar class embeddings.
- It can help you to group different classes together. This is particularly useful for building Reddit-based models and calculating RedditScores. There are a lot of different subreddits, but a lot of them are quite similar to each other (say, /r/Conservaitve and /r/republicans). Visualizations can help you to identify similar subreddits, which can be grouped together for improved predictive performance.

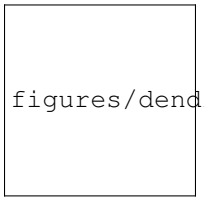

Fig. 1: Dengrogram for class embeddings

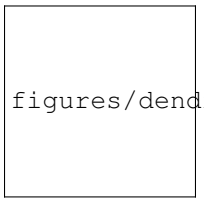

Fig. 2: t-SNE visualization

CHAPTER 6

APIs

6.1 CrazyTokenizer

CrazyTokenizer: spaCy-based tokenizer with Twitter- and Reddit-specific features

Splitting hashtags is based on the idea from https://stackoverflow.com/questions/11576779/how-to-extract-literal-words-from-a-consecutive-string-efficiently

Author: Evgenii Nikitin <e.nikitin@nyu.edu>

Part of https://github.com/crazyfrogspb/RedditScore project

class tokenizer.**CrazyTokenizer**(*lowercase=True*, *keepcaps=False*, *normalize=3*, *ignore_quotes=False*, *ignore_reddit_quotes=False*, *ignore_stopwords=False*, *stem=False*, *remove_punct=True*, *remove_breaks=True*, *decontract=False*, *twitter_handles=False*, *urls=False*, *hashtags=False*, *numbers=False*, *subreddits=False*, *reddit_usernames=False*, *emails=False*, *extra_patterns=None*, *keep_untokenized=None*, *whitespaces_to_underscores=True*, *remove_nonunicode=False*, *pos_emojis=None*, *neg_emojis=None*, *neutral_emojis=None*, *print_url_warnings=False*, *latin_chars_fix=False*, *ngrams=1*)

Tokenizer with Reddit- and Twitter-specific options

Parameters

- **lowercase** (*bool, optional*) – If True, lowercase all tokens. Defaults to True.
- **keepcaps** (*bool, optional*) – If True, keep ALL CAPS WORDS uppercased. Defaults to False.
- **normalize** (*int or bool, optional*) – If not False, perform normalization of repeated charachers ("awesooooooome" -> "awesooome"). The value of parameter determines the number of occurences to keep. Defaults to 3.

- **ignore_quotes** (*bool, optional*) – If True, ignore tokens contained within double quotes. Defaults to False.
- **ignore_reddit_quotes** (*bool, optional*) – If True, remove quotes from the Reddit comments. Defaults to False.
- **ignore_stopwords** (*str, list, or boolean, optional*) – Whether to ignore stopwords
  - str: language to get a list of stopwords for from NLTK package
  - list: list of stopwords to remove
  - True: use built-in list of the english stop words
  - False: keep all tokens

  Defaults to False
- **stem** (*{False, 'stem', 'lemm'}, optional*) – Whether to perform word stemming
  - False: do not perform word stemming
  - 'stem': use PorterStemmer from NLTK package
  - 'lemm': use WordNetLemmatizer from NLTK package
- **remove_punct** (*bool, optional*) – If True, remove punctuation tokens. Defaults to True.
- **remove_breaks** (*bool, optional*) – If True, remove linebreak tokens. Defaults to True.
- **decontract** (*bool, optional*) – If True, attempt to expand certain contractions. Defaults to False. Example: "'ll" -> " will"
- **subreddits, reddit_usernames, emails** (*numbers,*) –
- **or str, optional** (*False*) – Replacement of the different types of tokens
  - False: leaves these tokens intact
  - str: replacement token
  - '': removes all occurrences of these tokens
- **twitter_handles** (*False, 'realname' or str, optional*) – Processing of twitter handles
  - False: do nothing
  - str: replacement token
  - 'realname': replace with the real screen name of Twitter account
  - 'split': split handles using Viterbi algorithm

  Example: "#vladimirputinisthebest" -> "vladimir putin is the best"
- **hashtags** (*False or str, optional*) – Processing of hashtags
  - False: do nothing
  - str: replacement token
  - 'split': split hashtags according using Viterbi algorithm
- **urls** (*False or str, optional*) – Replacement of parsed URLs

- False: leave URL intact
- str: replacement token
- dict: replace all URLs stored in keys with the corresponding values
- ’‘: removes all occurrences of these tokens
- ’domain’: extract domain (“http://cnn.com” -> “cnn”)
- ’domain_unwrap_fast’: extract domain after unwraping links

for a list of URL shorteners (goo.gl, t.co, bit.ly, tinyurl.com) - ‘domain_unwrap’: extract domain after unwraping all links - ‘title’: extract and tokenize title of each link after unwraping it

Defaults to False.

- **extra_patterns** (*None or list of tuples, optional*) – Replacement of any user-supplied extra patterns. Tuples must have the following form: (name, re_pattern, replacement_token):
  - name (str): name of the pattern
  - re_pattern (_sre.SRE_Pattern): compiled re pattern
  - replacement_token (str): replacement token

  Defaults to None
- **keep_untokenized** (*None or list, optional*) – List of expressions to keep untokenized

  Example: [“New York”, “Los Angeles”, “San Francisco”]
- **whitespaces_to_underscores** (*boolean, optional*) – If True, replace all whitespace characters with underscores in the final tokens. Defaults to True.
- **remove_nonunicode** (*boolean, optional*) – If True, remove all non-unicode characters. Defaults to False.
- **neg_emojis, neutral_emojis** (*pos_emojis,*) – Replace positive, negative, and neutral emojis with the special tokens
  - None: do not perform replacement
  - True: perform replacement of the default lists of emojis
  - list: list of emojis to replace
- **print_url_warnings** (*bool, optional*) – If True, print URL-related warnings. Defaults to False.
- **latin_chars_fix** (*bool, optional*) – Try applying this fix if you have a lot of xe2x80x99-like or U+1F601-like strings in your data. Defaults to False.
- **ngrams** (*int, optional*) – Add ngrams of tokens after tokenizing

tokenize(*text*)

Tokenize document

Parameters **text** (*str*) – Document to tokenize

Returns List of tokens

Return type list

6.1.0.0.1 Examples

```
>>> from redditscore.tokenizer import CrazyTokenizer
>>> tokenizer = CrazyTokenizer(splithashtags=True, hashtags=False)
>>> tokenizer.tokenize("#makeamericagreatagain")
["make", "america", "great", "again"]
```

6.2 Models

Popular models for text classification

6.2.1 BoWModel

bow_mod: A wrapper for Bag-of-Words models

Author: Evgenii Nikitin <e.nikitin@nyu.edu>

Part of https://github.com/crazyfrogspb/RedditScore project

class models.bow_mod.**BoWModel**(*estimator*, *ngrams=1*, *tfidf=True*, *random_state=24*)

A wrapper for Bag-of-Words models with or without tf-idf re-weighting

Parameters

- **estimator** (*scikit-learn model*) – Estimator object (classifier or regressor)
- **ngrams** (*int, optional*) – The upper boundary of the range of n-values for different n-grams to be extracted
- **tfidf** (*bool, optional*) – If true, use tf-idf re-weighting
- **random_state** (*integer, optional*) – Random seed
- ****kwargs** – Parameters of the multinomial model. For details check scikit-learn documentation.

params

dict – Dictionary with model parameters

cv_score(*X*, *y*, *cv=0.2*, *scoring='accuracy'*, *k=3*)

Calculate validation score

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels
- **cv** (*float, int, cross-validation generator or an iterable, optional*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - float, to use holdout set of this size
  - None, to use the default 3-fold cross validation,
  - integer, to specify the number of folds in a StratifiedKFold,
  - An object to be used as a cross-validation generator.

– An iterable yielding train, test splits.

- **scoring** (*string, callable or None, optional, optional*) – A string (see sklearn model evaluation documentation) or a scorer callable object or 'top_k_accuracy'
- **k** (*int, optional*) – k parameter for 'top_k_accuracy' scoring

Returns Average value of the validation metrics

Return type float

fit (*X*, *y*)
Fit model

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Fitted model object

Return type RedditModel

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*numpy array of shape [n_samples, n_features]*) – Training set.
- **y** (*numpy array of shape [n_samples]*) – Target values.

Returns **X_new** – Transformed array.

Return type numpy array of shape [n_samples, n_features_new]

get_params (*deep=None*)
Get parameters of the model

Returns Dictionary with model parameters

Return type dict

plot_analytics (*classes=None*, *fig_sizes=((20, 15), (20, 20))*, *linkage_pars=None*, *dendrogram_pars=None*, *clustering_pars=None*, *tsne_pars=None*, *legend_pars=None*, *label_font_size=17*)
Plot hieracical clustering dendrogram and T-SNE visualization based on the learned class embeddings

Parameters

- **classes** (*iter, optional*) – Iterable, contains list of class labels to include to the plots. If None, use all classes
- **fig_sizes** (*tuple of tuples, optional*) – Figure sizes for plots
- **linkage_pars** (*dict, optional*) – Dictionary of parameters for hieracical clustering. (scipy.cluster.hierarchy.linkage)
- **dendrogram_pars** (*dict, optional*) – Dictionary of parameters for plotting dendrogram. (scipy.cluster.hierarchy.dendrogram)
- **clustering_pars** (*dict, optional*) – Dictionary of parameters for producing flat clusters. (scipy.cluster.hierarchy.fcluster)

- **tsne_pars** (*dict, optional*) – Dictionary of parameters for T-SNE. (sklearn.manifold.TSNE)
- **legend_pars** (*dict, optional*) – Dictionary of parameters for legend plotting (matplotlib.pyplot.legend)
- **label_font_size** (*int, optional*) – Font size for the labels on T-SNE plot

predict(*X*)

Predict the most likely label

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Predicted class labels

Return type array, shape (n_samples,)

predict_proba(*X*)

Predict the most likely label

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Predicted class probabilities

Return type array, shape (n_samples, num_classes)

save_model(*filepath*)

Save model to disk.

Parameters **filepath** (*str*) – Path to the file where the model will be sabed.

set_params(***params*)

Set the parameters of the model.

Parameters ****params** (*{'tfidf', 'ngrams', 'random_state'} or*) – parameters of the corresponding models

tune_params(*X*, *y*, *param_grid=None*, *verbose=False*, *cv=0.2*, *scoring='accuracy'*, *k=3*, *refit=False*)

Find the best values of hyperparameters using chosen validation scheme

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels
- **param_grid** (*dict, optional*) – Dictionary with parameters names as keys and lists of parameter settings as values. If None, loads deafult values from JSON file
- **verbose** (*bool, optional*) – If True, print scores after fitting each model
- **cv** (*float, int, cross-validation generator or an iterable, optional*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - float, to use holdout set of this size
  - None, to use the default 3-fold cross validation,

- integer, to specify the number of folds in a StratifiedKFold,
- An object to be used as a cross-validation generator.
- An iterable yielding train, test splits.

- **scoring** (*string, callable or None, optional*) – A string (see sklearn model evaluation documentation) or a scorer callable object or 'top_k_accuracy'
- **k** (*int, optional*) – k parameter for 'top_k_accuracy' scoring
- **refit** (*boolean, optional*) – If True, refit model with the best found parameters

Returns

- **best_pars** (*dict*) – Dictionary with the best combination of parameters
- **best_value** (*float*) – Best value of the chosen metric

models.bow_mod.**load_model**(*filepath*)

Loan pickled instance of SklearnModel.

Parameters **filepath** (*str*) – Path to the pickled model file.

Returns Unpickled model.

Return type SklearnModel

6.2.2 FastTextModel

FastTextModel: A wrapper for Facebook fastText model

Author: Evgenii Nikitin <e.nikitin@nyu.edu>

Part of https://github.com/crazyfrogspb/RedditScore project

class models.fasttext_mod.**FastTextClassifier**(*lr=0.1, dim=100, ws=5, epoch=5, minCount=1, minCountLabel=0, minn=0, maxn=0, neg=5, wordNgrams=1, loss='softmax', bucket=2000000, thread=12, lrUpdateRate=100, t=0.0001, label='__label__', verbose=2*)

get_params(*deep=True*)

Get parameters for this estimator.

Parameters **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

score(*X*, *y*, *sample_weight=None*)

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like, shape = (n_samples, n_features)*) – Test samples.

- **y** (*array-like, shape = (n_samples) or (n_samples, n_outputs)*) – True labels for X.
- **sample_weight** (*array-like, shape = [n_samples], optional*) – Sample weights.

Returns **score** – Mean accuracy of self.predict(X) wrt. y.

Return type float

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns

Return type self

class models.fasttext_mod.**FastTextModel**(*random_state=24*, ***kwargs*)
Facebook fastText classifier

Parameters

- **random_state** (*int, optional*) – Random seed (the default is 24).
- ****kwargs** – Other parameters for fastText model. Full description can be found here: https://github.com/facebookresearch/fastText

cv_score(*X*, *y*, *cv=0.2*, *scoring='accuracy'*, *k=3*)
Calculate validation score

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels
- **cv** (*float, int, cross-validation generator or an iterable, optional*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - float, to use holdout set of this size
  - None, to use the default 3-fold cross validation,
  - integer, to specify the number of folds in a StratifiedKFold,
  - An object to be used as a cross-validation generator.
  - An iterable yielding train, test splits.
- **scoring** (*string, callable or None, optional, optional*) – A string (see sklearn model evaluation documentation) or a scorer callable object or 'top_k_accuracy'
- **k** (*int, optional*) – k parameter for 'top_k_accuracy' scoring

Returns Average value of the validation metrics

Return type float

fit(*X*, *y*)
Fit model

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Fitted model object

Return type *FastTextModel*

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*numpy array of shape [n_samples, n_features]*) – Training set.
- **y** (*numpy array of shape [n_samples]*) – Target values.

Returns **X_new** – Transformed array.

Return type numpy array of shape [n_samples, n_features_new]

get_params(*deep=None*)
Get parameters of the model

Returns Dictionary with model parameters

Return type dict

plot_analytics(*classes=None*, *fig_sizes=((20, 15), (20, 20))*, *linkage_pars=None*, *dendrogram_pars=None*, *clustering_pars=None*, *tsne_pars=None*, *legend_pars=None*, *label_font_size=17*)
Plot hieracical clustering dendrogram and T-SNE visualization based on the learned class embeddings

Parameters

- **classes** (*iter, optional*) – Iterable, contains list of class labels to include to the plots. If None, use all classes
- **fig_sizes** (*tuple of tuples, optional*) – Figure sizes for plots
- **linkage_pars** (*dict, optional*) – Dictionary of parameters for hieracical clustering. (scipy.cluster.hierarchy.linkage)
- **dendrogram_pars** (*dict, optional*) – Dictionary of parameters for plotting dendrogram. (scipy.cluster.hierarchy.dendrogram)
- **clustering_pars** (*dict, optional*) – Dictionary of parameters for producing flat clusters. (scipy.cluster.hierarchy.fcluster)
- **tsne_pars** (*dict, optional*) – Dictionary of parameters for T-SNE. (sklearn.manifold.TSNE)
- **legend_pars** (*dict, optional*) – Dictionary of parameters for legend plotting (matplotlib.pyplot.legend)
- **label_font_size** (*int, optional*) – Font size for the labels on T-SNE plot

predict(*X*)
Predict the most likely label

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents

- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Predicted class labels

Return type array, shape (n_samples,)

predict_proba(*X*)
Predict the most likely label

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels

Returns Predicted class probabilities

Return type array, shape (n_samples, num_classes)

save_model(*filepath*)
Save model to disk.

Parameters **filepath** (*str*) – Path to the file where the model will be saved. NOTE: The model will be saved in two files: with '.pkl' and 'bin' file extensions.

set_params(***params*)
Set parameters of the model

Parameters ****params** – Model parameters to update

tune_params(*X*, *y*, *param_grid=None*, *verbose=False*, *cv=0.2*, *scoring='accuracy'*, *k=3*, *refit=False*)
Find the best values of hyperparameters using chosen validation scheme

Parameters

- **X** (*iterable, shape (n_samples,)*) – Sequence of tokenized documents
- **y** (*iterable, shape (n_samples,)*) – Sequence of labels
- **param_grid** (*dict, optional*) – Dictionary with parameters names as keys and lists of parameter settings as values. If None, loads deafult values from JSON file
- **verbose** (*bool, optional*) – If True, print scores after fitting each model
- **cv** (*float, int, cross-validation generator or an iterable, optional*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - float, to use holdout set of this size
  - None, to use the default 3-fold cross validation,
  - integer, to specify the number of folds in a StratifiedKFold,
  - An object to be used as a cross-validation generator.
  - An iterable yielding train, test splits.
- **scoring** (*string, callable or None, optional*) – A string (see sklearn model evaluation documentation) or a scorer callable object or 'top_k_accuracy'
- **k** (*int, optional*) – k parameter for 'top_k_accuracy' scoring
- **refit** (*boolean, optional*) – If True, refit model with the best found parameters

Returns

- **best_pars** (*dict*) – Dictionary with the best combination of parameters

- **best_value** (*float*) – Best value of the chosen metric

models.fasttext_mod.**load_model**(*filepath*)
Load pickled model.

Parameters **filepath** (*str*) – Path to the file where the model will be saved. NOTE: the directory has to contain two files with provided name: with '.pkl' and 'bin' file extensions.

Returns Unpickled model object.

Return type *FastTextModel*

6.2.3 Neural networks

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[Nikitin2018] Nikitin Evgenii, Identyifing Political Trends on Social Media Using Reddit Data, in progress

[Nikitin2018] Nikitin Evgenii, Identyifing Political Trends on Social Media Using Reddit Data, in progress

Python Module Index

Index

Symbols

B

C

F

G

L

M

P

S

T