cReddit score Documentation

Release 0.1.0

Gautam Sisodia

December 17, 2016

Contents

Contents:

CHAPTER 1

cReddit score

An automatic comment flagger for online forums

- Free software: BSD license
- Documentation: https://cRedditscore.readthedocs.org.

1.1 TODO

- Update tests for make_model to include parameters
- Add test data functionality to Evaluate.compute_scores
- Add tests for Evaluate.compute_scores

CHAPTER 2

cRedditscore package

2.1 Submodules

2.2 cRedditscore.cRedditscore module

A module of tools to train and package predictive models for the quality of comments on reddit.

class cRedditscore.cRedditscore.**TermFreqModel**(*comments_df*, *low_thresh=0*, *high_thresh=15*)

Bases: object

A Naive Bayes model predicting the quality of comments on Reddit. The data is input as a pandas dataframe with columns

- *comment_id*: a unique identifier for each comment
- *score* (int): the score of the comment
- *content*: the comment itself

For example, we make a small comments data set

```
>>> import pandas as pd
>>> test_df = pd.DataFrame([
...     [1, 1, "That's cool", 1],
...     [2, -3, 'boo you', 2],
...     [3, 4, 'I love you', 2],
...     [3, 16, 'I love you', 4],
...     ], columns=['comment_id', 'score', 'content', 'timestamp'])
```

and build a *TermFreqModel* object from it.

```
>>> tfm = TermFreqModel(comments_df = test_df)
```

Parameters

- **comments_df** (*pandas.core.frame.DataFrame*) – the dataframe containing the comment data
- **low_thresh** (*int*) – the lower bound for the score of a neutral comment. Anything lower is considered a bad comment
- **high_thresh** (*int*) – the upper bound for the score of a neutral comment. Anything higher is considered a good comment

add_qual_feature(*df*)
Add the comment quality feature to the data as a new column named *qual*. This will be our outcome variable.

Parameters **df** (*pandas.core.frame.DataFrame*) – the data frame to add the *qual* column to

dump_model(*pickle_name='text_mnb_model'*)
Dump the model object to file with *pickle*.

Parameters **pickle_name** (*string*) – the name of the file to dump the object to

fit()
Fit the model.

get_data()
Get the full data set of the model.

Returns the full data set underlying the model

Return type pandas.core.frame.DataFrame

get_good_bad(*df*)
Get the good and bad comments in a data set.

Parameters **df** (*pandas.core.frame.DataFrame*) – the comments data set

Returns the dataframes containing only the good and bad comments from *df*

Return type pandas.core.frame.DataFrame, pandas.core.frame.DataFrame

make_model(*test_size=0.2*, *ngram_range=(1*, *4)*, *max_features=1000*)
Make a new class attribute

- ***model*:** the Naive Bayes model as an *sklearn.pipeline.Pipeline* object

For example, we make a small comments data set,

```
>>> import pandas as pd
>>> test_df = pd.DataFrame([
...      [1, 1, "That's cool", 1],
...      [2, -3, 'boo you', 2],
...      [3, 4, 'I love you', 2],
...      [3, 16, 'I love you', 4],
...      ], columns=['comment_id', 'score', 'content', 'timestamp'])
```

build a *TermFreqModel* object from it,

```
>>> tfm = TermFreqModel(test_df)
```

and train a model on it predictive of the quality of a comment.

```
>>> tfm.train_test(test_size=0.1)
>>> tfm.make_model(ngram_range=(1, 3), max_features=10)
>>> tfm.fit()
>>> prediction = tfm.model.predict(['Thanks for a great post!'])
>>> prediction in ['good', 'bad']
True
```

Parameters

- **test_size** (*int*) – the percentage of data points to hold out for testing
- **ngram_range** (*tuple*) – the range of n for ngrams to include as features

- **max_features** (*int*) – the maximum number of features to include

most_recent_obs(*df*)

Select the most recent observation of each comment in a data set.

Parameters **df** (*pandas.core.frame.DataFrame*) – The data set of comments

Returns the data set containing only the most recent observation of each comment in *df*

Return type pandas.core.frame.DataFrame

setup_data()

Set up the data for model training. In detail:

•Remove all but the most recent observation for each comment

•Add the quality feature to the data, which will be our outcome variable

•Separate out the good and bad comments. This will be the data we train the model on.

This function adds a new class attribute

•***good_bad_df*:** the dataframe containing only the most recent observations of the good and bad comments

train_test(*test_size=0.2*)

Split the data into train and test sets.

This function adds new class attributes

•***X_train* and *X_test*,** The features of the training and test parts of the data set

•***y_train* and *y_test*,** The outcomes of the training and test parts of the data set

Parameters **test_size** (*int*) – the percentage of data points to hold out for testing

cRedditscore.cRedditscore.**get_quality**(*score*, *low_thresh=0*, *high_thresh=15*)

Get the quality (good, bad, neutral) of a score based on the score thresholds.

For example,

```
>>> get_quality(score=15)
'neutral'
>>> get_quality(score=2, low_thresh=5, high_thresh=20)
'bad'
>>> get_quality(score=2, low_thresh=-10, high_thresh=1)
'good'
```

Parameters

- **score** (*int*) – the score of the comment
- **low_thresh** (*int*) – the low threshold of a neutral comment
- **high_thresh** (*int*) – the high threshold of a neutral comment

Returns the quality of the comment (good, bad or neutral)

Return type string

2.3 cRedditscore.collection module

A module of tools to collect comment data from Reddit.

class cRedditscore.collection.**Collect**(*user_agent=None, subreddits=['dataisbeautiful', 'programming', 'technology', 'python', 'cpp', 'funny', 'news', 'science'], subm_table=None, comm_table=None, conn=None, debug=False*)

Bases: object

A class to handle the collecting and storing of comment data from Reddit.

In practice, *collection* requires the name of a Reddit web app to connect to the Reddit api. By default, no web app is passed to the class, in which case the class is useless.

The database for storing comments has a table *subm_table* for storing submission data with columns

- *submission_id* (varchar(45)),
- *subreddit* (varchar(45)), and
- *timestamp* (int(11))

and a table *comm_table* for storing comment data with columns

- *submission_id* (varchar(45)),
- *subm_title* (varchar(45)),
- *subm_content* (blob),
- *subm_created* (int(11)),
- *subm_created_local* (int(11)),
- *subm_score* (int(11)),
- *subm_author* (varchar(45)),
- *subm_num_comments* (int(11)),
- *comment_id* (varchar(45)),
- *user_id* (varchar(45)),
- *prev_comment_id* (varchar(45)),
- *created* (int(11)),
- *created_local* (int(11)),
- *timestamp* (int(11)),
- *content* (blob),
- *subreddit* (varchar(45)),
- *score* (int(11)),
- *ups* (int(11)),
- *downs* (int(11)), and
- *controversiality* (int(11)).

Parameters

- **user_agent** (*string*) – the name of the web app to connect to the api through

- **subreddits** (*list*) – the list of subreddits to collect from
- **subm_table** (*sqlalchemy.sql.schema.Table*) – the submissions sql table
- **comm_table** (*sqlalchemy.sql.schema.Table*) – the comments sql table
- **conn** (*sqlalchemy.engine.base.Connection*) – the connection to the sql database

For example,

```
>>> col = Collect()
>>> col.subreddits[0]
'dataisbeautiful'
```

comment_to_db(*comment=None*, *subm_values=None*)
Add *comment* to the database.

fetch_subm_from_table(*subm*)
Fetch the rows corresponding to *subm* in *self.subm_table*.

Returns the rows in *self.subm_table* that come from *subm*

Return type list

get_random_subm()
Get a random submission from Reddit.

Returns a random submission from a random subreddit in *self.subreddits* if *self.red* exists, else None

Return type praw.objects.Submission

rand_subm_to_db()
Add a random submission to the database.

subm_to_db(*subm*)
Add *subm* to the database with all of its comments.

2.4 cRedditscore.evaluation module

A module of tools to evaluate predictive models.

exception cRedditscore.evaluation.**EvalError**(*msg*)
Bases: exceptions.Exception

Errors in evaluating the model, for example a missing predict function.

class cRedditscore.evaluation.**Evaluate**(*model=None*, *data_features=None*, *data_responses=None*, *pos_label=None*)
Bases: object

Evaluate a predictive binary classification model on a choice of metrics including accuracy, AUC, precision and recall. The model is assumed to have functions

- *fit* (for cross validation): takes as input the training set features and responses and fits the model to the training set
- *predict* (for accuracy, precision, recall and cross validation): takes as input a list of observations and outputs a list of predictions

•*predict_proba* (for AUC and drawing the ROC curve): takes as input a list of observations and outputs a list of probabilities that the response belongs to the first class

For example, we make a dummy test set and model:

```
>>> model = GenModel(predict = lambda x : ['blue' for i in x])
>>> test_features = range(20)
>>> test_responses = np.array(
...     ['blue' if i%2==0 else 'green' for i in range(20)]
...     )
```

and make an *Evaluate* object to test its accuracy.

```
>>> eval = Evaluate(model)
>>> eval.accuracy(test_features=test_features,
...     test_responses=test_responses)
0.5
```

Parameters

- **model** – the model to evaluate, described above
- **data_features** (*array-like*) – the features of the data to evaluate the model on, generally the training set features
- **data_responses** (*array-like*) – the responses of the data to evaluate the model on
- **pos_label** – the class to be considered positive for auc, precision and recall; if None, the first class is picked

accuracy(*test_responses*, *test_features=None*, *predictions=None*, *train=None*)

Find the accuracy of the model on a given test set.

Parameters

- **test_responses** (*array-like*) – the responses of the test set
- **test_features** (*array-like*) – the features of the test set to predict on. If None, use the pre-made *predictions*
- **predictions** (*array-like*) – the predictions to test. If None, use *test_features* to predict on
- **train** (*array-like*) – the data set to train the model on (optional)

build_scores_df()

Build the scores dataframe.

compute_curves()

Plot ROC and precision recall curves.

compute_scores(*test_features=None, test_responses=None, metrics=['accuracy', 'auc', 'precision', 'recall', 'f1']*)

Compute the metric scores for the model on the cross-validation folds of the training data or on the test data, storing the results in a dataframe.

Add a new class attribute

•***scores***, the results of the computations as a *pandas.core.frame.DataFrame* object

Parameters

- **test_features** (*array-like*) – The features of the test data. If none, evaluate the model on the cv folds.

- **test_responses** (*array-like*) – The responses of the test data. If none, evaluate the model on the cv folds.

cv_split(*k=10*)
Make k folds of the data using stratified cross validation.

Parameters **k** (*int*) – The number of folds to divide the data into

class cRedditscore.evaluation.**GenModel**(*fit=None*, *predict=None*, *predict_proba=None*)
Bases: object

A general (binary classification) model class, mostly for testing and explanatory purposes.

Parameters

- **fit** (*function*) – the fit function of the model
- **predict** (*function*) – the prediction function of the model
- **predict_proba** (*function*) – the probability prediction function of the model, computes the probability that an observation belongs to the first class

2.5 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

C

A

B

C

D

E

F

G

M

R

S

T