HASHTAG-DRIVEN AGGRESSION DETECTION ON TWITTER

VOLKER STROBEL

Text Mining, Radboud University Nijmegen

s4491491

v.strobel@student.ru.nl

January 24, 2015

Abstract

Hashtags on microblogging platforms can give strong cues for the category of a message. In this paper, a categorization approach, known as distant supervision, is used to classify a training set with respect to certain hashtags. Using this hashtag-driven approach, a data set of 714 Dutch tweets was split into aggressive and non-aggressive messages. In this paper, the F_1 scores of combinations of feature extraction methods (bag-of-words, linguistic markers, swear words) are compared to find a suitable combination. The best classifier (tested on manually annotated tweets) using distant supervision reached a F_1 score of 73 %.

1 INTRODUCTION

Annotating data for text classification is a labor-intensive task and, therefore, most of the time, only a couple of hundred or thousand of items can be categorized. However, hashtags on Twitter can function as categories, labeled by the users themselves. This can be conveniently utilized for text categorization purposes: using a 'hashtag-driven distant supervision' approach the training data can a split into different categories by means of their hashtags. This dataset is then used to train a classifier. This gives the advantage of drastically reducing the time for the labour-intensive task of manually labeling the training data: instead incoming tweets can increase the training data set without effort. Once trained, arbitrary tweets (i.e., Twitter messages) can be classified. Another advantage is, that the users themselves have labeled the data, so that inter-rater reliability or ambiguity is of little concern. However, this approach is inherently noisy and a particular challenge consists in choosing the right hashtags resulting in a good data set.

In this paper a hashtag-driven distant supervision approach for classifying aggressive and non-aggressive tweets will be used. I will show the validity of the used classifier, trained on a manually selected set of hashtags, in a combinatorial manner by combining bag-of-word, linguistic marker, and swear word extraction methods.

In Section 2, I will present related work with respect to distant supervision, text classification and aggression detection. Section 3 discusses details of the logistic

regression classifier, the considered hashtags and the set of evaluation approaches. The results are shown in Section 4 and discussed in terms of the evaluation criteria in Section 5. General conclusions and future work for obtaining higher accuracy scores are given in 6.

2 RELATED WORK

Since the popularity of microblogging platforms and social networking services, the amount of user-generated data and, parallely the interest in utilizing it, has strongly increased.

Distant supervision is the 'automated' labeling of a training data set by means of a marker in [1]. Several studies examined the usefulness of distant supervision on Twitter. While most of them used hashtags only [2], some studies have been compared or combined hashtags and emoticons – and both approaches have been proven useful [3, 4]. Mohammad [5] shows that, in general, emotional hashtags in tweets have a high inter-rater consistency. Emotional hashtags can have two different functions [5, 2]: They can be used as tool for strengthening the emotion of a hashtag (e.g., *Fuck you! #hate*) but can also be used to add an emotion to an otherwise neutral or ambiguous statement (e.g., *I have school tomorrow. #shit)*. In order to successfully train a classifier, hashtags of the first category are needed. Kunneman, Liebrecht, and Bosch [2] present a way for objectively determining the two usages of emotion hashtags. In contrast, for the present paper, a predefined set of hashtags has been used. In English, swearing on Twitter is primarily related to sadness and anger [6]. Consistent with that, the used hashtags for the distant supervision approach in the present paper are primarily based on swear words.

3 METHODS

3.1 *Selection of Hashtags*

A crucial step in hashtag-driven distant supervision is the selection of aggression-bearing hashtags. To this end, the Wikipedia entry about 'Dutch profanity' has been studied[1] and the most prominent hashtags of aggressive tweets in a manually labeled data set, consisting of 3150 tweets, were examined. A total of eight profanity words (*#fuckyou, #idioot, #idiot, #sukkel, #gvd, #grr, #grrr, #kut, #bek, #asshole*), and two onomatopoeia (*#grr, #grrr*) have been selected. Since many hashtags can be ambivalent (e.g., *#wtf*) or used in different context (e.g., *#kanker* as an expletive or for a topic related to cancer, only a small set of hashtags was selected.

3.2 *Data*

The training data (a total of 714 tweets) has been obtained on January 23, 2015 using the Twitter Rest API in two steps:

[1] `http://en.wikipedia.org/wiki/Dutch_profanity`, visited on 17 January, 2014.

aggressive	
hashtag	frequency
#grr	66
#kut	60
#grrr	53
#idioot	42
#sukkel	42
#fuckyou	18
#idiot	12
#bek	5

Table 1: Frequency distribution of hashtags in the aggressive data set

1. An aggressive set, obtained by a search query that ensured that all tweets with at least one of the hashtags in the aggressive list are retrieved.

2. A control set that contained none of the hashtags in the aggressive list.

Afterwards, meta data (e.g., date, author) have been striped off of the tweets, so that they solely consisted of their text messages. Table 1 represents the frequencies of the hashtag in the aggressive set and the frequencies of the 7 most frequent hashtags in the control set.

Another data set was, consisting of 3150 tweets that were categorized by human annotators over time, was at my disposal. It consisted of an equal part of aggressive and non-aggressive messages.

3.3 *Feature Extraction*

The classifier used combinations of feature extraction methods: besides the features from the bag-of-words model, it used linguistic markers and swear words as emotional cues. Dependening on the used feature set, the message of each tweet was processed in parallelly to obtain the features:

3.3.1 *Bag-of-words feature extraction*

Each message was tokenized and lowercased using `Ucto` [7]. To reduce the feature space, user names (beginning with an @-sign) and web links were changed to dummy variables[2]. Hash signs of hashtags were removed, so that a word and its related hashtag represented the same feature. Subsequently, a bag-of-words model on unigram level was used.

[2] The same had already been done with the manually annotated data set.

3.3.2 *Linguistic Marker feature Extraction*

Due to the restricted length of tweets (140 characters) and the missing markers of spoken text, users may use textual cues to express aggressiveness. For this reason, the following markers have been extracted for each message:

CAPITALIZATION The amount of capitalized letters

CAPITALIZATION RATIO The amount of capitalized letters divided by the amount of characters

EXCLAMATION MARKS The amount of exlamation marks

EXCLAMATION MARK RATIO The amount of exlamation marks divided by the amount of characters

ADDRESSED USERS The amount of addressed user in this tweet

3.3.3 *Swear Words Feature Extraction*

A collection of 117 Dutch swear words has been created based on the Wikipedia entry about 'Dutch profanity'. For each tweet, the following features have been extracted:

SWEAR WORDS The amount of swear words

SWEAR WORDS RATIO The amount of swear words divided by the amount of words

3.3.4 *Implementation of the Classifiers*

A logistic regression classifier was implemented using Python 2.7 and the package *scikit-learn* [8]. The classifier used a L2 penalty norm. For each tweet, a count matrix for each feature was created and normalized using a tf-idf (term frequency - inverse document frequency) representation.

3.4 *Evaluation*

The evaluation used a combinatorial approach to detect the usefulness of feature sets. Since both a manually labeled set, and a distantly supervised training set were at hand, four different categorization tasks have been performed:

1. Cross-validation on manually labeled data set
2. Cross-validation on distantly supervised data set
3. Validation of distant supervision using manually labeled data set as training set
4. Validation of manually labeled data set using distant supervision as training set

This way, it is validated, that (1) the classifier is able to reliably categorize aggressive and non-aggressive tweets, (2) that a consistent data set has been obtained that shares features with the manually annotated set, (3) and (4) that the distantly supervised

data set is actually useful for classifying data (given that the human categorization is correct).

For each categorization all combinations of the the bag-of-words feature extraction (BOW), linguistic marker feature extraction (LM), and swear words feature extraction (SW) have been used.

For the cross-validations, the data sets were randomly split into 10 folds. Since the data sets and every fold consisted of an equal amount of aggressive and non-aggressive tweets, chance level for classification was 50, %. A leave-one-out cross-validation was conducted on fold-level. To this end, each of the ten folds served as a test set once, and the other nine folds as training sets. The words included in the search query to obtain the tweets for the distantly supervised training set were treated as stopwords (and therefore, excluded from the BOW model).

3.4.1 *Evaluation Metrics*

The performance of the classifier was evaluated using precision (p), recall (r), and their harmonic mean, the F_1 score [9]. In the evaluation, *positive* is the label for *aggressive* and *negative* for *'non-aggressive*. The F_1 score is a prominent evaluation metric in text classification. Since the acquired data sets are balanced in terms of categories, it reflects a convenient measurement for respecting both the precision, and the recall of the data, in contrast to the accuracy. The scores are calculated and reported for the positive category 'aggressive'.

4 RESULTS

The results of the validations can be found in Tables 2, 3, 4, and 5. The best score per column (i.e., precision, recall, F_1) is highlighted. The abbreviations stand for bag-of-words (BOW), linguistic markers (LM), and swear words (SW), precision (p), and (*recall*).

The cross-validation of the manually labeled tweets is listed in Table 2. The best F_1 score is achieved by using the BOW model (independently of taking LM or SW into account). LW is just above change level for F_1. SW alone performs at a rather higher level.

In Table 3, the results of the cross-validation for the distantly supervised tweets are displayed. Here, the combination of BOW and SW performs best, and at a high level, for the three evaluation metrics. Similar pattern can be seen for the other combinations. Only using LW alone results in a F_1 score around change level. It is important to note again, that the hashtags that have been used to acquire the tweets, were treated as stopwords for the cross-validation and therefore, excluded from the BOW model.

Table 4 depicts the results of the evaluation of a classifier that was trained on a distantly supervised training set. The test set was annotated by humans. It seems that the rather higher F_1 is mainly due to the contributions of SW. LW alone performs under chance level, with $F_1 = 0.37$.

The scores for a classifer that was trained on a manually annotated data set are shown in Table 5. For the test set, the distantly supervised data set was used. Here, the contributions of SW were important again. The BOW model perform only

Crossvalidation: Manual			
Feature Selection Method	p	r	F_1
BOW	0.81	**0.80**	**0.81**
BOW + LM	0.79	0.80	0.80
BOW + SW	**0.83**	0.79	0.81
BOW + LM + SW	0.82	0.79	0.81
LM	0.54	0.56	0.55
LM + SW	0.71	0.74	0.73
SW	0.71	0.75	0.73

Table 2: Cross-validation for manually annotated data set. The best F_1 score is written in **bold**. If multiple scores have the same value, the one with less features is marked.
(BOW = bag-of-words, LM = linguistic features, SW = Swear words, p = precision, r = recall)

Crossvalidation: Distant			
Feature Selection Method	p	r	F_1
BOW	0.77	0.85	0.81
BOW + LM	0.75	0.80	0.77
BOW + SW	**0.87**	**0.88**	**0.88**
BOW + LM + SW	0.87	0.88	0.88
LM	0.41	0.60	0.48
LM + SW	0.78	0.86	0.82
SW	0.77	0.86	0.81

Table 3: Cross-validation for distantly supervised data set.

slightly above chance, however with a high r. LW performed better than in the other evaluations.

5 DISCUSSION

The combinatorial approach that was used for the evaluation, depicts the pros and cons of the classifiers and their combinations. The results show that high accuracy measurements are achieved when using the bag-of-unigrams approach on supervised learning with manually annotated data. For distantly supervised data, in particular the use of swear words (SW) features increased the F_1 scores (expect for the cross-validation of the manually annotated tweets). This could be due to small amount of tweets that were collected (714 tweets in total). With the small corpus, the classifier with the bag-of-words approach could not suffienctly learn the frequencies of swear

Train: Distant → Test: Manual			
Feature Selection Method	p	r	F_1
BOW	0.57	0.65	0.61
BOW + LM	0.64	0.60	0.62
BOW + SW	0.73	0.73	**0.73**
BOW + LM + SW	0.73	0.73	0.73
LM	0.31	0.46	0.37
LM + SW	0.64	**0.75**	0.69
SW	**0.73**	0.72	0.72

Table 4: Validation for a manually labeled data set, trained on a distantly supervised data set.

Train: Manual → Test: Distant			
Feature Selection Method	p	r	F_1
BOW	0.39	**0.95**	0.56
BOW + LM	0.38	0.92	0.54
BOW + SW	0.71	0.93	0.81
BOW + LM + SW	0.69	0.93	0.80
LM	0.71	0.47	0.56
LM + SW	0.52	0.83	0.64
SW	**0.77**	0.85	**0.81**

Table 5: Validation for a for distantly supervised data set, trained on a manually labeled data set.

words in aggression-bearing tweets. The support of a predefined word list compensates for this.

The linguistic markers did not contribute to higher scores on the evaluation criteria. They rather performed at chance level in all evaluations. This could be due to markers being used in a similar fashion for aggressive and non-aggressive statements.

The cross-validation, on both the manually labeled set and the distantly supervised set, yields high F_1 values (0.81 and 0.88, respectively). Additionally, similar patterns in their evaluation scores can be found. This indicates a properly chosen set of hashtags for the distant supervision approach.

The F_1 value of 0.73 for classifying manually annotated tweets with a distantly supervised training set is a further indication for the validity of distant supervision. This must be seen in the context of the small corpus that was collected. This could indicate that the majority of Dutch-speaking users rarely use hashtags to express aggression or they do not express aggression to a greater degree in general.

In this paper, a comparison of different feature unions has been conducted in a combinatorial approach. With a F_1 score of 73 % (using the bag-of-words approach and swear word features), distant supervision offers a suitable alternative for the labour-intensive tak of manually assigning categories to data.

Future research offers manifold opportunities to further increase the scores of the evaluation criteria. The obtained data set (714 tweets) is small, since the Twitter API limits searches to the last seven days. A first step would be to collect more data by continuously filtering incoming tweets over a longer period of time. Additionally, the list of swear words could be further extended by intentional misspellings of swear words (e.g., *biatch, b!tch*). The choice of suitable tweets has been particularly difficult, as many swear words have different meanings in different contexts or have not been used in the last seven days (Twitter's time restriction for searching in its history). Using Kunneman, Liebrecht, and Bosch [2]'s approach could help to restrict the search query to significant hashtags. Finally, more feature extraction methods could be used: gender or time of post would give additional clues, in order to reliably classify tweets.

REFERENCES

[1] Mike Mintz et al. "Distant Supervision for Relation Extraction without Labeled Data". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics. 2009, pp. 1003–1011.

[2] FA Kunneman, CC Liebrecht, and APJ van den Bosch. "The (Un) Predictability of Emotional Hashtags in Twitter". In: *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM), EACL 2014*. Association for Computational Linguistics, 2014, pp. 26–34.

[3] Matthew Purver and Stuart Battersby. "Experimenting with Distant Supervision for Emotion Classification". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2012, pp. 482–491.

[4] Dmitry Davidov, Oren Tsur, and Ari Rappoport. "Enhanced Sentiment Learning using Twitter Hashtags and Smileys". In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics. 2010, pp. 241–249.

[5] Saif M Mohammad. "# Emotional tweets". In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics. 2012, pp. 246–255.

[6] Wenbo Wang et al. "Cursing in English on Twitter". In: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM. 2014, pp. 415–425.

[7] Maarten van Gompel, Ko van der Sloot, and Antal van den Bosch. *Ucto: Unicode Tokeniser*. Tech. rep. Citeseer, 2012.

[8] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Yiming Yang and Xin Liu. "A Re-Examination of Text Categorization Methods". In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1999, pp. 42–49.

A CODE

The code is hosted at GitHub and can also be found at the end of this document:
`https://github.com/Pold87/hashpy`

1 Import

In [252]:
```
# General
import os, csv, time
import numpy as np
import pandas as pd

# Twitter API
import tweepy
from tweepy.streaming import StreamListener
from tweepy import Stream

# unicode tokenizer
import ucto

# scikit-learn for machine learning algortihms
from sklearn.base import BaseEstimator
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import FeatureUnion
from sklearn.cross_validation import LeaveOneOut
from sklearn import cross_validation

from sklearn.feature_selection import chi2

from random import shuffle

# Saving and Reading Python objects
import pickle

# import logging # not sure
```

1.1 Settings

In [253]:
```
os.chdir("/home/pold/Dropbox/Uni/Radboud/Text_Mining/Endterm/hashpy")
```

1.2 General Classifier

1.3 Read Aggressive Tweets from TSV files

In [254]:
```
def create_key(x, path = "tweets_better/aggressive_quest"):
    '''Create a file name from a number'''
    return os.path.join(path, "fold_" + str(x) + ".txt")

# Create a list of file names of the manually labeled tweets.
files = [create_key(x) for x in xrange(0,10)]

# Store everything in a pandas data frame.
list = []
for file in files:
    df = pd.read_table(file.encode('utf-8'),
```

```
                                names = ('category', 'user', 'date', 'time', 'message'),
                                header = None,
                                index_col = None)
            list.append(df)

        frame = pd.concat(list, axis=0, keys = files)
```

1.4 Use Ucto for Dutch tokenization

```
In [255]: def remove_non_ascii(s):
              return "".join(i for i in s if ord(i) < 128)

          # Ucto settings
          settingsfile = "/etc/ucto/tokconfig-nl-twitter"
          tokenizer = ucto.Tokenizer(settingsfile)

          def ucto_tokenizer(str):
              '''Tokenize string'''
              remove_non_ascii(str)
              tokenizer.process(unicode(str))
              tokens = [unicode(token).encode('utf-8') for token in tokenizer]
              for pos, token in enumerate(tokens):
                  # Create dummy values for users and urls.
                  if token.startswith('http:'): tokens[pos] = 'url'
                  if token.startswith('@'): tokens[pos] = 'user'
              return tokens
```

1.5 Aggressive Hashtags - Wordlist

```
In [256]: # List a selected hashtags
          with open("wordlists/wordlist_selection.txt") as f:
              selected_hashtags = f.read().splitlines()
```

2 Twitter

2.1 Authentication

```
In []: consumer_key = 'zdprSqld7JxkSjaeqQppemW5y'
       consumer_secret = ''
       access_token = '1612711080-64a9w2e4BOTV4id4HTqLpzaO8Z6ag0pNVE9axZv'
       access_token_secret = ''
```

2.2 Connect to Twitter

```
In [258]: auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
          auth.set_access_token(access_token, access_token_secret)
          api = tweepy.API(auth)
```

2.3 Create a Query

```
In [236]: def create_query(wordlist):
              '''Create a string for an OR query on Twitter'''
              return " OR ".join(wordlist)
```

```
    def create_negate_query(wordlist):
        '''Create a string for a negated query on Twitter'''
        return " ".join(["-" + word for word in wordlist])
```

2.4 Collect Tweets

```
In []: results_aggressive = []
       results_non_aggressive = []

       # Create queries
       query_pos = create_query(selected_hashtags)
       query_neg = create_negate_query(selected_hashtags)

       # Connect to Twitter and collect aggressive tweets
       for tweet in tweepy.Cursor(api.search, q = query_pos,  lang = "nl").items(1500):
           results_aggressive.append(tweet)

       # Wait some time to get not rate limited
       time.sleep(60)

       # Same for non-aggressive Tweets (equal amount to aggressive tweets)
       for tweet in tweepy.Cursor(api.search
                                  , q = query_neg
                                  ,  lang = "nl").items(len(results_aggressive)):
           results_non_aggressive.append(tweet)
```

2.5 Count Hashtags

```
In [259]: hash_dict = { }

          # Initialize dictionary
          for word in selected_hashtags:
              hash_dict[word] = 0

          # Count occurances of hashtags
          for tweet in results_aggressive:
              for word in selected_hashtags:
                  if word in tweet.text: hash_dict[word] += 1

          # Print collected same stats about collected tweets
          #print("Amount of aggressive tweets:", len(results_aggressive))
          #print("Amount of non-aggressive tweets:", len(results_non_aggressive))
          #print("Hashtags frequencies:")
          #print(hash_dict)
```

2.6 Additional Feature Extractor (Linguistic Markers)

```
In [260]: # Transformer for extracting linguistic features in the message
          # Inspired by (and uses code of) Andreas Mueller:
          # https://github.com/amueller/kaggle_insults/
          class LinguisticMarkers(BaseEstimator):

              def fit(self, documents, y=None):
                  return self
```

```
        def transform(self, documents):

            n_words = [len(ucto_tokenizer(remove_non_ascii(tweet))) for tweet in documents]
            n_chars = [len(tweet) for tweet in documents]

            # number of uppercase words
            allcaps = [np.sum([w.isupper() for w in comment.split()])
                   for comment in documents]

            allcaps_ratio = (np.array(allcaps) / np.array(n_chars, dtype=np.float))

            # Total number of exlamation marks
            exclamation = [c.count("!") for c in documents]

            exclamation_ratio = (np.array(exclamation) / np.array(n_chars, dtype=np.float))

            # Total number of addressed users
            users = [c.count("@") for c in documents]

            return np.array([allcaps
                          , allcaps_ratio
                          , exclamation
                          , exclamation_ratio
                          , users
    ]).T
```

2.7 Additional Features (Swear Words)

```
In [261]: # Transformer for extracting swear words in the message
          # Inspired by (and uses code of) Andreas Mueller:
          # https://github.com/amueller/kaggle_insults/
          class SwearWords(BaseEstimator):

              def __init__(self):
                  with open("wordlists/swearwords-nl.txt") as f:
                      swearwords = [l.strip() for l in f.readlines()]
                  self.swearwords_ = swearwords

              def fit(self, documents, y=None):
                  return self

              def transform(self, documents):

                  # number of words
                  n_words = [len(ucto_tokenizer(remove_non_ascii(tweet))) for tweet in documents]

                  # number of swear words:
                  n_swearwords = [np.sum([c.lower().count(w) for w in self.swearwords_])
                                                          for c in documents]

                  swearwords_ratio = np.array(n_swearwords) / np.array(n_words, dtype=np.float)

                  return np.array([n_swearwords
```

```
                                    , swearwords_ratio]).T
```

2.8 k-Fold Cross Validation

```
In [262]: def leave_one_out_evaluation(dataset
                                       , testset = None
                                       , pipe = 0
                                       , add_selection_to_stopwords = False):

              if add_selection_to_stopwords:
                  stopwords_path_to_file = "wordlists/dutch-stop-words-added.txt"
              else:
                  stopwords_path_to_file = "wordlists/dutch-stop-words.txt"

              with open(stopwords_path_to_file) as f:
                  dutch_stop_words = f.read().splitlines()

              # Feature Extractors
              count_vect = CountVectorizer(stop_words = dutch_stop_words
                                           , tokenizer = ucto_tokenizer
                                           , lowercase = True)

              linguisticmarkers = LinguisticMarkers()
              swearwords = SwearWords()

              # Combined features
              combined0 = count_vect

              combined1 = FeatureUnion([("feat1", count_vect)
                                       , ("feat2", linguisticmarkers)])

              combined2 = FeatureUnion([("feat1", count_vect)
                                       , ("feat3", swearwords)])

              combined3 = FeatureUnion([("feat1", count_vect)
                                       , ("feat2", linguisticmarkers)
                                       , ("feat3", swearwords)])



              combined4 = linguisticmarkers
              combined5 = FeatureUnion([("feat2", linguisticmarkers)
                                       , ("feat3", swearwords)])

              combined6 = swearwords


              pipe_list = [combined0, combined1, combined2
                         , combined3, combined4, combined5, combined6]

              tfidf = TfidfTransformer(sublinear_tf = True
                                       , use_idf = True)
```

```
# Create classifier
clf = LogisticRegression(penalty = 'l2', dual = False)


# Create pipelines
pipeline = Pipeline([('vect', pipe_list[pipe]),
                     ('tfidf', tfidf),
                     ('clf', clf)])

# Create collectors
predicted_per_fold = []
target_per_fold = pd.Series()

# If no test set is given, train the classifier do a k-fold cross-validation.
if testset is None:

    # Create random test and fold sets
    kf = cross_validation.KFold(n=len(dataset), n_folds=10, shuffle=True)

    for train_num, test_num in kf:

        messages_train = dataset['message'][train_num]
        categories_train = dataset['category'][train_num]

        messages_test = dataset['message'][test_num]
        categories_test = dataset['category'][test_num]


        target_per_fold = pd.concat([target_per_fold, categories_test])

        # train classifier

        # Remember to chose combined 1, 2 or 3 in Pipeline
        pipeline = pipeline.fit(messages_train, categories_train)

        predicted = pipeline.predict(messages_test)
        predicted_per_fold.append(predicted)

    # flatten predicted_per_fold
    predicted_flattened = [item for sublist in predicted_per_fold for item in sublist]
    return (np.array(predicted_flattened), target_per_fold)


# If a test set is available, train the classifier on the dataset and
# evaluate it on the test set
else:

    # Remember to chose combined 1, 2 or 3 in Pipeline
    pipeline = pipeline.fit(dataset['message'], dataset['category'])

    predicted = pipeline.predict(testset['message'])

    return (predicted, testset['category'])
```

3 Evaluation

```
In [269]: def labels2ints(ls):
              '''Convert aggressive to 1 and non-aggressive to 0'''
              out = []
              for label in ls:
                  if label == 'aggressive': out.append(1)
                  else: out.append(0)
              return out

          def evaluation(tpf, ppf):
              ''' Perform the complete evaluation'''
              predicted_per_fold_array = np.array(ppf)

              target_int = labels2ints(tpf)
              predicted_int = labels2ints(predicted_per_fold_array)

              #print("Accuracy")
              #print(metrics.accuracy_score(target_int, predicted_int))

              #print("Classic Report")
              #print(metrics.classification_report(target_int, predicted_int))
```

3.1 Run Evaluation of manually labeled data set

3.2 Run Cross-validation with function (distantly supervised)

```
In []: # Extract text from tweets
       aggressive_train =  [tweet.text for tweet in results_aggressive]
       non_aggressive_train = [tweet.text for tweet in results_non_aggressive]

       messages_train = aggressive_train + non_aggressive_train

       categories_train = ['aggressive'] * len(aggressive_train) \
                        + ['non_aggressive'] * len(non_aggressive_train)

       distant_frame = pd.DataFrame()
       distant_frame['message'] = messages_train
       distant_frame['category'] = categories_train

       for i in xrange(0,7):
           # print "i is", i
           # Pay attention to add_selection_to_stopwords!
           predicted_per_fold, target_per_fold = \
           leave_one_out_evaluation(frame
                                    , distant_frame
                                    , pipe = i
                                    , add_selection_to_stopwords = False)
           evaluation(predicted_per_fold, target_per_fold)
```

3.3 Validation: Distant Supervision -> Manually Labeled

3.4 Interactive test for getting a feeling for the accuracy (using the manually labeled data)

```
In []: interactive_test = [raw_input('Please type message that should be categorized: ')]

       train = frame.ix[[create_key(y) for y in range(0,10)]]

       messages_train = train['message']
       categories_train = train['category']

       # train classifier
       pipeline = pipeline.fit(messages_train, categories_train)

       predicted = pipeline.predict(interactive_test)
       #print(predicted)
```

3.5 Inspect Bag of Words

```
In []: aggressive_train =  [tweet.text for tweet in results_aggressive]

       non_aggressive_train = [tweet.text for tweet in results_non_aggressive]

       # messages_train = aggressive_train + non_aggressive_train

       categories_train = ['aggressive'] * len(aggressive_train) \
                        + ['non_aggressive'] * len(non_aggressive_train)

       matrix = count_vect.fit_transform(aggressive_train, ['aggressive'] * len(aggressive_train))
       freqs = [(word, matrix.getcol(idx).sum()) for word, idx in count_vect.vocabulary_.items()]
       #sort from largest to smallest
       # print sorted (freqs, key = lambda x: -x[1])
```

3.6 Pickle

```
In []: # Getting back the objects:
       with open('results_854.pickle') as f:
           results_aggressive, results_non_aggressive = pickle.load(f)
```

```
In []: # Saving the objects:
       with open('final_evaluation.pickle', 'w') as f:
           pickle.dump([results_aggressive, results_non_aggressive], f)
```