Distant Supervision - The End

January 23, 2015

1 Import

```
In [830]: import os, csv, tweepy, ucto, time
          import numpy as np
          import pandas as pd
          from sklearn.ensemble import ExtraTreesClassifier
          from sklearn.feature_extraction.text import TfidfTransformer
          from sklearn.feature_extraction.text import CountVectorizer
          from sklearn.linear_model import SGDClassifier, LogisticRegression
          from sklearn.cross_decomposition import PLSRegression
          from sklearn import tree
          from sklearn.svm import LinearSVC, NuSVC
          from sklearn.ensemble import BaggingClassifier
          from sklearn import metrics
          from sklearn.grid_search import GridSearchCV
          from sklearn.pipeline import FeatureUnion
          from pprint import pprint
          from sklearn.base import BaseEstimator
          import pickle
          from scipy import sparse


          from sklearn.feature_selection import SelectKBest, chi2
          from sklearn.linear_model import RidgeClassifier
          from sklearn.linear_model import SGDClassifier
          from sklearn.linear_model import Perceptron
          from sklearn.linear_model import PassiveAggressiveClassifier
          from sklearn.naive_bayes import MultinomialNB
          from sklearn.neighbors import NearestCentroid

          import logging

          from tweepy.streaming import StreamListener
          from tweepy import Stream

          from pynlpl.clients.frogclient import FrogClient
```

1.1 Settings

```
In [831]: os.chdir("/home/pold/PycharmProjects/twitter-tutorial/")
```

1.2 General Classifier

1.3 Read Aggressive Tweets from TSV files

```
In [878]: def create_key(x):
              '''Create a file name from a number'''
              return "tweets/fold_" + str(x) + ".txt"

          # Create a list of file names of the aggressive 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,
                                 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 Dutch tokeniser rules

```
In [879]: settingsfile = "/etc/ucto/tokconfig-nl-twitter"
          tokenizer = ucto.Tokenizer(settingsfile)

          def remove_non_ascii(s):
              return "".join(i for i in s if ord(i) < 128)

          def ucto_tokenizer(str):
              remove_non_ascii(str)
              tokenizer.process(unicode(str))
              tokens = [unicode(token).encode('utf-8') for token in tokenizer]
              for pos, token in enumerate(tokens):
                  if token.startswith('http:'): tokens[pos] = 'url'
                  if token.startswith('@'): tokens[pos] = 'user'
              return tokens


          ucto_tokenizer("Was geht #leute #heute @POld hey? http://tinyurl.com/dasd")
```

```
Out[879]: ['Was', 'geht', '#', 'leute', '#', 'heute', 'user', 'hey', '?', 'url']
```

1.5 Aggressive Hashtags - Wordlist

```
In [880]: # The wordlist is a dictionary of profanity words.
          wordlists = {}
          # for name in ["illness", "religion", "sexuality", "slurs", "misc"]:
          for name in ["selection"]:
              with open("wordlist_" + name + ".txt") as f:
                  words = f.read().splitlines()
                  wordlists[name] = words
```

2 Twitter

2.1 Authentication

```
In [881]: consumer_key = 'zdprSqld7JxkSjaeqQppemW5y'
          consumer_secret = 'QMZCPjMZnqJjkXQxXerHGCP11PGXIrl7iXHOHnrlLUjwNVOQdY'
          access_token = '1612711080-64a9w2e4BOTV4id4HTqLpzaO8Z6ag0pNVE9axZv'
          access_token_secret = 'iGBJ7jTo7LjqdMWyU8sJT5xX6CllWX0YoeQScYxpbGjMr'
```

2.2 Connect to Twitter

```
In [882]: 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 [883]: def create_query(wordlist):
              return " OR ".join(wordlist)

          def create_negate_query(wordlist):
              return " ".join(["-" + word for word in wordlist])
```

2.4 Collect Tweets

```
In [838]: results_aggressive = []
          results_non_aggressive = []

          flattened_wordlist = [item for sublist in wordlists.values() for item in sublist]

          query_pos = create_query(flattened_wordlist)
          query_neg = create_negate_query(flattened_wordlist)

          # Aggressive Tweets

          for tweet in tweepy.Cursor(api.search, q = query_pos,  lang = "nl").items(1500):
              results_aggressive.append(tweet)

          time.sleep(60)

          # Non-aggressive Tweets

          for tweet in tweepy.Cursor(api.search, q = query_neg,  lang = "nl").items(len(results_aggressi
              results_non_aggressive.append(tweet)
```

2.5 Count Hashtags

```
In [885]: flattened_wordlist = [item for sublist in wordlists.values() for item in sublist]
          hash_dict = { }

          for word in flattened_wordlist:
              hash_dict[word] = 0

          for tweet in results_aggressive:
              for word in flattened_wordlist:
```

```
                if word in tweet.text: hash_dict[word] += 1

        print(len(results_aggressive))
        print(len(results_non_aggressive))
        print(hash_dict)
```

```
593
593
{'#idiot': 12, '#fuckyou': 19, '#gvd': 15, '#sukkel': 42, '#grrr': 60, '#idioot': 48, '#grr': 75}
```

2.6 Additional Feature Extractor

```
In [842]: # The wordlist is a dictionary of profanity words.
          badwords = []
          for name in ["illness", "religion", "sexuality", "slurs", "misc"]:
              with open("wordlist_" + name + ".txt") as f:
                  word = f.read().splitlines()
                  badwords.append(word)

          class MyWordCounter(BaseEstimator):

              def __init__(self):
                  with open("my_badlist.txt") as f:
                      badwords = [l.strip() for l in f.readlines()]
                  self.badwords_ = badwords

              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]

                  # longest word
                  max_word_len = [np.max([len(w) for w in c.split()]) for c in documents]
                  # average word length
                  mean_word_len = [np.mean([len(w) for w in c.split()])
                                                      for c in documents]
                  # number of badwords:
                  n_bad = [np.sum([c.lower().count(w) for w in self.badwords_])
                                                          for c in documents]
                  exclamation = [c.count("!") for c in documents]
                  addressing = [c.count("@") for c in documents]
                  spaces = [c.count(" ") for c in documents]

                  allcaps_ratio = np.array(allcaps) / np.array(n_words, dtype=np.float)
                  bad_ratio = np.array(n_bad) / np.array(n_words, dtype=np.float)

                  return np.array([n_words
                                   , n_chars
```

```
                                , allcaps
                                , max_word_len
                                , mean_word_len
                                , exclamation
                                , addressing
                                , spaces
                                , bad_ratio
                                , n_bad
                                , allcaps_ratio
        ]).T
```

2.7 Define Pipeline

```
In [913]: with open("dutch-stop-words.txt") as f:
              dutch_stop_words = f.read().splitlines()

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

          my_Counter = MyWordCounter()

          combined = FeatureUnion([("clf1", count_vect), ("clf2", my_Counter)])

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


          clf = LogisticRegression(penalty = 'l2', dual = False)# NuSVC(nu = 0.4)

          # or logistic regression ! (could be easier and better to explain)

          from sklearn.pipeline import Pipeline
          pipeline = Pipeline([('vect', combined),
                               ('tfidf', tfidf),
                               ('clf', clf)])
```

2.8 Leave-one-out cross-validation

```
In [914]: accuracy_per_fold = []

          for x in xrange(0, 10):
              train = frame.ix[[create_key(y) for y in range(0,10) if x != y]]
          #     test = frame.ix[create_key(x)]

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

              messages_test = test['message']
              categories_test = test['category']

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

```
            predicted = pipeline.predict(messages_test)
            accuracy = np.mean(predicted == categories_test)
            accuracy_per_fold.append(accuracy)
            print(accuracy)

        print("Total average accuracy: ", np.mean(accuracy_per_fold))
```

```
0.879365079365
0.879365079365
0.888888888889
0.88253968254
0.87619047619
0.879365079365
0.873015873016
0.87619047619
0.869841269841
0.806349206349
('Total average accuracy: ', 0.87111111111111117)
```

2.9 Run Classification and Evaluation

```
In [915]: 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)

          pipeline = pipeline.fit(messages_train, categories_train)
          #
          accuracy_per_fold = []
          for x in xrange(0, 10):
              test = frame.ix[create_key(x)]

              messages_test = test['message']
              categories_test = test['category']

              predicted = pipeline.predict(messages_test)
              accuracy = np.mean(predicted == categories_test)

              accuracy_per_fold.append(accuracy)
              print(metrics.classification_report(categories_test, predicted))

          print("Total average accuracy: ", np.mean(accuracy_per_fold))
```

```
precision    recall  f1-score   support

 aggressive       0.68      0.78      0.73       158
non_aggressive       0.74      0.64      0.68       157

avg / total       0.71      0.71      0.71       315

             precision    recall  f1-score   support
```

```
 aggressive       0.76      0.72      0.74       158
non_aggressive       0.73      0.77      0.75       157

avg / total       0.75      0.75      0.75       315

             precision    recall  f1-score   support

 aggressive       0.79      0.75      0.77       158
non_aggressive       0.76      0.80      0.78       157

avg / total       0.77      0.77      0.77       315

             precision    recall  f1-score   support

 aggressive       0.72      0.72      0.72       158
non_aggressive       0.72      0.72      0.72       157

avg / total       0.72      0.72      0.72       315

             precision    recall  f1-score   support

 aggressive       0.71      0.77      0.74       158
non_aggressive       0.75      0.69      0.72       157

avg / total       0.73      0.73      0.73       315

             precision    recall  f1-score   support

 aggressive       0.70      0.75      0.72       157
non_aggressive       0.73      0.68      0.70       158

avg / total       0.71      0.71      0.71       315

             precision    recall  f1-score   support

 aggressive       0.66      0.76      0.71       157
non_aggressive       0.72      0.62      0.67       158

avg / total       0.69      0.69      0.69       315

             precision    recall  f1-score   support

 aggressive       0.70      0.71      0.71       133
non_aggressive       0.71      0.70      0.71       135

avg / total       0.71      0.71      0.71       268

             precision    recall  f1-score   support

 aggressive       0.68      0.73      0.70       154
non_aggressive       0.72      0.67      0.69       158

avg / total       0.70      0.70      0.70       312
```

```
             precision    recall  f1-score   support

 aggressive       0.70      0.74      0.72       157
non_aggressive       0.73      0.69      0.71       158

avg / total       0.71      0.71      0.71       315

('Total average accuracy: ', 0.71949747599001335)
```

```
In [924]: 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)
```

```
Please type message that should be categorized: eikel
['non_aggressive']
```

2.10 Inspect Bag of Words

```
In [851]: matrix = count_vect.fit_transform(messages_train, categories_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])
```

```
[('user', 814), ('url', 517), ('...', 199), ('fuck', 156), ('(', 155), (')', 154), ('kleur', 152), ('ja
```

2.11 Pickle Load

```
In [642]: # Getting back the objects:
          with open('results_854.pickle') as f:
              results_aggressive, results_non_aggressive = pickle.load(f)
```

```
In [94]: # Saving the objects:
         with open('results_479.pickle', 'w') as f:
             pickle.dump([results_aggressive, results_non_aggressive], f)
```