# NewzTrader: Autonomous Trading Agent Implementation Using Natural Language Processing Of News Headlines

William Lyon

December 6, 2012

**Abstract**

Natural Language Processing techniques are used to examine financial news headlines and generate predictions of stock price movements. News headlines from the Wall Street Journal from January 1, 2009 to November 1, 2012 are collected and paired with daily S&P 500 index returns. This information is used to train a text classifier and generate BUY/SELL trading signals. This simple trading strategy is then backtested using Wall Street Journal news headlines as a predictor for movement in the price of the S&P 500 index.

## 1 Introduction

In finance, the Efficient Market Hypothesis states that all publicly available information is reflected in financial market prices. As new information becomes available, prices adjust to take this new information into account. In accordance with the Efficient Market Hypothesis, financial news should directly influence stock price movements as the news items become publicly available. This paper uses financial news headlines to train a machine-learning agent to predict short term fluctuations in the value of the S&P 500 stock market index. Naive Bayes and Maximum Entropy classifiers are built and used to classify news headlines as indicative of an increase, decrease, or no movement of the value of the S&P 500 index (within a 1% threshold). These classifiers are then used to generate Buy and Sell trading signals and this trading strategy is backtested by trading against the S&P 500 index.

### 1.1 Motivation

This tool could be used as a component of an autonomous trading agent that will make BUY/SELL decisions for trading financial instruments. An autonomous trading agent is more likely to be effective if buy/sell signals are generated by an ensemble model, where the ultimate buy/sell signal is taken by polling many independent predictive models and combining a weighted average of predictions [5]. Therefore it should not be expected that the Natural Language Processing component alone should be profitable during backtesting.

### 1.2 Literature Review

Daskalopoulos 2003 presents a method of predicting short term stock price fluctuations of individual company stocks by classifying company-specific news headlines using a Naive Bayes classifier[3]. He reports accuracy of between 0.988 and 0.945. However, the threshold for identifying a RISE or FALL instance is set at a one day change in the stock price of 10%, which is very rare. In fact, he indicates the prior probability of a NOTHING instance (a daily stock return of between -9.99% and 9.99%) is 0.978. Thus a classifier that only predicts NOTHING classes would have accuracy of 0.978. Skiena, et al. 2010 uses a more complicated sentiment analysis approach and wider selection of text data (including blogs, Twitter, and daily newspapers) to predict individual stock price movements and evaluates the model using trading backtesting. The paper finds their approach to have consistent returns over the course of five years of backtesting[6].

### 1.3 Initial Attempt

Initially I set out with the goal of duplicating Daskalopoulos's[3] results and building on that by using trading backtesting to further evaluate the model. I collected stock specific data by using a Google Finance API which returned an XML document listing news headlines relevant to a specific company. After building a Naive Bayes classifier using the company specific news headlines, accuracy was unfortunately quite low: around 0.10 on average. I believe these poor results relative to Daskalopoulos can be explained by 1) using a lower threshold for identifying UP/DOWN/NONE classes and 2) removal of news items from Google Finance. I chose to use a much lower threshold for identifying UP/DOWN/NONE instances than Daskalopoulos (+/- 3% vs. +/- 10% daily returns). Since daily returns exceeding 10% absolute value are quite rare, a trading strategy using this model would be quite ineffective. Also, it appears Google for some reason removes news items from Google Finance after a relatively short period of time. Therefore the same data contained many recent news items but few news items dating beyond one year. This made building a multi-year collection of test data difficult. After this initial setback I moved to the model described below.

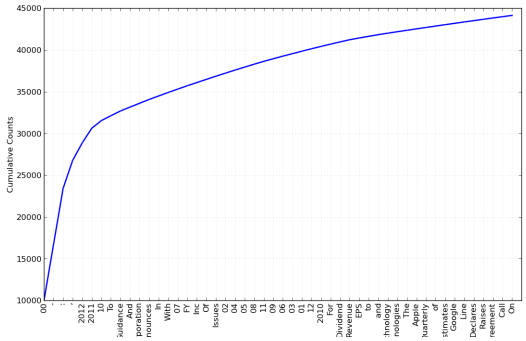


Figure 1: Cumulative frequency plot

# 2 The Data

For this experiment news headlines and stock quotes from Jan 1, 2009 to Nov 1, 2012 were collected. News headlines were limited to the Wall Street Journal Business section. Stock quotes were daily open, low, high, and close of the S&P 500 index.

## 2.1 Historical Stock Market Quotes

Historical stock price data is available from Yahoo! Finance and was downloaded using the Python pandas data analysis library, which includes a method for importing Yahoo! Finance data directly into a pandas object. The S&P 500 index was chosen because it is a broad market index that incorporates companies from all major industries and is a good proxy for the market as a whole. Daily quotes were collected, however quotes of higher frequency (hourly, 15-minute, etc.) would have been preferred and would more accurately reflect how a trading system such as NewzTrader would perform in a real trading environment. Unfortunately stock quote data in frequency greater than daily is more difficult to obtain and expensive. For the purposes of this initial experiment daily quotes are acceptable, but it should be noted that higher frequency data is ideal.

## 2.2 News Headlines

Wall Street Journal Business section news headlines were collected using an API provided by NewsCred [7]. NewsCred is a company that provides access to full-text news articles, including The Wall Street Journal. A limited-use API key was provided by NewsCred, which allowed for access to headlines only. The NewsCred API was used to retrieve daily Wall Street Journal news headlines, which were then limited to articles tagged by NewsCred as belonging to the "Business" category. A total of 101,618 news headlines over 1416 days were collected with an average of 71 headlines per day.

## 2.3 Initial Data Exploration

Each news headline's publication date was aligned with the S&P 500 stock quote for that date. Headlines were removed from the corpus if no stock quote was available for that day (non-trading day, missing data, etc.). After aligning with stock quote data, a total of 85,775 news headlines remained in the corpus. These were stored in a dictionary data structure where the key corresponded to a date and the values were a list of the news headlines for that date. The corpus of news headlines consisted of a total of 199,511 words. Figure 1 shows a cumulative frequency plot of the 50 most common words in the news headline corpus.

# 3 Classification Methodology

Every observed trading day was classified as either UP, DOWN, or NONE depending on the return of the S&P 500 index for that day, relative to the previous trading day, using a threshold of 1.5% of the previous day's value. Days where the S&P 500 closed at least 1.5% higher than the previous day's close were classified as UP. Days where the S&P500 close was at least 1.5% lower than the previous day's close were classified as DOWN. Days closing within 1.5% were classified as NONE. The distribution of classifications were as follows: UP: 14651 DOWN: 13718 NONE: 57405 TOTAL: 85775. Resulting in the following prior probabilities: UP: 0.1708 DOWN: 0.1599 NONE: 0.6693. These tuples of {classification, listOfNewsHeadlines} were used as the training data for a Naive Bayes classifier and a Maximum Entropy classifier. The Python Natural Language Toolkit (NLTK) was used for handling text processing as well as the built-in classifiers.

## 3.1 Naive Bayes

Summarize NB from NLTK book

## 3.2 Maximum Entropy

Summarize ME from NLTK book

# 4 Implementation

This initial version of NewzTrader is implemented in Python using the libraries listed below. Eventually NewzTrader could be implemented as a web based application that processes news headlines in real time and makes live trading decisions. This paper however only explores the initial implementation and backtesting of a trading strategy. This initial version is implemented in four separate Python programs. The first program (newsCredScraper.py) simply collects Wall Street Journal news headlines and stores them in a dictionary data structure then writes that structure to disk for persistence. The second program (dataGetter.py) collects

## 4.1 Dependencies

Several Python packages are required to run NewzTrader. The required packages and their purpose are described below.

**NLTK** The Python Natural Language Toolkit is used for text processing, as well as the built-in classifiers Naive Bayes and Maximum Entropy.

**pandas** Data analysis library

**LXML** Used for parsing XML

**Dateutil** Used for parsing date strings

**Matplotlib** Used for plotting

**Zipline** Financial backtesting library used for backtesting of NewzTrader

**Numpy** Numeric Python library

## 4.2 Data Collection & Munging - newsCredScraper.py

## 4.3 Training NLP Classifier -

### 4.3.1 Bag of words

### 4.3.2 Filtering stopwords

### 4.3.3 Include significant bigrams

## 4.4 Naive Bayes Classifier

## 4.5 Maximum Entropy Classifier

## 4.6 Backtesting

# 5 Evaluation

## 5.1 Accuracy

## 5.2 Precision

## 5.3 Recall

## 5.4 Most informative features

| Classifier | Accuracy | Recall(NONE) | Recall(UP) | Recall(DOWN) | Precision(NONE) | Precision(UP) | Precision(DOWN) |
|---|---|---|---|---|---|---|---|
| NB-full | 0.48 | 0.5814 | 0.3076 | 0.2945 | 0.7054 | 0.2292 | 0.2150 |
| NB-0910 | 0.45 | 0.5361 | 0.3110 | 0.2947 | 0.6695 | 0.2374 | 0.2151 |
| ME-full | 0.6680 | 0.9335 | 0.1139 | 0.1035 | 0.6883 | 0.3857 | 0.3944 |
| ME-0910 | 0.6155 | 0.9499 | 0.0439 | 0.0503 | 0.6374 | 0.2636 | 0.3025 |

Table 1: Evaluation metrics. NB-full=Naive Bayes, full data; NB-0910=Naive Bayes 2009-2010 data only; ME-full=Maximum Entropy, full data; ME-0910=Maximum Entropy 2009-2010 data only

| | Naive Bayes | Maximum Entropy |
|---|---|---|
| Accuracy | 0.6381 | 0.4091 |
| Recall(NONE) | 0.9594 | 0.4698 |
| Recall(UP) | 0.0929 | 0.5161 |
| Recall(DOWN) | 0.0908 | 0.0782 |
| Precision(NONE) | 0.6508 | 0.6288 |
| Precision(UP) | 0.5035 | 0.1996 |
| Precision(DOWN) | 0.4452 | 0.4375 |

Table 2: Evaluation metrics, high information words only for 2009-2010 data only

# 6 Trading Model

## 6.1 Trading Signals

## 6.2 Backtesting

This is the backtestinfg. Where is the rest of the backtesting?

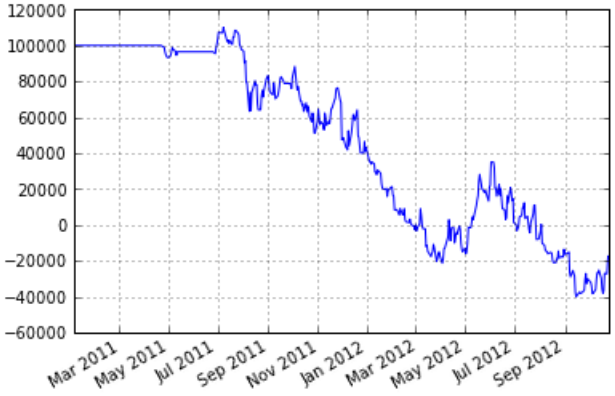


Figure 2: Portfolio value with backtesting strategy

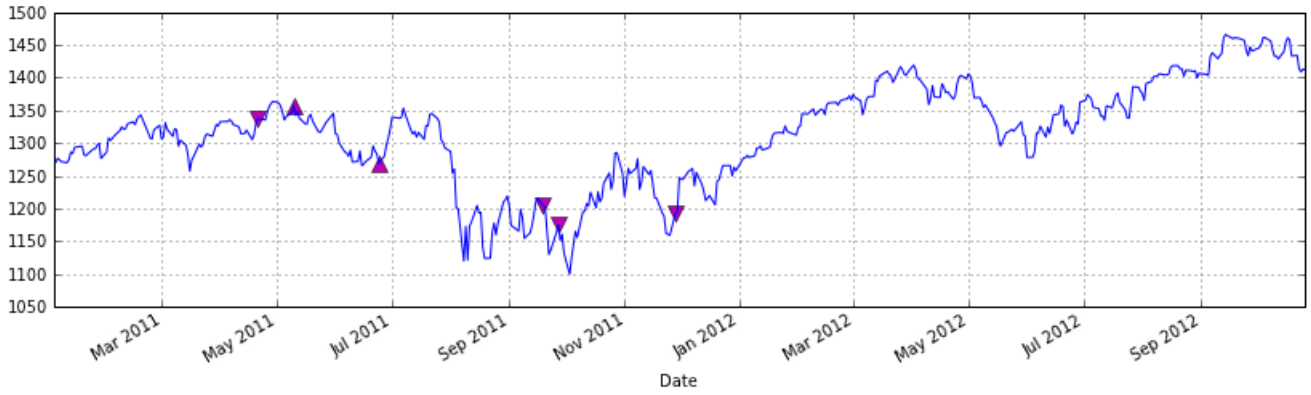


Figure 3: Backtest buy/sell signals

# 7 Conclusions

In backtesting, the trading strategy tested by NewzTrader is not profitable. This can be attributed to the relatively low accuracy of the classifier and specifically the very low recall measures of the UP and DOWN classifications.

## 7.1 Further research

# References

[1] Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing with Python*. O'Reilly Media Inc, California 1st Edition, 2009.

[2] Jacob Perkins, *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing, Birmingham, UK 1st Edition, 2010

[3] Daskalopoulos, V, *Stock Price Prediction From Natural Language Understanding of News Headlines*. Presented at Information Conference on Machine Learning (iCML03), 2003. `http://www.cs.rutgers.edu/~mlittman/courses/ml03/iCML03/papers/daskalopoulos.pdf`.

[4] Peramunetilleke, D. et al., *Currency Exchange Rate Forecasting From News Headlines*. Australian Computer Science Communications. Vol 24:2 Jan-Feb 2002, pp131-139.

[5] Barbosa, R., et al., *Algorithmic Trading Using Intelligent Agents*. 2008 International Conference on Image Processing, Computer Vision, and Pattern Recognition.

[6] Skiena, S. et al., *Trading Strategies To Exploit Blog and News Sentiment*. Fourth International Conference on Weblogs and Social Media 2010, Washington, DC, May 23-26, 2010. `http://www.cs.sunysb.edu/~skiena/lydia/blogtrading.pdf`

[7] `http://newscred.com/developer/docs/module/article/doc`

# 8 Source Code Listings

Listing 1: newsCredScraper.py: scrapes WSJ news headlines

```
1
2  from lxml import etree
3  from datetime import datetime
4  from datetime import date
5  from datetime import timedelta
6  #import datetime
7  from dateutil.parser import parse
8  #import pandas as pd
9  import pickle
10 #from pandas.io.data import DataReader
11 #from pandas.io.data import DataReader
12 #from pandas.io.data import DataReader
13
14 #import pandas as pd
15 #from datetime import datetime
16
17 #iimport pandas as pd
18 #
19 #from pandas.io.data import DataReader
```

```
20
21  def daterange(start_date, end_date):
22      for n in range(int ((end_date - start_date).days)):
23          yield start_date + timedelta(n)
24
25  def only_alphanum(s):
26      #s = unicode(s, "utf-8")
27      return '_'.join(c for c in s.split() if c.isalnum())
28  def only_alpha(s):
29      return '_'.join(c for c in s.split() if c.isalpha())
30  def removeNonAscii(s): return "".join(i for i in s if ord(i)<128)
31
32  #sp500 = DataReader("SPY", "yahoo", datetime(2009, 1, 1))
33  news={}
34  #descs = {}
35  start_date=date(2011, 7, 27)
36  end_date = date(2012, 11, 9)
37  news[date(2009, 11, 9)] = []
38  news[date(2009, 11, 10)] = []
39  for single_date in daterange(start_date, end_date):
40      news[single_date]=[]
41  for single_date in daterange(start_date, end_date):
42      #print "Starting date: " + str(single_date)
43      year = single_date.year
44      month = single_date.month
45      day = single_date.day
46      path = "http://api.newscred.com/articles?access_key=c4bcc3f7c9bf9ec159f51da0a86ca658&sources=104
            afa30d811d37a5582a39e1662a311&pagesize=99&from_date=%d-%d-%d&to_date=%d-%d-%d_23:59:59" % (year
            , month, day, year, month, day)
47      while True:
48          try:
49              root = etree.parse(path)
50              break
51          except etree.XMLSyntaxError:
52              pass
53      myRoot = root.getroot()
54
55      #news[date(year, month, day)]=[]
56      #descs[date(year, month, day)]=[]
57      for element in myRoot.iter("article"):
58          #for item in element.iter("description"):
59           # desc = item.text
60          for item in element.iter("title"):
61              title = item.text
62          for item in element.iter("created_at"):
63              pubDate = parse(item.text)
64
65          news[pubDate.date()].append(only_alphanum(removeNonAscii(title)))
66  #import pickle
67      output = open('newsDict2.pkl', 'wb')
68      pickle.dump(news, output)
69      output.close()
```

Listing 2: dataGetter.py: downloads historical S&P 500 index price data and joins with WSJ news headlines / identifies news headline classifications and saves in format for corpus reader

```
1  # dataGetter.py
2  # William Lyon
3  # AI Grad Project
4  # NewzTrader
5  # dataGetter.py
6  # 1) Loads pickled news dict
7  # 2) Downloads historical stock price data
8  # 3) Joins news dict and stock data in a pandas dataframe
9  # 4) Set UP/DOWN/NONE classifications for every trading day
10 # 5) Save news headlines in .CSV files for corpus reader
11
```

```
import pickle
from datetime import datetime
from datetime import timedelta
from datetime import date

def daterange(start_date, end_date):
    for n in range(int ((end_date - start_date).days)):
        yield start_date + timedelta(n)

fkl_file = open('combinedNewsDictFull.pkl', 'rb')
news = pickle.load(fkl_file)
fkl_file.close()
start_date=date(2009, 1, 1)
end_date = date(2012, 11, 17)

import pandas as pd
#
from pandas.io.data import DataReader

def only_alphanum(s):
    #s = unicode(s, "utf-8")
    return '_'.join(c for c in s.split() if c.isalnum())
def only_alpha(s):
    return '_'.join(c for c in s.split() if c.isalpha())
def removeNonAscii(s): return "".join(i for i in s if ord(i)<128)

sp500 = DataReader("^GSPC", "yahoo", datetime(2009, 1, 1))
newsframe = pd.Series(news, name='News')
#descframe = pd.Series(descs, name='Desc')
frameWithNews = sp500.join(pd.DataFrame(newsframe))
#sp500Frame = frameWithNews.join(pd.DataFrame(descframe))
newsframe = pd.Series(news, name='News')
#descframe = pd.Series(descs, name='Desc')
frameWithNews = sp500.join(pd.DataFrame(newsframe))

newsReturns = frameWithNews['Adj_Close'].pct_change()
newsReturns.name='Returns'
returnsFrame = frameWithNews.join(pd.DataFrame(newsReturns))

returnsFrame['UP'] = returnsFrame.Returns > 0.01
returnsFrame['DOWN'] = returnsFrame.Returns < -0.01
returnsFrame['NONE'] = (returnsFrame['UP']==False) & (returnsFrame['DOWN']==False)

droppedFrame = returnsFrame

newsUP_frame = droppedFrame[droppedFrame['UP']==True]


newsDOWN_frame = droppedFrame[droppedFrame['DOWN']==True]
newsNONE_frame = droppedFrame[droppedFrame['NONE']==True]

newsNONE_frame = newsNONE_frame.dropna()
newsUP_frame = newsUP_frame.dropna()
newsDOWN_frame = newsDOWN_frame.dropna()

# DO THIS FOR NONE, UP, and DOWN
i = 1
for row in newsNONE_frame.iterrows():
    i+=1
    if len(row[1].ix['News'])>0:
        for line in row[1].ix['News']:
            i+=1
            writeFile = open('%d_news_NONE.csv' % i, 'w')
            writeFile.write(line+'\n')

# DO THIS FOR NONE, UP, and DOWN
i = 1
```

```
for row in newsUP_frame.iterrows():
    i+=1
    if len(row[1].ix['News'])>0:
        for line in row[1].ix['News']:
            i+=1
            writeFile = open('%d_news_UP.csv' % i, 'w')
            writeFile.write(line+'\n')

# DO THIS FOR NONE, UP, and DOWN
i = 1
for row in newsDOWN_frame.iterrows():
    i+=1
    if len(row[1].ix['News'])>0:
        for line in row[1].ix['News']:
            i+=1
            writeFile = open('%d_news_DOWN.csv' % i, 'w')
            writeFile.write(line+'\n')
```

Listing 3: nbTrainer.py: loads news corpus / trains Naive Bayes classifier

```
# nbTrainer.py
# William Lyon
# AI Grad Project
# NewzTrader
# nbTrainer.py
# 1) Load NLTK corpus reader for naive bayes classifier
# 2) Train NB classifier with random 90% of corpus features
# 3) Test NB classifier with remaining 10% of corpus features and report
# accuracy
# TODO: recall, precision reports; improve classifier (only strong words?)

# NLTK - train nb_classifier


import random
import nltk as nltk
#nltk.download()
from nltk.corpus import stopwords
import os, os.path
path = os.path.expanduser('~/nltk_data')
if not os.path.exists(path):
    os.mkdir(path)
os.path.exists(path)
import nltk.data
path in nltk.data.path
from nltk.corpus.reader import CategorizedPlaintextCorpusReader
reader = CategorizedPlaintextCorpusReader('.', r'.*_news_.*\.csv', cat_pattern=r'.*_news_(\w+)\.csv')
reader.categories()

def bag_of_words(words):
    return dict([(word, True) for word in words if word[0].isalpha()])
import collections
def bag_of_words_not_in_set(words, badwords):
    return bag_of_words(set(words)-set(badwords))

def bag_of_non_stopwords(words, stopfile='english'):
    badwords = stopwords.words(stopfile)
    return bag_of_words_not_in_set(words, badwords)

from nltk.metrics import BigramAssocMeasures
from nltk.collocations import BigramCollocationFinder

def bag_of_bigrams_words(words, score_fn=BigramAssocMeasures.chi_sq, n=2000):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(score_fn, n)
    dictOfBigrams = bag_of_words(bigrams)
    dictOfBigrams.update(bag_of_non_stopwords(words))
```

```
48      return dictOfBigrams
49
50  def label_feats_from_corpus(corp, feature_detector=bag_of_bigrams_words):
51      label_feats = collections.defaultdict(list)
52      for label in corp.categories():
53          for fileid in corp.fileids(categories=[label]):
54              feats = feature_detector(corp.words(fileids=[fileid]))
55              label_feats[label].append(feats)
56      return label_feats
57
58  def split_label_feats(lfeats, split=0.90):
59      train_feats = []
60      test_feats = []
61      for label, feats in lfeats.iteritems():
62          random.shuffle(feats, random.random)
63          cutoff = int(len(feats) * split)
64          train_feats.extend([(feat, label) for feat in feats[:cutoff]])
65          test_feats.extend([(feat, label) for feat in feats[cutoff:]])
66      return train_feats, test_feats
67
68
69
70
71  reader.categories()
72
73  lfeats = label_feats_from_corpus(reader)
74  lfeats.keys()
75  train_feats, test_feats = split_label_feats(lfeats)
76  len(train_feats)
77  len(test_feats)
78
79  from nltk.classify import NaiveBayesClassifier
80  nb_classifier = NaiveBayesClassifier.train(train_feats)
81  nb_classifier.labels()
82
83  from nltk.classify.util import accuracy
84  accuracy(nb_classifier, test_feats)
```

Listing 4: backTest.py: simulates trading using trading signals generated from WSJ news headlines using Naive Bayes classifier

Listing 5: NewzTrader.py