CS230 Stock Price Prediction (Milestone)

Ryan Almodovar
Stanford University
ralmodov@stanford.edu

1 Introduction

The financial market has been shown to be informationally efficient so stock prices reflect all known information and the price movement is in response to news or events [Fama, 1965]. Several Natural Language Processing (NLP) techniques have been applied over recent years to explore financial news for predicting market volatility, such as bags-of-words, noun phrases, named entities, and sentiment analysis [Kogan et al., 2009; Schumaker and Chen, 2009]. For this project, we explore predicting the Dow Jones Industrial Average (DJIA) as well as the stock prices for some selected companies found in headlines.
Note: Due to time constraints, the baseline model for this milestone only contains the generalized DJIA prediction using the sentiment analysis approach. The alternate architectures (CNN, Neural Tensor Network [Ding, 2015], LSTM, GRU, etc.) and select companies stock price predictions are planned to be included in the finalized report.

2 Dataset Details

For DJIA, a dataset with the top 25 of news headlines extracted from news articles. The current data was provided through Kaggle (https://www.kaggle.com/aaron7sun/stocknews) and included data points over a period of roughly 8 years.

3 Approach

The data is first pre-processed to remove Split the dataset train, dev, and test sets with distributions 80%, 10%, and 10% respectively. Using NLTK's Sentiment Intensity Analyzer as a simple baseline model, it determined the average sentiment of the top 25 news headlines and scored a 55% prediction rate. Some other architectures The main issue with this approach is that the average sentiment may result in a loss of information which could be a reason for poor accuracy. In order to improve the prediction rate, I plan to gather more data specific to particular companies/industries and predict the stock prices of these rather than only the generalized DJIA, as well as using a more in-depth architecture than sentiment analysis from the NLTK Sentiment Analyzer. implement and experiment using LSTM/GRU cells, as well as the Neural Tensor Network (NTN) described in Ding et al 2015 (https://www.ijcai.org/Proceedings/15/Papers/329.pdf) for event-driven embedding. For the NTN to be implemented, each headline needs to be transformed into an event tuple described in the NTN architecture. Some candidate options to be able to transform the headline tokens into the event tuple are to use SpaCy's library I further plan to expand the dataset and combine it with

4 Baseline Code

```
import numpy as np
import pandas as pd

from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

data_file = "Combined_News_DJIA.csv"

def main():
  print("Importing data...")
  data = import_data("./../data/" + data_file)
  pre_processed = pre_process(data)
  sentiment_included = analyze_sentiment(pre_processed)

  train_set, test_set = split_dataset(sentiment_included)

  train_labels = ravel_labels(train_set)
  test_labels = ravel_labels(test_set)

  train_sentiments = process_sentiments(train_set)
  test_sentiments = process_sentiments(test_set)

  print("Begin Training...")
  clsfr = train(train_sentiments, train_labels.astype('int'))

  results = cross_validate(test_sentiments, test_labels.astype('int'),
                                        clsfr)

  print("Test Label Mean: " + str(test_labels.mean()))
  print("Results: " + str(results))
  print("Avg sentiment: " + str(results.mean()))

def import_data(filename):
  return pd.read_csv(filename, header=0).fillna('').values

def pre_process(data):
  print("Preprocessing...")
  for row in data[0:476]:
    for field in row[2:]:
      if field:
        field = field[1:] # Remove first 'b'
  return data

def analyze_sentiment(data):
  print("Analyzing sentiment...")
  sid = SentimentIntensityAnalyzer()
  avgs = np.empty((len(data),1))
  for i in range(0,len(data)):
    sentiments = []
    for field in data[i][2:]:
      sentiments.append(sid.polarity_scores(field)['compound'])
    avg = float(sum(sentiments))/len(sentiments)
    avgs[i] = avg
  return np.append(data, avgs, axis=1)

def split_dataset(data):
  # Split 80/10/10
  n_d = len(data)
  p_train = int(n_d * 0.8)
  p_dev = p_train + int(n_d * 0.1)

  train = data[:p_train]
  dev = data[p_train:p_dev]
  test = data[p_dev:]

  print("Train: {0} / Dev: {1} / Test: {2}".format(len(train), len(dev),
                                        len(test)))
```

```
  return train, test

def train(data, labels):
  return svm.SVC(kernel='linear', C=1).fit(data, labels)

def cross_validate(data, labels, clsfr):
  return cross_val_score(clsfr, data, labels, cv=5)

def ravel_labels(data):
  return data[:,1].ravel()

def process_sentiments(data):
  return data[:,27].reshape(len(data), 1)

if __name__ == "__main__":
  main()
```