SYS 6016: Machine Learning

Narges Tabari

Final Project, Phase 2 Document

NLP Fine-Grained Numeral Understanding in Financial Tweets

Due April 2, 2019

Naive Baes

Lucas Beane (lhb7tz@virginia.edu)

Will Gleave (swg8jq@virginia.edu)

Rakesh Ravi (rk9cx@virginia.edu)

Justin Ward (jw8vw@virginia.edu)

1. Introduction

To begin, we will attempt to vectorize the tweets using normal word embeddings then classify using a feed forward neural network. Due to the nature of the tweets, we don't expect these embeddings to produce accurate results and may attempt character embeddings before moving on to future approaches.

We then plan to use convolutional layers in a neural network in order to obtain a proper vectorization of our financial tweets with potential application of recurrent neural networks depending upon performance. As the above papers address, the CNN and RNN (GRU) embeddings both have the potential to capture information from the unstructured tweets and we plan to try each of these individually as well as ensembles.

Finally, we will use ULMFiT's approach to LSTMs as a method of generating a language model for our financial tweets in the hopes of improving accuracy by narrowing down the field of possibilities in text. That being said, Prusa and Khoshgoftaar seem confident that LSTMs in a CNN for character-level embedding do not add to such a neural network, so we expect that we may need to tweak our embedding level in order to achieve reliable results with this technique.

2. Literature Review

I. Universal Language Model Fine-tuning for Text Classification

By Jeremy Howard and Sebastian Ruder

In this paper, Howard and Ruder examine the problem of building a language model with a small dataset. The paper proposes using transfer learning in Natural Language Processing (NLP), with a new method called Universal Language Model Fine-Tuning. This methodology can be used on any NLP problem, and has outperformed the state-of-the-art for six different widely studied text classification tasks, with an error reduction of 18-24% on most of the datasets.

This central idea of this paper is Universal Language Model Fine-tuning (ULMFiT), "which pretrains a language model (LM) on a large general-domain corpus and fine-tunes it on the target task using novel techniques." The authors initially train a LSTM on Wikitext-103, a corpus of 28,595 Wikipedia articles. They then fine-tune the language model using specific data for the task they are working on, and finally train a classifier on top of the language model. The largest performance gains seem to come from the techniques used to fine-tune the language model, including discriminative fine-tuning and triangular learning rates. The authors test their methodologies on a number of common text

classification tasks, and outperformed the state-of-the-art for each task. They also conducted experiments, and concluded that each of the steps in their described process is necessary to achieve optimal results, including pre-training a language model on a large corpus, fine-tuning that model on the specific dataset, and training the classifier on top of the language model.

II. Tweet2Vec: Character-Based Distributed Representations for Social Media

By Bhuwan Dhingra, Zhong Zhou, Dylan Fitzpatrick, Michael Muehl, & William W. Cohen

Unstructured text, specifically that originating from social media sources, is generally littered with issues that many traditional word vectorization methods have trouble dealing with. Things like abbreviations, spelling mistakes, and special character usage are all common in this type of text. The authors of the paper offer up a new method of document embedding, tweet2vec, which trains on text at the character level unlike most common techniques which try to look at the document as a word model. The system works by breaking the input tweet down into a sequence of encoded characters. Following this, the sequence is used as input for a GRU, or Gated Recurrent Unit, where the end goal is to predict a hashtag following the tweet. The authors compared this system to a word2vec-like model they implemented for the same tweet data and found that the predicted hashtags were consistently more accurate in the tweet2vec model compared to the traditional word embedding method.

III. Deep Neural Network Architecture for Character-Level Learning on Short Text

By Joseph D. Prusa, Taghi M. Khoshgoftaar

In this paper, Prusa and Khoshgoftaar approach the problem of developing a vectorization method for short text documents by using neural networks on the character-level instead of the more popular word- or even sentence-level models. The paper proposes using convolutional neural networks (specifically networks with convolutional layers followed by dense layers and those with convolutional layers followed by long short-term memory layers). Their results showed that convolutional layers seem beneficial in addressing the lack of length in tweets, especially when the language (like the almost unintelligible abbreviations of those in the financial sector) is unknown.

The paper's dataset was the sentiment140 corpus of 1.6 million labeled positive- and negative-sentiment tweets (in equal numbers). They use two different types of character embedding (1-Hot and 2D) and test their effect in addition to the varying network architectures. Results indicated that the networks with LSTM layers were outperformed (in time and accuracy) by the networks using solely convolutional followed by dense layers. A max of 50 epochs was run for each model, and the time for each epoch was also measured, which the aforementioned models also did better on. Though

this work is done in a binary case, it is likely possible to extend to a multi-class case, which we hope to accomplish in our project.

IV. Numeral Understanding in Financial Tweets for Fine-grained Crowd-based Forecasting

By Chung-Chi Chen, Hen-Hsen Huang, Yow-Ting Shiue, Hsin-Hsi Chen

The motivation for the research in this paper is that numerals in financial textual data, especially social media data, are valuable information that could potentially aid in financial decision making. There has been a considerable rise in social trading and this has enabled the investors make better decisions on their trade. Financial social media data can provide an insight into movement of the stock market, financial health of publicly traded companies and importantly help understand the sentiments of the investors. The authors of the paper propose a taxonomy for fine-grained opinion mining on financial social media data, provide a methodology for feature extraction annotate the numerals in the social media data, conduct experiments to compare different classification models and use FinNum's Numeral Corpus to evaluate the models. The goal of the experiment to classify the numerals found in the tweets in one of the 7 categories and further classify the tweets along with the first level of classification into 17 types of sub-categories.

The authors used a character based approach to overcome the problem of informal word forms such as abbreviations and word based approach to capture semantic and syntactic patterns to represent data. They employed SVM, CNN and RNN to model the data with micro and macro averaged F-scores to compare the models. The word based CNN model performed the best out of all models in the coarse-grained classification (Broad Categorical Classification) task and character based CNN model performed the best in fine-grained classification (Sub-Categorical Classification).

3. Data

Raw data extracted from the API contains 4429 tweets accompanied with the target numeral captured in the tweet, the category of the numeral and the subcategory of the numeral. The category and subcategory are the primary and secondary labels. The main problem that we faced with the data is that some of the tweets contained more than one numeral. Below is a snapshot of the data set.

Tweet	Category	Sub-Category	target_num
$ARNA APD334 for Amyotrophic Lateral Sclerosis...	Product Number	Product Number	334
$OCLR Noob investor that i am, put a 7.38 stop...	Monetary	stop loss	7.38
*ESF*ESFSPY Bias-2 bearish and the DLT-1 DRR ar...	Product Number	Product Number	1

$TMUS its acquisition of Layer3 TV The purchas...	Product Number	Product Number	5
*TWTRBuy*TWTRBuyWSTL 68c up 14% 4 time avg vol. ...	Percentage	relative	14

Below is a snapshot of the data size

Characteristics	Train	Validation	Test
Size	1.1 MB	0.1 MB	0.1 MB
Rows	6706	744	1232
Columns	4	4	2
Number of Tweets	3980	449	768
Labelled	Yes	Yes	No

4. Pre-Processing Step

As the data extracted was in the form of a JSON object, we created copies of the tweets in the case of multiple numerals which ensured that each occurence of numeral in a tweet was represented as a separate row in the data. We then parsed the JSON object into a machine-legible format using pandas. As the data had only textual content, we trained a word vector or word embedding model using Gensim's Fasttext library to represent each word in the tweet in a 100 dimensional euclidean space.

Preprocessing for Feedforward Neural Network

As the FFNN requires a feature matrix with each row representing a tweet, we decided to average the word vector dimensions for all the words in a tweet to have a single vector representation for each tweet. We additionally decided to build a matrix of word vectors for each tweet and flatten it into a single vector representation for input.

Preprocessing for Convolutional Neural Network

In order to convert the dataset into an image or a tensor, we decided to represent the words as columns and 100-dimensional word vectors as rows. The maximum number of words in the tweet across the data set was 26 and we used that as the height of the image. For the tweets that had a word count less than 26, we padded the matrix with zeros to ensure uniform dimensions across all tweets. Table below illustrates the image representation of each tweet

$Word_1$	$n_{1,1}$	$n_{1,2}$	.	.	$n_{1,100}$
$Word_2$	$n_{2,1}$	$n_{2,2}$	.	.	$n_{2,100}$
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$Word_{26}$	$n_{26,1}$	$n_{26,2}$	.	.	$n_{26,100}$

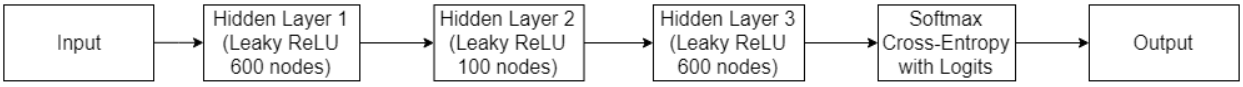
where $n_{i,j}$ is a word vector for the ith word and jth dimension. Additionally, this process could be done for the characters as well.

5. Baseline Model Description

For this report, we trained and tested model only for the primary labels in the data (category of numeral). We used random forest classification as the baseline model and trained it on the averaged word vector representation for each tweet. The model predicted 42% of the validation tweets with the correct category and the model did not learn too much from some of the labels as majority of the tweets belonged to just three categories (Over 80%).

6. Method/Analysis

To implement a feedforward neural network, we created a simple model with 3 layers (600, 100, 600 nodes in each successive layer), which was our best performing out of the tested FF models. We used a leaky ReLU activation function for each layer, terminating in a logits calculation to compute cross-entropy as our loss function. Due to its efficiency at reaching maximum accuracy in the fewest number of epochs, we used the Adam algorithm as our optimizer and added no dropout layers.



Mathematical Equation

Input - X

Layer 1 (L1) - $Z_1 = \text{Leaky ReLU}(w_1X + b_1)$

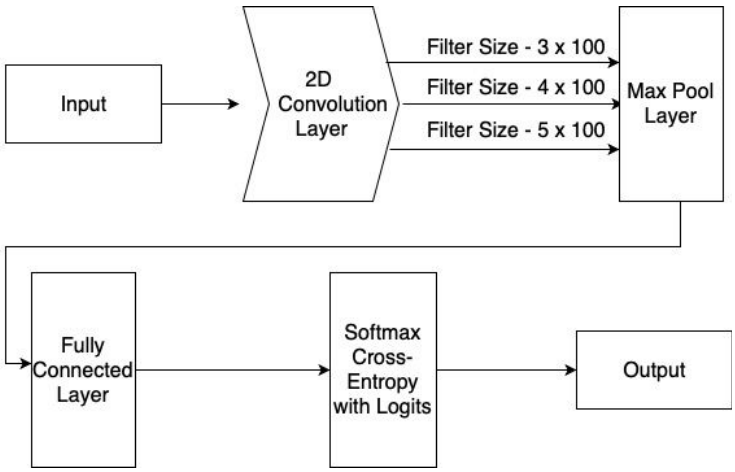
Layer 2 (L2) - $Z_2 = \text{Leaky ReLU}(w_2Z_1 + b_2)$

Layer 3 (L3) - $Z_3 = \text{Leaky ReLU}(w_3Z_2 + b_3)$

Layer 4 (Softmax) - $\widehat{y} = \text{Softmax}(Z_3)$

Final Equation - $\widehat{y} = \text{Softmax}(L3(L2(L1(X))))$

Our convolution neural network consists of one 2D convolution layer, a max pool layer, a fully connected layer and a softmax unit. The forward propagation starts with the inputs being passed through the 2D convolution layer. We used three filters with different sizes (3 X 100, 4 X 100, 5 X 100) for traversing each data point. The output of the convolution layer from all the filters is summed up and passed through the maxpool layer. The maxpool layer is followed by a fully connected layer of 100 units which uses ReLU as the activation function. Finally, we used a softmax classifier to make the predictions. We used cross-entropy loss function and adam optimizer for the model.



7. Result Analysis

For our feedforward neural network, we used the averaged word vectors as our input, which led to a significant loss of information, explaining the worse performance when compared with our convolutional neural networks. When testing with the standard ReLU activation function, we found that a significant amount of values were being ignored because they were negative, so we settled on a leaky ReLU activation function for each layer, instead, leading to a higher accuracy on the validation set. We also attempted to add dropout layers after each activation layer and found that it actually decreased the model's performance on the validation set, so we decided to not use any dropout layers. When modifying the architecture of the model by adjusting the number of layers and number of nodes in each layer, we found that three layers is our sweet spot with respective node numbers of (600, 100, 600). More layers led to overfitting on the training set, while fewer layers didn't pick up on any deeper trends, with a similar pattern for the number of nodes in each layer.

For the convolution neural network, we implemented a word embedding based model with 1 convolution layer and 1 maxpool layer. This model performed better than all the feed forward networks we had built. In general, character embeddings are known to represent more granular information than word embeddings. We tried a 1-layer CNN with character embeddings, and the

accuracy improved significantly. We also implemented a 2-layer CNN with character embeddings which delivered an accuracy of 81.2%. Both accuracy and loss indicate the 2-layer CNN as the best model. We summarize the performance of each model on the validation set below.

Model	**Val Loss**	**Val Accuracy**
Baseline (Random Forest)	1.88	42.1%
FFNN 2-layer	1.26	47.8%
FFNN 2-layer with dropout	1.28	48.2%
FFNN 3-layer	1.30	49.8%
FFNN 3-layer with dropout	1.31	47.5%
FFNN 3-layer with 1000 nodes in first layer	1.28	47.5%
Word Embedding CNN 1 - layer	1.86	58.6%
Character Embedding CNN 1 - layer	0.67	75.6%
Character Embedding CNN 2 - layer	0.62	81.2%

8. Improvement

a. Regularization

To address the problem of regularization in our FFNN, we attempted a number of different architectures, adjusting both the number of layers and the number of nodes in layers and found our best model at 3 layers with (600, 100, 600) nodes, respectively. We also attempted to add dropout layers in order to better generalize to the validation set but found that it only excluded information from our already relatively small dataset, ultimately lowering the validation accuracy, so we opted to not include dropout. Additionally, batch normalization was implemented after convolutional layers in multilayer CNNs.

b. Model size and layers

For our feedforward network, we attempted adjusting the number of nodes in our layers for an array of values but found that as we increased the number, our model tended to overfit on the training data, and as we decreased the number, it couldn't pick up on deeper trends (a picture of an example at (1000, 100) nodes is attached). We also varied the number of layers but found

a similar issue at numbers of layers different than three; two was not enough to learn deeper trends and four layers would overfit.

For CNNs, we used a 1-layer CNN with 1 fully connected layer (100 units) and 1 maxpool layer as the baseline model. To improve the model, we added in another convolution layer and bumped up the number of units in the fully connected layer to 800 which positively impacted the accuracy. In addition to tweaking the layers, we used three different filters with varying heights for convolution layers and sum the outputs from all filters before it gets passed to the next layer. Using multiple filters also improved the performance of the model.

c. Optimization algorithms

We tested several different optimization algorithms to train our models, stochastic gradient descent, sgd with momentum, and adam. We tested the optimizers on a single-layer CNN keeping other parameters constant. The best performing optimizer was the Adam optimizer. Validation loss quickly reached a minimum in nine training epochs, and had the lowest validation loss of the three methods tested. The momentum optimizer performed well, but still had inferior results to the adam optimizer. The best validation loss was a bit higher than the adam optimizer and it took longer to converge. Stochastic gradient descent trained much more slowly than the other optimizers, requiring an increased learning rate. The higher learning rate caused the large fluctuations in the validation loss, making it difficult to decide when the model is fully trained. Stochastic gradient descent was clearly the inferior optimizer for this model. See below for a summary of our results.

Algorithm	**Best Validation Loss**	**Training Epochs to Reach Optimum**
Adam	.667	9
Momentum	.689	19
Stochastic Gradient Descent	.693	35

d. Choice of embedding (optional)

To represent the tweets for this project phase, two types of embeddings were used. The first embeddings used were FastText word embeddings. These embeddings were chosen because of their known flexibility and usefulness in being able to represent unstructured text, where

spelling mistakes and inconsistent abbreviations are common. The FastText model was built using Gensim and trained on the corpus of 38,000 financial tweets provided for the NLP projects The model was then used to generate vectors for all words in every tweet in our corpus. Character embeddings were also used. For this embedding type it is common to create one hot encoded vectors built for every character in the corpus, but for our implementation we chose to implement log-of-m embeddings due to their compact width.

e. Any other cool thing you can think of (optional)

To maximize performance, we implemented a convolutional neural network that used multiple filter sizes instead of a uniform filter size. This required extra engineering work on our part, but allowed the network more options in detecting features to be used in the model. This implementation borrows ideas from Inception network, in which different filter sizes are applied to the same convolutional layers of two dimensional CNNs. The idea of applying varying layers to text is that different sequences of words, known as n-grams, can capture different meaning in the text. By applying filters over different numbers of words these important sequences can be captured.