Machine Learning - CS582

Group 2:
986191 - Boldkhuu Batbaatar
986264 - Batjargal Bayarsaikhan (Alex)
986159 - Unubold Tumenbayar

PROJECT REPORT

Sentiment Analysis

May 25, 2018

Introduction

Project idea

We developed a mobile app that tells if someone is pessimistic or optimistic using Sentiment Analysis based on his/her Facebook posts history. A user must log in to the app via Facebook and must allow required access permissions. Our app considers the person who usually posts positive things as optimistic, otherwise as pessimistic.

We used various Machine learning algorithms from naive to advanced and compared their results. The best algorithm (or combined) was used for our mobile app.

Project goal

- Learn how to use machine learning algorithms in Natural Language Processing
- How to deploy machine learning model as a web service
- How to apply machine learning model to a mobile application

Sentiment analysis

Sentiment analysis determines the emotional tone behind a piece of text which refers to the use of Natural Language Processing to identify, extract affective states and subjective information. For example, ”This movie is so good” which is positive and ”That book was waste of time” which is a negative sentence.

Natural Language Processing (NLP)

NLP is all about creating systems that process or understand language in order to perform certain tasks. For example:

- Question Answering - The main job of technologies like Siri, Alexa, and Cortana
- Machine Translation - Translating a paragraph of text to another language

- Speech Recognition - Having computers recognize spoken words
- Sentiment Analysis

Data set

We used "Sentiment Labelled Sentences Data Set" from UCI machine learning repository that was created for the Paper "From Group to Individual Labels using Deep Features", Kotzias et. al, KDD 2015.

It contains sentences labeled with positive (1) or negative sentiment (0). There are 500 positive and 500 negative sentences that come from IMDB, Amazon, and Yelp about movies, products, and restaurants.

Tools

Machine learning algorithms:

- Word2Vec
- Gradient descent
- Deep learning
- Recurrent Neural Networks
- Long Short Term Memory

Programming language: Python, Javascript

Frameworks: Tensorflow, Node.js, React Native

Additional libraries: NumPy, Jupyter, matplotlib, Facebook API

Approaches we used

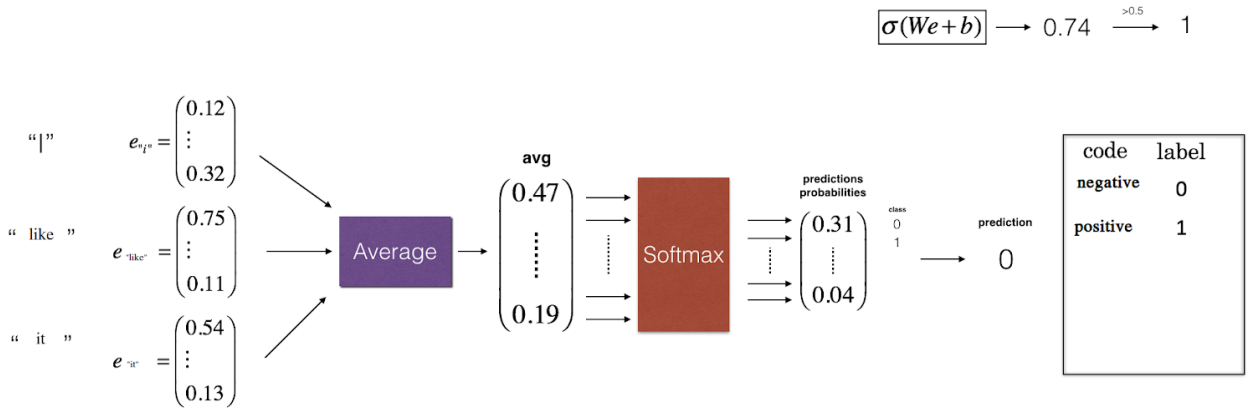
Naive approach

As a first approach, we used a simple classification and Word embedding for analyzing the sentiment. Well known Word embedding tool Word2Vec was chosen for converting the words to word vectors.

Word2Vec

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.

Following diagram shows how first approach works:



Disadvantages

- **Not accurate** - Using simple classification with word2vec has a low accuracy.

- **Does not consider connection** - It does not consider on a connection between the words. e.g. Negations: Good - Positive, Not good - Negative.
- **Does not consider impact** - It does not determine whether which the parts of sentences are important or not.

Improved approach

To solve above major issues, we used Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) algorithms.

Recurrent Neural Network (RNN)

A RNN is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This solves our first issue which is a connection problem between the words.

Long Short Term Memory (LSTM)

Long short-term memory (LSTM) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell. This solves our second issue which is about ignoring unimpactful part of the sentences.

How to run

You can find all python notebooks and application codes from a git repository.

Python notebooks

There are five notebooks in '04. Notebooks' section. Jupyter notebook is required in order to run.

- 01. Word2Vec implementation in Python using Numpy (Scratch).ipynb
  Implementation of Word2Vec from Scratch using Numpy
- 02. Word2Vec implementation in Python using Tensorflow.ipynb
  Implementation of Word2Vec using Tensorflow
- 03. Recurrent Neural Network from Scratch.ipynb
  Implementation of Recurrent Neural Network from Scratch using Numpy.
- 04. Sentiment Analysis with a Baseline Classification.ipynb
  A naive implementation of Sentiment analysis.
- 05. Sentiment analysis with LSTM.ipynb
  Amore sophisticated model with LSTMs using word embeddings in Tensorflow.

Applications

Our project consists of two sub applications, a backend service and a mobile app. There are the applications' codes in '05. Applications' section.

Backend-service

The service is written on Python using Web.py web framework. Its responsibilities are that to receive a request with an user's facebook API token from mobile app and to get the user information from facebook using the token.

Docker Setup

- Install Docker
- Run 'git clone https://github.com/bbatjargal/sentiment-analysis'
- Open docker terminal and navigate to /path/to/sentiment-analysis/05. Applications/backend-service
- Run docker build . -t sentiment-api
- Run docker run -p 8180:8180 sentiment-api
- Access http://0.0.0.0:8180/sentiment?inputtext=i love it from your browser [assuming you are on windows and docker-machine has that IP. Otherwise just use localhost]
- OR access 'http://0.0.0.0:8180/getfriendinfo?accesstoken=YOUR_FB_ACCESS_TOKEN&userid=YOUR_FB_USER_ID' from your browser [assuming you are on windows and docker-machine has that IP. Otherwise just use localhost]

Native Setup

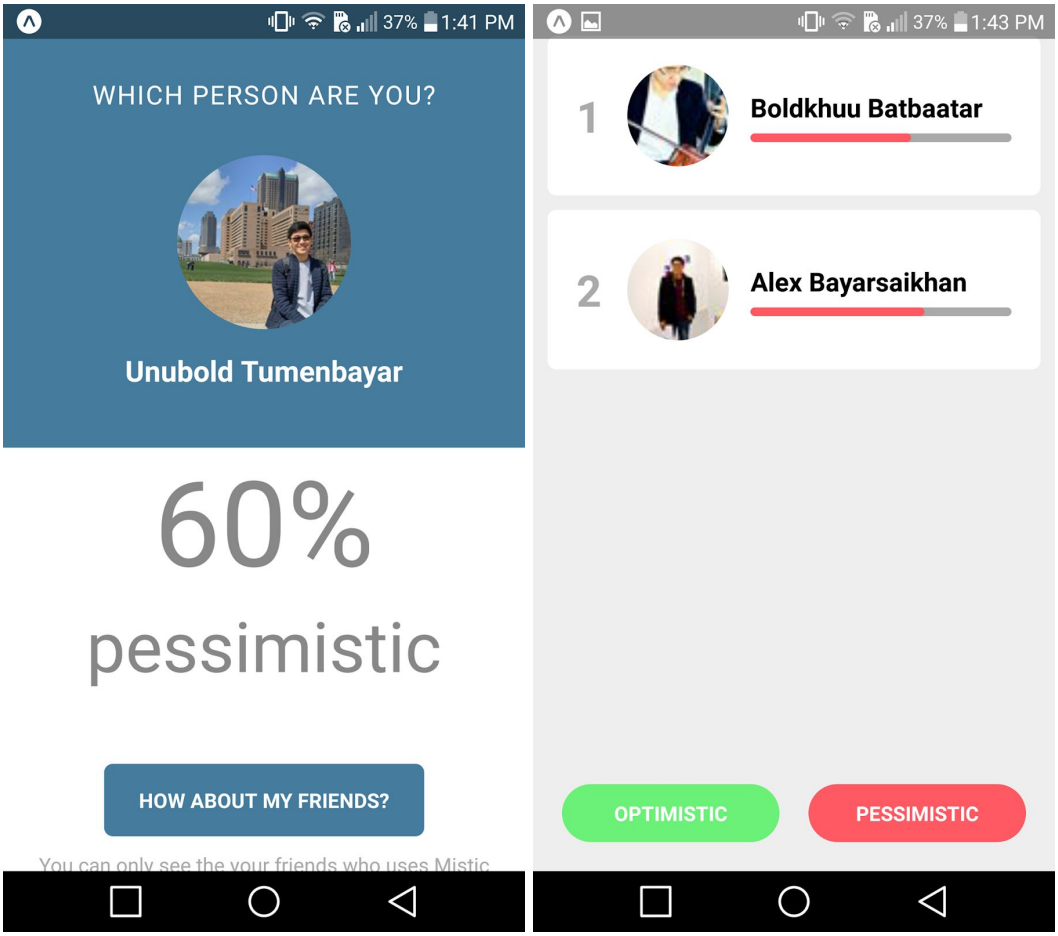
- Anaconda distribution of python 3.6
- 'pip install web.py==0.40-dev1' which installs Web.py for Python 3

For local testing

- Open terminal and navigate to /path/to/sentiment-analysis/05. Applications/backend-service/sentiment
- Run 'python sentiment_api.py'

Mobile app

- Using terminal, go to the “mobile” directory
- ‘npm install’ or ‘yarn’ to install required packages
- Create .env configuration file from .env.sample and configure it with your information.
- Run ‘exp start’ to start the mobile server
- It shows useful information on the terminal how to run it on your mobile device or on simulator



Result and discussion

Sentiment Analysis is implemented in two different approaches, using a simple network with a softmax classifier and using a recurrent neural network with LSTMs.

In the first approach with a softmax classifier, the result was not able to classify a simple negation sentence such as ‘I don’t like it’ etc. In order to determine correctly that a sentence whether positive or not, the approach with LSTMs is chosen for the mobile app that determines someone whether the optimistic or pessimistic based on his/here Facebook posts.

Moreover, the naive model did not correctly label "I do not like it" but our implementation with LSTMs got it right. The current model in the '05. Sentiment analysis with LSTM.ipynb' still isn't very robust at understanding negation (like "not like, don't like") because the training set is small and so doesn't have a lot of examples of negation. But if the training set were larger, the LSTM model would be much better than the naive model at understanding such complex sentences.

Summary

Sentiment analysis can be implemented using simple classification with softmax and word2vec but it has the some drawbacks which mentioned above. A Recurrent Neural Network with LSTM is more powerful and robust type of neural networks. It is the most promising approach for Sentiment Analysis at the moment.

References

- Adit Deshpande. (July 13, 2017). Perform sentiment analysis with LSTMs, using TensorFlow. https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow
- Sebastian Ruder. (September 02, 2018). An overview of gradient descent optimization algorithms. http://ruder.io/optimizing-gradient-descent/
- McCormick, C. (April 19, 2016). Word2Vec Tutorial - The Skip-Gram Model. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- Christopher Olah. (August 27, 2015). Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- NSS. (June 4, 2017). An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Papers

1. **Word2Vec**

   Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. It was introduced in 2013 by team of researchers led by Tomas Mikolov at Google - Read the paper from https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

   Additional reading from
   https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/.

2. **Visualizing and Understanding Recurrent Networks**

   Andrej Karpathy, Justin Johnson, Li Fei-Fei
   (Submitted on 5 Jun 2015 (v1), last revised 17 Nov 2015 (this version, v2))

   Recurrent Neural Networks (RNNs), and specifically a variant with Long Short-Term Memory (LSTM), are enjoying renewed interest as a result of successful applications in a wide range of machine learning problems that involve sequential data. However, while LSTMs provide exceptional results in practice, the source of their performance and their limitations remain rather poorly understood. Using character-level language models as an interpretable testbed, we aim to bridge this gap by providing an analysis of their representations, predictions and error types. In particular, our experiments reveal the existence of interpretable cells that keep track of long-range dependencies such as line lengths, quotes and brackets.

   https://arxiv.org/pdf/1506.02078.pdf

3. Long Short Term Memory

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997.

http://www.bioinf.jku.at/publications/older/2604.pdf

Additional reading from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

4. Automated Hyperparameter Tuning for Effective Machine Learning

Patrick Koch, Brett Wujek, Oleg Golovidov, and Steven Gardner
SAS Institute Inc, 2017

Machine learning predictive modeling algorithms are governed by "hyperparameters" that have no clear defaults agreeable to a wide range of applications. The depth of a decision tree, number of trees in a forest, number of hidden layers and neurons in each layer in a neural network, and degree of regularization to prevent overfitting are a few examples of quantities that must be prescribed for these algorithms. Not only do ideal settings for the hyperparameters dictate the performance of the training process, but more importantly they govern the quality of the resulting predictive models. Recent efforts to move from a manual or random adjustment of these parameters include rough grid search and intelligent numerical optimization strategies.

https://pdfs.semanticscholar.org/ce60/a2a90be9ba9088f91b1bd51a2c1cabade1ae.pdf

Additional reading from
https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

5. An overview of gradient descent optimization algorithms

Sebastian Ruder

(Submitted on 15 Sep 2016 (v1), last revised 15 Jun 2017 (this version, v2))

Gradient descent optimization algorithms, while increasingly popular, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by. This article aims to provide the reader with intuitions with regard to the behaviour of different algorithms that will allow her to put them to use. In the course of this overview, we look at different variants of gradient descent, summarize challenges, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent.

https://arxiv.org/pdf/1609.04747.pdf

Additional reading from http://ruder.io/optimizing-gradient-descent/.