Word2Vec: Learning Text Features for Sentiment Analysis

Team 3: weekly update

Nov. 11, 2015

Outline

- `Word2Vec` intro
- The experiment
  - Exploring the dataset
  - Workflow
  - Feature Learning and Model Evaluation
  - Classifiers used for the sentiment analysis task
  - Results

Features are important ..

Not so good features

- has color
- moves
- big

Is it possible to identify what's been described above?

Good features

- has four wheels
- has engine
- a box-like shape with ~(1.5m X 3m) dimensions

What about now?

Features for representing *things* . . .

e.g. objects:

- color, width, height, shape, temperature . . .

. . .

Features for representing *words (text)* . . .

e.g. the word ‘great’ ?!

What are some good features to represent words and their meanings?

Historically, statistical based techniques are common (e.g. tf-idf, bag-of-word, n-gram . . .)

> *For instance, counting the word occurrence in text then assign its importance value based on that.*

Word2Vec, The basic idea

Word2Vec is a neural-network-based method used for word embedding in text representation.

> *Where, word **meaning** and **relationships** between words are encoded spatially (in vectors).*

- "The vector for each word is a semantic description of how that word is used in context."
- "Similar words are nearby vectors." i.e. ***grouping***
- *Similarity*: associate words with points in space. The spatial distance between words (vectors) then describes the relation (similarity) between these words. i.e. ***adding*** and ***subtracting***

Example: see word galaxy 2-D visualization.

`Word2Vec`, The model

Predictive model (unlike other count-based 'statistical' model).

Learns from input texts.

Based on two NN methods:

- CBOW: "predicts a word based on its context."
- Skip-gram: "predicts surrounding words given the current word."

Learns analogy

- syntactic: (*quickly* is to *quick* as *slowly* is to *slow*)
  - `quickly - quick + slow = slowly`
- semantic: (*king* is to *man* as *queen* is to *woman*)
  - `king - man + woman = queen`

Dataset overview

- Labeled: 50K IMDB movie reviews
  - sentiment is 0 if movie rating < 5
  - sentiment is 1 if movie rating $>= 7$
- Unlabeled: 50K IMDB movie reviews

The assumption of our sentiment analysis task is that ...

- the sentiment (of all reviews) is either positive '1' or negative '0' (no neutral sentiment). Thus, it is a binary Classification.

  source: Bag of Words Meets Bags of Popcorn @ Kaggle

The experiment workflow

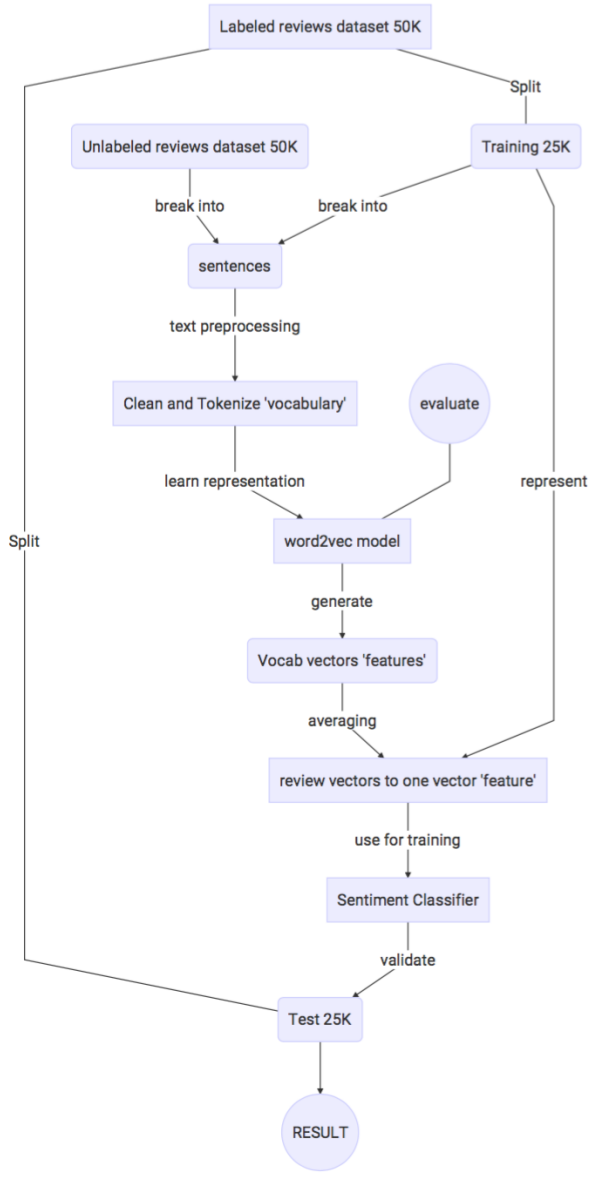


Figure 1: Our feature learning and sentiment workflow

Feature learning, training `word2vec` on IMDB dataset

Input text (to `word2vec`):

- 75K reviews: 25K labeled training reviews + 50K unlabeled reviews
- 811264 is the total number of sentences.

Output vocabulary (of `word2vec` model):

- 16717 vectors (words) of learned vocabulary.

The learned vocabulary is, then, used as input features for the classification task.

Evaluating the learned model

To evaluate our model accuracy we used Google's testing set:

- 19544 syntactic and semantic test examples
- see: **questions-words.txt**

Our model (size=300, window=10, min_count=40) scores:

- Total: 7509 of 19544 test examples
  - Correct: 1613 (21.48%),
  - Incorrect: 5896 (78.52%)

NOTE: word2vec training is an unsupervised task, thus there is no good way to objectively evaluate the result.

Exploring the model results

On some sentiment-related vocabulary:

```
model.doesnt_match("ugly bad beautiful crazy nasty")
# 'beautiful'
model.doesnt_match("good bad nice cool fantastic")
# 'bad'
model.most_similar("awesome", topn=2)
#[('amazing', 0.650), ('fantastic', 0.563)]
model.most_similar("horrible", topn=2)
#[('terrible', 0.740), ('awful', 0.625)]
```

Preparing the classification features

After training `word2vec` model on the IMDB dataset, all reviews' vocabulary are included (vectorized) in the pre-trained model.

Averaging the reviews' words (vectors)

- The feature of a review is calculated as the average (word) vector of that review's (words) vectors.

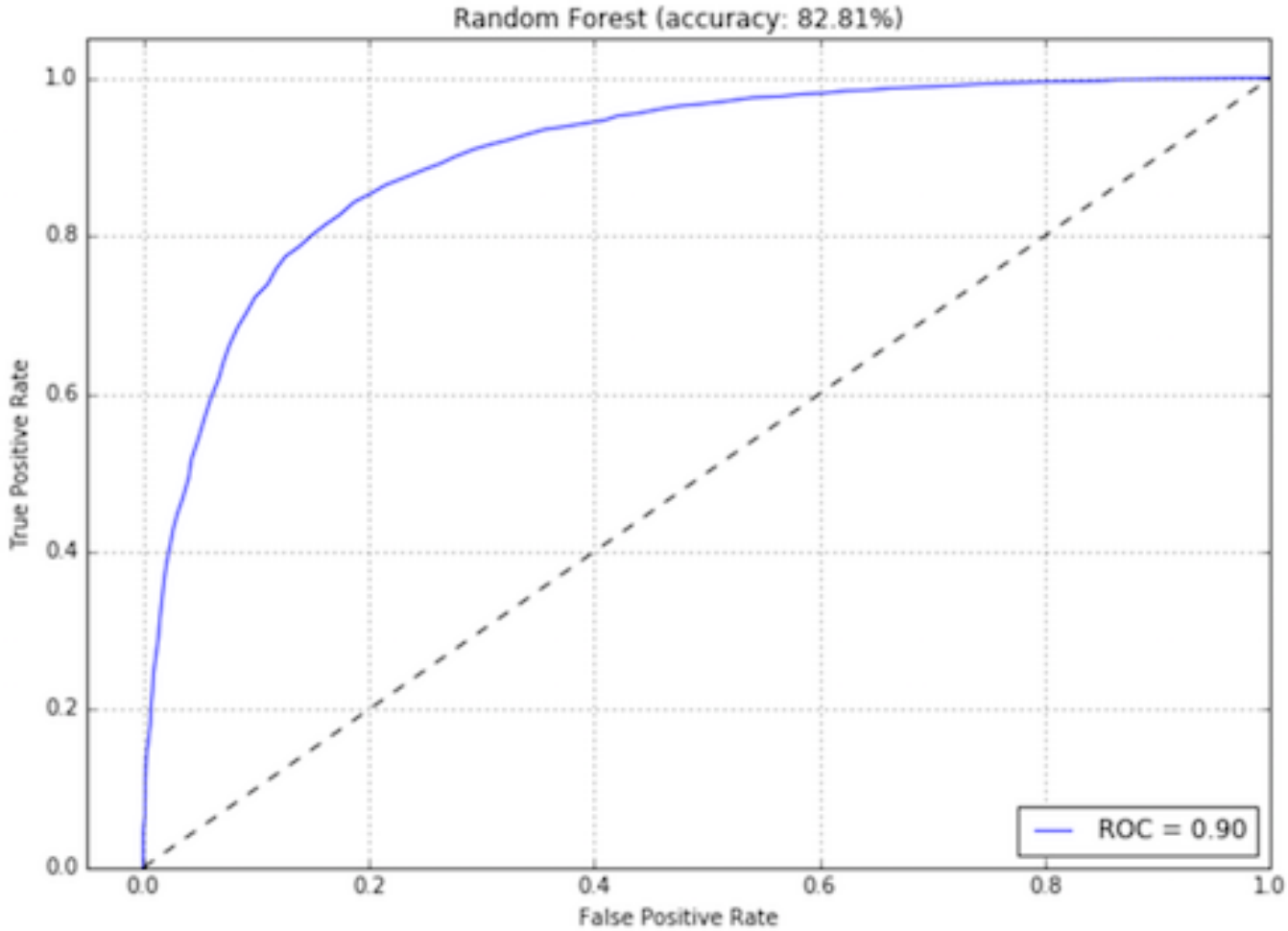
Example review: "good movie!"

```
model["good"]
model["movie"]
get_avg_vec(["good", "movie"])
# [ 0.11087843  0.04439273  0.04955978]
# [-0.08714154 -0.02466415 -0.03661531]
# [ 0.01186845  0.00986429  0.00647223]
```

Linear classifiers results (In progress . . .)

Results on 30% of the test dataset

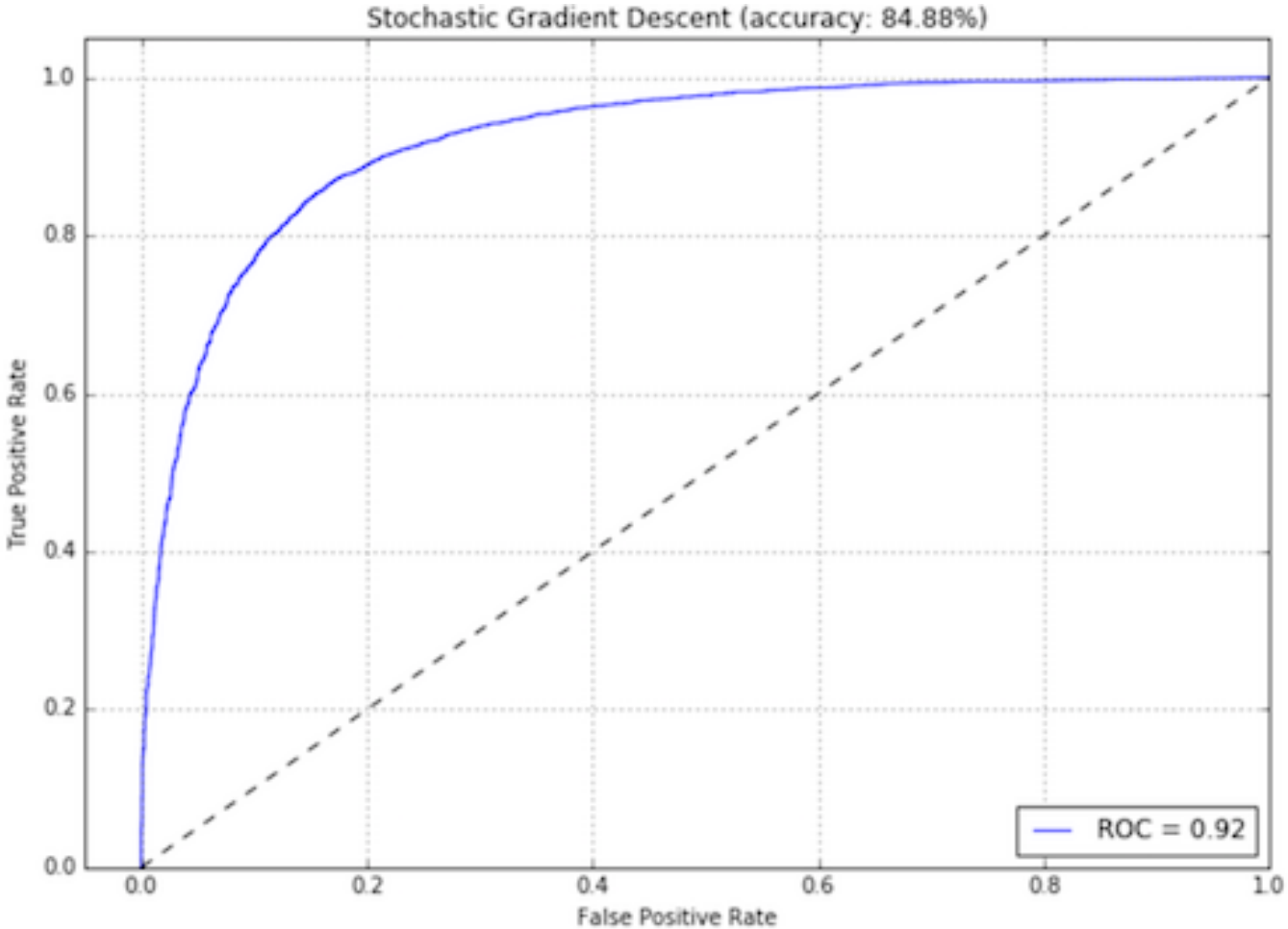
- **Random forest**: 83%
- prediction: 3656 negative, 3844 positive



Linear classifiers results (In progress . . .)

Results on 30% of the test dataset

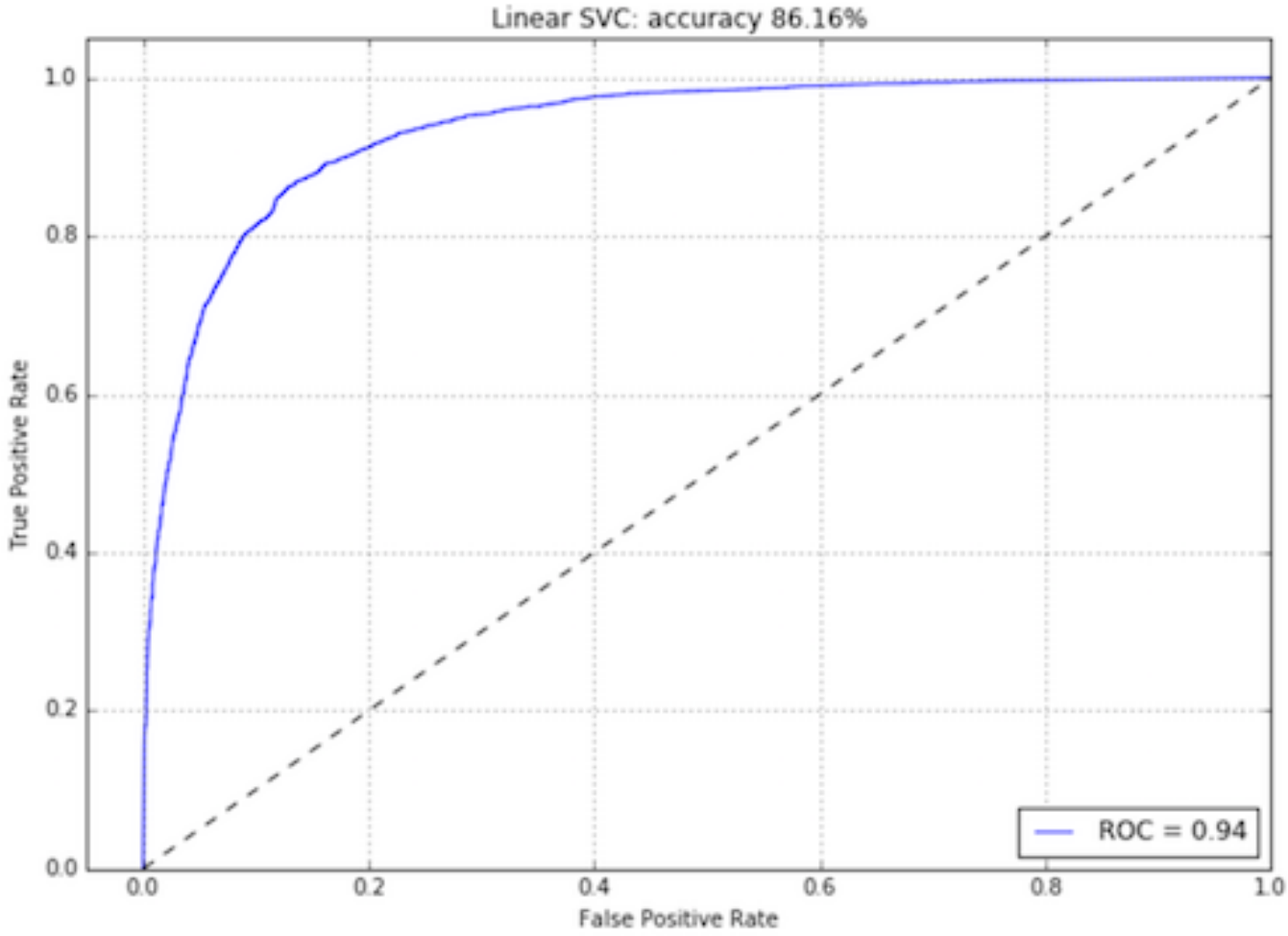
- **SGD**: 84%
- prediction: 3724 negative, 3776 positive



Linear classifiers results (In progress . . .)

Results on 30% of the test dataset

- **LinearSVC**: 86%
- prediction: 3880 negative, 3620 positive



References

1. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.
2. H. Wang, "Introduction to Word2vec And its Application to Find Predominant Word Senses," 2014. August, 1–41.
3. Word2vec Tutorial, RaRe Technologies http://rare-technologies.com/word2vec-tutorial/
4. https://www.kaggle.com/c/word2vec-nlp-tutorial
5. http://scikit-learn.org/
6. http://streamhacker.com/