浙江大学 Zhejiang University

Berkeley UNIVERSITY OF CALIFORNIA

LEHIGH UNIVERSITY

# TextBugger: Generating Adversarial Text Against Real-world Applications

**Jinfeng Li** Shouling Ji Tianyu Du Bo Li Ting Wang

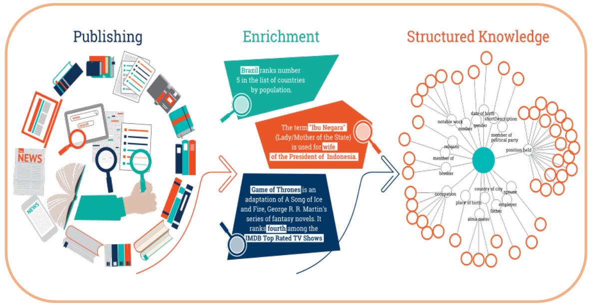
NDSS 2019

# Machine Learning For Natural Language Processing

## Sentiment Analysis


## Information Retrieval

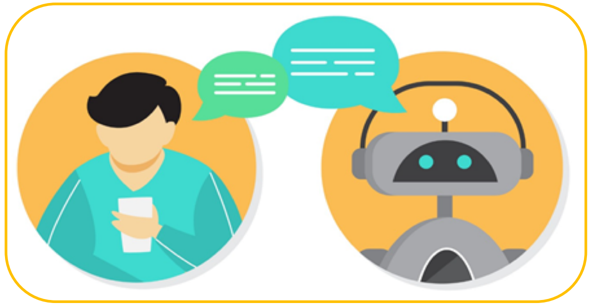

## Information Extraction




## Machine Learning For Multiple Tasks

## Machine Translation


## Question Answering

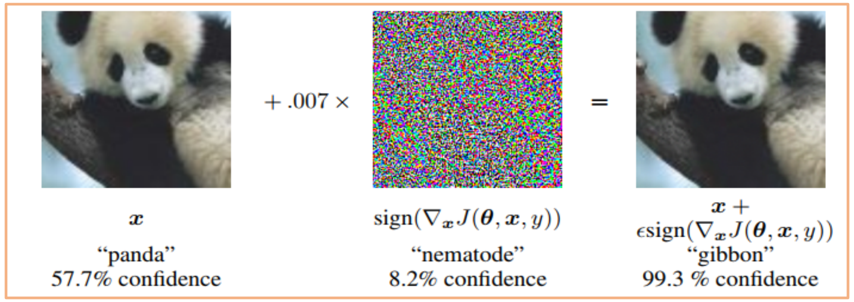


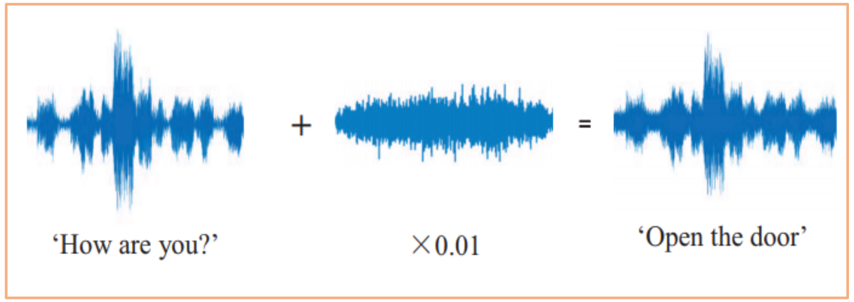
# Machine Learning As A Service For NLP

# Breaking Thing Is Easy

**Recent works have revealed the vulnerabilities of DNNs in image and speech domain**

- The DNNs for image classification are vulnerable to adversarial images. [Goodfellow *et al.*, ICLR'15]
- Automatic speech recognition systems can be broken down by adversarial audios in physical world. [Yuan et al., USENIX'18]




**Do the adversarial examples also exist in text domain?**

**Are the MLaaS for NLP also vulnerable to adversarial examples?**

# Preliminaries

# Adversarial Text

## What is the adversarial text?

Carefully generated by adding small perturbations to the legitimate text.

***Task:*** Sentiment Analysis. ***Classifier:*** Amazon AWS. ***Original label:*** 100% Negative. ***Adversarial label:*** 89% Positive.

***Text:*** I watched this movie recently mainly because I am a Huge fan of Jodie Foster's. I saw this movie was made right between her 2 Oscar award winning performances, so my expectations were fairly high. ~~Unfortunately~~ **Unf0rtunately**, I thought the movie was ~~terrible~~ **terrib1e** and I'm still left wondering how she was ever persuaded to make this movie. The script is really ~~weak~~ **wea k**.

## What is the challenge for generating adversarial texts?

- The discrete property of text makes it hard to optimize.
- Small perturbations in text are usually clearly perceptible.
- Replacement of a single word may drastically alter the semantics of the sentence.

# Related Works For Generating Adversarial Texts

## Gradient-based Methods

- Modifying an input text repetitively until it is misclassified. [Papernot *et al.,* MILCOM' 16]
- Changing one token to another by a gradient-based optimization method. [Ebrahimi *et al.* ,NAACL' 18]
- Perturbing the important words determined by embedding gradient with hand-crafted synonyms. [Samanta *et al.,* arXiv'17]

## Out-of-vocabulary Words

- Breaking machine learning systems down by random character manipulations. [Belinkov *et al*., ICLR' 18]
- Attacking black-box models by applying random character perturbations. [Gao et al. SPW' 18]
- Changing the toxicity score of the texts by adding spaces or dots between characters. [Hosseini et al., arXiv' 17]

# Related Works For Generating Adversarial Texts

## Replace with Semantically/Syntactically Similar Words

- Only replacing words with semantically similar ones. [Alzantot *et al*.., arXiv’ 18]
- Replacing tokens by random words of the same POS tag with a probability proportional to the embedding similarity. [Ribeiro *et al.,* ACL’ 18]

## Other Methods

- Attacking reading comprehension systems by adding distracting sentences to the input document. [Jia *et al.,* EMNLP’ 17]
- Generating adversarial sequence by Generative Adversarial Networks (GANs). [Zhao *et al.,* ICLR’ 18]

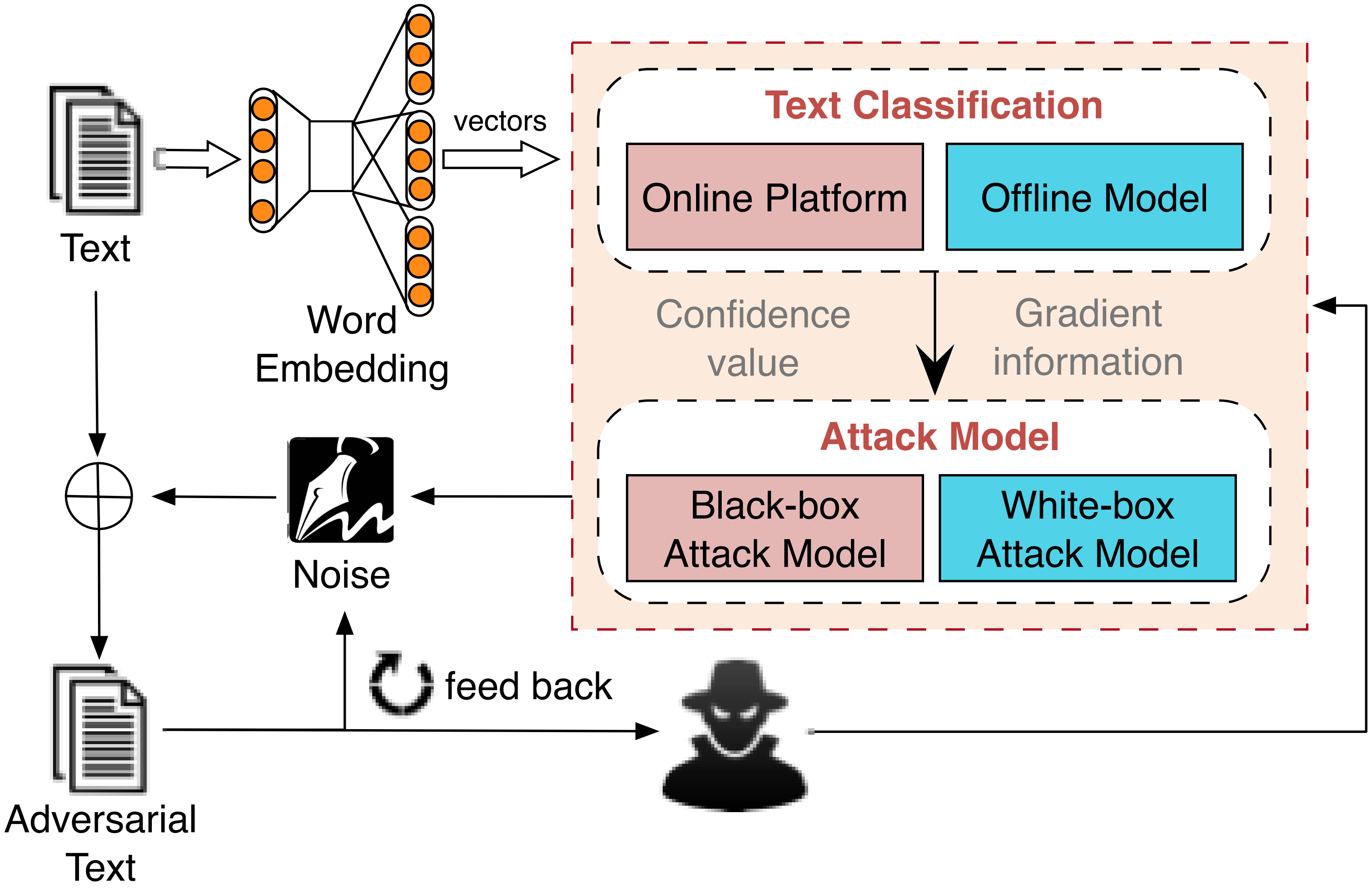
# Limitations

**These works are limited in practice due to at least one of the following reasons:**

- Limited to short texts
- Significantly affect the original meaning
- Need hand-crafted synonyms and typos
- Requires manual intervention to polish the added sentences
- Not computationally efficient

# TextBugger

# Framework For TextBugger



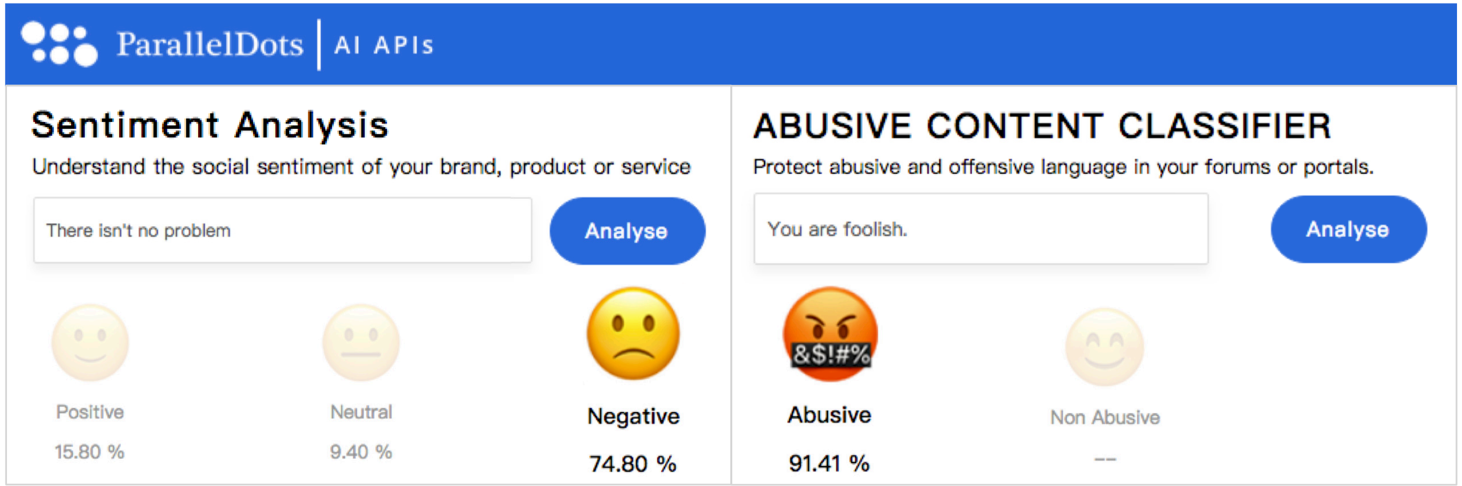
# Threat Model

## White-box

- Have complete knowledge about the targeted model

## Black-box

- Do not know the model architecture, parameters or training data
- Only capable of querying the targeted model with output as the prediction or confidence scores



# Step 1: Finding Important Words

## White-box attack

- Find important words by gradient information.

$$C_{x_i} = J_{\mathcal{F}(i,y)} = \frac{\partial \mathcal{F}_y(\boldsymbol{x})}{\partial x_i}$$

$$J_{\mathcal{F}}(\boldsymbol{x}) = \frac{\partial \mathcal{F}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[ \frac{\partial \mathcal{F}_j(\boldsymbol{x})}{\partial x_i} \right]_{i \in 1..N, j \in 1..K}$$

**Denotes:**

- $\boldsymbol{x}$ is the input text, $x_i$ is the $i^{th}$ word in $\boldsymbol{x}$.
- $\mathcal{F}_j(\boldsymbol{x})$ is the confidence value of the $j^{th}$ class.
- $C_{x_i}$ is the importance of word $x_i$.
- $N$ is the total number of words in $\boldsymbol{x}$.
- $K$ is the total number of classes.

# Step 1: Finding Important Words

## Black-box attack

- Find important sentences

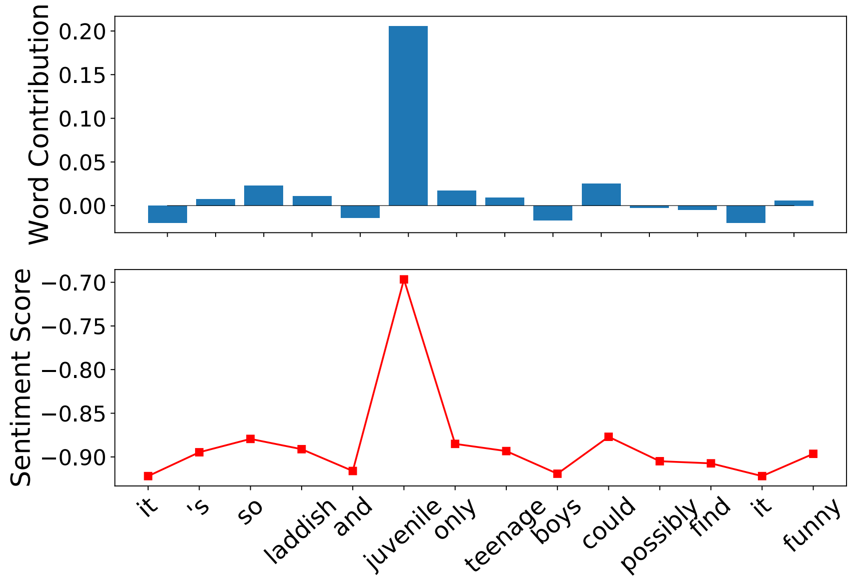
  $C_{sentence}(i) = \mathcal{F}_y(\boldsymbol{s_i})$

  $S_{ordered} \leftarrow Sort(\boldsymbol{s})$ according to $C_{sentence}(i)$

  Delete sentences in $S_{ordered}$ if $\mathcal{F}_l(\boldsymbol{s_i}) \neq y$

- Find important words for each sentence in $S_{ordered}$

  $$C_{w_j} = \mathcal{F}_y(w_1, w_2, \cdots, w_m) - \mathcal{F}_y(w_1, \cdots, w_{j-1}, w_{j+1}, \cdots, w_m)$$




**Denotes:**

- $\boldsymbol{s_i}$ is the $i^{th}$ sentence in the input text $\boldsymbol{x}$.
- $\mathcal{F}_y(\boldsymbol{s_i})$ is $\boldsymbol{s_i}$'s confidence value of the predicted class $y$.
- $S_{ordered}$ is the important sentences set.
- $C_{sentence}(i)$ is the importance of word $\boldsymbol{s_i}$, $C_{w_j}$ is the importance of the $j^{th}$ word in $\boldsymbol{s_i}$.

**Sentence**: *It is so laddish and* ***juvenile****, only teenage boys could possibly find it funny .*

# Step 2: Bugs Generation

**Character-level perturbation**: out-of-vocabulary phenomenon

- **Insert**: Insert a space into the word.
- **Delete**: Delete a random character of the word.
- **Swap**: Swap random two adjacent letters in the word.
- **Substitute-C** (**Sub-C**) : Replace characters with visually similar characters or adjacent characters in the keyboard.

**Word-level perturbation**: nearest neighbor searching in the embedding space

- **Substitute-W (Sub-W)** : Replace a word with its top *k* nearest neighbors in a context-aware word vector space.

| Original | Insert | Delete | Swap | Sub-C | Sub-W |
|---|---|---|---|---|---|
| foolish<br>awfully<br>cliches | f oolish<br>awfull y<br>clich es | folish<br>awfuly<br>clichs | fooilsh<br>awfluly<br>clcihes | fo0lish<br>awfu1ly<br>c1iches | silly<br>terribly<br>cliche |

# Step 3: Replacing Important Word By Generated Bug

## Optimal bug selection

- choose the optimal bug according to the change of the confidence value

$$\boldsymbol{candidate(k)} = \text{replace } w \text{ with } b_k \text{ in } \boldsymbol{x}$$
$$score(k) = \mathcal{F}_y(\boldsymbol{x}) - \mathcal{F}_y(\boldsymbol{candidate(k)})$$

## Important word replacement

- Replace the important word by the selected optimal bug
- Repeat until “convergence”
  - the semantic similarity is below the threshold
  - the new text is misclassified by the classifier

# Attack Evaluation

# Case Study

Sentiment Analysis

Toxic Content Detection

# Attack Evaluation: Sentiment Analysis

## Dataset

- **IMDB:** 50,000 positive and negative movie reviews
- **Rotten Tomatoes Movie Reviews (MR):** 5,331 positive and 5,331 negative snippets

## Targeted Model

- **White-box models:** LR, CNN, LSTM
- **Real-world Online Platforms:**

## Baseline Algorithms

- **White-box:** Random, FGSM+NNS (Nearest Neighbor Search), DeepFool+NNS
- **Black-box:** DeepWordBug

# Attack Evaluation: Sentiment Analysis

## Evaluation Metrics

- **Edit Distance**
- **Jaccard Similarity Coefficient**

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

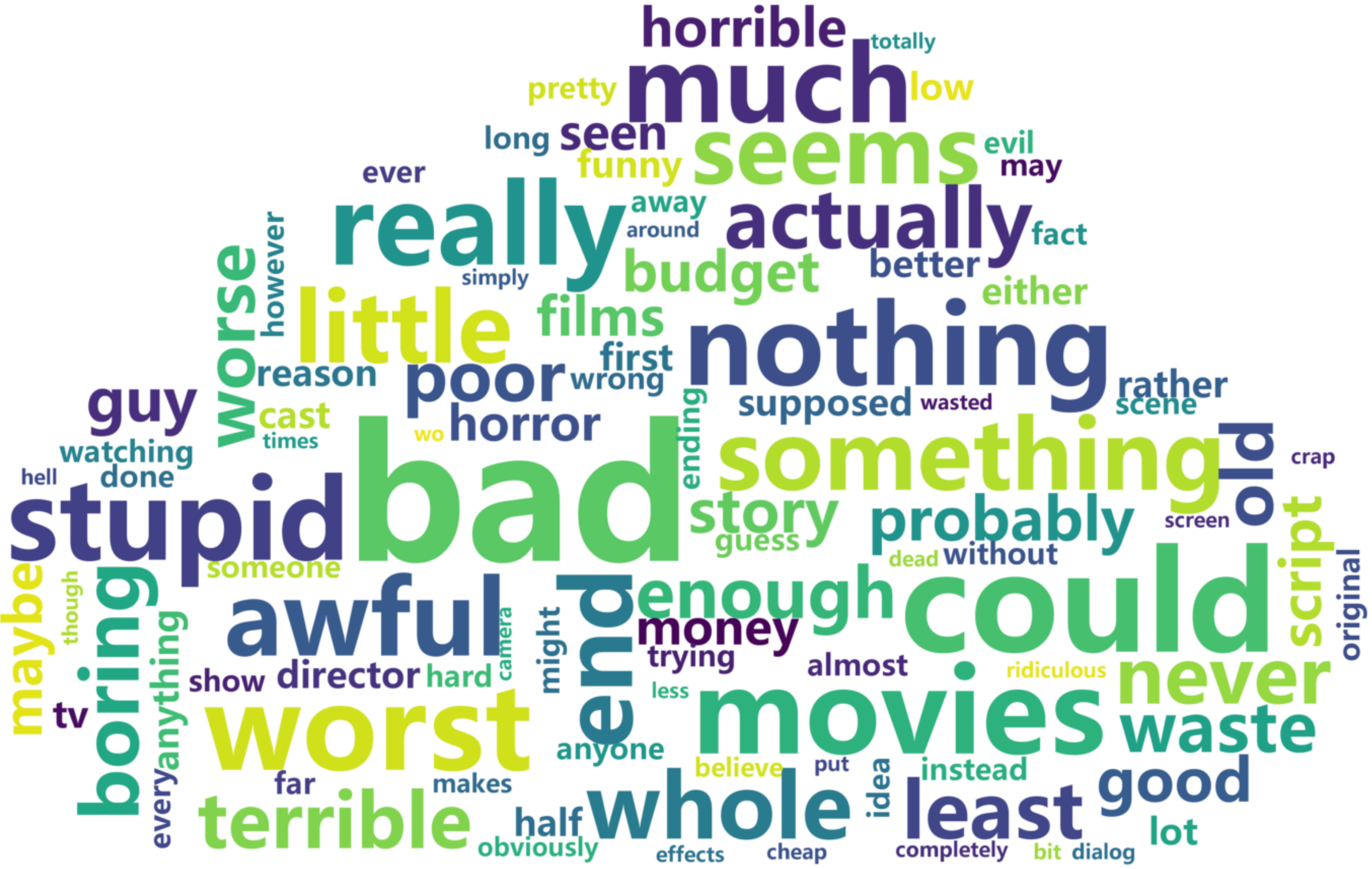
- **Euclidean Distance**

$$d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

- **Semantic Similarity**

$$S(\boldsymbol{p},\boldsymbol{q}) = \frac{\boldsymbol{p} \cdot \boldsymbol{q}}{||\boldsymbol{p}|| \cdot ||\boldsymbol{q}||} = \frac{\sum_{i=1}^{n} p_i \times q_i}{\sqrt{\sum_{i=1}^{n} (p_i)^2} \times \sqrt{\sum_{i=1}^{n} (q_i)^2}}$$

# Important Words Selected By TextBugger



# Generated Adversarial Texts

## Successful Attack Examples

***Task:*** Sentiment Analysis. ***Classifier:*** CNN. ***Original label:*** 99.8% Negative. ***Adversarial label:*** 81.0% Positive.

***Text:*** I love these ~~awful~~ **awf ul** 80's summer camp movies. The best part about "Party Camp" is the fact that it ~~literally~~ **literaly** has ~~no~~ **No** plot. The ~~cliches~~ **clichs** here are limitless: the nerds vs. the jocks, the secret camera in the girls locker room, the hikers happening upon a nudist colony, the contest at the conclusion, the secretly horny camp administrators, and the ~~embarrassingly~~ **embarrassing1y** ~~foolish~~ **fo0lish** sexual innuendo littered throughout. This movie will make you laugh, but never intentionally. I repeat, never.

***Task:*** Sentiment Analysis. ***Classifier:*** Amazon AWS. ***Original label:*** 100% Negative. ***Adversarial label:*** 89% Positive.

***Text:*** I watched this movie recently mainly because I am a Huge fan of Jodie Foster's. I saw this movie was made right between her 2 Oscar award winning performances, so my expectations were fairly high. ~~Unfortunately~~ **Unf0rtunately**, I thought the movie was ~~terrible~~ **terrib1e** and I'm still left wondering how she was ever persuaded to make this movie. The script is really ~~weak~~ **wea k**.

# Attack Performance: Effectiveness And Efficiency

## White-box Attack

TABLE II. RESULTS OF THE WHITE-BOX ATTACKS ON IMDB AND MR DATASETS.

| Model | Dataset | Accuracy | Random | | FGSM+NNS [12] | | DeepFool+NNS [12] | | TEXTBUGGER | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Success Rate | Perturbed Word | Success Rate | Perturbed Word | Success Rate | Perturbed Word | Success Rate | Perturbed Word |
| **LR** | MR | 73.7% | 2.1% | 10% | 32.4% | 4.3% | 35.2% | 4.9% | **92.7%** | 6.1% |
| | IMDB | 82.1% | 2.7% | 10% | 41.1% | 8.7% | 30.0% | 5.8% | **95.2%** | 4.9% |
| **CNN** | MR | 78.1% | 1.5% | 10% | 25.7% | 7.5% | 28.5% | 5.4% | **85.1%** | 9.8% |
| | IMDB | 89.4% | 1.3% | 10% | 36.2% | 10.6% | 23.9% | 2.7% | **90.5%** | 4.2% |
| **LSTM** | MR | 80.1% | 1.8% | 10% | 25.0% | 6.6% | 24.4% | 11.3% | **80.2%** | 10.2% |
| | IMDB | 90.7% | 0.8% | 10% | 31.5% | 9.0% | 26.3% | 3.6% | **86.7%** | 6.9% |

## Remarks

- **Choosing important words to modify is necessary.**
- **Effective:** TextBugger has high attack success rate on all models and performs better than baselines.
- **Evasive :** TextBugger perturbs few words to fool the models.

# Attack Performance: Effectiveness And Efficiency

## Black-box Attack

TABLE III. RESULTS OF THE BLACK-BOX ATTACK ON IMDB.

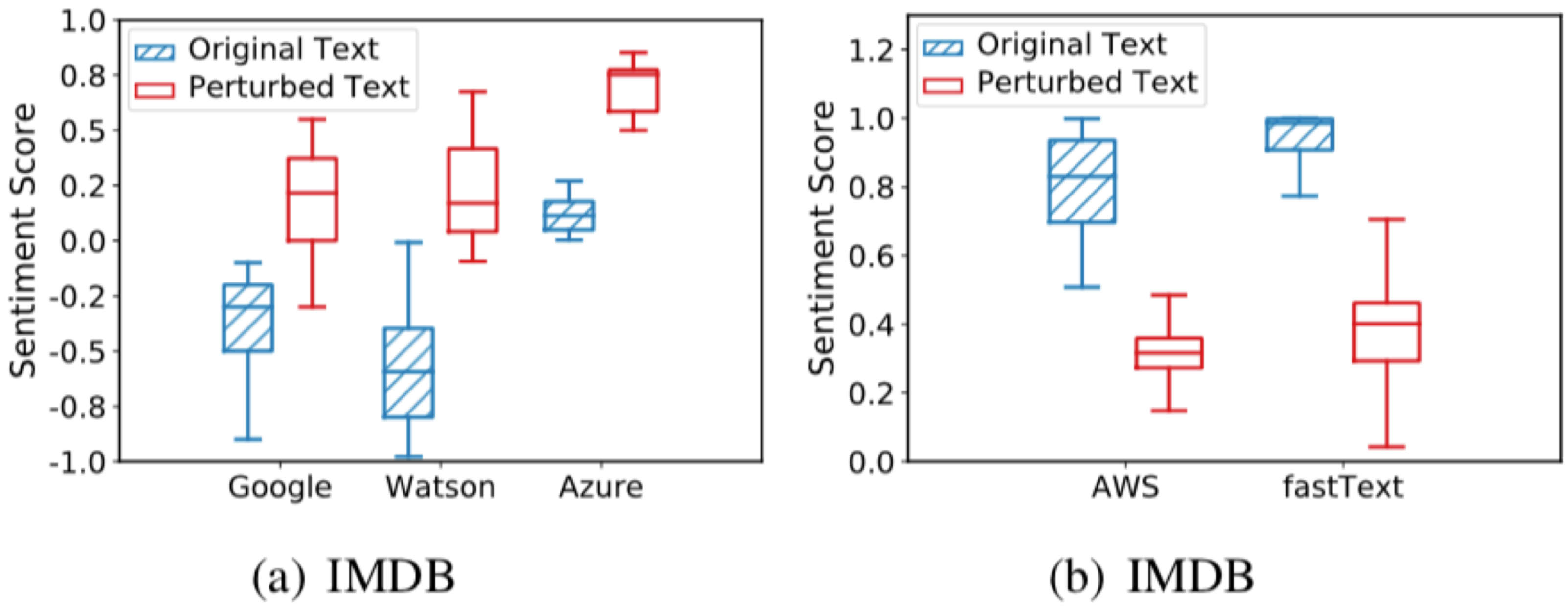
| Targeted Model | Original Accuracy | DeepWordBug [11] | | | TEXTBUGGER | | |
|---|---|---|---|---|---|---|---|
| | | Success Rate | Time (s) | Perturbed Word | Success Rate | Time (s) | Perturbed Word |
| Google Cloud NLP | 85.3% | 43.6% | 266.69 | 10% | **70.1%** | 33.47 | 1.9% |
| IBM Waston | 89.6% | 34.5% | 690.59 | 10% | **97.1%** | 99.28 | 8.6% |
| Microsoft Azure | 89.6% | 56.3% | 182.08 | 10% | **100.0%** | 23.01 | 5.7% |
| Amazon AWS | 75.3% | 68.1% | 43.98 | 10% | **100.0%** | 4.61 | 1.2% |
| Facebook fastText | 86.7% | 67.0% | 0.14 | 10% | **85.4%** | 0.03 | 5.0% |
| ParallelDots | 63.5% | 79.6% | 812.82 | 10% | **92.0%** | 129.02 | 2.2% |
| TheySay | 86.0% | 9.5% | 888.95 | 10% | **94.3%** | 134.03 | 4.1% |
| Aylien Sentiment | 70.0% | 63.8% | 674.21 | 10% | **90.0%** | 44.96 | 1.4% |
| TextProcessing | 81.7% | 57.3% | 303.04 | 10% | **97.2%** | 59.42 | 8.9% |
| Mashape Sentiment | 88.0% | 31.1% | 585.72 | 10% | **65.7%** | 117.13 | 6.1% |

## Remarks

- **Effective:** TextBugger has higher attack success rate against all online platforms than DeepWordBug.
- **Evasive:** TextBugger only perturbs fewer words than DeepWordBug.
- **Efficient:** TextBugger spends less time than DeepWordBug.

# Attack Performance: Change Of Confidence

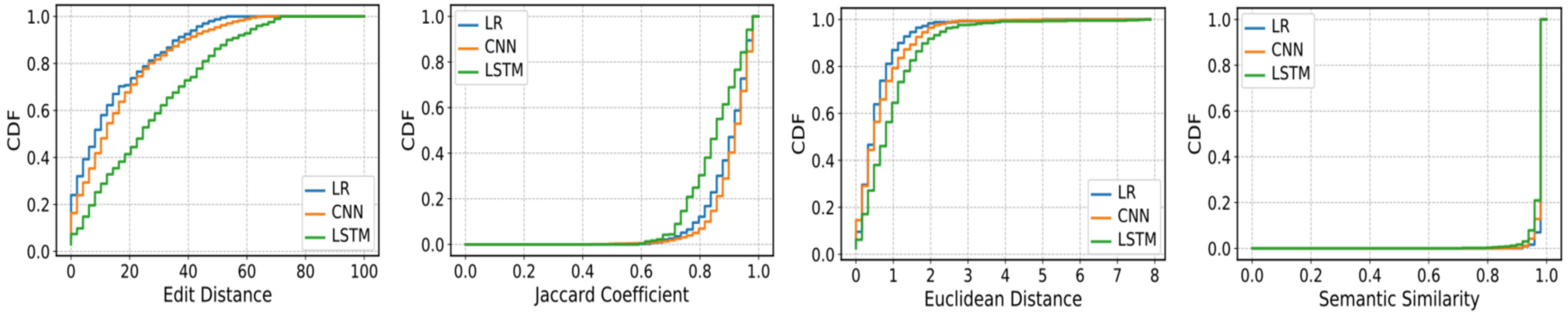
## Sentiment Score Distribution




## Remarks

- TextBugger greatly changes the confidence value of the classification results.
- IBM Watson is more sensitive to the adversarial texts generated by TextBugger.

# Utility Analysis: White-box Attack

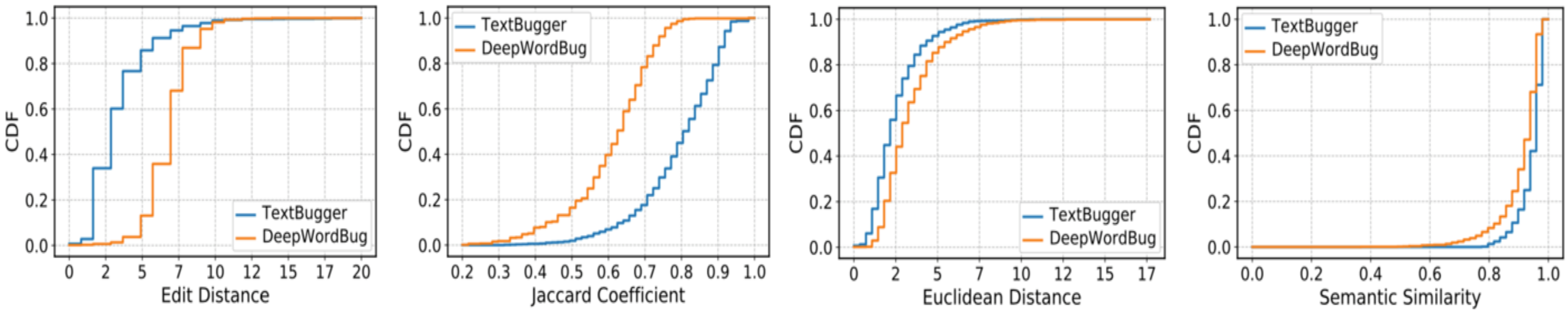



(a) IMDB

## Remarks

- The generated adversarial texts preserve good word-level and vector-level utility.

# Utility Analysis: Black-box Attack



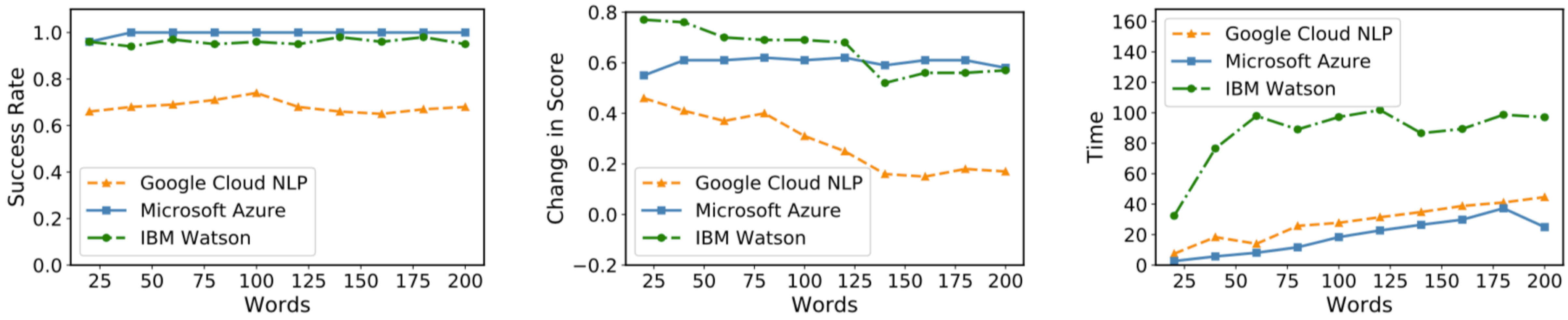

(a) IMDB

## Remarks

- TextBugger generates higher quality adversarial texts than DeepWordBug.

# The Impact Of Document Length

## The Impact of Document Length on Attack Performance



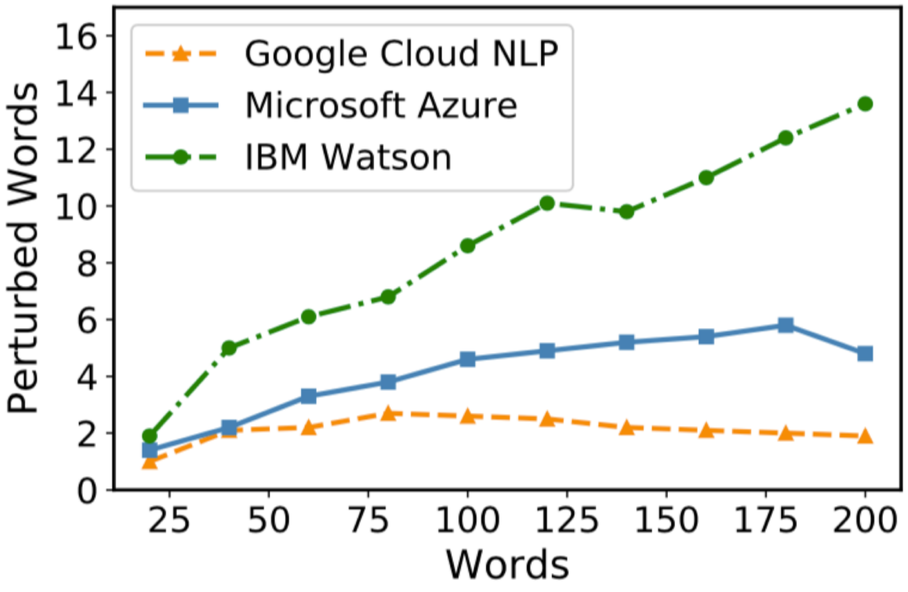

(a) Success Rate (b) Score (c) Time

## Remarks

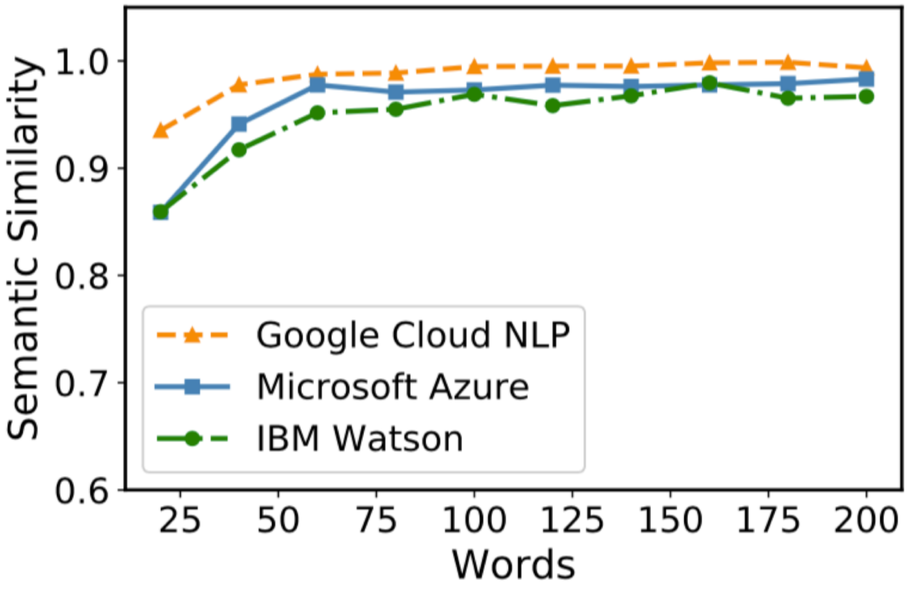
- Length has little impact on the success rate, but may decrease the change of negative class's confidence value.
- The time required for generating one adversarial text increases slightly as the length grows.

# The Impact Of Document Length

## The Impact of Document Length on The Utility of Generated Adversarial Texts.




(a) Number of Perturbed Words

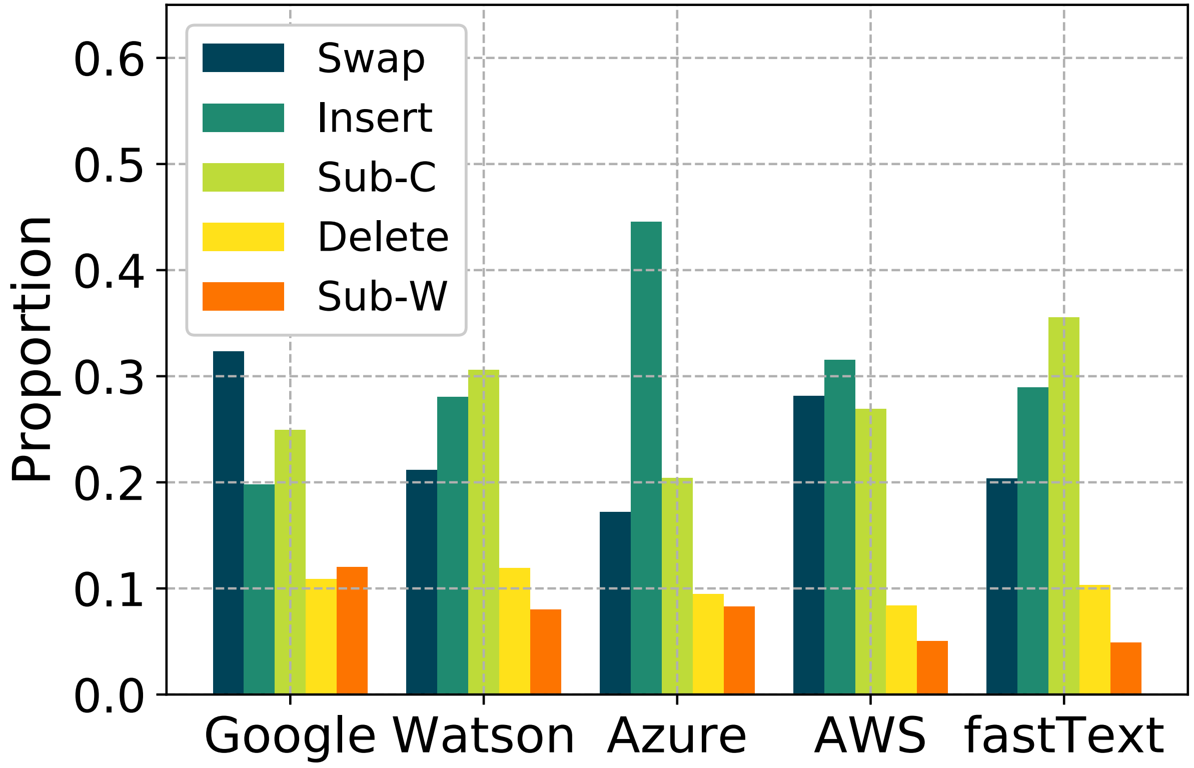



(b) Semantic Similarity

## Remarks

- Longer document length leads to more perturbed words.
- The increasing perturbed words do not decrease the semantic similarity of the adversarial texts.

# Bug Distribution




## Remarks

- Azure and AWS are sensitive to the insert bug
- Watson and fastText are sensitive to Sub-C
- Delete and Sub-W are used less than others

# Further Analysis

Transferability


User Study

# Transferability

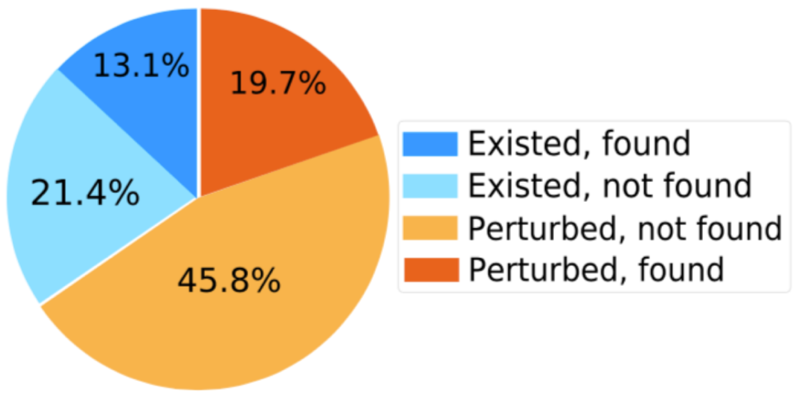
TABLE VII. TRANSFERABILITY ON IMDB AND MR DATASETS.

| Dataset | Model | White-box Models | | | Black-box APIs | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LR | CNN | LSTM | IBM | Azure | Google | fastText | AWS |
| **IMDB** | LR | 95.2% | 20.3% | 14.5% | 14.5% | 24.8% | 15.1% | 18.8% | 19.0% |
| | CNN | 28.9% | 90.5% | 21.2% | 21.2% | 31.4% | 20.4% | 25.3% | 20.0% |
| | LSTM | 28.8% | 23.8% | 86.6% | 27.3% | 26.7% | 27.4% | 23.1% | 25.1% |
| **MR** | LR | 92.7% | 18.3% | 28.7% | 22.4% | 39.5% | 31.3% | 19.8% | 29.8% |
| | CNN | 26.5% | 82.1% | 31.1% | 25.3% | 28.2% | 21.0% | 19.1% | 20.5% |
| | LSTM | 21.4% | 24.6% | 88.2% | 21.9% | 17.7% | 22.5% | 16.5% | 18.7% |

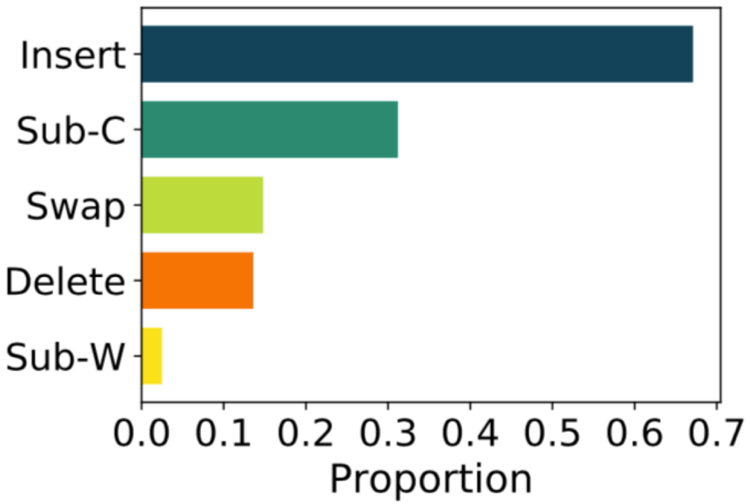
## Remarks

- Transferability also exists in adversarial texts among models and online platforms.
- Transferability can be used to attack online platforms even they have call limits.

# User Study




(a) The distribution of all mistakes in the samples.

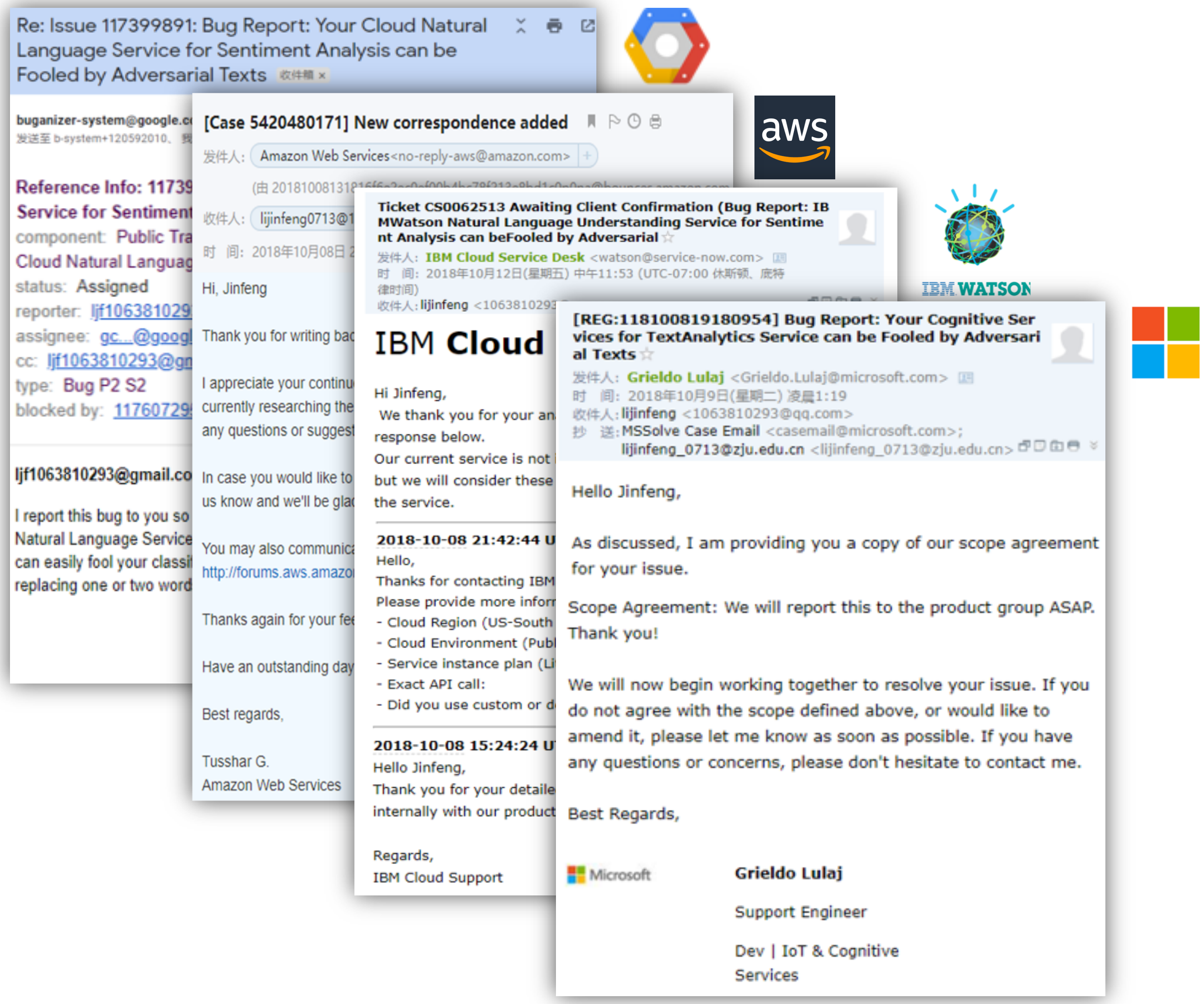



(b) The proportion of found bugs accounting for each kind of bug added in the samples.

## Remarks

- Adversarial texts generated by TextBugger are hard to distinguish.
- The insert bug is human-perceptible .
- Sub-W is the most robust bug.

# Vulnerability Report



# Summary

**We proposed TextBugger, a framework for generating adversarial texts effectively and efficiently**

- **Effective:** It outperforms state-of-the-art attacks in terms of attack success rate under both white-box and black-box settings.
- **Evasive:** It preserves the utility of benign text.
- **Efficient:** It generates adversarial text with computational complexity sub-linear to the text length**.**

**We evaluated TextBugger on 15 real-world online applications**

- **Dataset:** IMDB, MR and Kaggle.
- **Application:** Includes sentiment analysis and toxic content detection.

**We conducted a user study on our generated adversarial texts**

- **Utility-preserving:** TextBugger has little impact on human understanding.

**We further discuss two potential defense strategies to defend against such attacks**

**lijinfeng0713@zju.edu.cn**