Reducing Plaintext Input Size While Maintaining Machine Learning Model Accuracy

For Application Towards Machine Learning Over Encrypted Data

FirstName Surname†  
 Department Name  
 Institution/University Name  
 City State Country  
 email@email.com

Shreya Mohanty  
 Computer Science  
 University of California, Berkeley  
 Berkeley, California, USA  
 [shreyamohanty@berkeley.edu](mailto:email@email.com)

FirstName Surname  
 Department Name  
 Institution/University Name  
 City State Country  
 email@email.com

ABSTRACT

Machine learning already plays an integral part in several real world applications, from search engine recommendations to autonomous driving. However, a large chunk of data, such as medical records or transactional history, is sensitive and must be kept private. Such data can be protected by encryption, but performing machine learning over encrypted data poses an obstacle. Homomorphic encryption shows promising potential as a solution. Operations performed on homomorphically encrypted data will yield an encrypted result which can be decrypted by the original data owner. Thus, the original data need never be seen by a third party providing the machine learning services. Running machine learning algorithms on homomorphically encrypted data comes with some setbacks. Addition and multiplication are the only types of operations which can be performed on this data. Furthermore, encrypting data entails adding noise, which greatly increases the input’s size. Performing multiplication on this data augments noise and size even more, incurring a heavy computation tax.[1] The research presented in this paper focuses on tackling the latter problem. Specifically, I investigate ways to reduce plaintext data size while maintaining meaningful information and machine learning model accuracy. Reducing the size of an unencrypted dataset will in turn reduce size and computation tax for encrypted data. Tests were run on the IMDB review dataset provided by Keras, which contains 25,000 movie reviews labelled with either a positive or negative sentiment. I first examined trends arising from modifying the maximum length of a review to be read by the model, the maximum vocabulary size to use for the dataset, and the section of the review from which to read. I found that increasing the maximum length of a review to be read initially shows significant improvement in model accuracy, but gains start to taper. A similar trend is shown when increasing the vocabulary size. However, the initial gain in accuracy is much sharper and tapers off very quickly, even decreasing slightly. Finally, reading from the beginning, end, and a split (both the beginning and the end) allowed for a higher model accuracy than reading from the middle of a review. Next, I examined the effect of removing stop words, a list of common and sometimes unimportant words generally removed from textual data during preprocessing. As stop words are some of the most common words in the English language, I first tested ignoring a certain number of top most frequent words but found that model accuracy decreased as more words were ignored. Then, I explicitly removed stop words from the reviews and found that not using stop words initially showed a higher model accuracy but was outperformed by a model trained on data with stop words when the maximum length of the review to be read surpassed 100 words. Next, I investigated using convolutional neural networks (CNNs) to shorten text data intelligently and found that using CNNs yielded a much higher model accuracy than without using them.

KEYWORDS

machine learning, neural network, homomorphic encryption, IMDB, Keras, stop words, convolutional neural network, CNN

1 Technical Information

All tests were run on an Intel® Core™ i7-8550U CPU running at 1.80 GHz, with 12.0 GB of RAM, running the Windows 10 operating system. All machine learning models were written using the Keras API which is written in Python and uses TensorFlow backend. The IMDB dataset is provided by Keras. The dataset consists of 25000 movie reviews labelled with a positive or negative sentiment. The data is automatically preprocessed, and each word is converted to an integer index based on its frequency in the dataset. Each review is represented as a list of integers. If needed, sequences were padded with 0s. The start of a review was marked with 1. Unknown characters were marked with 2s.

2 Key Terms

**maxlen** – a variable that determines the maximum length of a review to read from the IMDB dataset. Shorter reviews are padded while longer reviews are cut off.

**max\_features** – a variable that determines the vocabulary size a machine learning model uses with the IMDB dataset

**stop words** – common words such as ‘the’ in a language that are usually removed during text preprocessing.

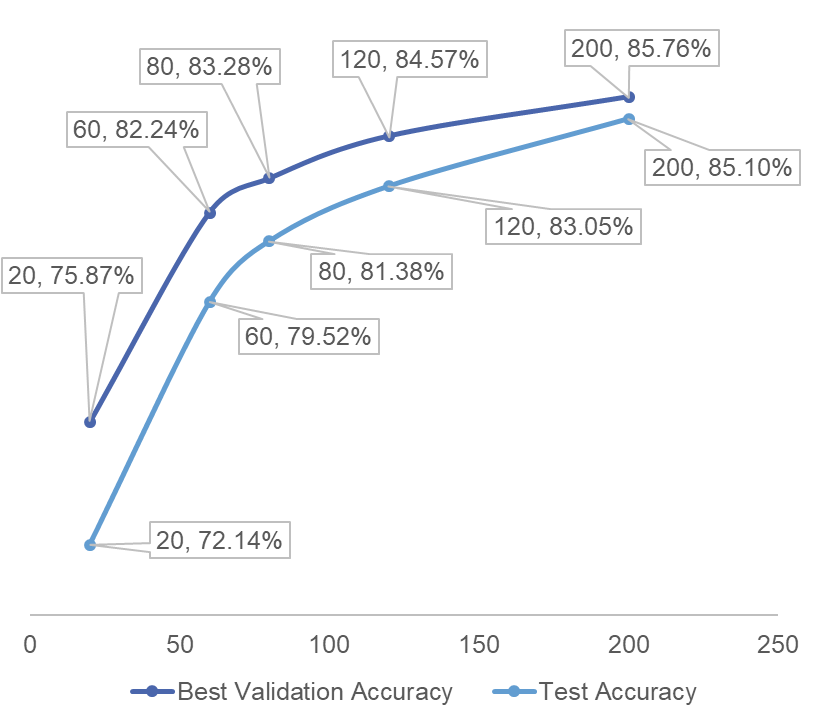
**skip\_top** – a variable that determines the top most frequent words to ignore in the IMDB dataset.

3 Review Processing and Splicing

The first part of my investigation consisted of analyzing different ways of processing and reading the reviews. Specifically, I modified variables determining the maximum length of a review read by the machine learning model and the vocabulary size considered by the model, and I modified the code to read from the beginning, end, end with text reversed, middle, and both the beginning and end of a review (split). The neural network model I used was an example downloaded from the Keras GitHub.[2] It had maxlen set to 80 and the max\_features variable set to 20000. The model consisted of an embedding layer, LSTM layer, and a dense output layer with a sigmoid activation function. Training was set to run for 15 epochs. The training, validation, and testing sets consisted of all 25000 reviews. The model default was to read from the beginning of each review.

3.1 Maximum Length

I first tested the maxlen variable. Because default model had maxlen = 80, I decided to test a range centering on this value. I tested maxlen = 20, 60, 80, 120, and 200. I kept max\_features = 20000 (the default).



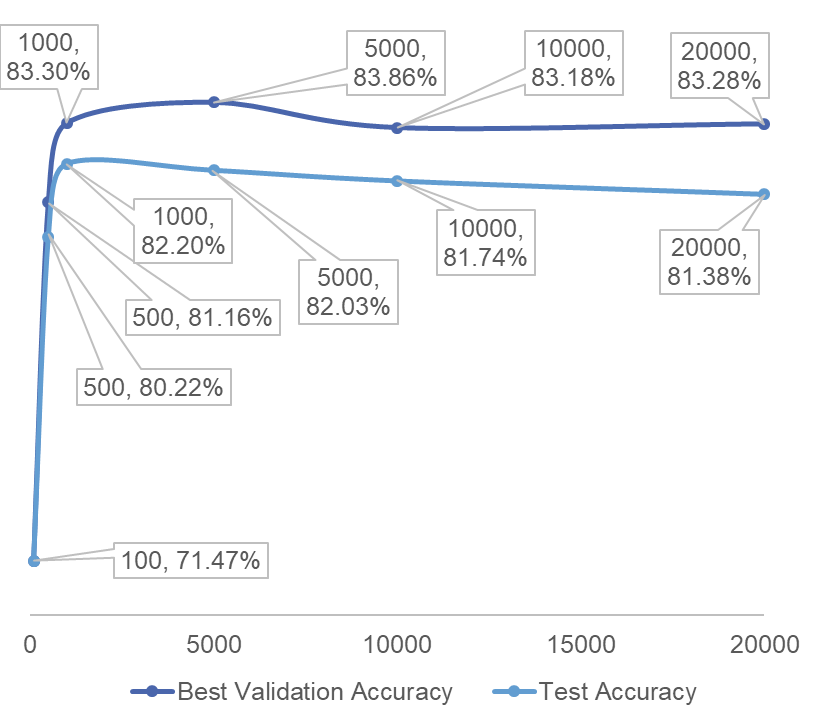
**Figure 1: Results for varying the maxlen variable.**

Because training over 15 epochs often led to over fitting the model, I consider both the final test accuracy and best validation accuracy out of all the epochs in my results. The trend is similar, although the best validation accuracy was higher than the test accuracy. The model accuracy initially shows significant increases as the maxlen variable is set higher. However, gains start to taper after around 80 words. The average review length is 239 words, so accuracy growth is minimal after 200 words. Increasing the maxlen variable led to significant increases in computation time

The average time per training epoch was 99.6 s for 80 words, 179.5 s for 120 words, and 224.8 s for 200 words. However, these times can be greatly reduced on a device with a higher processing power (such as a GPU). It should also be noted that different background processes were running while these tests were and run times may have been slightly affected by this. Therefore, specific times are not reported in future trials.

3.2 Vocabulary Size

Next, I tested the max\_features variable, which was set to 20000 in the original code. The dataset has about 88500 unique words. However, I began by testing max\_features = 100, 500, 1000, 5000, 10000, and 20000 to see if large increases in the variable would affect model accuracy. I kept maxlen = 80.

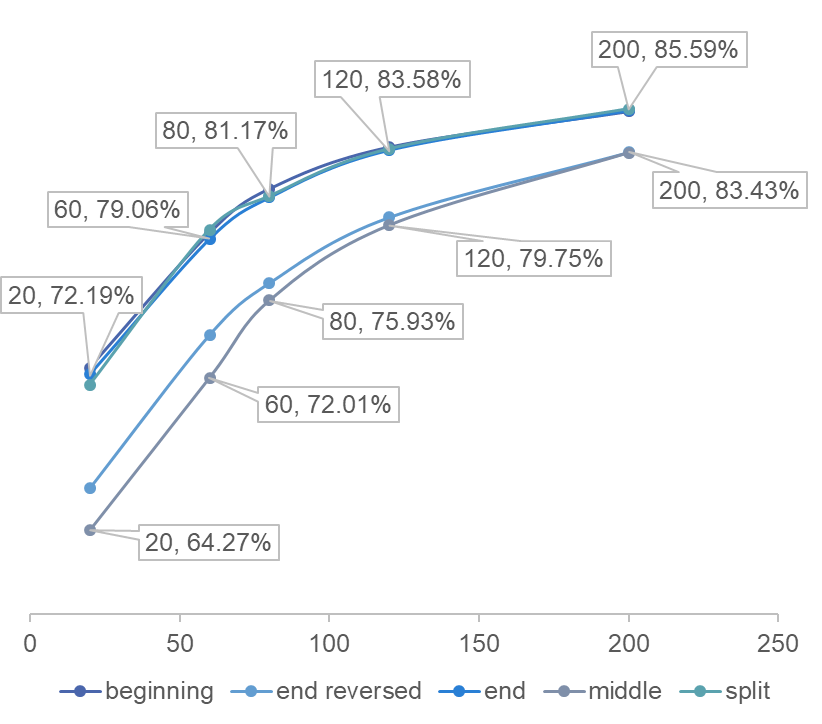


**Figure 2: Results for varying the max\_features variable.**

Again, I consider both the test accuracy and best validation accuracy in my results. Using the best validation accuracy generally shows the same trend as test accuracy with better results. In future trials, I will only display the best validation accuracy. Increasing the number of features measured from 100 to 1000 increases the best validation accuracy by 11.71%. Model accuracy begins to decrease slightly when the number of features is increased from 1000 to 20000, suggesting too many features may lead to superfluous data and overtraining. Computation time generally increased as the number of features increased.

3.3 Review Splicing

Next, I tested reading the reviews from different sections, specifically the beginning, end, end with text reversed, middle, and both the beginning and end of a review. I used max\_features = 500 as this value performed well in terms of accuracy and efficiently in terms of computation time. I used the same distribution for maxlen as my previous trials (maxlen = 20, 60, 80, 120, 200). Because the original code only read reviews from the beginning, I had to modify the model to read from different sections. I did this by changing the order of the words (represented as integer indexes) in each sequence through Python list processing functions.



**Figure 3: Results for reading reviews from different sections.**

Reading from the beginning, end, and both beginning and end showed extremely similar results and the highest accuracies. At maxlen = 200, the beginning and end models both had an accuracy around 85.6%. The model performed more poorly when reading from the end with text reversed and when reading from the middle. The accuracies converge as maximum length increases probably due to overlapping coverage.

3.4 Training and Validation Trends

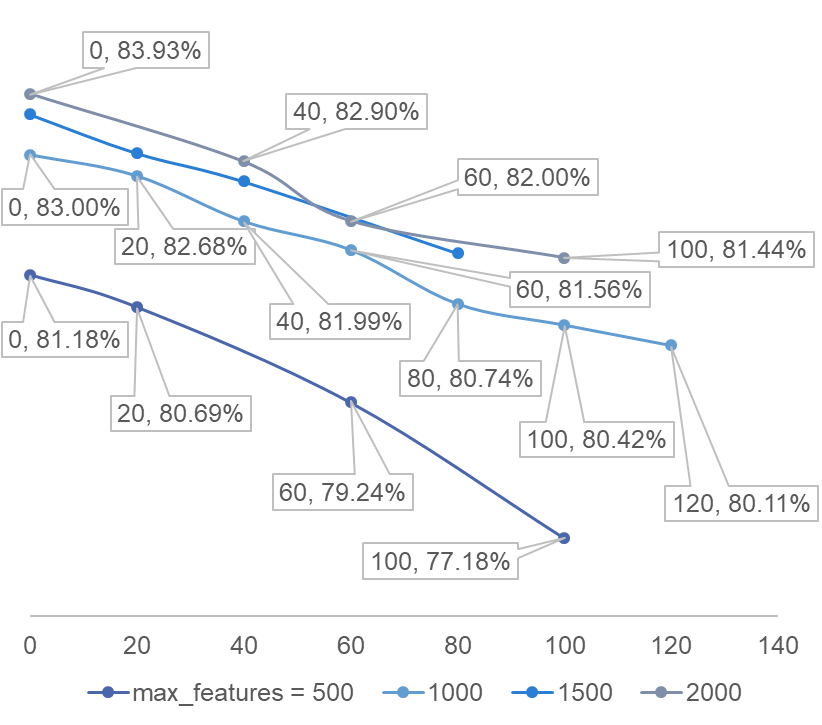
Training loss decreased, and training accuracy increased over the training epochs in the aforementioned trials, as expected. Validation loss increased, and validation accuracy decreased when max\_features was greater than 5000 (maxlen = 80). Using max\_features = 500, most of the trials for different text splices showed validation loss decreasing and accuracy increasing. We can infer that measuring too many features may lead to overtraining, as the model is trained on word frequencies specific to its training set. When measuring maxlen with max\_features set to 20000, most trials were affected by the high number of features and showed validation loss increase and accuracy decrease. Reading from the end with text reversed with maxlen = 20 and max\_features = 500 showed the same trend.

4 Word Frequency and Stop Word Analysis

The second part of my investigation was prompted by an article on feature engineering which mentioned that stop words are generally removed in text data preprocessing.[3] The reviews in the IMDB dataset do not have stop words removed. I used the same model as my previous trials.

4.1 Top Most Frequent Words to Ignore

As stop words are among the most common words in the English language, I first decided to vary the skip\_top variable and remove a number of most frequent words in the IMDB dataset. The 1000 most frequent words in the English language cover 87.8% of oral speech. The 2000 most frequent words cover 92.7%, and the 3000 most frequent words cover 94%. As such, I decided to test skip\_top = 0, 20, 40, 60, 80, 100, and 120 over max\_features = 500, 1000, 1500, and 2000. However, because of computation time, I did not complete a trial for each skip\_top value in all cases. Rather, I focused on finding trends.

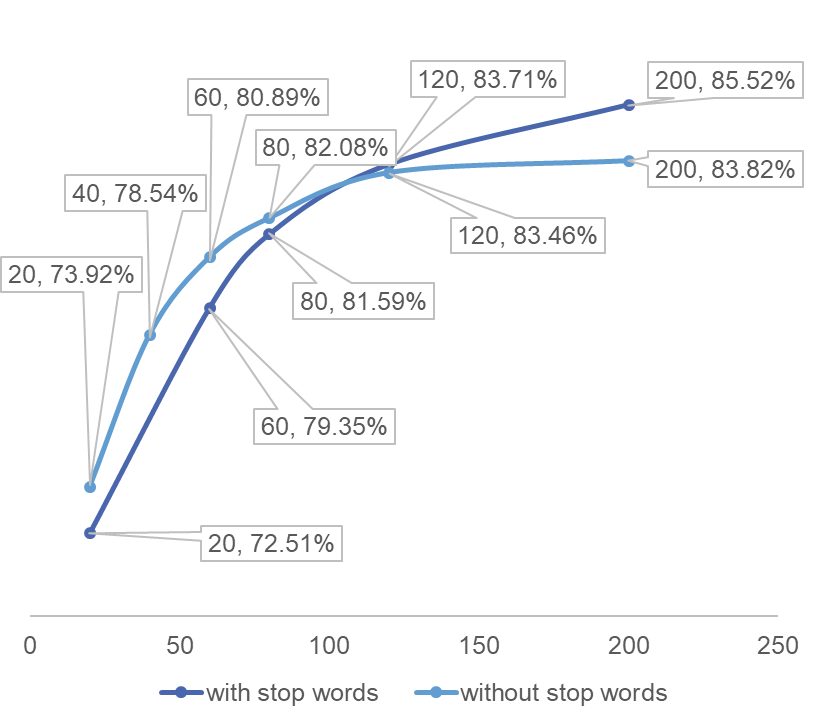


**Figure 4: Results for varying skip\_top.**

As more top frequent words in the dataset are ignored, model accuracy tends to decrease. Although stop words are among the most common words in the English language, they are not necessarily the most common in this dataset. Removing top frequent words may cause the model to ignore key sentiment indicators. Another reason for the decrease in accuracy may be because of the intervals of skip\_top tested. Future testing should be done on values between 1 and 20 to see if ignoring a smaller number of words may increase accuracy. Validation loss tended to increase, and validation accuracy decreased for max\_features = 2000.

4.2 Stop Words

Next, I explicitly removed stop words from the dataset. I used NLTK’s list of stop words which is used as a common reference. I converted each processed review back to its text form using a dictionary inherent to the Keras code, removed the stop words, and converted the sequences back to their processed form.



**Figure 5: Results for removing stop words.**

Removing stop words from the dataset makes the model more accurate for maxlen less than around 100. Afterwards, the model performs better when stop words are included. Some stop words such as ‘not’ may affect the sentiment of surrounding words, and the resulting effect can add up as review length increases.

5 Convolutional Neural Networks

After completing the previous trials, I spent a few weeks reading about convolutional neural networks (CNNs) and the Keras documentation.[4] I decided to investigate using convolutional and pooling layers to reduce text sequence size. In practice, the layers could be used to shorten a sequence of text without classifying it. The shortened sequence could then be encrypted and put through a machine learning algorithm. I tested a number of different combinations of convolutional and max pooling layers followed by either a global max pooling layer or flattening layer, a dropout layer, a dense layer, and a dense output layer with a sigmoid activation function. I used a randomized search to tune the hyperparameters.[5] For convolutional layers, the options for the number of filters were 32, 64, and 128. The options for the kernel size were 5, 10, and 20. For max pooling layers, the options for pool size were 5, 10, and 20, and the options for strides were None, 2, and 3. The options for the number of nodes for the first dense layer were 10, 20, and 30. I chose these based on common values seen in readings and tutorials as well as computation time. However, for the model with 3 convolutional layers, I had to set the kernel size for each layer to 2, as a larger number would have reduced the size of the data to the point where no data was left. I kept the vocabulary size at 1000 because showed the highest accuracy in previous trials. Maxlen = 250 because the average review length is 239 words. I trained the model over 5 epochs, as that is usually when validation accuracy started increasing again in trials with vocab\_size = 1000. The batch\_size was kept at the default 32. The embedding layer had 50 dimensions as this is standard and not too data intensive. I set the cross validation variable to 5, as this divides the 25000 reviews in the dataset evenly. However, when testing the model, I had to test it on the original split of using all 25000 reviews for training, validation, and testing. Finally, I left the number of iterations to the default 10.

|  |  |  |
| --- | --- | --- |
| 2 convolutional | | 87.72% |
| 2 convolutional, 1 max pooling (at end) | | 87.50% |
| 1 convolutional, 1 global max pooling | | 87.32% |
| 3 convolutional layers (modified parameter selection) | | 87.20% |
| 1 convolutional, 1 max pooling | | 86.86% |
| 2 convolutional, 1 max pooling (in between) | | 86.80% |
| 1 convolutional | | 86.60% |
| 2 convolutional, 1 global max pooling | | 86.52% |
| 1 global max pooling | | 84.82% |
| none |  | 84.42% |

**Figure 6: Results for different CNNs.**

The results show that convolutional layers can be used to reduce text sequence size while maintaining meaningful information and model accuracy.

6 Future Work

6.1 Extending Convolutional Neural Networks

Future work could involve implementing the model laid out in this paper by training a CNN on textual data and stopping the process before classifying the output. The shortened sequences can be encrypted and run through a machine learning algorithm.

6.2 Genetic Algorithms

I read a paper about using genetic algorithms to optimize a convolutional neural network architecture for image classification and began my own investigation into creating one for text classification.[6] Although I haven’t completed the code, it is something I plan to continue working on. Currently I have created the population initialization function which creates a number of convolutional neural networks built out of layer blocks. The blocks consist of a convolutional layer block, a max pooling block, a global block, which contains a global max pooling layer, a dense layer, a dropout layer, and an output dense layer, and a flatten block, which contains a flattening layer, a dense layer, a dropout layer, and an output dense layer. The convolutional and max pooling block are favored with a higher probability when creating individuals. A check is also implemented to prevent the models from having too many layers, and thus erasing all the data. The fitness function is simply a measure of model accuracy after being trained on the data. I implemented an early stopping checkpoint to prevent over fitting. I am currently writing the crossover function, which essentially takes two parents, splits them at a random point, and combines opposite parents to create two children.

7 Areas for Improvement

One of the biggest problems I faced was the computation tax required to run my tests. The text processing tests took from 5 – 30 minutes, and the hyperparameter optimization for the CNNs took about 5 hours on average. Therefore, I was not able to run multiple trials for each specific configuration of variables. I was also unsure what ranges to test in my initial experiments with text processing and would have liked to test smaller intervals given more time. Another obstacle I faced was a lack of guidance on matters which would have made my work more efficient and results more accurate. For example, I could have implemented early stopping checks in my initial trials with text processing to save time and prevent over fitting. Because I learned many of these concepts as I was doing my work, the results took longer to obtain than they would if I retry my experiments a second time. Should work in this area be extended, it would be prudent to use a GPU to run tests and implement checkpoints in the machine learning models.

ACKNOWLEDGMENTS

Acknowledgements are extended to Dr. Hassan Takabi and Dr. Renee Bryce for hosting this REU site, as well as to Robert Podschwadt and Ehsan Hesamifard for introducing us to many of the concepts used in my research and helping guide and troubleshoot the process.

REFERENCES

[1] Thore Graepel, Kristin Lauter, and Michael Naehrig, Ml confidential: Ma-

chine learning on encrypted data

https://eprint.iacr.org/2012/323.pdf, Information Security and Cryptology–

ICISC 2012, Springer, 2013, pp. 1–21.

[2] Keras-Team.“Keras-Team/Keras.” *GitHub*,

github.com/keras-team/keras/blob/master/examples/imdb\_lstm.py.

[3] Sarker, Dipanzan. “Traditional Methods for Text Data - Towards Data Science.” *Medium*, Towards Data Science, 27 Mar. 2019, towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41?gi=920677979e23.

[4] “Keras: The Python Deep Learning Library.” Home - Keras Documentation, keras.io/.Conference Name:ACM Woodstock conference

[5] “Sklearn.model\_selection.RandomizedSearchCV¶.” *Scikit*,scikit-learn.org/stable/modules/generated/sklearn.model\_selection.RandomizedSearchCV.html.

[6] Sun, Yanan et. al. “Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification”

https://arxiv.org/pdf/1808.03818.pdf

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00