Tagging Stack Overflow Questions

Susan Thomas
Fernando Avalos Garcia
Dhaval Khamar

March 28, 2020

The Problem

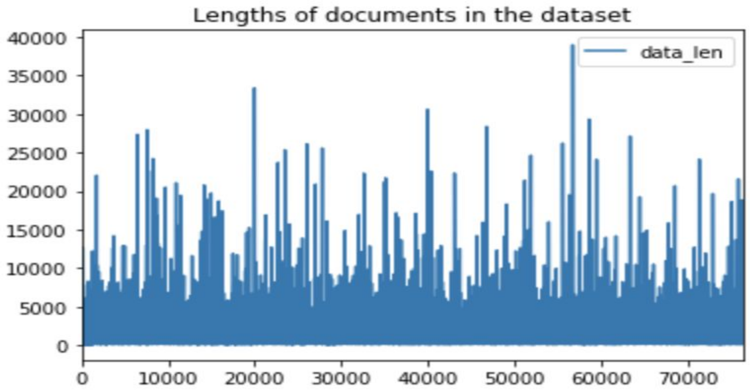
“Help Stack Overflow’s content managers to manage tags on new questions through automated tag generation process using deep learning techniques”

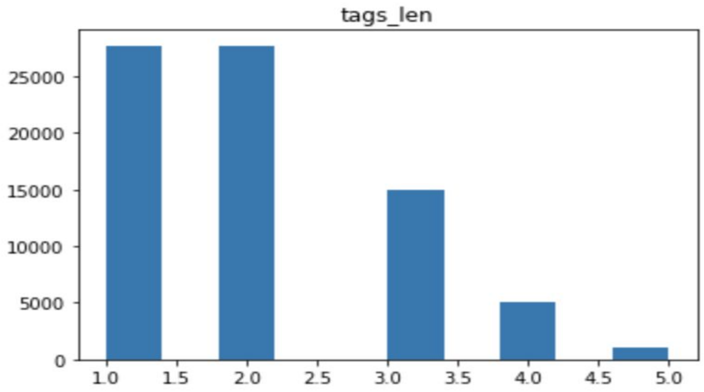
- Stack Overflow is a question and answer site for professional and programming enthusiasts
- Stack Overflow has over 16.5 million questions and is a great resource for programmers.
- Every question on Stack Overflow is marked with relevant tags. There is usually at least one tag per post.
- Tags are useful for users to find similar questions and for programming experts to find and answer open questions.
- However they have to be entered and maintained manually.
- Deep Learning algorithms can be used to generate tags on new questions learning tags information from existing questions.

The Data

The Data

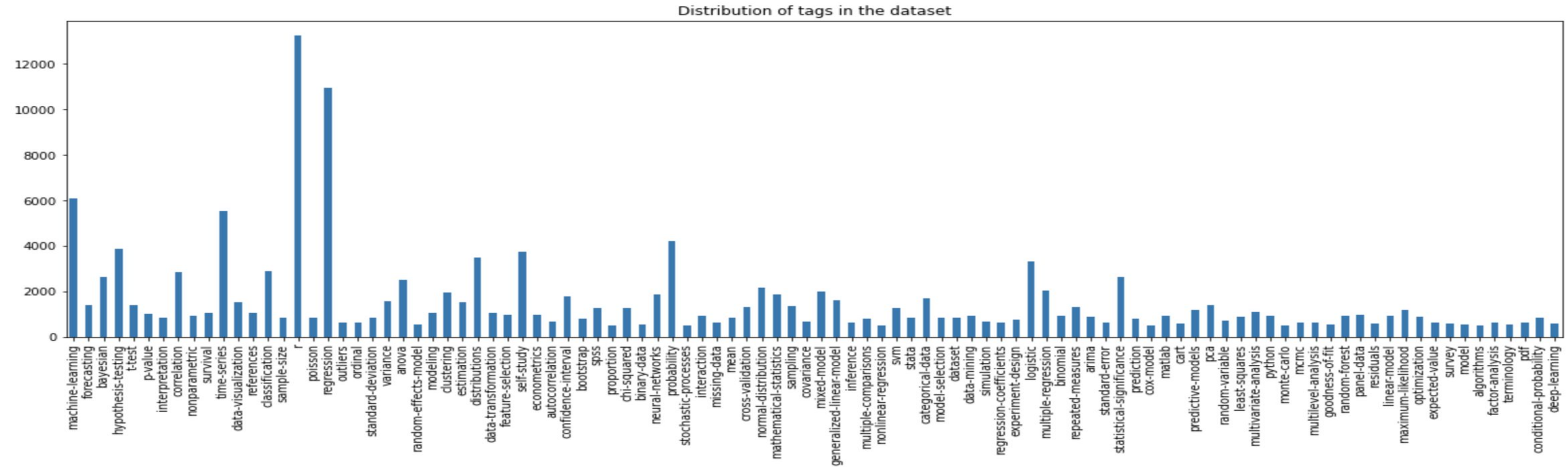
- To make a Stack Overflow data tag learning model, we took a data set of Machine Learning related questions and tags (76364).
- These Questions were of varied different length (500-24000) and had different number of tags (1-5).





The Data

- There are 100 possible tags for Machine Learning questions on Stack Overflow.
- Since the data set is related to Machine Learning - tags like “r”, “Machine Learning”, “Regression”, “Time Series” has larger occurrence in the data.



The Data

- Before consuming the data, we performed data cleanup and normalization activities.
- As part of cleanup -
    - All non-english words were removed
    - All special characters, punctuation marks, control characters were removed.
    - All text was converted into lower case.
- As part of normalization all questions were truncated or padded to 500 characters.
- For processing all words were encoded as integers.

```
[ ] df['data_clean']

0        two cultures statistics machine learning last ...
1        forecasting demographic census ways forecast d...
2        bayesian frequentist reasoning plain english w...
3        meaning values values statistical tests taking...
4        examples teaching correlation mean causation o...
                               ...
76360    interpretation global test value interaction t...
76361    testing linear model used fit linear model dat...
76362    know simple validation result statistically si...
76363    kendall conditional independence test statisti...
76364    heteroskedasticity regression model testing he...
Name: data_clean, Length: 76365, dtype: object
```

```
[ ] padded_data.shape

(76365, 500)

[ ] vocab_size

81139
```

The Solution

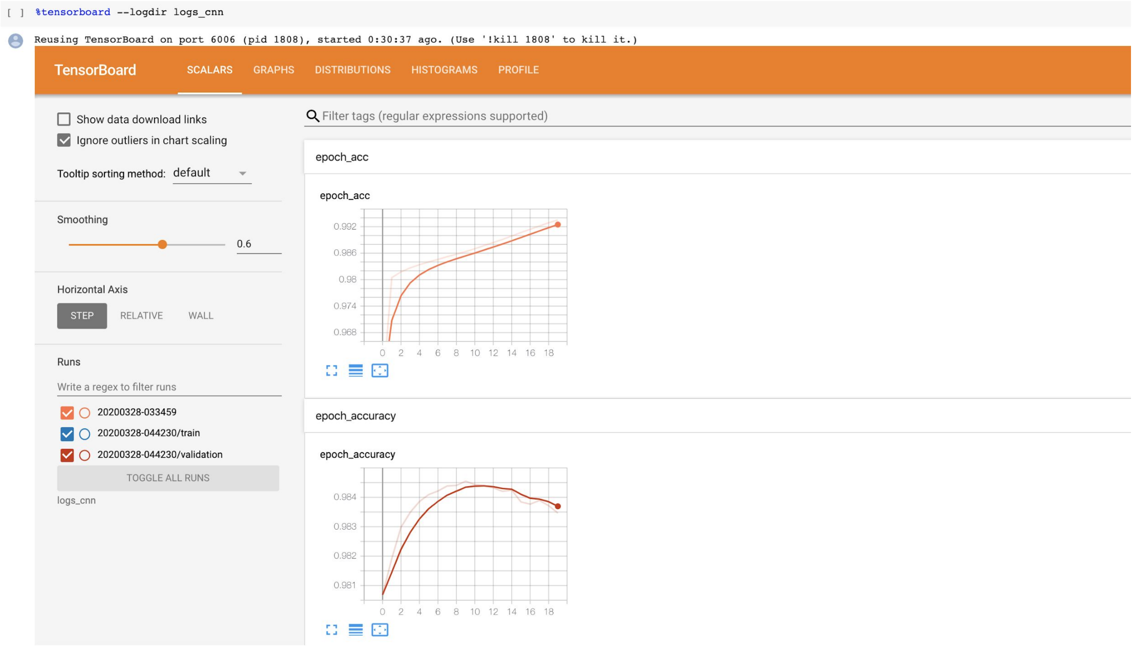
Solution Architecture

- We applied two different deep learning techniques on the identified data set.
    - CNN
        - Embedding Layer
        - Dropout - Introduce a 15% dropout to prevent overfitting
        - Conv1D - 128 filters of size 5 applied to capture essential features
        - Global MaxPooling - Applies max-pooling over each document
        - Dense - Single fully-connected layer to 100 output nodes
    - RNN
        - Embedding Layer
        - Dropout - Introduce a 15% dropout to prevent overfitting
        - Bidirectional LSTM
            - Two hidden layers of opposite direction with 64 hidden layers to better understand the context of the text.
            - The architecture of the recurrent neural network is Many-To-One.
            - Apply recurrent regularizer L2.
            - Apply recurrent dropout.
        - Dense - Single fully-connected layer to 100 output nodes

CNN Model Training and Accuracy

```
==================================================================
Total params: 20,948,452
Trainable params: 20,948,452
Non-trainable params: 0
__________________________________________________________________
None
Train on 54982 samples, validate on 6110 samples
Epoch 1/20
54982/54982 [==============================] - 41s 739us/sample - loss: 0.1790 - categorical_accuracy: 0.0468 - accuracy: 0.9573 - val_loss: 0.0876 - val_categorical_accuracy: 0.0736 - val_accuracy: 0.9802
Epoch 2/20
54982/54982 [==============================] - 37s 677us/sample - loss: 0.0826 - categorical_accuracy: 0.1245 - accuracy: 0.9806 - val_loss: 0.0755 - val_categorical_accuracy: 0.2295 - val_accuracy: 0.9814
Epoch 3/20
54982/54982 [==============================] - 38s 684us/sample - loss: 0.0695 - categorical_accuracy: 0.2692 - accuracy: 0.9819 - val_loss: 0.0653 - val_categorical_accuracy: 0.2961 - val_accuracy: 0.9825
Epoch 4/20
54982/54982 [==============================] - 37s 681us/sample - loss: 0.0610 - categorical_accuracy: 0.3152 - accuracy: 0.9828 - val_loss: 0.0595 - val_categorical_accuracy: 0.3260 - val_accuracy: 0.9830
Epoch 5/20
54982/54982 [==============================] - 38s 684us/sample - loss: 0.0554 - categorical_accuracy: 0.3482 - accuracy: 0.9835 - val_loss: 0.0560 - val_categorical_accuracy: 0.3566 - val_accuracy: 0.9834
Epoch 6/20
54982/54982 [==============================] - 38s 685us/sample - loss: 0.0515 - categorical_accuracy: 0.3752 - accuracy: 0.9841 - val_loss: 0.0539 - val_categorical_accuracy: 0.3763 - val_accuracy: 0.9836
Epoch 7/20
54982/54982 [==============================] - 37s 678us/sample - loss: 0.0485 - categorical_accuracy: 0.3983 - accuracy: 0.9847 - val_loss: 0.0528 - val_categorical_accuracy: 0.3789 - val_accuracy: 0.9837
Epoch 8/20
54982/54982 [==============================] - 37s 673us/sample - loss: 0.0460 - categorical_accuracy: 0.4158 - accuracy: 0.9853 - val_loss: 0.0518 - val_categorical_accuracy: 0.3897 - val_accuracy: 0.9839
Epoch 9/20
54982/54982 [==============================] - 36s 663us/sample - loss: 0.0437 - categorical_accuracy: 0.4325 - accuracy: 0.9858 - val_loss: 0.0513 - val_categorical_accuracy: 0.3894 - val_accuracy: 0.9839
Epoch 10/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0414 - categorical_accuracy: 0.4477 - accuracy: 0.9864 - val_loss: 0.0513 - val_categorical_accuracy: 0.3894 - val_accuracy: 0.9840
Epoch 11/20
54982/54982 [==============================] - 37s 674us/sample - loss: 0.0391 - categorical_accuracy: 0.4664 - accuracy: 0.9871 - val_loss: 0.0512 - val_categorical_accuracy: 0.3925 - val_accuracy: 0.9839
Epoch 12/20
54982/54982 [==============================] - 37s 668us/sample - loss: 0.0368 - categorical_accuracy: 0.4810 - accuracy: 0.9878 - val_loss: 0.0514 - val_categorical_accuracy: 0.3915 - val_accuracy: 0.9839
Epoch 13/20
54982/54982 [==============================] - 37s 676us/sample - loss: 0.0345 - categorical_accuracy: 0.4981 - accuracy: 0.9885 - val_loss: 0.0519 - val_categorical_accuracy: 0.3902 - val_accuracy: 0.9838
Epoch 14/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0322 - categorical_accuracy: 0.5111 - accuracy: 0.9893 - val_loss: 0.0524 - val_categorical_accuracy: 0.3895 - val_accuracy: 0.9837
Epoch 15/20
54982/54982 [==============================] - 37s 678us/sample - loss: 0.0298 - categorical_accuracy: 0.5248 - accuracy: 0.9901 - val_loss: 0.0528 - val_categorical_accuracy: 0.3917 - val_accuracy: 0.9837
Epoch 16/20
54982/54982 [==============================] - 37s 670us/sample - loss: 0.0276 - categorical_accuracy: 0.5390 - accuracy: 0.9908 - val_loss: 0.0543 - val_categorical_accuracy: 0.3933 - val_accuracy: 0.9833
Epoch 17/20
54982/54982 [==============================] - 37s 671us/sample - loss: 0.0253 - categorical_accuracy: 0.5500 - accuracy: 0.9916 - val_loss: 0.0550 - val_categorical_accuracy: 0.3879 - val_accuracy: 0.9833
Epoch 18/20
54982/54982 [==============================] - 37s 672us/sample - loss: 0.0232 - categorical_accuracy: 0.5601 - accuracy: 0.9924 - val_loss: 0.0562 - val_categorical_accuracy: 0.3854 - val_accuracy: 0.9834
Epoch 19/20
54982/54982 [==============================] - 37s 669us/sample - loss: 0.0213 - categorical_accuracy: 0.5689 - accuracy: 0.9931 - val_loss: 0.0575 - val_categorical_accuracy: 0.3892 - val_accuracy: 0.9832
Epoch 20/20
54982/54982 [==============================] - 37s 670us/sample - loss: 0.0195 - categorical_accuracy: 0.5774 - accuracy: 0.9938 - val_loss: 0.0588 - val_categorical_accuracy: 0.3802 - val_accuracy: 0.9830
```

CNN Model Tensor Board



CNN Sample Output

```
# Predict
sample_pred = model.predict(padded_data)

# Binarize output
sample_pred = (sample_pred >= THRESHOLD).astype(int)

print('Real tags: ', df['Tags'][post_index])
print('Predicted tags: ', mlb.inverse_transform(sample_pred))
```

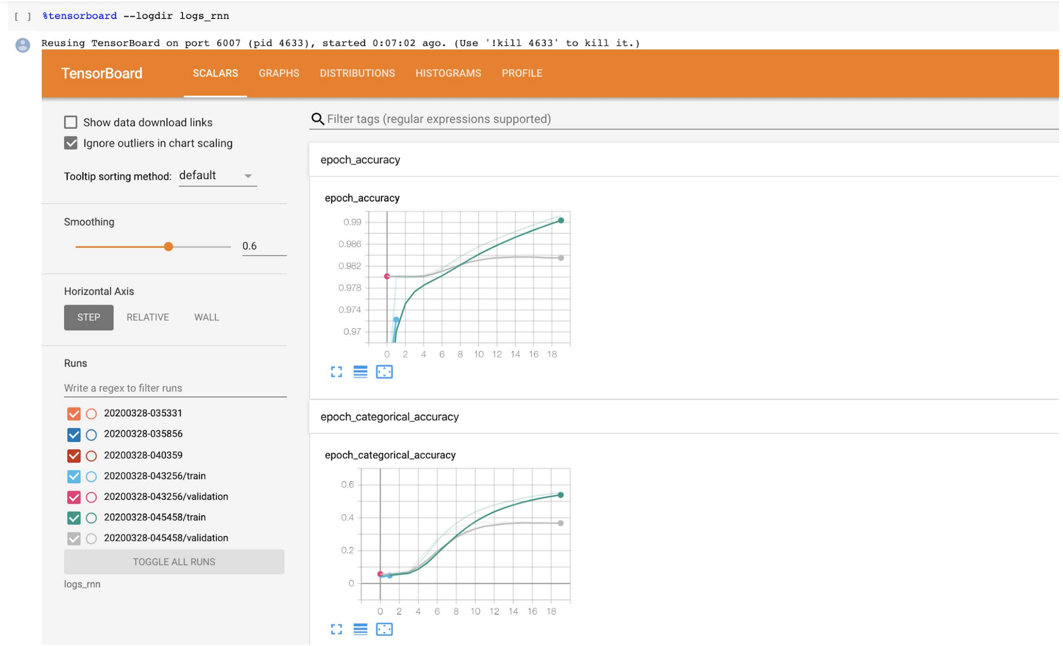
```
[ ] # Helper method to test the model
    get_tags_for_id(75300)
```

```
Original post: How to Generate a Cloud of 3 dimensional points distributed according to a gaussian? <p>How can I generate a cloud of N[100] 3-dimensional points that are isotropically and identically distribu

<p>I know that box muller can generate the following 2d cloud but I would like to know how I can do something similiar in 3d. </p>

<p><a href="http://i.stack.imgur.com/IYflU.png" rel="nofollow"><img src="http://i.stack.imgur.com/IYflU.png" alt="https://upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Box-Muller_transform_visualisation.s

Real tags:  ['distributions' 'normal-distribution']
Predicted tags:  [('distributions', 'normal-distribution')]
```

RNN Model Training and Accuracy

```
=================================================================
Total params: 20,948,836
Trainable params: 20,948,836
Non-trainable params: 0
_________________________________________________________________
None
Train on 54982 samples, validate on 6110 samples
Epoch 1/20
54982/54982 [==============================] - 222s 4ms/sample - loss: 0.9494 - categorical_accuracy: 0.0410 - accuracy: 0.9536 - val_loss: 0.2052 - val_categorical_accuracy: 0.0566 - val_accuracy: 0.9801
Epoch 2/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.1225 - categorical_accuracy: 0.0545 - accuracy: 0.9800 - val_loss: 0.0922 - val_categorical_accuracy: 0.0566 - val_accuracy: 0.9801
Epoch 3/20
54982/54982 [==============================] - 218s 4ms/sample - loss: 0.0900 - categorical_accuracy: 0.0615 - accuracy: 0.9800 - val_loss: 0.0881 - val_categorical_accuracy: 0.0682 - val_accuracy: 0.9801
Epoch 4/20
54982/54982 [==============================] - 216s 4ms/sample - loss: 0.0863 - categorical_accuracy: 0.0683 - accuracy: 0.9800 - val_loss: 0.0835 - val_categorical_accuracy: 0.0768 - val_accuracy: 0.9801
Epoch 5/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.0804 - categorical_accuracy: 0.1203 - accuracy: 0.9800 - val_loss: 0.0774 - val_categorical_accuracy: 0.1445 - val_accuracy: 0.9803
Epoch 6/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0734 - categorical_accuracy: 0.1855 - accuracy: 0.9806 - val_loss: 0.0713 - val_categorical_accuracy: 0.2051 - val_accuracy: 0.9810
Epoch 7/20
54982/54982 [==============================] - 214s 4ms/sample - loss: 0.0662 - categorical_accuracy: 0.2609 - accuracy: 0.9815 - val_loss: 0.0656 - val_categorical_accuracy: 0.2704 - val_accuracy: 0.9818
Epoch 8/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0598 - categorical_accuracy: 0.3184 - accuracy: 0.9827 - val_loss: 0.0613 - val_categorical_accuracy: 0.3005 - val_accuracy: 0.9826
Epoch 9/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0546 - categorical_accuracy: 0.3664 - accuracy: 0.9837 - val_loss: 0.0587 - val_categorical_accuracy: 0.3419 - val_accuracy: 0.9831
Epoch 10/20
54982/54982 [==============================] - 216s 4ms/sample - loss: 0.0502 - categorical_accuracy: 0.4048 - accuracy: 0.9847 - val_loss: 0.0568 - val_categorical_accuracy: 0.3571 - val_accuracy: 0.9835
Epoch 11/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0467 - categorical_accuracy: 0.4358 - accuracy: 0.9855 - val_loss: 0.0559 - val_categorical_accuracy: 0.3638 - val_accuracy: 0.9836
Epoch 12/20
54982/54982 [==============================] - 215s 4ms/sample - loss: 0.0436 - categorical_accuracy: 0.4579 - accuracy: 0.9863 - val_loss: 0.0553 - val_categorical_accuracy: 0.3730 - val_accuracy: 0.9838
Epoch 13/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0409 - categorical_accuracy: 0.4776 - accuracy: 0.9870 - val_loss: 0.0552 - val_categorical_accuracy: 0.3637 - val_accuracy: 0.9838
Epoch 14/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0385 - categorical_accuracy: 0.4932 - accuracy: 0.9877 - val_loss: 0.0556 - val_categorical_accuracy: 0.3738 - val_accuracy: 0.9837
Epoch 15/20
54982/54982 [==============================] - 210s 4ms/sample - loss: 0.0362 - categorical_accuracy: 0.5053 - accuracy: 0.9883 - val_loss: 0.0559 - val_categorical_accuracy: 0.3714 - val_accuracy: 0.9837
Epoch 16/20
54982/54982 [==============================] - 210s 4ms/sample - loss: 0.0342 - categorical_accuracy: 0.5190 - accuracy: 0.9889 - val_loss: 0.0566 - val_categorical_accuracy: 0.3733 - val_accuracy: 0.9837
Epoch 17/20
54982/54982 [==============================] - 212s 4ms/sample - loss: 0.0323 - categorical_accuracy: 0.5294 - accuracy: 0.9896 - val_loss: 0.0574 - val_categorical_accuracy: 0.3664 - val_accuracy: 0.9836
Epoch 18/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0305 - categorical_accuracy: 0.5375 - accuracy: 0.9901 - val_loss: 0.0581 - val_categorical_accuracy: 0.3676 - val_accuracy: 0.9835
Epoch 19/20
54982/54982 [==============================] - 213s 4ms/sample - loss: 0.0288 - categorical_accuracy: 0.5455 - accuracy: 0.9907 - val_loss: 0.0590 - val_categorical_accuracy: 0.3664 - val_accuracy: 0.9834
Epoch 20/20
54982/54982 [==============================] - 217s 4ms/sample - loss: 0.0272 - categorical_accuracy: 0.5523 - accuracy: 0.9912 - val_loss: 0.0600 - val_categorical_accuracy: 0.3674 - val_accuracy: 0.9835
```

RNN Model Tensor Board



Model Comparison

Results summary

> Comparing the results of the two architectures tested for the project.

> Metrics:

>> Precision: $\frac{True-Positive}{Actual-Results}$

>> Recall: $\frac{True-Positive}{Predicted-Results}$

>> F1 Score: $\frac{True-Positive+True-Negative}{Total}$

Model	Precision	Recall	F1 Score
Conv Network	0.6009195193008011	**0.4316026940430262**	**0.5023785059177228**
RNN Network	**0.6470524017467248**	0.3875629372915713	0.48476669529301103

Project Code

- Coleb Notebook: https://colab.research.google.com/drive/1oBKK2cDySrh2Oc6tR0Ls7-Gtz0K_0mpK
- Jupyter Notebook: https://drive.google.com/open?id=1xmMhGAgvKHEnbFDS2tQM_5nfh4nYWTTT
- Executed Code PDF Output: https://drive.google.com/open?id=1x3vDJkMC42bQZkH1RipTNg2Padmw2eb_