OUTLINE

# What to Expect ?

1. How to read into your NLP Problem ?
2. Defining your Business Metrics
3. How to read & evaluate your dataset ?
4. Understanding and stocking your option for Text Pre-processing
5. Creating a solid baselines model : At times you will be surprised with the results.
6. Understanding the importance of embeddings
7. Winning smaller battles leads to winning the War : Break your problem statement
8. Deep Neural Network Modelling : Step by Step Process
9. The Codebase for all the above experiments & pipelines

# What NOT to EXPECT ?

1. SOTA results as this is a dummy subsample data
2. SOTA results as that will be published later in our ongoing papers
3. Deep Learning as a Silver Bullet to your problems

Dive-In

## Reading into your NLP Problem

1. *Text Classification* or *Topic Modelling*
2. *Multi Class* or/and *Multi Label*
3. *Divide and Conquer*
   1. Can the problem be broken ?
      1. Compartmentalize you problem.
   2. Define Intrinsic & Extrinsic level of your solution
      1. NER-POS tags are intrinsic component for entity coreference
      2. Word-Embedding are intrinsic component for text classification

## Defining your Business Metrics

1. *Precision-Recall Tradeoff*
2. Any new metrics that needs to be defined
3. *Inference Time Requirement*
   1. *Fast Response Time*: chatbot interaction, face detection
   2. *Slow Response Time*: disease prediction, credit fault detection
4. Frequency of Production releases
   1. *Bi/weekly intervals*: fast iterations, less experiments, quick feedback
   2. *Bi/monthly intervals*: study covariance shift, long training

## Understanding your Dataset

1. Understand whether your dataset or/and source is :
   1. Inherently biased because of the problem.
      1. Cancer Detection
   2. Data Collection techniques is biased.
      1. Gender Detection (male, female, others) built on images of fair-skinned
      2. Predicting road accident rates across the country, data collected only from 2 states
2. *Word Count Distribution*
   1. Exploring general characteristics of the document
   2. Twitter : Short and highly abbreviated
   3. Stackoverflow : Moderate # of words with high jargon
   4. News: High # of words with a descriptive narrative
3. *Class/Label Distribution*
   1. Determining the skewness in the data.
   2. Word count distribution within subgroups
4. *NOTEBOOK*

## Text Preprocessing

This can have many-many options. As Data Science practitioners we need to decide what all is required ?

1. Clean HTML Tags
2. Stemming & Lemmatization
3. Remove Stop words
   1. Generic/Corpus dependent
   2. Example : We should not remove HTML Tags from this corpus
4. Character Encoding
5. Normalizing entities
6. String formatting
7. Alphanumeric - Numeric - Alphabets
8. Spelling correction (Generic as well as Corpus Dependent)
9. Grammar correction
10. Semi structured to proper text

## Strong Baselines

With this power of Deep Learning let’s not forget standard practices to our baselines. They serve as a strong guide for our experiments in Deep Learning.

1. If you are from the industry : Be resistive to your clients or managers to ask time for baselines
2. If you are a researcher : I hope you agree to this
3. If you are a student : Don’t jump into the ocean without knowing how to swim
4. Based on our experience , try the following first :
   1. Parameterized approaches like Naive Bayes, Support Vector Machines
   2. Non parameterized Tree based like Decision Trees, Random Forests
5. Understand and list the problems faced by these classical models
6. Try to address these in Deep Learning models
7. *NOTEBOOK*

## Embeddings

This is the engine oil for any/most NLP tasks. But blindly using a standard out of the box embedding or the Latest & Greatest SOTA can also be of no use unless you understand them.

1. Most of the publicly available embeddings are Trained on Wikipedia or conversation or QA
2. Crawled HTML web pages are also used for many embeddings.
3. Does it relate to this corpus ?
4. If yes ! Go forward & use them.
5. Otherwise train a custom embeddings on your corpus
   1. Either from Scratch.
      1. Given enough data, atleast over 100K thousands.
   2. Fine-tune the above on your corpus. Also called transfer learning
   3. Stack multiple embeddings
      1. Usage of average, max, min, concatenate
6. Always benchmark your problem against 2-3 recent SOTA embeddings and not just the latest one.
7. Our order :
   1. Glove
   2. Flair
   3. ELMO
   4. BERT
   5. Custom Skipgram/CBOW
   6. Custom ELMO
8. *NOTEBOOK*

## Break your Problem Statement

*“Let’s train a model that can do X + Y + Z.”, said a* ***manager****.*

“We need to predict A,B,C and custom D fields. Here is the data train a model and let’s release”, said another ***manager***

Well, just like a simple software engineering, we should divide our machine learning problem into specific smaller problems that are more achievable.

That has been our key to moving to production level AI Systems accepted by clients.

Here we break the problem statement to a hierarchical(pyramid) problem statement :

1. We group the tags/labels different buckets. This can be done via
   1. Data-filtering, i.e. tags that co-occur
   2. Tags distribution , i.e. highly occurring tags v/s rare tags
   3. Grouping based on similarity of text of the questions of the tags
2. Once you group, do manual verification and address any outliers or exceptional case.
3. We build classifiers for each group separately to predict labels of that group.
4. Made the problem statement more handable now.

## Modelling : Step by Step Process

Now that we are loaded with everything, let’s get to the real game.

We will be modelling for each and every group. It should be noted that all the experiments we do are applicable to every group

1. *Outlay your project folder/structure for versioning*
   1. Plan to store & version data
   2. Plan to store & version models
   3. Plan to store & model configuration & results
2. Splitting the dataset of each group
   1. *Ratio of splits*
      1. *Classical Train Dev Test* : 70-15-15
      2. *Custom Train Dev Test*: 80-10-10
      3. *For Large datasets(>3M):* 90-5-5
   2. *Distribution of Labels*
      1. Determining the skewness in the data.
   3. *Handling skewness*
      1. Clipping Distribution
      2. SMOTE
      3. Artificial Boosting
3. Tokenize and create the corpus in the format required
4. *Recurrent Neural Network : Parameters*
   1. Batch Size & Learning Rate
   2. # Hidden Layers
   3. # RNN Layers
   4. # RNN Type
   5. Which Dropout ?
   6. Word Embeddings
   7. Optimizer
5. *Quick Prototyping* 
   1. Build your architecture and run for 3-5 epochs
   2. Note every observation & result
   3. Distribute it to your team
   4. Re-group & try to find reason for the results
6. Select 2-3 most promising architecture and train longer
   1. Error analysis for the promising architectures
7. *Training RNN is unstable & tricky* . **Handy Tips** :
   1. *Learning Rate* : Keep it low and train longer. High lr makes training unpredictable and losses jump around alot
   2. *Batch Size* : Affects RNNs more than you can imagine. Tuning Batch Size & Learning rate go hand in hand
   3. *Early Stopping* : Have a high value for it as training loss moves around a lot. So this might interrupt training with a very bad model
   4. *Checkpoints* : Save only the best model
   5. *Epochs* : High Epoch count with high Early-stopping
   6. Sequence to Sequence are better than Sequence to Vector
      1. Sequence to vector face major vanishing gradient issue to backpropagation through time
      2. In sequence to sequence have strong gradient at every time step
8. Select the best threshold via ROC
9. Ensemble your model and test on both training & test data
10. Check if your business requirements are fulfilled
11. **Next improve your model via continuous learning**
    1. Negative Sampling
    2. Stratified Sampling
    3. Regrouping the labels
    4. Merging some small groups
    5. Using new architectures.

Resources & References

1. Data Source 1 : <https://www.kaggle.com/stackoverflow/stacksample>
2. Data Source 2 : <https://www.kaggle.com/stackoverflow/stack-overflow-tag-network>
3. <https://ruder.io/deep-learning-nlp-best-practices/>
4. <https://ruder.io/a-review-of-the-recent-history-of-nlp/>
5. <https://github.com/keon/awesome-nlp>
6. Code Inspirations :
   1. Text Preprocessing : [nlp\_workshop\_odsc19/Module02 - Text Wrangling at master · dipanjanS/nlp\_workshop\_odsc19](https://github.com/dipanjanS/nlp_workshop_odsc19/tree/master/Module02%20-%20Text%20Wrangling)
   2. Classical ML : [Machine-Learning-with-Python/Multi label text classification.ipynb at master · susanli2016/Machine-Learning-with-Python](https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Multi%20label%20text%20classification.ipynb)
   3. Flair : <https://github.com/flairNLP/flair>
7. <https://ruder.io/word-embeddings-1/index.html>
8. <https://medium.com/genei-technology/richer-sentence-embeddings-using-sentence-bert-part-i-ce1d9e0b1343>