# **Problem Statement**

The challenge is to learn a machine learning model that classifies a given line as belonging to one of the 12 novels encoded in numbers (0-11). This is a text classification case however, data is obfuscated and contains continuous sequence of characters for each sentence therefore, usual NLP pipeline (tokenisation, lemmatization, stemming and stop word removal) is not applicable in my opinion. This challenge consists of the following steps.

# **1-Data Preparation for Model**

1. Data is loaded used Pandas modules. On further inspection of target i.e y\_train , I found that classes are not balanced. For example, class 7 has 15% of occurrence whereas class 0 has only 1.6% frequency. Due to this reason, finding a model that could predict class 7 as good as class 0 will be challenging.
2. Data is divided into training and validation sets in 75%-25% ratio for experiment.
3. Vectorization (converting text into matrices) is done based on characters using countvectorizer, tf-idf, word embedding i.e GloVe. In our case, Characters are considered as individual tokens whereas t counts and frequency is calculated on character level. Also, SVD is used to factorize matrices. All these vectorization methods are used with machine learning models.

# **2- Implementation of Machine Learning Models**

I have implemented classical models such as Naive Bayes, Logistic Regression classifier, Support Vector classifier, and XGBoost Classifier to check performance on given data. I used log loss and accuracy as performance metrics.

Formula for accuracy percentage = e^(-logloss)

|  |  |  |
| --- | --- | --- |
| **Models** | **log\_loss** | **accuracy** |
| Logistic classifier using CountVectorizer | 0.915 | 40.05% |
| Logistic classifier using TFIDF | 1.105 | 33.13% |
| Naive Bayes using CountVectorizer | 1.480 | 22.77% |
| Naive Bayes using TFIDF | 4.991 | 6.8% |
| SVC using SVD | 1.098 | 33.35% |
| XGB classifer using tf-idf | 0.775 | 46.08% |
| XGB classifier using tfidf\_svd | 1.24 | 28.74% |
| XGB classifier using countvector\_svd | 0.736 | 47.87% |
| XGB classifier using glove embedding | 1.938 | 14.40% |

These models are only used as base idea to check model scores. For reliable results, I applied deep learning algorithms.

# **3- Implementation of Deep Learning Models**

For our deep learning algorithms, key assumption is that this text data even though obfuscated is in sequence. That's why RNN (LSTMS, GRU) were implemented. We used Keras tokenizer on character level for processing input sentences

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **# of Epochs** | **Training loss** | **Training accuracy** | **Validation loss** | **Validation accuracy** | **Comment** |
| **Sequential Neural Net** | 50 | 1.6322 | 42.87% | 1.7170 | 40.36% | Overfitting |
| LSTM | 100 | 1.3040 | 54.55% | 1.2884 | 55.19% | balanced |
| Bi-Direct\_ LSTM | 10 | 1.7404 | 37.36% | 1.6641 | 40.51% | Overfitting |
| GRU | 10 | 2.2270 | 19.87% | 2.1895 | 21.96% | underfitting |
| CNN-1D convnet | 10 | 0.4081 | 85.97% | 1.4948 | 61.82% | overfitting |
| CNN with glove | 100 | 0.2288 | 97.23% | 1.2606 | 61.11% | overfitting |
| BERT | 5 | 2.486 | - | 2.485 | 23.01% | balanced |

1. As of above results, we can find that CNN models perform very well with high accuracy and low loss score however, I did not consider them due to overfitting.
2. In my results, BERT and LSTM were the most balanced ones.
3. BERT model did not show over or under fitting with loss function. This is a good sign. But I have not trained this model on enough iteration to consider it suitable model. For this reason, I called it 2nd most suitable model. I wish to have more computing resources so that I may experiment more.
4. Finally, I used LSTM as “model” to predict given test data(xtest\_obfuscated.txt). Other evaluation matrices such as precision, recall, f1 score were also calculated and found consistent.

# **4-Limitation**

1. I must confess that I didn't run this model with enough epochs. This was one issue working with colab as it was breaking down too often with these models. For example, on BERT model working with colab I went out of RAM once and had to restart all notebook code again. It was hell of an experience for running all models again. So, I didn't run them again. I ran pytorch code. That is why one can find code that reads pandas, train\_test and preprocessing steps once again in that section. The code looks very messy in that part.
2. I could not do sanity check as data was obfuscated.

# **5-Possible Improvements**

1. Though I did not use consistent epochs, my aim was to observe if model is overfitting/underfitting. I could see in some model after 10 epochs. I did give some model more epochs such as CNN because their computing time was faster and also, I was getting very good scores.
2. I used standard scheme i.e relu as activation between layers, softmax for output layer and adam as optimizer. But there could be more combinations made to check performance.
3. In my opinion, models are only alive when they are used in production. Model deployment part is crucial for organizations. I could not perform that but, I would like to provide a conceptual framework with methods and tool for model deployment.

There is an emphasis on creating automated pipelines for machine learning projects. This is my suggested framework.

# **6-Results submitted**

Results are submitted as.

1. ***y\_test.txt*** file containing predicted class results on format on y\_train.txt file,
2. a ***result.csv*** file containing test text, predicted class results, and probability of that class,
3. notebook code implemented on google colab. I have also added comments to show few concepts, explanation of code, model selection,
4. ***Github link*** for code,
5. ***requirement.txt*** file to show what dependencies were needed,
6. ***description*** file: I used this file as explanation of my approaches, methods, and reasoning. It is not a tutorial on “how” part. I tried to explain that detail in notebook with comments and notes as much as I could.

# **References**