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 problem 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

a. Data was loaded using Pandas module. On further inspection of target i.e y_train , I found that classes were not balanced. For example, class 7 had 15% of occurrence whereas class 0 had only 1.6% frequency. Due to this reason, finding a model that could predict class 7 as good as class 0 will be challenging.
b. Data was divided into training and validation sets in 75%-25% ratio for experiment.
c. Vectorization (converting text into matrices) was done based on characters using countvectorizer, tf-idf, word embedding i.e GloVe. In our case, characters are considered as individual tokens whereas 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 = 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 were 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 is why RNN (LSTMS, GRU) were implemented. I used Keras tokenizer on character level for processing input sentences. For class imbalance, I used "*class_weight*". For saving extra computation, I used "*earlystopping*" i.e if model stops improving based on validation loss.

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	13.01%	balanced

a. As of above results, I can find that CNN models performed very well with high accuracy and low loss score however, I did not consider them suitable model due to overfitting.
b. In my results, BERT and LSTM were the most balanced ones.
c. BERT model did not show over or under fitting as could be seen in table above. This is a good model too. 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 I could have more computing resources so that I might experiment with more epochs.
d. 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.

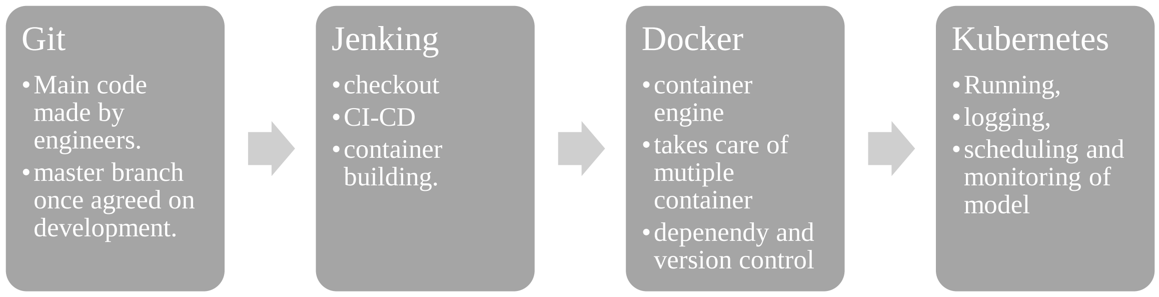
4-Limitation

a. I must confess that I didn't run these models with enough or consistent 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 might find repeating code that reads pandas, train_test and preprocessing steps once again in section 6.6. The code looks very messy in that part as well.
b. I could not do sanity check as data was obfuscated.
c. Some code from machine learning part was also removed at last moment due to computing overweight of notebook. It could be better to create two notebooks- one with ML basic models and one with deep learning models.

d. Finally, I could not use my personal PC due to computing limitations for this project. I would have created a docker image to avoid any dependency issues. I am not sure if provided notebook will run smoothly on other systems. To avoid this, I am also sharing Github link.

5-Possible Improvements

a. 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 that there was either over or under fitting. I did give some model more epochs such as CNN because their computing time was faster and also, I was getting very good scores. Eventually, they showed overfitting and hence, rejected.
b. 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 and variations to check performance.
c. 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.


Git
•Main code made by engineers.
•master branch once agreed on development.
Jenking
•checkout
•CI-CD
•container building.
Docker
•container engine
•takes care of mutiple container
•depenendy and version control
Kubernetes
•Running,
•logging,
•scheduling and monitoring of model

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

6-Results submitted

Results are submitted as.

i. ***y_test.txt*** file containing predicted class results based on format of y_train.txt file.
ii. ***result.csv*** file containing test text, predicted class results, and probability of that class.
iii. notebook code implemented on google colab. I have also added comments to show concepts, explanation of code, and implementation steps.
iv. ***Github link*** for code,
v. ***requirement.txt*** file to show what modules and dependencies were needed to run this project.
vi. ***description*** file: I used this file as explanation of 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

- Convolutional Neural network for text classification:[accessed : 08.09.2020]

http://mbenhaddou.com/2019/12/29/convolutional-neural-network-for-text-classification/

- Classification of obfuscated text data:[accessed : 08.09.2020]

https://datascience.stackexchange.com/questions/23608/classification-of-obfuscated-text-data

- Classification of obfuscated text data(2): [accessed: 06.09.2020]

https://github.com/dupree/obfuscated-text-classification

- MLOps with a Feature Store:[accessed : 08.09.2020]

https://towardsdatascience.com/mlops-with-a-feature-store-816cfa5966e9