Kaggle Competition

H6751 - Text and Web Mining

AY 2019-20

Submitted By

Name	Matriculation Number
Thant Htoo Zin	G1801166F

1. Data preprocessing

Data preprocessing process is the first and critical step to create a Machine Learning model. The data preprocessing step involves importing training and testing data from a csv file and some basic data cleaning processes which increase the accuracy and efficiency of a machine learning model. Given the dataset are user's reviews so that contains unstructured, noise data and unusable format which cannot be directly used for machine learning models. Therefore to minimize the unstructured data, clean all the data with the following step by step procedure.

- Data contains html entities such as <html> <header> </head></html> that are removed by using regular expression.
- Removed Stopword because Stopwords considered as low information or non critical information in the text classification.
- To improve model accuracy, remove extra noises such as hyperlink, special character, numeric and single character by using regular expression.
- As the given data sets are user's reviews from the internet so that difference people can use different ways of writing for the same meaning. To reduce different forms of word to core meaning, I decided to use lemmatization because lemmatization is more accurate and intelligent than stemming. Lemmatization changes the word to its meaningful base form, whereas stemming just removes some characters that may cause the incorrect meanings.

1.1 Feature Engineering

The second step is a feature engineering step that transforms raw text data into numerical structure. There are a lot of feature engineering techniques such as Count Vector (Vectorization), n-grams, TF - IDF, Cosine Similarity, Jaccard Similarity, Levenshtein Distance, Feature Hashing and so on. Among them I decided to use **CountVector**, **TF - IDF** and **n-grams** as features.

By using the **CountVectorizer** function, it tokenizes the text document and converts it into a matrix of word frequency count that is used as weight. The structure of the CountVector matrix is that all the rows represent the row of the dataset, all the columns represent a term of the dataset and every cell represents the frequency count of each term in the dataset. For parameter tuning, I decided to use word level or **1-gram** in the CountVercor feature because in the given dataset contain more individual words rather than the combined word.

TF-IDF (Term Frequency-Inverse Document Frequency) can provide important words or terms from the dataset. The formula of TF and IDF as following:

- TF - No of terms that appear in the documents / Total no of terms in the documents
- IDF - Log (total no of documents / no of documents with the term in it)
- TF-IDF - TF * IDF

For the TF-IDF feature, I generated two different levels such as TF-IDF with word level and TF-IDF with word level by using **N-gram** (2-3 range). The parameter for max_fetaures set as 5,000 that is neither too large nor too small and that can perform reasonably well. For **max_df** parameter, it was set by 0.8 which means that if the terms appear in more than 80% of the document that will be ignored. And the integer value 5 sets for the min_df which mean that it will ignore terms appear in less than 5 documents in the dataset.

2. Model Validation

There are many techniques to validate the accuracy of the model predictions. Among them I used the **cross Validation technique** that enables us to estimate whether our model got the correct pattern or not. In the cross validation technique I decided to use the **Validation method** and **K-Fold Cross Validation Method** for this assignment.

Validation method: To validate the model prediction, split the data into two portions as training dataset (70%) and testing dataset (30%). The 70% training dataset is used to train the model then the 30% testing dataset is to test the model performance and accuracy. This method may have some bias on splitting so that I decided to use another validation method called K-Fold Cross Validation method.

K-Fold Cross Validation Method: For the K-Fold Cross Validation, I split the given dataset into 2 subset or 2 folds (K fold). We need to iterate 2 times with different subsets and each time 1 subset (K-1 fold) becomes the training data and the remaining reserved subset becomes the testing data. So that by using the K-Fold cross validation method all data were used for both testing and training data.

3. Model Building

In this step I will choose the classification algorithm and train the machine learning model by using clean datasets and features that were created in the previous step 1. There are various choices of machine learning models to estimate accuracy, among them I chose the following algorithms to apply in this assignment.

Naive Bayes Classifier: This algorithm is simple to implement and the results are good in most cases especially for very large dataset. It performs well not only in binary class prediction but also multi class prediction. Moreover its processing time is faster than the other algorithm.

Linear Classifier (Logistic Regression): It is easy and quick to implement and it does not require many computational resources.

Support Vector Machine: it is very effective in high dimensional space especially for text dataset. It is more suitable with binary classification and works well in unstructured data and semi-structured data.

Random Forest Classifier (Bagging): Random Forest Classifier uses Ensemble Learning technique and based on bagging algorithm. It takes more time to train rather than other models. Random Forest reduces the overfitting problem so that it improves the accuracy of the model.

Xtereme Gradient Boosting Classifier (Boosting): Boosting that reduces variance & bias and that help to convert weak learners to strong learners. There are many types of boosting algorithms such as Gradient Booting, Adaptive Boosting, Catboostclassifier, and Extreme Gradient Boosting and so on. I chose the Extreme Gradient Boosting because it is simple and effective. It also delivers high performance and accuracy.

3.1 Select Best Model

Finally each model predicted the accuracy of the data, among them the SVM model with WordLevvel TFIDF feature got the highest accuracy (70%) and considered quite good. So I decided to select and submit to the Kaggle competition. For the accuracy result comparison chart and table, please refer to the appendix section.

4. Appendix

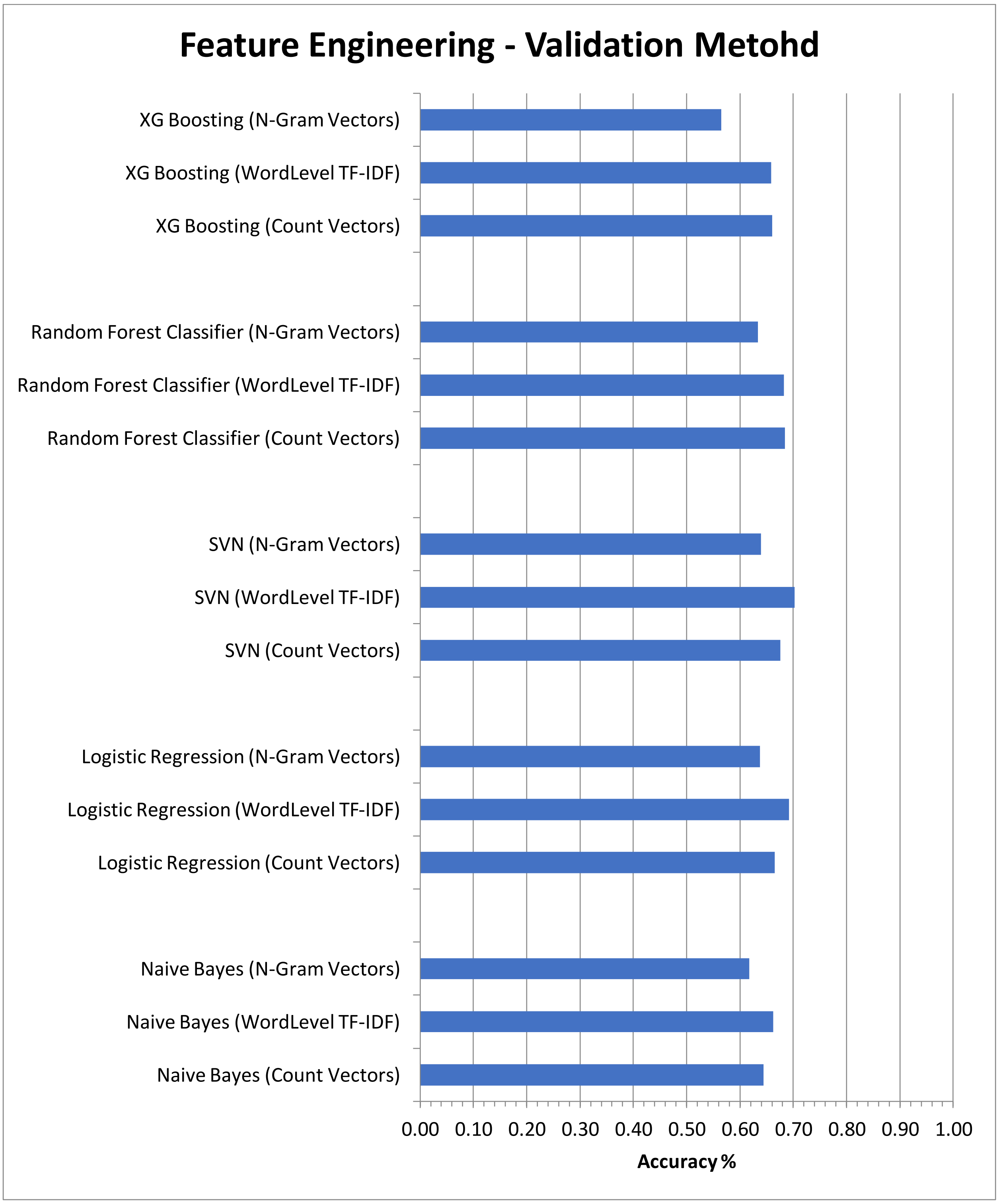

Figure-1: Comparison between the accuracy of the model by using Validation

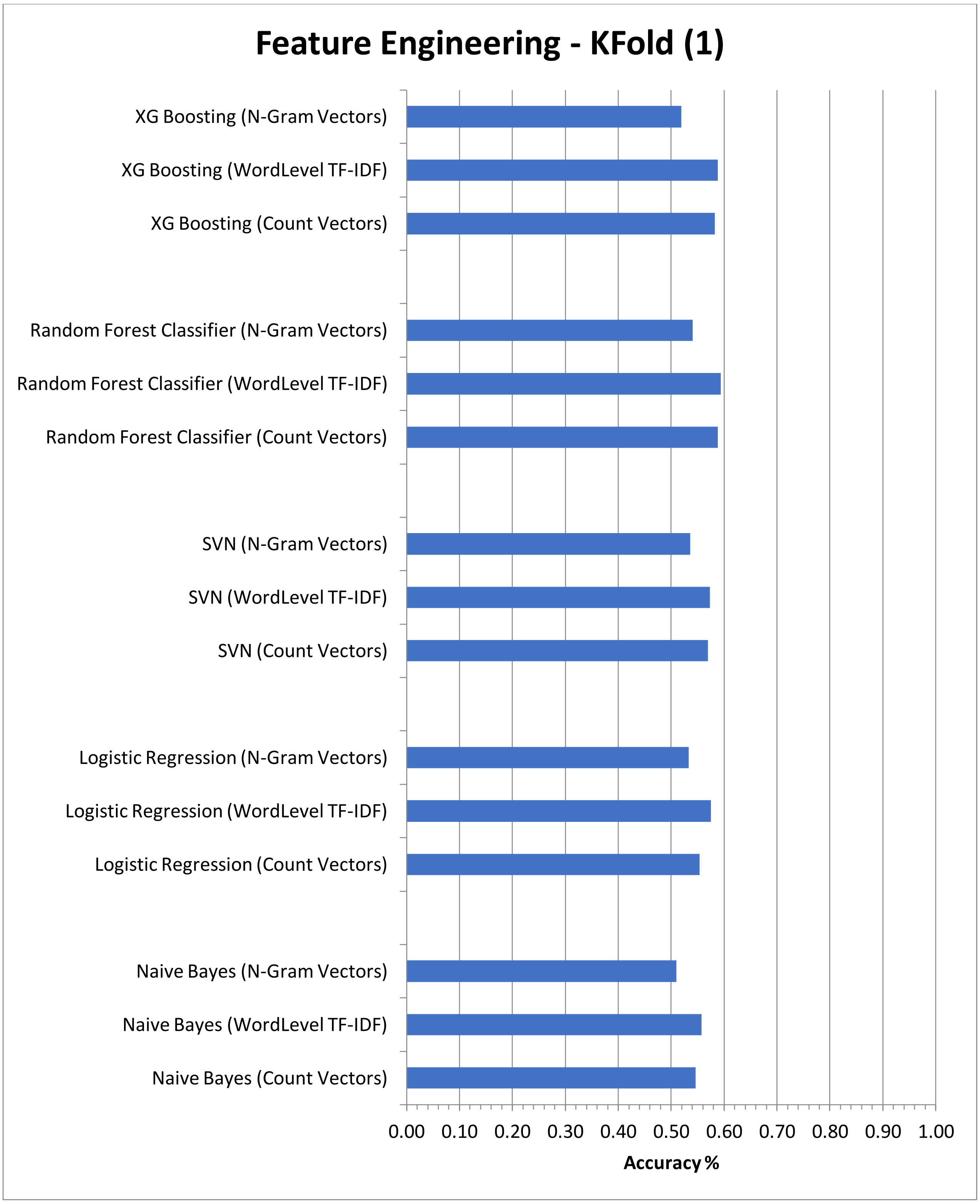

Figure-2: Comparison between the accuracy of the model by using K-fold 1st Iteration

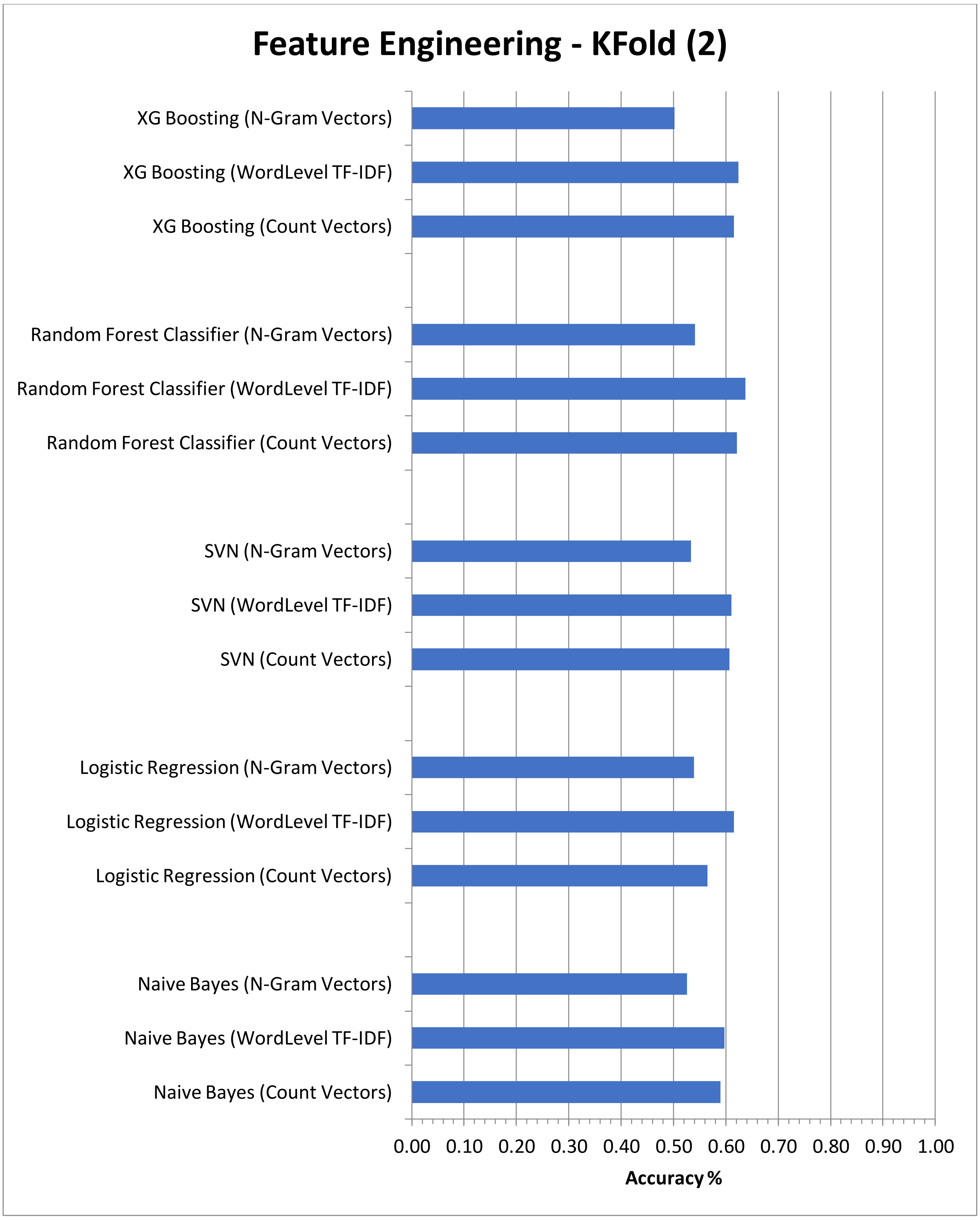

Figure-3: Comparison between the accuracy of the model by using K-fold 2st Iteration

	Validation method	K-Fold	
		1-Fold	2-Fold
Naive Bayes (Count Vectors)	0.64	0.55	0.59
Naive Bayes (WordLevel TF-IDF)	0.66	0.56	0.60
Naive Bayes (N-Gram Vectors)	0.62	0.51	0.53
Logistic Regression (Count Vectors)	0.66	0.55	0.56
Logistic Regression (WordLevel TF-IDF)	0.69	0.57	0.62
Logistic Regression (N-Gram Vectors)	0.64	0.53	0.54
SVN (Count Vectors)	0.67	0.57	0.61
SVN (WordLevel TF-IDF)	**0.70**	0.57	0.61
SVN (N-Gram Vectors)	0.64	0.54	0.53
Random Forest Classifier (Count Vectors)	0.68	0.59	0.62
Random Forest Classifier (WordLevel TF-IDF)	0.68	0.59	0.64
Random Forest Classifier (N-Gram Vectors)	0.62	0.54	0.54
XG Boosting (Count Vectors)	0.64	0.58	0.62
XG Boosting (WordLevel TF-IDF)	0.65	0.59	0.62
XG Boosting (N-Gram Vectors)	0.56	0.52	0.50

Table-1: Comparison between the accuracy of the model

PS: The green color cell is selected for Kaggle submission.

=== END ===