Apr 25, 2022

Machine Learning Final Project
Team – Shallow Learning

Spring 2022

Abhishek Kumar Bhagat, Panther ID: [redacted]
[redacted]
[redacted]

Abstract

Nature Project

Images now make up for a significant share of global data creation and image classification is probably the most important part of digital image analysis. Image classification applications are used in many areas, such as medical imaging, in satellite images, traffic control systems, and more. We wanted to have hands-on experience with real world image data classification problem and hence selected this Nature project which is about classification of images of flowers belonging to 35 different classes. We achieve validation accuracy of 45.61 for best model i.e., ResNet50, whereas for CNN we receive low validation accuracy i.e., 2.88 and for VGG 16 2.25 and VGG 19 2.50.

Topic Analysis Project

With the introduction of Google as the leading search engine, our world being more and more digitalized, NLP (Natural Language Processing) has crept into our lives almost unnoticed. This project is a real-world data NLP problem which deals with people's comments scraped from an online platform, belonging to 40 different categories and the goal is to build a model which can correctly predict which comment belongs to which category, on unseen data. We selected this project since it is related to topic classification and can be our gateway to more complex NLP problems like sentiment analysis, text summarization, trend extraction from social media websites and the list goes one. We achieved an overall accuracy of 45% on the test data provided to us with an AUC score of 91%. Due to computational constraint, we carried our experiments on a smaller dataset, the code for which has also been attached.

Chapter 1 - Nature Project

Introduction

People have practiced landscaping and gardening for centuries. As far back as the ancient Mayans, humans were manipulating the land for both aesthetic and practical reasons. Many of us wish there was an app that could take a picture of the plant and would tell us what the plant type is. While there have been some efforts on that front, we still have a long way to develop reliable software for that.

In this project, we are planning to come up with a prototype system. Due to the limited computational resources, we would only explore 35 class labels, although upon the availability of more data the approach can be extended to a more commercialized product.

Since we have to classify images based on labels, we plan to use neural network models such as CNN, Resnet50, VGG16, and VGG19. These are the best models available to work with classification problems.

Data Description

- We have 15000 train images.
- For each image, we have an image number associated with a class label number.
- Totally, we have 35 class labels.

Distribution of each class label.

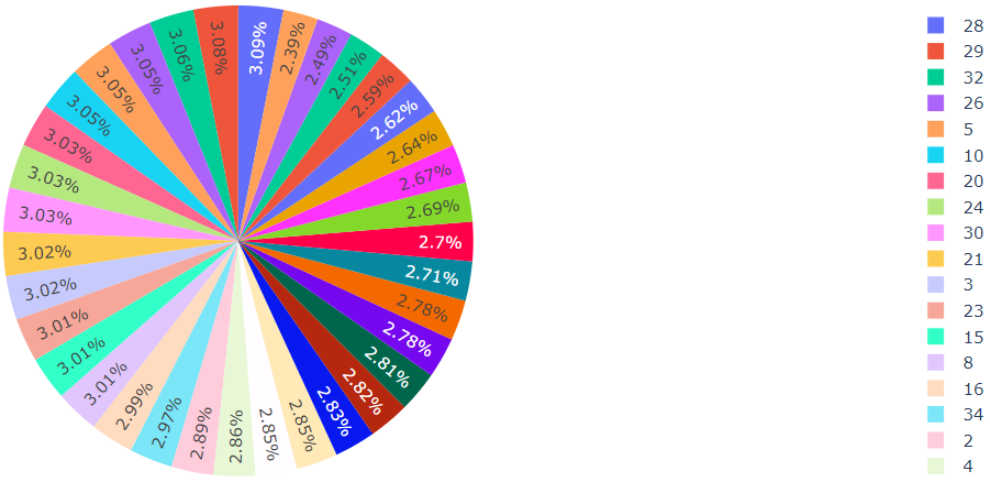


Fig. 1 Distribution of each class label

Batch visualization of Image with labels

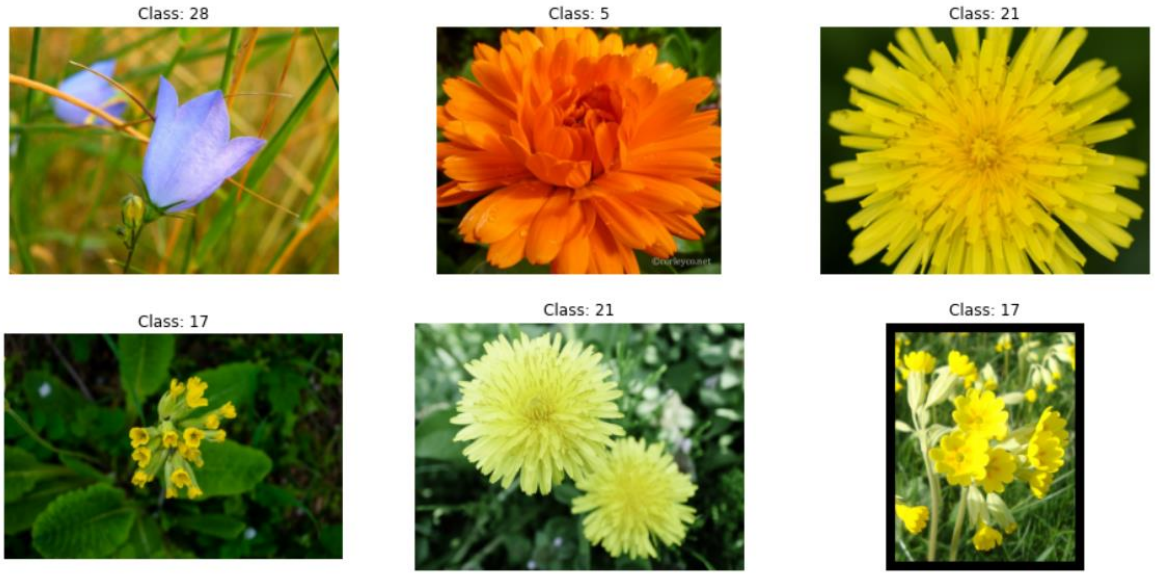


Fig. 2 Batch visualization of Image with labels

Feature Engineering:

We used Keras TensorFlow to optimize the image dataset. We used ImageDataGenerator and flow_from_dataframe functions to optimize the image dataset.
HEIGHT = 128
WIDTH=128
SEED = 45
BATCH_SIZE= 64

Data Modeling

Convolutional neural network (CNN):

A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification and have also found success in natural language processing for text classification.

A convolutional neural network is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.

Types of convolutional neural networks

- AlexNet
- VGGNet
- GoogLeNet
- ResNet

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully connected (FC) layer

ResNet50

ResNet50 is a variant of the ResNet model which has 48 Convolution layers along

with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations. It is a widely used ResNet model and we have explored ResNet50 architecture in depth.

VGG-16

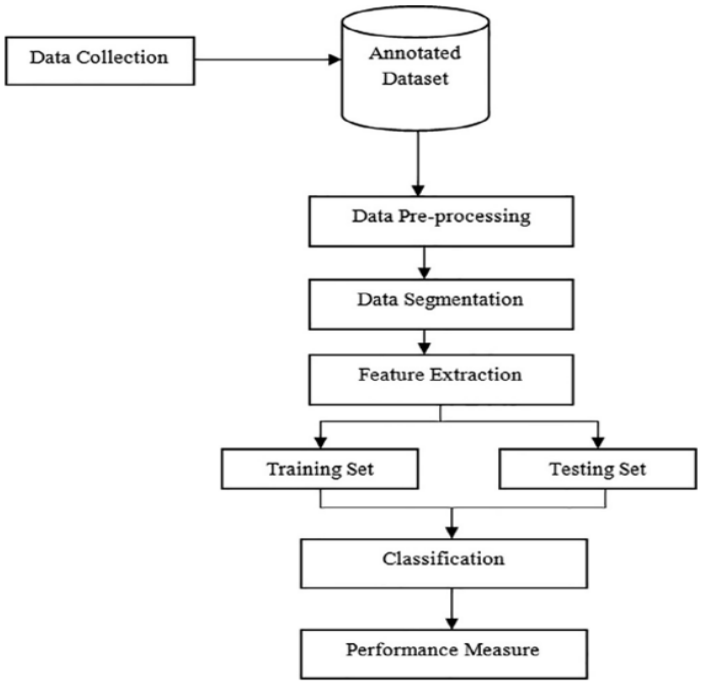
It is a Convolutional Neural Network (CNN) model proposed by Karen Simonyan and Andrew Zisserman at the University of Oxford. The idea of the model was proposed in 2013, but the actual model was submitted during the ILSVRC ImageNet Challenge in 2014. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was an annual competition that evaluated algorithms for image classification (and object detection) at a large scale. They did well in the challenge but couldn't win.

VGG-19

It is a trained Convolutional Neural Network, from Visual Geometry Group, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers.

Check this link for more detail.

Below is the architecture diagram of Process:



Model Summary:

CNN:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 119, 119, 25)      7525

 dropout (Dropout)           (None, 119, 119, 25)      0

 conv2d_2 (Conv2D)           (None, 117, 117, 20)      4520

 max_pooling2d (MaxPooling2D  (None, 58, 58, 20)       0
 )

 conv2d_3 (Conv2D)           (None, 57, 57, 15)        1215

 dropout_1 (Dropout)         (None, 57, 57, 15)        0

 flatten (Flatten)           (None, 48735)             0

 dense (Dense)               (None, 256)               12476416

 dense_1 (Dense)             (None, 64)                16448

 dense_2 (Dense)             (None, 35)                2275

=================================================================
Total params: 12,508,399
Trainable params: 12,508,399
Non-trainable params: 0
_________________________________________________________________
```

Model Training details:

```
Epoch 144: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 151s 644ms/step - loss: 0.8252 - accuracy: 0.7293
Epoch 145/150
235/235 [==============================] - ETA: 0s - loss: 0.8455 - accuracy: 0.7265
Epoch 145: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 153s 650ms/step - loss: 0.8455 - accuracy: 0.7265
Epoch 146/150
235/235 [==============================] - ETA: 0s - loss: 0.8331 - accuracy: 0.7325
Epoch 146: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 151s 642ms/step - loss: 0.8331 - accuracy: 0.7325
Epoch 147/150
235/235 [==============================] - ETA: 0s - loss: 0.8348 - accuracy: 0.7311
Epoch 147: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 153s 651ms/step - loss: 0.8348 - accuracy: 0.7311
Epoch 148/150
235/235 [==============================] - ETA: 0s - loss: 0.8480 - accuracy: 0.7266
Epoch 148: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 152s 646ms/step - loss: 0.8480 - accuracy: 0.7266
Epoch 149/150
235/235 [==============================] - ETA: 0s - loss: 0.8592 - accuracy: 0.7175
Epoch 149: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 153s 650ms/step - loss: 0.8592 - accuracy: 0.7175
Epoch 150/150
235/235 [==============================] - ETA: 0s - loss: 0.8393 - accuracy: 0.7296
Epoch 150: saving model to ./drive/My Drive/Machine Learning/Trained_Model/cp.ckpt
235/235 [==============================] - 153s 650ms/step - loss: 0.8393 - accuracy: 0.7296
INFO:tensorflow:Assets written to: ram://b634461a-a687-410a-a8ab-2081ca3faff3/assets
```

ResNet50

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 batch_normalization (BatchN  (None, 2048)             8192
 ormalization)

 dense (Dense)               (None, 2048)              4196352

 batch_normalization_1 (Batc  (None, 2048)             8192
 hNormalization)

 dense_1 (Dense)             (None, 1024)              2098176

 batch_normalization_2 (Batc  (None, 1024)             4096
 hNormalization)

 dense_2 (Dense)             (None, 35)                35875

=================================================================
Total params: 29,938,595
Trainable params: 6,340,643
Non-trainable params: 23,597,952
_________________________________________________________________
```

```
100/100 [==============================] - 68s 683ms/step - loss: 1.9360 - accuracy: 0.4288 - val_loss: 2.2345 - val_accuracy: 0.4672
Epoch 135/150
100/100 [==============================] - 68s 677ms/step - loss: 1.9320 - accuracy: 0.4483 - val_loss: 2.6616 - val_accuracy: 0.4351
Epoch 136/150
100/100 [==============================] - 68s 677ms/step - loss: 1.9092 - accuracy: 0.4341 - val_loss: 3.3260 - val_accuracy: 0.4574
Epoch 137/150
100/100 [==============================] - 68s 678ms/step - loss: 1.9635 - accuracy: 0.4241 - val_loss: 4.0866 - val_accuracy: 0.4548
Epoch 138/150
100/100 [==============================] - 68s 683ms/step - loss: 1.9555 - accuracy: 0.4281 - val_loss: 3.0587 - val_accuracy: 0.4718
Epoch 139/150
100/100 [==============================] - 68s 679ms/step - loss: 1.9204 - accuracy: 0.4528 - val_loss: 2.7180 - val_accuracy: 0.4554
Epoch 140/150
100/100 [==============================] - 68s 677ms/step - loss: 1.9297 - accuracy: 0.4505 - val_loss: 2.3531 - val_accuracy: 0.4646
Epoch 141/150
100/100 [==============================] - 68s 679ms/step - loss: 1.9153 - accuracy: 0.4471 - val_loss: 2.1685 - val_accuracy: 0.4364
Epoch 142/150
100/100 [==============================] - 68s 679ms/step - loss: 1.9107 - accuracy: 0.4384 - val_loss: 2.4385 - val_accuracy: 0.4803
Epoch 143/150
100/100 [==============================] - 68s 680ms/step - loss: 1.9008 - accuracy: 0.4497 - val_loss: 2.8418 - val_accuracy: 0.4764
Epoch 144/150
100/100 [==============================] - 68s 678ms/step - loss: 1.9203 - accuracy: 0.4467 - val_loss: 2.3126 - val_accuracy: 0.4705
Epoch 145/150
100/100 [==============================] - 67s 674ms/step - loss: 1.9577 - accuracy: 0.4319 - val_loss: 2.5869 - val_accuracy: 0.4718
Epoch 146/150
100/100 [==============================] - 68s 678ms/step - loss: 1.9289 - accuracy: 0.4291 - val_loss: 2.5983 - val_accuracy: 0.4777
Epoch 147/150
100/100 [==============================] - 68s 678ms/step - loss: 1.9111 - accuracy: 0.4425 - val_loss: 2.4183 - val_accuracy: 0.4567
Epoch 148/150
100/100 [==============================] - 68s 676ms/step - loss: 1.8855 - accuracy: 0.4484 - val_loss: 2.2345 - val_accuracy: 0.4797
Epoch 149/150
100/100 [==============================] - 67s 674ms/step - loss: 1.9140 - accuracy: 0.4484 - val_loss: 2.1139 - val_accuracy: 0.4489
```

VGG 19

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_39 (Conv2D)          (None, 224, 224, 64)      1792

 conv2d_40 (Conv2D)          (None, 224, 224, 64)      36928

 max_pooling2d_15 (MaxPoolin  (None, 112, 112, 64)     0
 g2D)

 conv2d_41 (Conv2D)          (None, 112, 112, 128)     73856

 conv2d_42 (Conv2D)          (None, 112, 112, 128)     147584

 max_pooling2d_16 (MaxPoolin  (None, 56, 56, 128)      0
 g2D)

 conv2d_43 (Conv2D)          (None, 56, 56, 256)       295168

 conv2d_44 (Conv2D)          (None, 56, 56, 256)       590080

 conv2d_45 (Conv2D)          (None, 56, 56, 256)       590080

 max_pooling2d_17 (MaxPoolin  (None, 28, 28, 256)      0
 g2D)

 conv2d_46 (Conv2D)          (None, 28, 28, 512)       1180160

 conv2d_47 (Conv2D)          (None, 28, 28, 512)       2359808

 conv2d_48 (Conv2D)          (None, 28, 28, 512)       2359808

 max_pooling2d_18 (MaxPoolin  (None, 14, 14, 512)      0
 g2D)

 conv2d_49 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_50 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_51 (Conv2D)          (None, 14, 14, 512)       2359808

 max_pooling2d_19 (MaxPoolin  (None, 7, 7, 512)        0
 g2D)

 flatten_2 (Flatten)         (None, 25088)             0

 dense_6 (Dense)             (None, 4096)              102764544

 dense_7 (Dense)             (None, 4096)              16781312

 dense_8 (Dense)             (None, 35)                143395

=================================================================
Total params: 134,403,939
Trainable params: 134,403,939
Non-trainable params: 0
_________________________________________________________________
```

Model Evaluation

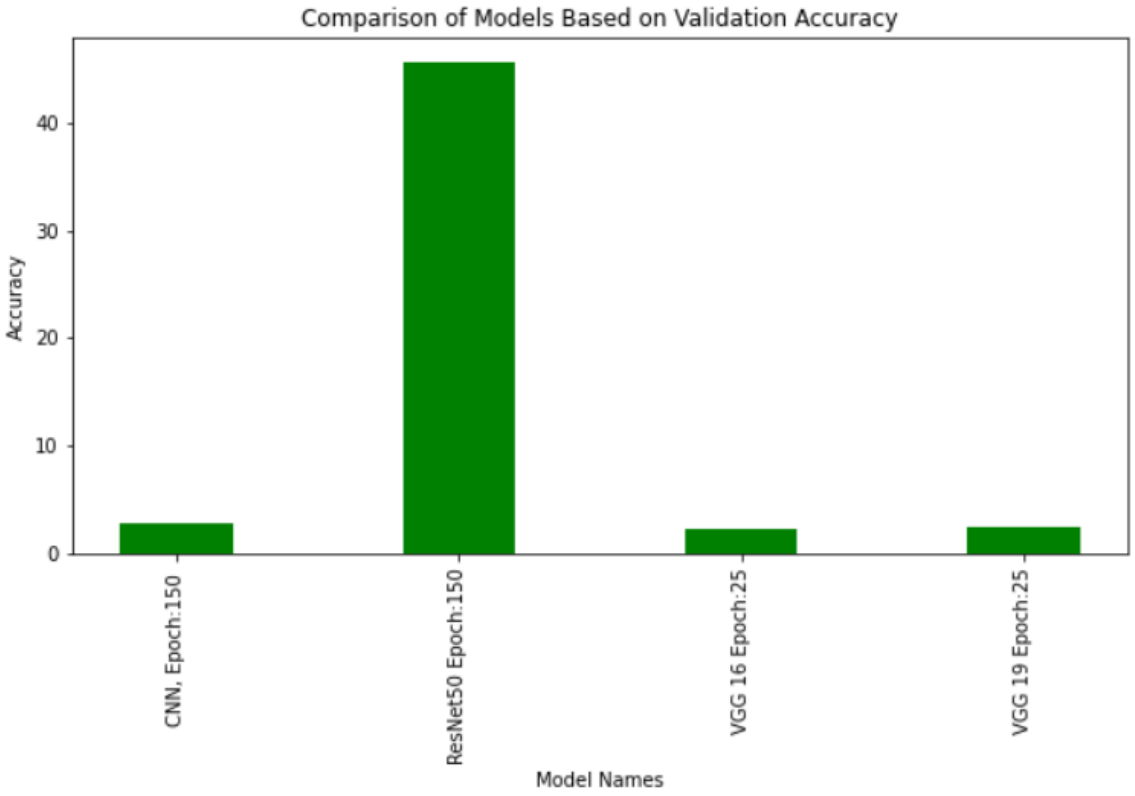


Fig. 3 Validation Accuracy Comparison

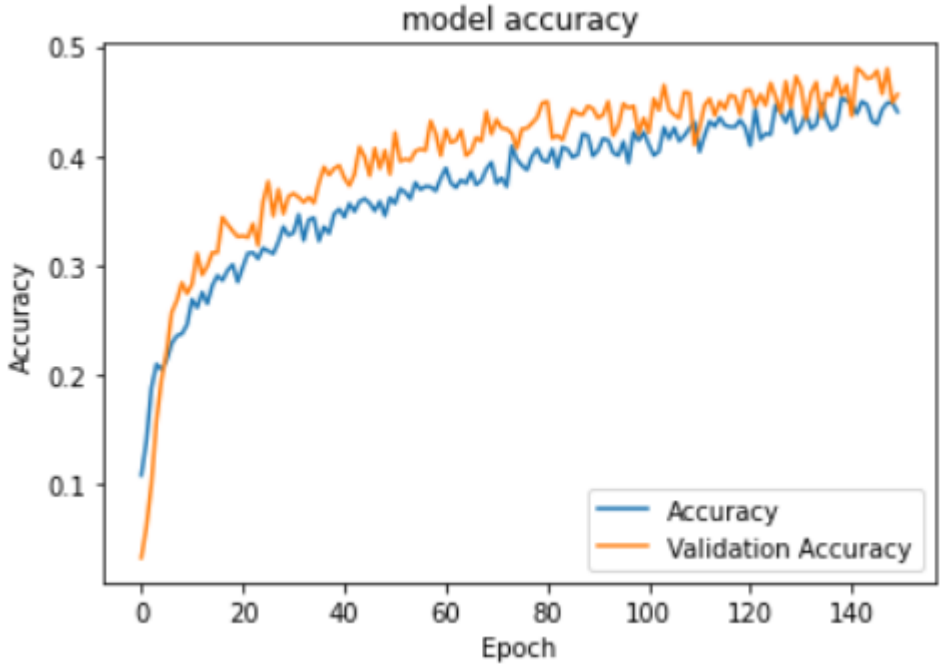


Fig. 4 Training and Validation Accuracy

Conclusion and Challenges

This model requires a significant amount of GPU and RAM power to function properly. Due to the limited available computational resources, our team could neither explore or experiment more complicated models or run-on higher number of epochs.

In conclusion, our model`s results for training and validation accuracy are closely aligned at around 45% for ResNet 50. In the future, we would like to utilize more GPU (Google Colab, Azure) to explore other options in improving our model.

Chapter 2 - Topic Analysis

Introduction

The ability to classify big data is becoming a great power nowadays. Technology has enabled mankind to collect data faster than ever before. However, all those data will be useless unless there is an efficient method to accurately classify the labels for upcoming input data based on the existing data labels. This is important in terms of minimizing human error and optimizing the output accuracy.

We used our given training dataset and train a model that could perform a multi-class prediction on a test dataset of 100,000 rows of text. We also explored and experimented a variety of models such as KNN (K nearest neighbor), Light gradient boosting, Extreme Gradient Boosting, etc.

Data Description

Our training dataset has 900,000 rows of textual data (sentences, paragraphs…) classified into 40 separate categories.

Data Exploration

We looked into the distribution for the number of sentences and the number of words for each sample. The number of sentences distribution seems to hover around 2 and 3. The number of words averages at around 10-15 words. Since the class distribution is relatively balanced, it is not required to oversample the minority classes.

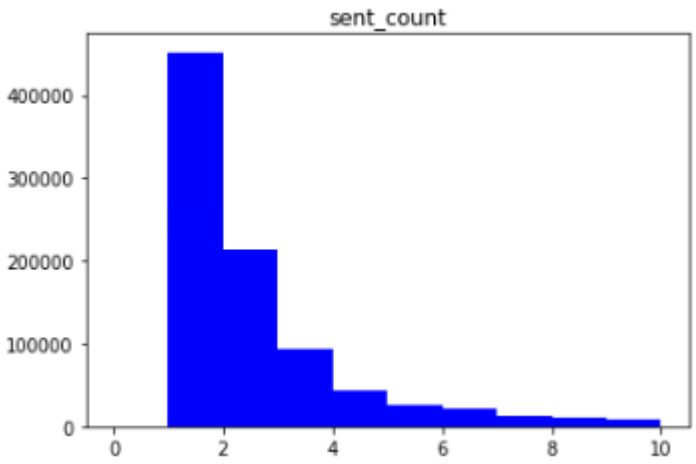


Fig. 5 Distribution for the number of sentence

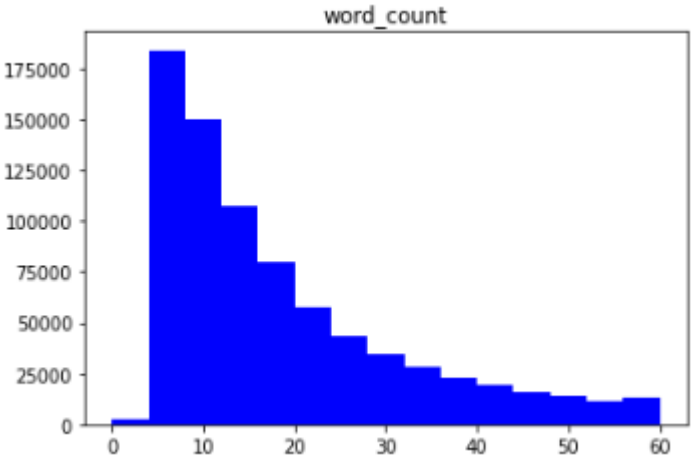


Fig. 6 Distribution for the number of word

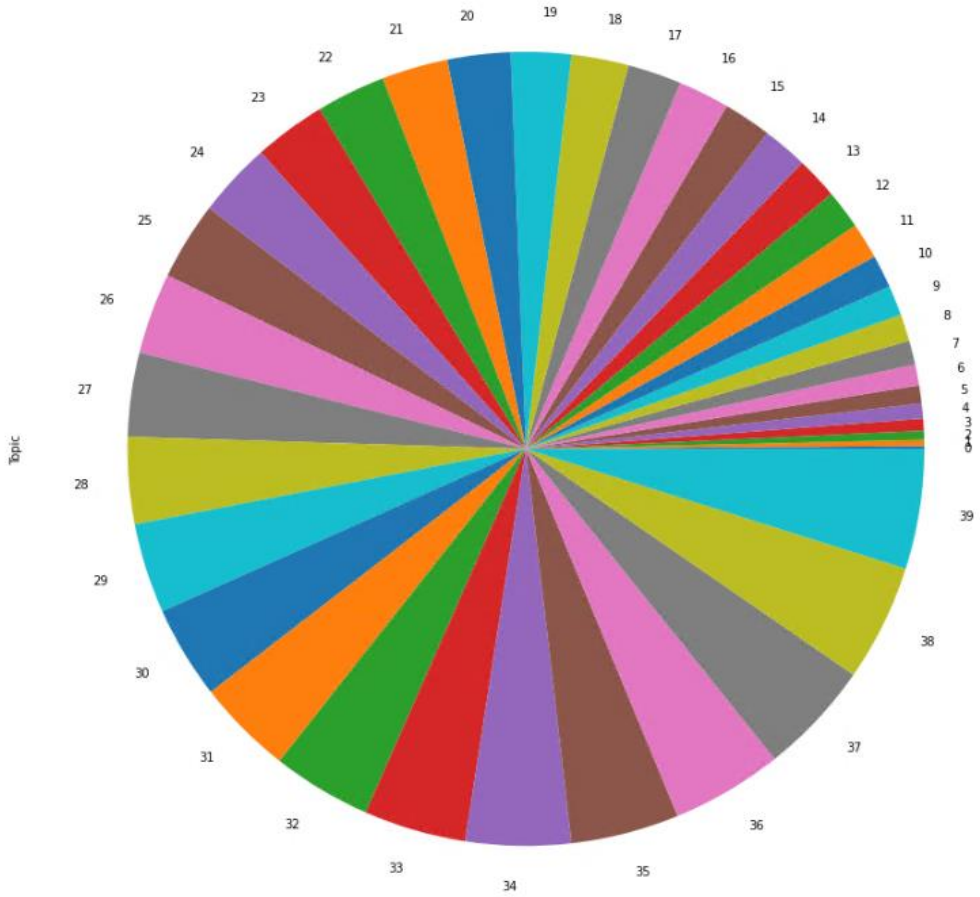


Fig. 7 Distribution for 40 labels

Preprocessing

For this experiment, we initially preprocessed the “comment” column. With custom functions, we next converted the texts to lowercase, removed punctuations, and removed stop words.

Now, since BERT is a language model that utilizes the structure of the sentence from both directions to connect every output element to every input element, and dynamically adjust weightings depending on this connection (this process is called attention), our hypothesis was that the lighter pre-processing or raw data will do better. As expected, we found out that feeding raw data to the model was giving a better accuracy if BERT vectorization is used. Hence, we decided to drop preprocessing and proceeded with the raw data only since BERT based ML model was eventually giving the highest accuracy.

Feature Engineering

Vectorization

The vectorization methods used include Word2Vec, TF-IDF and BERT (pre-trained, all-mpnet-base-v2). Word2Vec and TF-IDF are popular vectorization methods used for similar projects. For this reason, we wanted to implement each in our experiment. BERT is included because it is a transformer that learns in a parallel manner via its self-attention mechanism and contains the encoder portion of transformers’ encoder-decoder architecture. Descriptions of each can

be found below:

- TF-IDF: Tf-idf stands for *term frequency-inverse document frequency*, and the TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

- BERT: It stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based on their connection. (In NLP, this process is called *attention*.)

- Word2vec: It is not a singular algorithm, rather, it is a family of model architectures and optimizations that can be used to learn word embeddings from large datasets. Embeddings learned through word2vec have proven to be successful on a variety of downstream natural language processing tasks.

Data Modeling

For our modeling experiments, we compared the performance of the traditional machine learning models like KNN, Quadratic Discriminant Analysis (QDA) against the newer state of the art gradient boosting models, including AdaBoost Classifier (ABC), Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Classifier (LGB). We selected KNN as one of the experimenting models since KNN works well with multi-classification and also when the data are not linearly separable. We also tried with QDA since it assumes a quadratic decision boundary and can accurately model a wide range of problems. Regarding the gradient boosting models, we tried with ABC as it is a boosting ensemble model, and our team has seen the model perform well with text data in other projects. Finally, we also tried the state-of-the-art XGBoost and LGB, particularly LGB was tried since it has a lesser training time than other models with improved accuracy.

Furthermore, we also wanted to compare how machine learning algorithms performed with traditional vectorization methods versus transformers. We also ran all our experiments on smaller dataset to reduce the computation time. The combinations of vectorizer/transformers and algorithms are as follows:

- BERT + LGB
- BERT + KNN
- BERT + QDA
- BERT + XGBoost
- BERT + ABC
- TF-IDF + LGB
- Word2Vec + LGB

Model Evaluation and Results

We did not have labeled test data and also, we did not want to lose any training data by further splitting the training data into train and test data sets. Hence to evaluate each models' performances, we compared the accuracies for each model through K fold cross validation. The table below shows the scores for each model and the bar graph displays the accuracies.

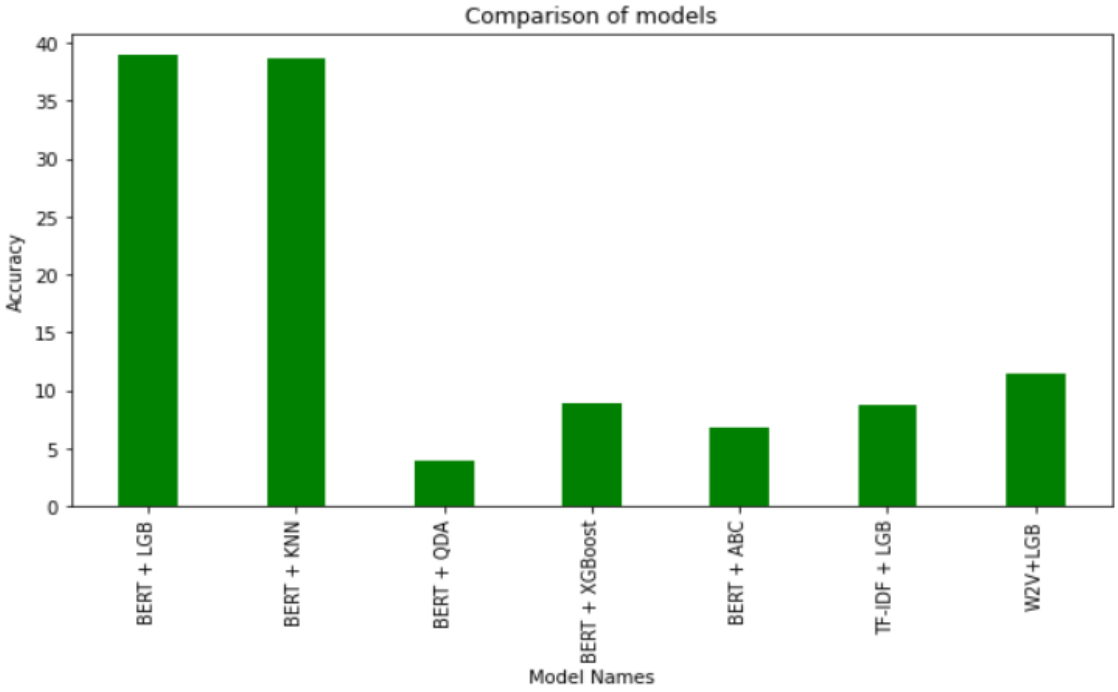


Fig. 8 Comparison of models

Accuracy: Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

accuracy			0.45	100000
macro avg	0.45	0.45	0.45	100000
weighted avg	0.45	0.45	0.45	100000

AUC score -
0.9115810470632557

Conclusion and Challenges

Multi-class prediction has always been a great challenge due to the learning curve of the model, and the computational power required to train with a lot of data. The lack of GPU and RAM disabled our team to effectively measure the accuracy for the whole training dataset based on k fold cross validation.

The model showing the highest performance accuracy was BERT + LGB (39%). This aligns with our expectations that the gradient boosting framework-based models would have better performance. The BERT transformer outperformed all of the traditional vectorizers, allowing the machine learning models to produce scores better with BERT. Hence, we selected BERT + LGB as our final model.
Interestingly when we ultimately got the labeled test data and ran the model (trained on full training data), we found that not only the model is yielding a decent accuracy (45%), also its AUC score is good (91.15%) which testifies the model has an excellent discrimination ability.

In conclusion, our model has the test accuracy of 45%. Similarly, we would like to have more GPU capacity to explore further options to solve this problem.

References:

https://doi.org/10.1016/j.matpr.2021.07.358

https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

https://www.ibm.com/cloud/learn/convolutional-neural-networks

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

https://ieeexplore.ieee.org/document/9658177