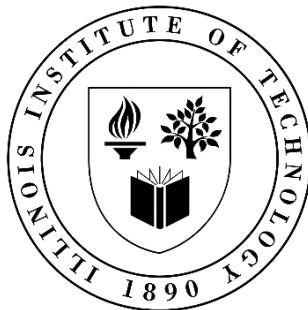


ILLINOIS INSTITUTE OF TECHNOLOGY



Emotion Sentiment Classification of Product Reviews on Twitter

Use of Big Data in Machine Learning Systems

BIG DATA TECHNOLOGIES
(CSP – 554)

Submitted by:

Ashlesh Khajbage	akhajbage@hawk.iit.edu	A20517913
Pradaap Shiva Kumar Shobha	pshivakumarshobha@hawk.iit.edu	A20512400
Shrilakshmi Pai Nallur	spainallur@hawk.iit.edu	A20523709

INTRODUCTION

Machine learning is the process through which an algorithm makes predictions without having that function explicitly coded into it. Data is the machine learning model's primary input. Large dataset management can be accomplished via big data. We can process vast amounts of data and generate predictions by combining Big Data with machine learning. Using techniques known as regression and classification, it may forecast numerical values or categorical values. The input data must be organized and clean before predictions can be made; this procedure is known as data cleaning. Exploratory data analysis can be used to learn more about the dataset. Divide the input dataset into train and test datasets after which you can develop a model and predict the target variable on new or test datasets using the training dataset.

Apache Spark is a data processing platform that, either independently or in combination with other distributed computing technologies, can efficiently analyze big datasets and distribute data processing tasks across several machines. Apache Spark employs Spark MLlib to carry out machine learning. A scalable Machine Learning toolkit built on Spark called MLlib puts equal emphasis on quick performance and high-quality algorithms.

At a high level, it provides tools such as:

- **ML Algorithms:** common learning algorithms such as classification, regression, clustering, and collaborative filtering
- **Featurization:** feature extraction, transformation, dimensionality reduction, and selection
- **Pipelines:** tools for constructing, evaluating, and tuning ML Pipelines
- **Consistency:** saving and loading algorithms, models, and Pipelines
- **Utilities:** linear algebra, statistics, data handling, etc.

MLlib is a program that supports Java, Python, R, and Scala that can analyze big data on Hadoop, Apache Mesos, Kubernetes, standalone, or on the cloud. Java, Python, and R can be used to implement it, and it can run on Hadoop, Apache Mesos, and Kubernetes. Faster than Map Reduce, practically all machine learning techniques are now available with Spark MLlib.

LITERATURE REVIEW

1. Big Data Machine Learning using Apache Spark MLlib

A study paper by Mehdi Assefi, Ehsun Behravesh, Guangchi Liu, and Ahmad P. Tafti was utilized as a reference for this project implementation in Spark-MLlib. To evaluate the ability of the Apache Spark MLlib 2.0 library in analyzing big data sets, the project focused on a set of supervised (classification) methods, including SVM (Support Vector Machine), Decision Tree, Naive Bayes, and Random Forest, along with a widely used unsupervised (clustering) machine learning algorithm, namely K-Means.

A total of Six various big datasets were used to analyze and compare the Apache Spark MLlib 2.0 performance. Five datasets from the UCI Machine Learning Repository, and a dataset from the US government's Bureau of Transportation Research and Innovative Technology Administration (RITA) Web sites.

2. Sentiment Analysis

Machine Learning models take numerical values as input. The reviews are made of sentences, so in order to extract patterns from the data; we need to find a way to represent it in a way that a machine learning algorithm can understand, i.e. as a list of numbers. Emotions are essential, not only in personal life but in business as well. How your customers and target audience feel about your products or brand provides you with the context necessary to evaluate and improve the product, business, marketing, and communications strategy. Sentiment analysis or opinion mining helps researchers and companies extract insights from user-generated social media and web content.

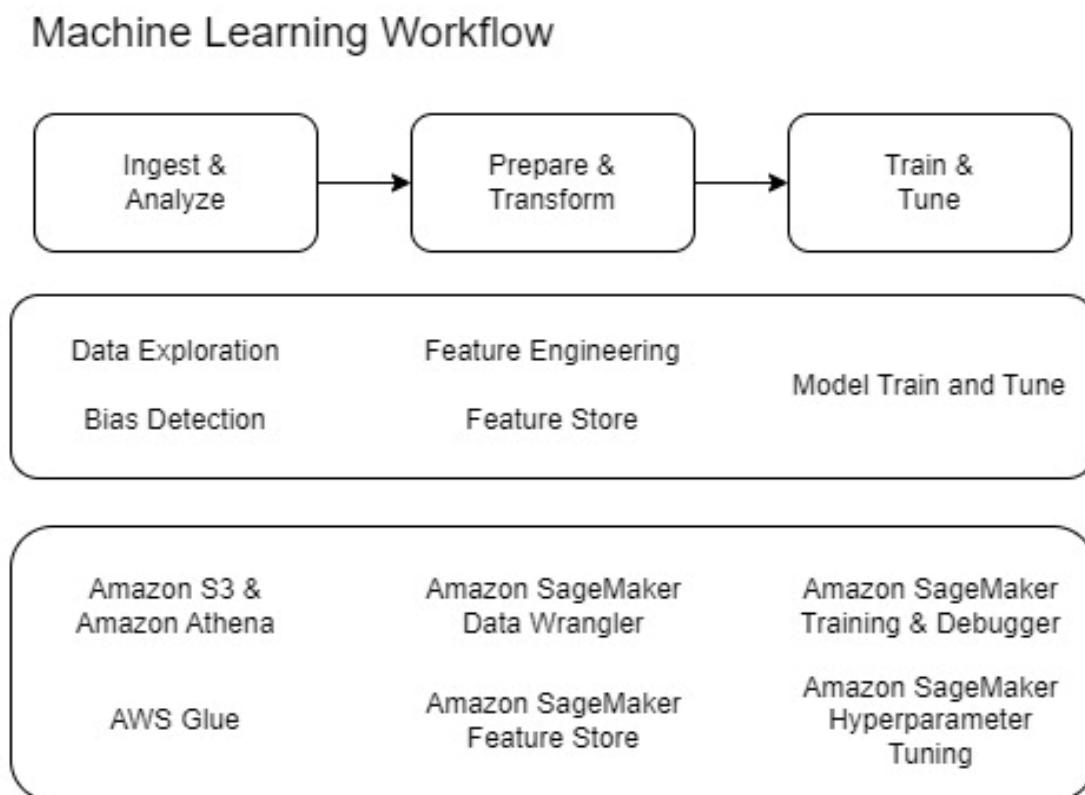
Irrespective of the industry or vertical, brands have become imperative to understand consumers' feelings about the brand and products. With cut-throat competition in the NLP and ML industry for high-paying jobs, a boring cookie-cutter resume might not just be enough. Instead, working on a sentiment analysis project with real datasets will help you stand out in job applications and improve your chances of receiving a call back from your dream company.

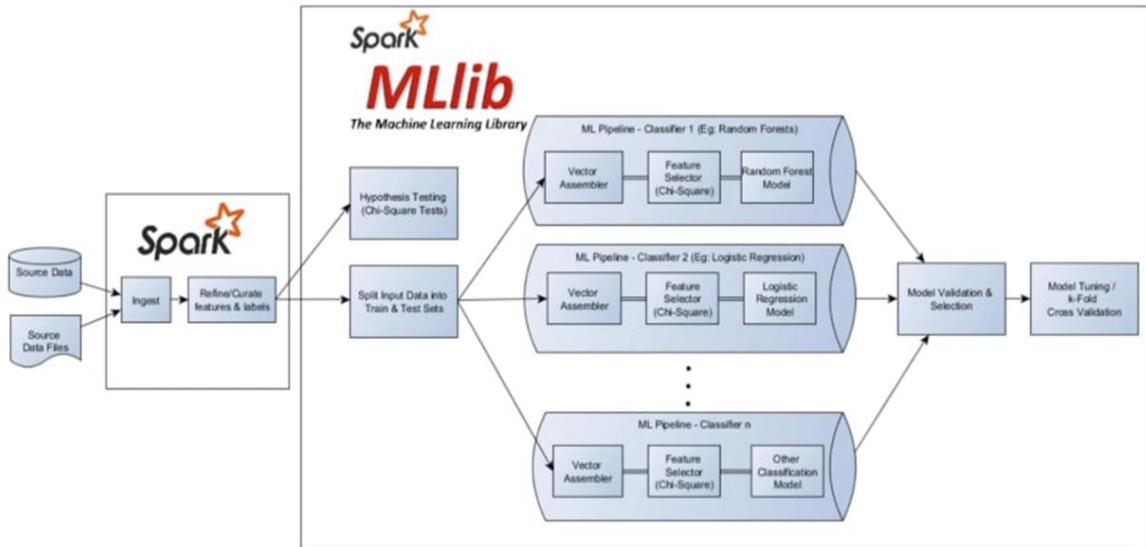
PROBLEM STATEMENT

It is considerably more difficult to discern someone's tone in writing than it is in conversation. For starters, you cannot infer an author's emotional condition from their facial expressions or body language. For instance, the tone of a message is typically not determined by a single characteristic. Numerous aspects are involved in effective communication. Because in some circumstances, what you say can be just as important as how you say it. Our approach is designed to offer feedback on a variety of keywords that represent the emotion of the message. It's also helpful in ensuring that your recipient will focus on the facts you are presenting rather than unintended consequences of your tone by making strategic judgments about how you convey your message.

WORK-FLOW

The steps implemented in this project are explained in the flowchart diagram below.





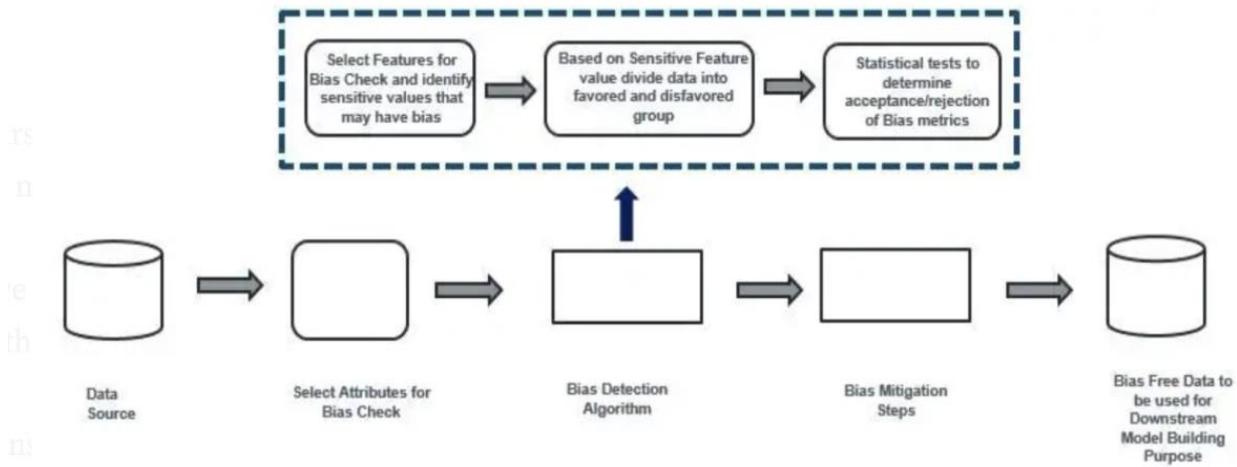
OBJECTIVE

To build a tone detector using Spark-MLibs and Sagemaker that performs better than or upto the state-of-the-art models.

Bias Detection using AWS SageMaker Clarify

To demonstrate the effectiveness of Bias Detection solution we applied AWS SageMaker Clarify for bias detection on two datasets:

- Problem Statement:
 1. Identify bias in datasets
 2. Identify bias in Model Predictions.
- Data Source: //add data source here
- Solution Approach for Bias Detection using SageMaker Clarify: We leverage SageMaker Clarify to identify biases on above datasets. Below is a high-level solution approach for Bias Detection in datasets.



Within the Bias algorithm, the dataset is divided into a favored and disfavoured group.

- i. In the first step we identify those attributes that may cause bias.
- ii. Once we have identified the sensitive attributes, we identify the range of values in these attributes which represent a disfavoured group.
- iii. The sensitive attribute and sensitive value is fed to the Bias Algorithm.
- iv. Within the Bias algorithm, the dataset is divided into a favored and disfavoured group based on the sensitive features and values specified.
- v. Statistical tests are done, and pre-training and post-training metrics are computed for bias detection. Based on interpretation of statistical metrics, we identify the presence/absence of Bias.

- Exploratory Analysis Results: Data Exploration was done on the two datasets before the bias detection process.

AWS Sagemaker

SageMaker

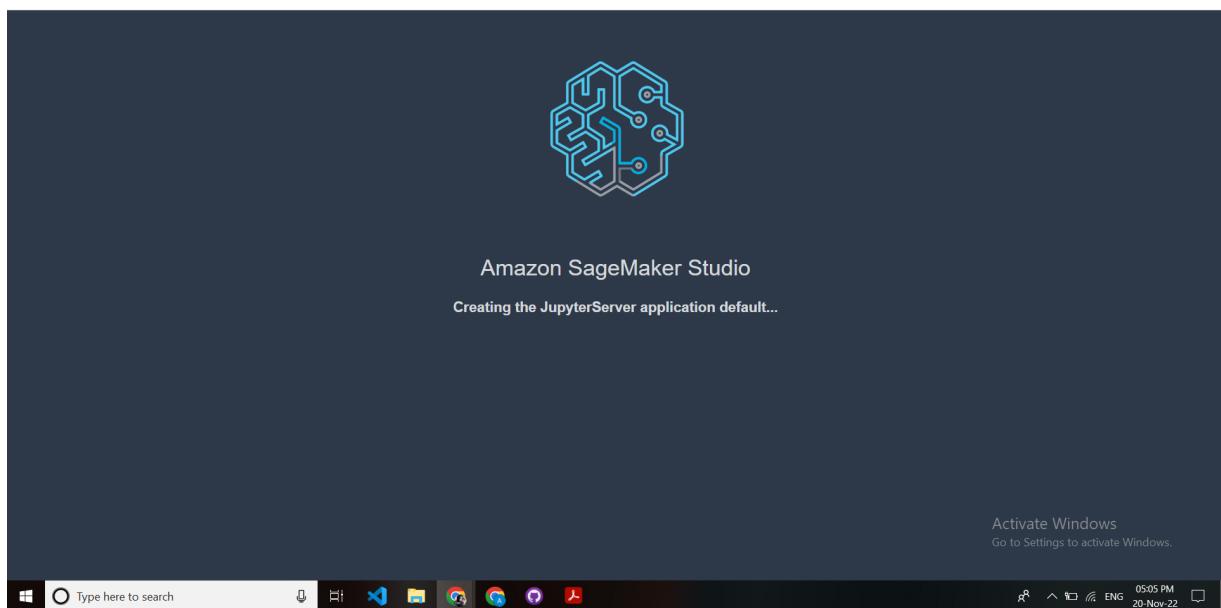
Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. Amazon SageMaker automatic model tuning can use Amazon EC2 Spot instance to optimize costs when running training jobs. Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

The complete machine learning pipeline, including labeling and prepping data, selecting an algorithm, training the model, fine-tuning and optimizing it for deployment, making predictions, and taking action, is covered by the fully-managed service Amazon SageMaker.

We have implemented the model with given dataset using sage maker, steps are represented below:

1. Starting up Amazon Sagemaker Studio



2.Create a notebook Instance

The screenshot shows the 'Create notebook instance' settings page in the AWS SageMaker console. The 'Notebook instance name' is set to 'demo'. The 'Notebook instance type' is 'ml.t3.large'. A note at the bottom of the form states: 'Amazon SageMaker Notebook Instance is ending its standard support on Amazon Linux AMI (AL1). Learn more'.

Notebook instance settings	
Notebook instance name	demo
Notebook instance type	ml.t3.large
Elastic Inference Learn more	none
Amazon SageMaker Notebook Instance is ending its standard support on Amazon Linux AMI (AL1). Learn more	
Platform identifier Learn more	notebook-al1-v1
► Additional configuration	

3. Configuring notebook instance for aws sagemaker

The screenshot shows the 'Permissions and encryption' configuration page. An IAM role 'AmazonSageMaker-ExecutionRole-20220501T133148' is selected. Under 'Root access - optional', the 'Enable' radio button is selected. Under 'Encryption key - optional', 'No Custom Encryption' is chosen. Below the main configuration are three expandable sections: 'Network - optional', 'Git repositories - optional', and 'Tags - optional'.

Permissions and encryption	
IAM role	AmazonSageMaker-ExecutionRole-20220501T133148
Root access - optional	<input checked="" type="radio"/> Enable - Give users root access to the notebook <input type="radio"/> Disable - Don't give users root access to the notebook Lifecycle configurations always have root access
Encryption key - optional	No Custom Encryption
► Network - optional	
► Git repositories - optional	
► Tags - optional	

4. Setting up Studio prerequisites

Control Panel

Configure and manage SageMaker domain, users, and apps.

Users

No users
To add a user, choose Add user and enter a user name.

Apps

Configure apps to manage your ML workflow using SageMaker.

Domain

- Status: Pending
- Authentication method: AWS Identity and Access Management (IAM)
- Domain ID: d-yjscd37hh1md
- Execution role: arn:aws:iam::394 42337255:role/service-role/AmazonSageMaker-ExecutionRole-20221119T23340

5. Setting up User & Cluster configuration

User Details

General details about this user profile.

App name	Status	App type	Created	Action
datascience-1-0-ml-t3-medium-1abf3407f667f989be9d86559395	Ready	KernelGateway	Sun Nov 20 2022 00:12:39 GMT-0600 (Central Standard Time)	<button>Delete app</button>
default	Ready	JupyterServer	Sun Nov 20 2022 00:09:23 GMT-0600 (Central Standard Time)	<button>Action ▾</button>

Details

- Name: csp554
- Execution role: arn:aws:iam::394423337255:role/service-role/AmazonSageMaker-ExecutionRole-20221119T23340
- Status: Ready
- ID: d-81xaynvmfjy
- Created On: Sun Nov 20 2022 00:09:17 GMT-0600 (Central Standard Time)
- Modified On: Sun Nov 20 2022 00:09:19 GMT-0600 (Central Standard Time)

6. Verify and check configuration for notebook instance we created

The screenshot shows the Jupyter Conda interface. At the top, there are tabs: Files, Running, Clusters, SageMaker Examples, and Conda (which is selected). Below the tabs, it says "23 Conda environments". A table lists environments with columns: Action, Name, Default?, and Directory. The environments listed are: root, JupyterSystemEnv (marked as default), R, amazonei_mxnet_p27, amazonei_mxnet_p36, and amazonei_pytorch_latest_p36. Below this, there are two sections: "Available packages" (empty) and "253 installed packages in environment *JupyterSystemEnv*". The installed packages table has columns: Name, Version, Build, and Available. The packages listed are: _libgcc_mutex (0.1, conda_forge), _openmp_mutex (4.5, 1_gnu), alsa-lib (1.2.3, h516909a_0), atk-1.0 (2.36.0, h3371d22_4), attrs (20.3.0, pypi_0), and autovizwidget (0.19.1, pyh6c4a22f_0).

Action	Name	Default?	Directory
root	/home/ec2-user/anaconda3		
JupyterSystemEnv	/home/ec2-user/anaconda3/envs/JupyterSystemEnv	✓	
R	/home/ec2-user/anaconda3/envs/R		
amazonei_mxnet_p27	/home/ec2-user/anaconda3/envs/amazonei_mxnet_p27		
amazonei_mxnet_p36	/home/ec2-user/anaconda3/envs/amazonei_mxnet_p36		
amazonei_pytorch_latest_p36	/home/ec2-user/anaconda3/envs/amazonei_pytorch_latest_p36		

Name	Version	Build	Available
_libgcc_mutex	0.1	conda_forge	
_openmp_mutex	4.5	1_gnu	
alsa-lib	1.2.3	h516909a_0	
atk-1.0	2.36.0	h3371d22_4	
attrs	20.3.0	pypi_0	
autovizwidget	0.19.1	pyh6c4a22f_0	

We also verified and checked configuration for notebook instances we created like the environment, programming language of your notebook, processing configuration (gpu, number of cores, etc.)

Implementation of Machine Learning Model in AWS Sagemaker

Stage-1 : Ingest And Analyze (Data Loading, Exploration & Data Preprocessing)

In this stage, we have cleaned and preprocessed data in order to fit the data into models. we started with following steps

- Installing required Python packages

The screenshot shows the Amazon SageMaker Studio interface. On the left, there's a file browser with a folder named 'Bert Data' containing 'Bert-Code.ipynb'. The main area displays a Jupyter notebook titled 'Bert-Code.ipynb' with the following code and output:

```
# Libraries and packages
!pip install torch
import torch
import pandas as pd
from tqdm.notebook import tqdm

Keyring is skipped due to an exception: 'keyring.backends'

Collecting torch
  Downloading torch-1.13.0-cp37-cp37m-manylinux1_x86_64.whl (890.2 MB)
    890.2/890.2 MB 895.9 kB/s eta 0:00:000:0100:01
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from torch) (4.4.0)
Collecting nvidia-cublas-cu11==11.10.3.66
  Downloading nvidia_cublas_cu11-11.10.3.66-py3-none-manylinux1_x86_64.whl (317.1 MB)
    317.1/317.1 MB 2.6 MB/s eta 0:00:000:0100:01
Collecting nvidia-cuda-runtime-cu11==11.7.99
  Downloading nvidia_cuda_runtime_cu11-11.7.99-py3-none-manylinux1_x86_64.whl (849 kB)
    849.3/849.3 kB 13.6 MB/s eta 0:00:000:01
Collecting nvidia-cuda-nvrtc-cu11==11.7.99
  Downloading nvidia_cuda_nvrtc_cu11-11.7.99-py3-none-manylinux1_x86_64.whl (21.0 kB)
    21.0/21.0 kB 23.7 MB/s eta 0:00:000:0100:01
Collecting nvidia-cudnn-cu11==8.5.0.96
  Downloading nvidia_cudnn_cu11-8.5.0.96-2-py3-none-manylinux1_x86_64.whl (557.1 kB)
    557.1/557.1 kB 1.4 MB/s eta 0:00:000:0100:01
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-packages (from nvidia-cublas-cu11==11.10.3.66->torch) (59.3.0)
Requirement already satisfied: wheel in /opt/conda/lib/python3.7/site-packages (from nvidia-cublas-cu11==11.10.3.66->torch) (0.34.2)
Installing collected packages: nvidia-cuda-runtime-cu11, nvidia-cuda-nvrtc-cu11, nvidia-cublas-cu11, nvidia-cudnn-cu11, torch
Successfully installed nvidia-cublas-cu11-11.10.3.66 nvidia-cuda-nvrtc-cu11-11.7.99 nvidia-cuda-runtime-cu11-11.7.99 nvidia-cudnn-cu11-8.5.0.96 torch-1.13.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

At the bottom of the notebook, there's a preview of the data:

```
# Load data
df = pd.read_csv('Bert_Data/smile-annotations-final.csv', names = ['id', 'text', 'category'])

#reset index
df.set_index('id', inplace = True)

#preview
df.head()
```

The status bar at the bottom right shows: Mode: Command, Ln 3, Col 15, Bert-Code.ipynb, 06:44 PM, 06-Dec-22.

- Then read the given data in your notebook (here we have used CSV format).

The screenshot shows the continuation of the Jupyter notebook from the previous image. The code cell [2] has been run, and the resulting data preview is shown in a table:

		text	category
	id		
[3]:	611857364396965889	@aandraous @britishmuseum @AndrewsAntonio Merc...	nocode
	614484565059596288	Dorian Gray with Rainbow Scarf #LoveWins (from...	happy
	614746522043973632	@SelectShowcase @Tate_StIves ... Replace with ...	happy
	614877582664835073	@SofabSports thank you for following me back. ...	happy
	611932373039644672	@britishmuseum @TudorHistory What a beautiful ...	happy

- after loading the data, we start with data Exploration with understanding on **column type**, checking for **null values**

```
In [9]: #info df.info()

Int64Index: 3085 entries, 611857364396965889 to 611566876762640384
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
 --- 
 0   text      3085 non-null   object  
 1   category  3085 non-null   object  
 dtypes: object(2)
 memory usage: 72.3+ KB

In [10]: #check for null df.isnull().sum()

Out[10]: text      0
category  0
dtype: int64
```

- Following On same Line, we **calculated categorical classes** already mentioned in dataset

```
In [12]: #count for each class df.category.value_counts()

Out[12]: nocode      1572
happy       1137
not-relevant  214
angry        57
surprise     35
sad          32
happy|surprise  11
happy|sad      9
disgust|angry   7
disgust       6
sad|disgust    2
sad|angry      2
sad|disgust|angry 1
Name: category, dtype: int64
```

- Dropping unwanted/undefined classes for category

```
In [13]: #drop irrelevant class df = df[~df.category.str.contains('|\|')]

In [14]: #drop irrelevant class df = df[df.category != 'nocode']

In [15]: #final classes df.category.value_counts()

Out[15]: happy       1137
not-relevant  214
angry        57
surprise     35
sad          32
disgust       6
Name: category, dtype: int64
```

- Visualizing final Prepared Data

```
In [16]:
import matplotlib.pyplot as plt
import seaborn as sns

#plot class distribution
plt.figure(figsize=(10, 5))
sns.countplot(df.category, palette='Spectral')
plt.xlabel('Classes')
plt.title('Class Distribution');

/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be 'data', and passing other arguments without an explicit keyword will result in an error or misinterpretation.
FutureWarning
```

Category	Count
happy	~1050
not-relevant	~200
angry	~50
disgust	~20
sad	~30
surprise	~30

- Finally, Converted labels to numerical values

```
In [18]:
#convert labels into numeric values
label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index

In [19]:
label_dict
```

```
Out[19]: {'happy': 0,
 'not-relevant': 1,
 'angry': 2,
 'disgust': 3,
 'sad': 4,
 'surprise': 5}
```

- Plotting histogram to understand Sentence length

```
In [21]:
#need equal length sentences
#plot hist of sentence length
plt.figure(figsize=(10, 5))
sns.histplot([len(s) for s in df.text], bins=100)
plt.title('Sentence Length')
plt.show()
```

Length Range (approx.)	Count (approx.)
10-20	1-2
20-30	2-3
30-40	3-4
40-50	4-5
50-60	5-6
60-70	6-7
70-80	7-8
80-90	8-9
90-100	10-12
100-110	12-14
110-120	14-16
120-130	16-18
130-140	18-20
140-150	1-2

- Splitting dataset into Training dataset, validation dataset and Test dataset

```
In [23]: from sklearn.model_selection import train_test_split
#train test split
X_train, X_val, y_train, y_val = train_test_split(df.index.values,
                                                df.label.values,
                                                test_size = 0.15,
                                                random_state = 17,
                                                stratify = df.label.values)

[20]: #fill in data type
df.loc[X_train, 'data_type'] = 'train'
df.loc[X_val, 'data_type'] = 'val'

[21]: df.groupby(['category', 'label', 'data_type']).count()
[21]:
   category  label  data_type
      angry    2       train    48
                  val     9
      disgust   3       train    5
                  val     1
      happy     0       train  966
                  val  171
      not-relevant  1       train  182
                  val  32
      sad       4       train   27
                  val     5
      surprise   5       train   30
                  val     5
```

- Now we are ready to develop models and implement them.
- Here we have implemented 3 machine learning models for the classification of the given dataset:
 - 1) BERT NLP model
 - 2) RoBERTa model
 - 3) RNN Model

Analyzed the performance of all three models on amazon sagemaker using 2 parameters accuracy and loss.

Implementing Sentiment Analysis using BERT NLP Model

BERT:

Although BERT(Bidirectional Encoder Representations from Transformers) can be used without any further training, it's helpful to understand how it learns and understands language by simultaneously using word masking and next sentence prediction. The model hides 15% of the words in each sentence while BERT reads fresh text. Then BERT guesses the masked words and self-corrects, updating the model weights when it makes a mistaken prediction. The term "masked language model" or "masked LM" refers to this phase. The model must learn the surrounding words for each sentence as a result of masking. BERT simultaneously predicts and masks words, or to be more accurate, input tokens. Additionally, it employs NSP, or next sentence prediction, on pairs of input sequences.

BERT randomly selects 50% of the sentence pairings and substitutes one of the two sentences with a random sentence from a different section of the document to conduct NSP. The validity of the two sentences as a sentence pair is then predicted by BERT. When it makes a mistaken prediction, BERT will once more correct itself. For the purpose of generating a single accuracy score for all of the training efforts, those two training tasks are carried out simultaneously. As a result, the model becomes more reliable and is able to anticipate words and sentences. Keep in mind that this pre-training phase is carried out using unsupervised learning. Large collections of unlabeled text make up the input data.

Following are steps involved in **Building Sentiment Analysis using the BERT model**.

1. We have already imported packages, explored, cleaned & prepared data. For NLP sentiment analysis, we are required to tokenize the words in sentences. **Tokenization** is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

```
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-packages (from transformers) (21.3)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.7/site-packages (from transformers) (6.0)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from transformers) (2.28.1)
Requirement already satisfied: typing_extensions>=3.6.0 in /opt/conda/lib/python3.7/site-packages (from loggingface-hub<1.0,>=0.1.0->transformers) (4.4.0)
Requirement already satisfied: idna<3.2,>=2.0.2 in /opt/conda/lib/python3.7/site-packages (from packaging>=20.0->transformers) (3.0.9)
Requirement already satisfied: rmpy>=0.5 in /opt/conda/lib/python3.7/site-packages (from importlib-metadata>transformers) (3.6.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests>transformers) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.7/site-packages (from requests>transformers) (1.26.12)
Requirement already satisfied: charset_normalizer<3,>=2 in /opt/conda/lib/python3.7/site-packages (from requests>transformers) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packages (from requests>transformers) (2022.9.24)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv-class-ansi-yellow

[28]: #Load tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
                                         do_lower_case = True)

Downloading: 100% [██████████] 226k/226k [00:00<00:00, 2.10MB/s]
Downloading: 100% [██████████] 28.0/28.0 [00:00<00:00, 953B/s]
Downloading: 100% [██████████] 570/570 [00:00<00:00, 19.0kB/s]

[29]: #tokenize train set
encoded_data_train = tokenizer.batch_encode_plus(df[df.data_type == 'train'].text.values,
                                                add_special_tokens = True,
                                                return_attention_mask = True,
                                                pad_to_max_length = True,
                                                max_length = 150,
                                                return_tensors = 'pt')

Truncation was not explicitly activated but 'max_length' is provided a specific value, please use 'truncation=True' to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GUMF-style) with the tokenizer you can select this strategy more precisely by providing a specific strategy to 'truncation'.
/opt/conda/lib/python3.7/site-packages/tokenization_utils/base.py:2307: FutureWarning: The 'pad_to_max_length' argument is deprecated and will be removed in a future version. Use 'padding=True' or 'pad_to_max_length' instead.
```

2. Result of Tokenization:

```
In [30]: #tokenizer val set
encoded_data_val = tokenizer.batch_encode_plus(df[df.data_type == 'val'].text.values,
                                              add_special_tokens = True,
                                              return_attention_mask = True,
                                              pad_to_max_length = True,
                                              max_length = 156,
                                              return_tensors = 'pt')

In [31]: encoded_data_train

Out[31]: {'input_ids': tensor([[ 101, 16892, 3897, ... , 0, 0, 0],
[ 101, 1030, 27034, ... , 0, 0, 0],
[ 101, 1030, 10682, ... , 0, 0, 0],
... ,
[ 101, 11047, 1030, ... , 0, 0, 0],
[ 101, 1030, 3680, ... , 0, 0, 0],
[ 101, 1030, 2120, ... , 0, 0, 0]]), 'token_type_ids': tensor([[0, 0, 0, ... , 0, 0, 0],
[0, 0, 0, ... , 0, 0, 0],
[0, 0, 0, ... , 0, 0, 0],
[0, 0, 0, ... , 0, 0, 0],
[0, 0, 0, ... , 0, 0, 0],
... ,
[1, 1, 1, ... , 0, 0, 0],
[1, 1, 1, ... , 0, 0, 0],
[1, 1, 1, ... , 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, ... , 0, 0, 0],
[1, 1, 1, ... , 0, 0, 0],
[1, 1, 1, ... , 0, 0, 0]])}
```

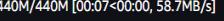
Activ

3. Set Up BERT Model on our Dataset: we are using pre trained **BERT model named- "bert-base-uncased"**, which was built on "BertForMaskedLM" architectures.

Set Up BERT Pretrained Model

```
[*]: from transformers import BertForSequenceClassification

#Load pre-trained BERT
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
                                                       num_labels = len(label_dict),
                                                       output_attentions = False,
                                                       output_hidden_states = False)
```

Downloading: 100%  440M/440M [00:07<00:00, 58.7MB/s]

4. Create Data loaders:

Create Data Loaders

```
In [43]: from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

batch_size = 4 #since we have Limited resource

#load train set
dataloader_train = DataLoader(dataset_train,
                             sampler = RandomSampler(dataset_train),
                             batch_size = batch_size)

#Load val set
dataloader_val = DataLoader(dataset_val,
                           sampler = RandomSampler(dataset_val),
                           batch_size = 32) #since we don't have to do backpropagation for this step
```

5. Set up Optimizer and Scheduler

Set Up Optimizer and Scheduler

```
In [44]:  
from transformers import AdamW, get_linear_schedule_with_warmup  
epochs = 10  
  
#Load optimizer  
optimizer = AdamW(model.parameters(),  
                  lr = 1e-5,  
                  eps = 1e-8) #2e-5 > 5e-5  
  
  
#opt/conda/lib/python3.7/site-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is deprecated and will be removed  
in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set 'no_deprecation_warning=True' to disable this warning  
FutureWarning,  
  
In [45]:  
#Load scheduler  
scheduler = get_linear_schedule_with_warmup(optimizer,  
                                            num_warmup_steps = 0,  
                                            num_training_steps = len(dataloader_train)*epochs)
```

6. Defining Evaluation Parameters

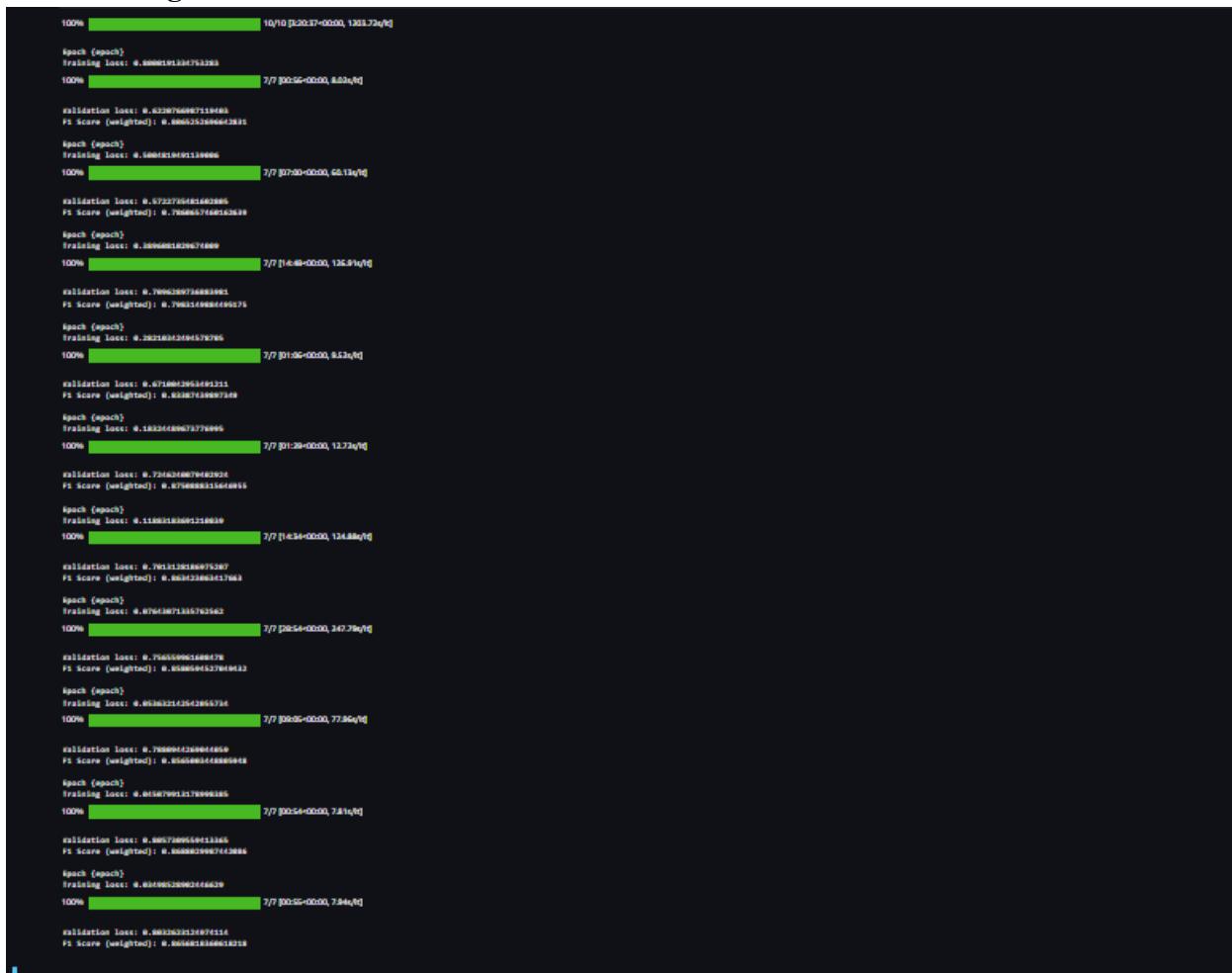
```
In [47]:  
import numpy as np  
from sklearn.metrics import f1_score  
  
#f1 score  
def f1_score_func(preds, labels):  
    preds_flat = np.argmax(preds, axis=1).flatten()  
    labels_flat = labels.flatten()  
    return f1_score(labels_flat, preds_flat, average = 'weighted')  
  
  
In [48]:  
#accuracy score  
def accuracy_per_class(preds, labels):  
    label_dict_inverse = {v: k for k, v in label_dict.items()}  
  
    #make prediction  
    preds_flat = np.argmax(preds, axis=1).flatten()  
    labels_flat = labels.flatten()  
  
    for label in np.unique(labels_flat):  
        y_preds = preds_flat[labels_flat==label]  
        y_true = labels_flat[labels_flat==label]  
        print(f"Class: {label_dict_inverse[label]}")  
        print(f"Accuracy: {(len(y_preds[y_preds==label]))/(len(y_true))}\n")
```

7. Training Model

```
(11):  
for epoch in tqdm(range(1, epochs+1)):  
    start_time = time.time()  
    model.train()  
    training_variable = 0  
    loss_train_total = 0  
  
    dataloader_train = tqdm(dataloader_train,  
                           desc=f'Epoch {epoch}',  
                           leave=False,  
                           disable=False)  
    for batch in progress_bar:  
        for param in model.parameters():  
            param.grad = None  
  
        global_step = 0  
        batch = to_device(batch, device) for b in batch  
  
        #Forward inputs  
        inputs = [batch[i].to(device) for i in range(len(batch))]  
        attention_mask = batch[1],  
        labels = batch[2]  
  
        outputs = model(**inputs)  
        loss = outputs.loss  
        loss_train_total += loss.item()  
  
        #backward pass to get gradients  
        loss.backward()  
  
        #clip the norm of the gradients to 1.0 to prevent exploding gradients  
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)  
  
        #update optimizer  
        optimizer.step()  
  
        #update scheduler  
        scheduler.step()  
  
        progress_bar.set_postfix({'training_loss': f'{(3f).format(loss.item())}/{len(batch)}'})  
    tqdm.write(f'\nEpoch {epoch}')  
  
    #print training results  
    loss_train_avg = loss_train_total/len(dataloader_train)  
    tqdm.write(f"\nTraining loss : {loss_train_avg}")  
  
    #evaluate  
    val_loss, predictions, true_vals = evaluate(dataloader_val)  
    if val_loss is not None:  
        val_f1 = f1_score_func(predictions, true_vals)  
        tqdm.write(f"\nValidation loss : {val_loss}")  
        tqdm.write(f"\nF1 Score (micro) : {val_f1}")  
        tqdm.write(f"\nF1 Score (weighted) : {val_f1}")
```

Activate Windows

8. Model Training results



9. Estimation : Accuracy

```
[53]: #get accuracy score
accuracy_per_class(predictions, true_vals)

Class: happy
Accuracy:164/171

Class: not-relevant
Accuracy:20/32

Class: angry
Accuracy:8/9

Class: disgust
Accuracy:0/1

Class: sad
Accuracy:1/5

Class: surprise
Accuracy:2/5

•[54]: print(f"Accuracy : {Accuracy}")

Accuracy : 0.97
```

10. Comparison of Performance on different Clusters:

After getting results on “**t3-medium**” (3 node) cluster instances, we also executed the above Machine Learning model on “**m5-large**” (5 node) cluster instances.

Observation:

- The Code execution was fast, epochs estimation time were smaller in **m5-large** than **t3-medium**.
- The CPU utilization for **t3-medium** did reach 100% during training, while it did not maxed for **m5-large** instances.
- Hence we can say that, with better CPU configuration the execution speed is improved.

Amazon SageMaker > Domains > Domain: BigDataProj > User Details: default-1670372824520

User Details

General details about this user profile.

Launch ▾

Apps

App name	Status	App type	Created	Action
datascience-1-0-ml-m5-large-ab1c8cf53029fb39f3ca96edc853	Ready	KernelGateway	Tue Dec 06 2022 18:52:49 GMT-0600 (Central Standard Time)	Delete app
datascience-1-0-ml-t3-medium-1abf3407f667f989be9d86559395	Ready	KernelGateway	Tue Dec 06 2022 18:40:33 GMT-0600 (Central Standard Time)	Delete app
default	Ready	JupyterServer	Tue Dec 06 2022 18:36:45 GMT-0600 (Central Standard Time)	Action ▾

Details

Name: default-1670372824520

Execution role: arn:aws:iam::394423337255:role/service-role/AmazonSageMaker-ExecutionRole-20221206T182617

Created On: Tue Dec 06 2022 18:32:18 GMT-0600 (Central Standard Time)

Status: InService

ID: d-yronwqstatepb

Modified On: Tue Dec 06 2022 18:32:21 GMT-0600 (Central Standard Time)

Activate Windows

Cancel

Edit

Feedback Looking for language selection? Find it in the new Unified Settings

Type here to search

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

10:25 PM 06-Dec-22 ENG

Implementing Sentiment Analysis using RoBERTa Model

RoBERTa: RoBERTa(Robustly Optimized BERT pre-training Approach) expands on the BERT model architecture while changing some model hyperparameters and the model's training process, including the use of more training data. In contrast to BERT, this can actually result in notable performance gains in a number of NLP tasks.

1. Sagemaker Domain-Cluster Setup for RoBERTa Model

The screenshot shows the 'User Details' page in the Amazon SageMaker console. The left sidebar includes links for Getting started, Control panel, SageMaker dashboard, and various notebooks and processing types. The main content area displays a table of 'Apps' with columns for App name, Status, App type, Created, and Action. There are four entries: 'datascience-1-0-ml-m5-xlarge-372beb14237fd194a4e4624f9d19' (KernelGateway, Ready, Sun Nov 20 2022 18:25:05 GMT-0600), 'datascience-1-0-ml-m5-large-ab1cfd5029fb39f3ca96edc853' (KernelGateway, Ready, Sun Nov 20 2022 18:22:38 GMT-0600), 'datascience-1-0-ml-t3-medium-1abf3407f667f989be9d86559395' (KernelGateway, Ready, Sun Nov 20 2022 17:07:20 GMT-0600), and 'default' (JupyterServer, Ready, Sun Nov 20 2022 17:04:36 GMT-0600). A 'Details' sidebar on the right provides specific information for the 'default' app, including Name (csp554bigdatabert), Execution role (arnawsiam:394423337255:role/service-role/AmazonSageMaker-ExecutionRole-20221120T165308), Status (Ready), ID (d-yjscd37hh1md), Created On (Sun Nov 20 2022 16:58:43 GMT-0600), and Modified On (Sun Nov 20 2022 16:58:45 GMT-0600). A 'Launch app' button is located at the top right of the main content area.

2. Installing required Packages for RoBERTa Implementation.

The screenshot shows a Jupyter Notebook cell with the title 'Step-1 Install packages'. The code input (In [2]) is:

```
In [2]:  
!pip3 uninstall keras-nightly  
!pip3 uninstall -y tensorflow  
!pip3 install keras==2.1.6  
!pip3 install tensorflow==2.7.0  
!pip3 install h5py==2.10.0
```

The output shows the results of the pip commands:

```
Keyring is skipped due to an exception: 'keyring.backends'  
WARNING: Skipping keras-nightly as it is not installed.  
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv  
Keyring is skipped due to an exception: 'keyring.backends'  
WARNING: Skipping tensorflow as it is not installed.  
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv  
Keyring is skipped due to an exception: 'keyring.backends'  
Collecting keras==2.1.6  
  Using cached Keras-2.1.6-py2.py3-none-any.whl (339 kB)  
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (2.10.0)
```

3. Load dataSet to Jupyter file using readCSV

```
In [8]:  
train_data = pd.read_csv('emotions-dataset-for-nlp/train.txt', names=['text', 'emotion'], sep=';')  
val_data = pd.read_csv('emotions-dataset-for-nlp/val.txt', names=['text', 'emotion'], sep=';')  
test_data = pd.read_csv('emotions-dataset-for-nlp/test.txt', names=['text', 'emotion'], sep=';')  
train_data.head()  
  
Out[8]:  
      text emotion  
0 i didnt feel humiliat... sadness  
1 i can go from feeling so hopeless to so damned... sadness  
2 im grabbing a minute to post i feel greedy wrong anger  
3 i am ever feeling nostalgic about the firepla... love  
4 i am feeling grouchy anger
```

4. Data Pre-Processing

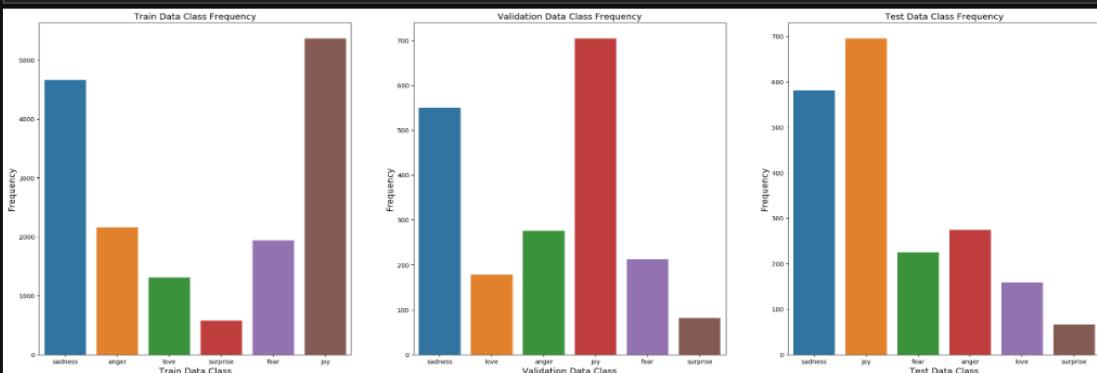
Data preprocessing

```
In [10]:  
data = {'Train Data': train_data, 'Validation Data': val_data, 'Test Data': test_data}  
for temp in data:  
    print(temp)  
    print(data[temp].isnull().sum())  
    print('*'*20)  
  
Train Data  
text      0  
emotion   0  
dtype: int64  
*****  
Validation Data  
text      0  
emotion   0  
dtype: int64  
*****  
Test Data  
text      0  
emotion   0  
dtype: int64  
*****
```

5. Class Distribution for Training, validation and Testing dataset

Class Distribution in Train, val and test dataset

```
In [11]:  
bar, ax = plt.subplots(1,3, figsize=(30, 10))  
for index, temp in enumerate(data):  
    sns.countplot(ax = ax[index],x = 'emotion', data = data[temp])  
    ax[index].set_title(temp+' Class Frequency', size=14)  
    ax[index].set_ylabel('Frequency', size=14)  
    ax[index].set_xlabel(temp+' Class', size=14)  
  
plt.show()
```



6. Data Processing, Label Creation and Sentence Listing

Data Processing, Label Creation and Seentence Listing

In [14]:

```
def preprocess(sentence):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    sentence = re.sub('[^A-z]', ' ', sentence)
    negative = ['not', 'neither', 'nor', 'but', 'however', 'although', 'nonetheless', 'despite', 'except',
    'even though', 'yet']
    stop_words = [z for z in stop_words if z not in negative]
    preprocessed_tokens = [lemmatizer.lemmatize(contractions.fix(temp.lower())) for temp in sentence.split() if temp not in stop_words] #lemmatization
    return ' '.join([x for x in preprocessed_tokens]).strip()
```

In [16]:

```
train_data['text'] = train_data['text'].apply(lambda x: preprocess(x))
val_data['text'] = val_data['text'].apply(lambda x: preprocess(x))
test_data['text'] = test_data['text'].apply(lambda x: preprocess(x))
```

Note: As class imbalance is evident, RandomOverSampler is used to add data(repetition) to all classes except highest frequency class.

In [17]:

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
train_x, train_y = ros.fit_resample(np.array(train_data['text']).reshape(-1, 1), np.array(train_data['emotion']).reshape(-1, 1))
train = pd.DataFrame(list(zip(x[0] for x in train_x), train_y)), columns = ['text', 'emotion'])
```

Applying OneHotEncoder on target of all dataset

In [18]:

```
from sklearn import preprocessing
le = preprocessing.OneHotEncoder()
y_train= le.fit_transform(np.array(train['emotion']).reshape(-1, 1)).toarray()
y_test= le.fit_transform(np.array(test_data['emotion']).reshape(-1, 1)).toarray()
y_val= le.fit_transform(np.array(val_data['emotion']).reshape(-1, 1)).toarray()
```

7. Tokenizing Text for Encoding

In [22]:

```
def roberta_encode(data,maximum_length) :
    input_ids = []
    attention_masks = []

    for i in range(len(data.text)):
        encoded = tokenizer.encode_plus(
            data.text[i],
            add_special_tokens=True,
            max_length=maximum_length,
            pad_to_max_length=True,
            return_attention_mask=True,
        )
        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])
    return np.array(input_ids),np.array(attention_masks)
```

In [23]:

```
max_len = max([len(x.split()) for x in train_data['text']])
train_input_ids,train_attention_masks = roberta_encode(train, max_len)
test_input_ids,test_attention_masks = roberta_encode(test_data, max_len)
val_input_ids,val_attention_masks = roberta_encode(val_data, max_len)
```

8. Creating Model:

```
In [24]:
def create_model(bert_model, max_len):
    input_ids = tf.keras.Input(shape=(max_len,), dtype='int32')
    attention_masks = tf.keras.Input(shape=(max_len,), dtype='int32')

    output = bert_model([input_ids, attention_masks])
    output = output[1]

    output = tf.keras.layers.Dense(6, activation='softmax')(output)
    model = tf.keras.models.Model(inputs=[input_ids, attention_masks], outputs=output)
    model.compile(Adam(lr=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

9. RobertA Model Description

The screenshot shows the Amazon SageMaker Studio interface. On the left, there's a file browser with a search bar. The main area displays the model architecture for 'Classify Emotions in text with BERT'. It shows two input layers (input_1 and input_2) and a single layer (tf_roberta_model). The tf_roberta_model layer has parameters like TFBModelOutputWithPoolingAndCrossAttention, hidden_size=768, and pooler_output=True. A note at the bottom of the code cell says: 'To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert Model: "model"'.

10. RoBERTa Model Training

Model Training

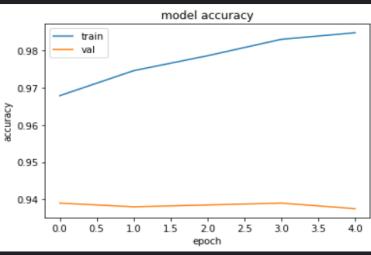
```
[29]:
history = model.fit([train_input_ids, train_attention_masks], y_train, validation_data=([val_input_ids, val_attention_masks], y_val), epochs=5, batch_size=100)

Epoch 1/5
322/322 [=====] - 154s 479ms/step - loss: 0.0795 - accuracy: 0.9679 - val_loss: 0.1593 - val_accuracy: 0.9390
Epoch 2/5
322/322 [=====] - 154s 479ms/step - loss: 0.0644 - accuracy: 0.9746 - val_loss: 0.1807 - val_accuracy: 0.9380
Epoch 3/5
322/322 [=====] - 154s 478ms/step - loss: 0.0552 - accuracy: 0.9786 - val_loss: 0.1616 - val_accuracy: 0.9385
Epoch 4/5
322/322 [=====] - 154s 479ms/step - loss: 0.0454 - accuracy: 0.9830 - val_loss: 0.1739 - val_accuracy: 0.9390
Epoch 5/5
322/322 [=====] - 154s 479ms/step - loss: 0.0403 - accuracy: 0.9848 - val_loss: 0.1906 - val_accuracy: 0.9375
```

Plotting Accuracy and Loss (Training and Validation)

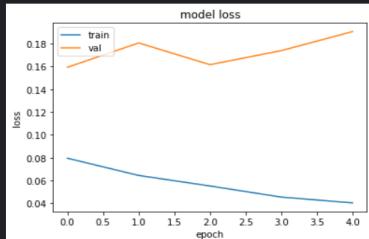
11. Plotting Model Accuracy of Training and Validation dataset

```
[38]:  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train', 'val'], loc='upper left')  
plt.show()
```



12. Plotting Model Loss of Training and validation dataset

```
[39]:  
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'val'], loc='upper left')  
plt.show()
```

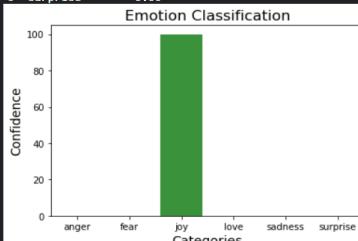


13. Test results on Different text example (Note: Text of inference is different)

```
[39]:  
result = inference("I am happy", max_len)  
print(result)
```

/opt/conda/lib/python3.7/site-packages/transformers/tokenizer.py:100: FutureWarning:
`padding=True` or `padding='longest'` to pad to the longer
`max_length` (e.g. `max_length=45`) or leave `max_length` to None

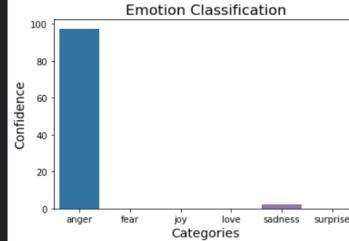
Category	Confidence
anger	0.92
fear	0.00
joy	99.97
love	0.00
sadness	0.01
surprise	0.00



```
[40]:  
result = inference("I am unhappy", max_len)  
print(result)
```

/opt/conda/lib/python3.7/site-packages/transformers/tokenizer.py:100: FutureWarning:
`padding=True` or `padding='longest'` to pad to the longer
`max_length` (e.g. `max_length=45`) or leave `max_length` to None

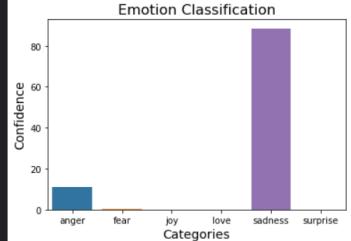
Category	Confidence
anger	97.39
fear	0.01
joy	0.14
love	0.00
sadness	2.45
surprise	0.00



```
[41]:  
result = inference("I am depressed", max_len)  
print(result)
```

/opt/conda/lib/python3.7/site-packages/transformers/tokenizer.py:100: FutureWarning:
`padding=True` or `padding='longest'` to pad to the longer
`max_length` (e.g. `max_length=45`) or leave `max_length` to None

Category	Confidence
anger	11.15
fear	0.19
joy	0.09
love	0.00
sadness	88.56
surprise	0.01



14. Capturing CPU & GPU utilization

Model Training

```
history = model.fit([train_input_ids,train_attention_masks], y_train, validation_data=[(val_input_ids,val_attention_masks), y_val], epochs=5)
```

Epoch 1/5 322/322 [=====] - 154s 479ms/step - loss: 0.0795 - accuracy: 0.9679 - val_loss: 0.1593 - val_accuracy: 0.9390 Epoch 2/5 322/322 [=====] - 154s 479ms/step - loss: 0.0644 - accuracy: 0.9746 - val_loss: 0.1807 - val_accuracy: 0.9380 Epoch 3/5 322/322 [=====] - 154s 478ms/step - loss: 0.0552 - accuracy: 0.9786 - val_loss: 0.1616 - val_accuracy: 0.9385 Epoch 4/5 35/322 [==>.....] - ETA: 2:15 - loss: 0.0424 - accuracy: 0.9840	CPU CPU 80.00% RAM 9.3 GB Max 13GB	GPU GPU 100.00% GPU Memory 15.5 GB Max 15.9GB
--	--	---

15. Comparison of Performance on different Clusters:

After getting results on “**t3-medium**” cluster instances (3 node), we also executed the above Machine Learning model on “**m5-large**” (3 node) and “**m5-xlarge**” (5 node) cluster instances.

Observation:

- The Code execution was fast, epochs estimation time were smaller in **m5-large** than **t3-medium**. it was even quicker in the “**m5-xlarge**”cluster.
- The CPU utilization for **t3-medium** did reach 100%(above snap) during training, while it reached 88% for **m5-large** and **m5-xlarge** instance.
- Hence we can again say that, with better CPU configuration the execution speed is improved.

Model Training

```
history = model.fit([train_input_ids,train_attention_masks], y_train, validation_data=[(val_input_ids,val_attention_masks), y_val], epochs=5)
```

Epoch 1/5 322/322 [=====] - 154s 479ms/step - loss: 0.0795 - accuracy: 0.9679 - val_loss: 0.1593 - val_accuracy: 0.9390 Epoch 2/5 177/322 [=====>.....] - ETA: 1:08 - loss: 0.0634 - accuracy: 0.9766	CPU CPU 81.00% RAM 9.3 GB Max 13GB	GPU GPU 88.00% GPU Memory 15.5 GB Max 15.9GB
--	--	--

Plotting Accuracy and Loss (Training and Validation)

Implementing Sentiment Analysis using RNN Model

The most fundamental and potent neural networks are recurrent neural networks (RNN). These algorithms have gained a lot of popularity since they have produced promising outcomes for several technologies. RNN's main goal is to process sequential data effectively. The idea of internal memory sets RNN apart from conventional neural networks.

Recurrent neural networks are somewhat more established than other types of neural networks, having existed since the 1980s, although recently gaining popularity. With the advancement of technology, RNN has taken the lead because we now have greater computing power and a lot of data that has been collected recently.

for the implementation of this, we wrote a class called ‘Emotions’ comprised of following steps:

1. Installing Required packages.

Step-1 Install packages

```
[25]: !pip3 uninstall keras-nightly
!pip3 uninstall -y tensorflow
!pip3 install keras==2.1.6
!pip3 install tensorflow==1.15.0
!pip3 install h5py==2.10.0

Keyring is skipped due to an exception: 'keyring.backends'
WARNING: Skipping keras-nightly as it is not installed.
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Keyring is skipped due to an exception: 'keyring.backends'
Found existing installation: tensorflow 1.15.0
Uninstalling tensorflow-1.15.0:
   Successfully uninstalled tensorflow-1.15.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Keyring is skipped due to an exception: 'keyring.backends'
Collecting keras==2.1.6
  Downloading Keras-2.1.6-py2.py3-none-any.whl (339 kB)
           130.6/130.6 kB 5.6 MB/s eta 0:00:00:00:01

Requirement already satisfied: pyyaml in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (6.0)
Requirement already satisfied: numpy<1.9.1 in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (1.21.6)
Requirement already satisfied: scipy<0.14 in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (1.4.1)
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (3.7.0)
```

2. Loading Dataset into Notebook

```
def __init__(self, datasetFolder, batch_size, validation_split, optimizer, loss, epochs):
    self.datasetFolder = datasetFolder
    self.batch_size = batch_size
    self.validation_split = validation_split
    self.optimizer = optimizer
    self.loss = loss
    self.epochs = epochs
def readDatasetCSV(self):
    trainDataset = pd.read_csv(os.path.join(self.datasetFolder, "train.txt"), names=['Text', 'Emotion'], sep=';')
    testDataset = pd.read_csv(os.path.join(self.datasetFolder, "test.txt"), names=['Text', 'Emotion'], sep=';')
    validDataset = pd.read_csv(os.path.join(self.datasetFolder, "val.txt"), names=['Text', 'Emotion'], sep=';')
    list_dataset = [trainDataset, testDataset, validDataset]
    self.dataset = pd.concat(list_dataset)
```

2. Feature Labeling & splitting dataset

```
def FeaturesLabels(self):
    self.features = self.dataset['Text']
    self.labels = self.dataset['Emotion']
def splitDataset(self):
    self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(self.features,
                                                                      self.labels,
                                                                      test_size = self.validation_split)
```

3. Cleaning Dataset feature

- in our dataset we had Capital letters, camel-cased Alphabets, for sentiment analysis we figured out we don't need these, hence converted data into lowercase.
- remove unwanted Spaces, punctuation etc. hence finishing off with the Data preparation stage.

```
def CleanFeatures(self):
    self.features = self.features.apply(lambda sequence:
                                         [ltrs.lower() for ltrs in sequence if ltrs not in string.punctuation])
    self.features = self.features.apply(lambda wrd: ''.join(wrd))
```

4. Tokenizing textual part of data and developing Word Embedding using GloVe

- **GloVe** : GloVe model is an [unsupervised learning](#) algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.[\[7\]](#) Training is performed on aggregated global word-word **co-occurrence statistics** from a corpus, and the resulting representations showcase interesting linear substructures of the **word vector space**.

```

def tokenizerDataset(self):
    self.tokenizer = Tokenizer(num_words=5000)
    self.tokenizer.fit_on_texts(self.features)
    train = self.tokenizer.texts_to_sequences(self.features)
    self.features = pad_sequences(train)
    le = LabelEncoder()
    self.labels = le.fit_transform(self.labels)
    self.vocabulary = len(self.tokenizer.word_index)
def label_categorical(self):
    self.labels = to_categorical(self.labels, 6)
def glove_word_embedding(self, file_name):
    self.embeddings_index = {}
    file_ = open(file_name)
    for line in file_:
        arr = line.split()
        single_word = arr[0]
        w = np.asarray(arr[1:], dtype='float32')
        self.embeddings_index[single_word] = w
    file_.close()
    max_words = self.vocabulary + 1
    word_index = self.tokenizer.word_index
    self.embedding_matrix = np.zeros((max_words, 300)).astype(object)
    for word , i in word_index.items():
        embedding_vector = self.embeddings_index.get(word)
        if embedding_vector is not None:
            self.embedding_matrix[i] = embedding_vector
    file_.close()

```

5. Defining function with activation function, Dropouts, Regularizers, metrics & fit function.

- here we have defined the function related dropouts, regularizers, function to compile and fit the model.
- We have called this function to compile and fit the model on training data

```

def model(self):
    m = Sequential()
    m.add(Input(shape=(self.features.shape[1], )))
    m.add(Embedding(self.vocabulary + 1, 300))
    m.add(GRU(128, recurrent_dropout=0.3, return_sequences=False, activity_regularizer = tf.keras.regularizers.L2(0.0001)))
    m.add(Dense(6, activation="softmax", activity_regularizer = tf.keras.regularizers.L2(0.0001)))
    self.m = m
def compiler(self):
    self.m.compile(loss= self.loss,optimizer=self.optimizer,metrics=['accuracy'])
def fit(self):
    earlyStopping = EarlyStopping(monitor = 'loss', patience = 20, mode = 'min', restore_best_weights = True)
    self.history_training = self.m.fit(self.X_train, self.Y_train, epochs= self.epochs,batch_size = self.batch_size,
                                      callbacks=[ earlyStopping])

```

6. Complete picture of Class Emotions

- After defining all the above functions, we decided to put everything together into a class called “**Emotion**”, whose object will be created to access all this functions.

```
In [10]:
class Emotion:
    def __init__(self, datasetFolder, batch_size, validation_split, optimizer, loss, epochs):
        self.datasetFolder = datasetFolder
        self.batch_size = batch_size
        self.validation_split = validation_split
        self.optimizer = optimizer
        self.loss = loss
        self.epochs = epochs
    def readDatasetCSV(self):
        trainDataset = pd.read_csv(os.path.join(self.datasetFolder, "train.txt"), names=['Text', 'Emotion'], sep=';')
        testDataset = pd.read_csv(os.path.join(self.datasetFolder, "test.txt"), names=['Text', 'Emotion'], sep=';')
        validDataset = pd.read_csv(os.path.join(self.datasetFolder, "val.txt"), names=['Text', 'Emotion'], sep=';')
        list_dataset = [trainDataset, testDataset, validDataset]
        self.dataset = pd.concat(list_dataset)
    def FeaturesLabels(self):
        self.features = self.dataset['Text']
        self.labels = self.dataset['Emotion']
    def splitDataset(self):
        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(self.features,
                                                                           self.labels,
                                                                           test_size = self.validation_split)
    def CleanFeatures(self):
        self.features = self.features.apply(lambda sequence:
                                             [ltrs.lower() for ltrs in sequence if ltrs not in string.punctuation])
        self.features = self.features.apply(lambda wrd: ''.join(wrd))
    def tokenizerDataset(self):
        self.tokenizer = Tokenizer(num_words=300)
        self.tokenizer.fit_on_texts(self.features)
        train = self.tokenizer.texts_to_sequences(self.features)
        self.features = pad_sequences(train)
        le = LabelEncoder()
        self.labels = le.fit_transform(self.labels)
        self.vocabulary = len(self.tokenizer.word_index)
    def label_categorical(self):
        self.labels = to_categorical(self.labels, 6)
    def glove_word_embedding(self, file_name):
        self.embeddings_index = {}
        file_ = open(file_name)
        for line in file_:
            arr = line.split()
            single_word = arr[0]
            w = np.asarray(arr[1:], dtype='float32')
            self.embeddings_index[single_word] = w
        file_.close()
        max_words = self.vocabulary + 1
        word_index = self.tokenizer.word_index
        self.embedding_matrix = np.zeros((max_words, 300)).astype(object)
        for word , i in word_index.items():
            embedding_vector = self.embeddings_index.get(word)
            if embedding_vector is not None:
                self.embedding_matrix[i] = embedding_vector
    def model(self):
        m = Sequential()
        m.add(Input(shape=(self.features.shape[1], )))
        m.add(Embedding(self.vocabulary + 1, 300))
        m.add(GRU(128, recurrent_dropout=0.3, return_sequences=False, activity_regularizer = tf.keras.regularizers.L2(0.0001)))
        m.add(Dense(6, activation="softmax", activity_regularizer = tf.keras.regularizers.L2(0.0001)))
        self.m = m
    def compiler(self):
        self.m.compile(loss= self.loss, optimizer=self.optimizer, metrics=[ 'accuracy'])
    def fit(self):
        earlyStopping = EarlyStopping(monitor = 'loss', patience = 20, mode = 'min', restore_best_weights = True)
        self.history_training = self.m.fit(self.X_train, self.Y_train, epochs= self.epochs,batch_size = self.batch_size,
                                         callbacks=[ earlyStopping])
```

7. Result of DataSet Reading, getting glimpse of Data using data.head()

```
In [25]:
emotion = Emotion(dataset_emotion, 256, 0.1, 'adam', 'categorical_crossentropy', 20)
emotion.readDatasetCSV()
emotion.dataset.head()
```

	Text	Emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

8. Result of Data Cleaning, feature labeling (step of Data Preparation)

```
In [26]: emotion.FeaturesLabels()
emotion.Cleanfeatures()
emotion.features.head()

Out[26]: 0           i didnt feel humiliated
1   i can go from feeling so hopeless to so damned...
2   im grabbing a minute to post i feel greedy wrong
3   i am ever feeling nostalgic about the fireplac...
4           i am feeling grouchy
Name: Text, dtype: object
```

9. Result of TokenizerDataSet function()

```
In [28]: emotion.tokenizerDataset()
emotion.features

Out[28]: array([[ 0,  0,  0, ..., 138,  2, 625],
   [ 0,  0,  0, ...,  3, 21, 1383],
   [ 0,  0,  0, ...,  2, 495, 420],
   ...,
   [ 0,  0,  0, ...,  5, 215, 191],
   [ 0,  0,  0, ..., 30,  57, 2181],
   [ 0,  0,  0, ..., 75,  5, 70]], dtype=int32)
```

10. Summary of RNN neural network

```
In [35]: emotion.compiler()
emotion.m.summary()

Model: "sequential_2"
-----  

Layer (type)      Output Shape       Param #
-----  

embedding_2 (Embedding)    (None, 63, 300)      5129100  

gru_2 (GRU)          (None, 128)        165120  

dense_2 (Dense)       (None, 6)         774  

-----  

Total params: 5,294,994
Trainable params: 165,894
Non-trainable params: 5,129,100
```

11. Training RNN Model

```
In [36]: emotion.fit()

Epoch 1/20
71/71 [=====] - 39s 519ms/step - loss: 1.3828 - accuracy: 0.4860
Epoch 2/20
71/71 [=====] - 35s 499ms/step - loss: 0.8182 - accuracy: 0.7173
Epoch 3/20
71/71 [=====] - 38s 542ms/step - loss: 0.4484 - accuracy: 0.8485
Epoch 4/20
71/71 [=====] - 38s 539ms/step - loss: 0.2896 - accuracy: 0.8990
Epoch 5/20
71/71 [=====] - 38s 532ms/step - loss: 0.2210 - accuracy: 0.9172
Epoch 6/20
71/71 [=====] - 38s 531ms/step - loss: 0.1796 - accuracy: 0.9279
Epoch 7/20
71/71 [=====] - 36s 512ms/step - loss: 0.1543 - accuracy: 0.9361
Epoch 8/20
71/71 [=====] - 35s 497ms/step - loss: 0.1355 - accuracy: 0.9435
Epoch 9/20
71/71 [=====] - 36s 504ms/step - loss: 0.1205 - accuracy: 0.9496
Epoch 10/20
71/71 [=====] - 35s 495ms/step - loss: 0.1127 - accuracy: 0.9524
Epoch 11/20
71/71 [=====] - 35s 495ms/step - loss: 0.1017 - accuracy: 0.9588
Epoch 12/20
71/71 [=====] - 35s 497ms/step - loss: 0.0951 - accuracy: 0.9601
Epoch 13/20
71/71 [=====] - 35s 499ms/step - loss: 0.0885 - accuracy: 0.9634
Epoch 14/20
71/71 [=====] - 35s 498ms/step - loss: 0.0830 - accuracy: 0.9666
Epoch 15/20
71/71 [=====] - 35s 499ms/step - loss: 0.0746 - accuracy: 0.9698
Epoch 16/20
71/71 [=====] - 36s 500ms/step - loss: 0.0752 - accuracy: 0.9694
Epoch 17/20
71/71 [=====] - 36s 505ms/step - loss: 0.0660 - accuracy: 0.9753
Epoch 18/20
71/71 [=====] - 36s 500ms/step - loss: 0.0626 - accuracy: 0.9754
Epoch 19/20
71/71 [=====] - 35s 498ms/step - loss: 0.0587 - accuracy: 0.9783
Epoch 20/20
71/71 [=====] - 35s 496ms/step - loss: 0.0577 - accuracy: 0.9779
```

12. Evaluation of Model:

a) Accuracy during Training



b) Plotting Loss Function During Training



c) Confusion Matrix

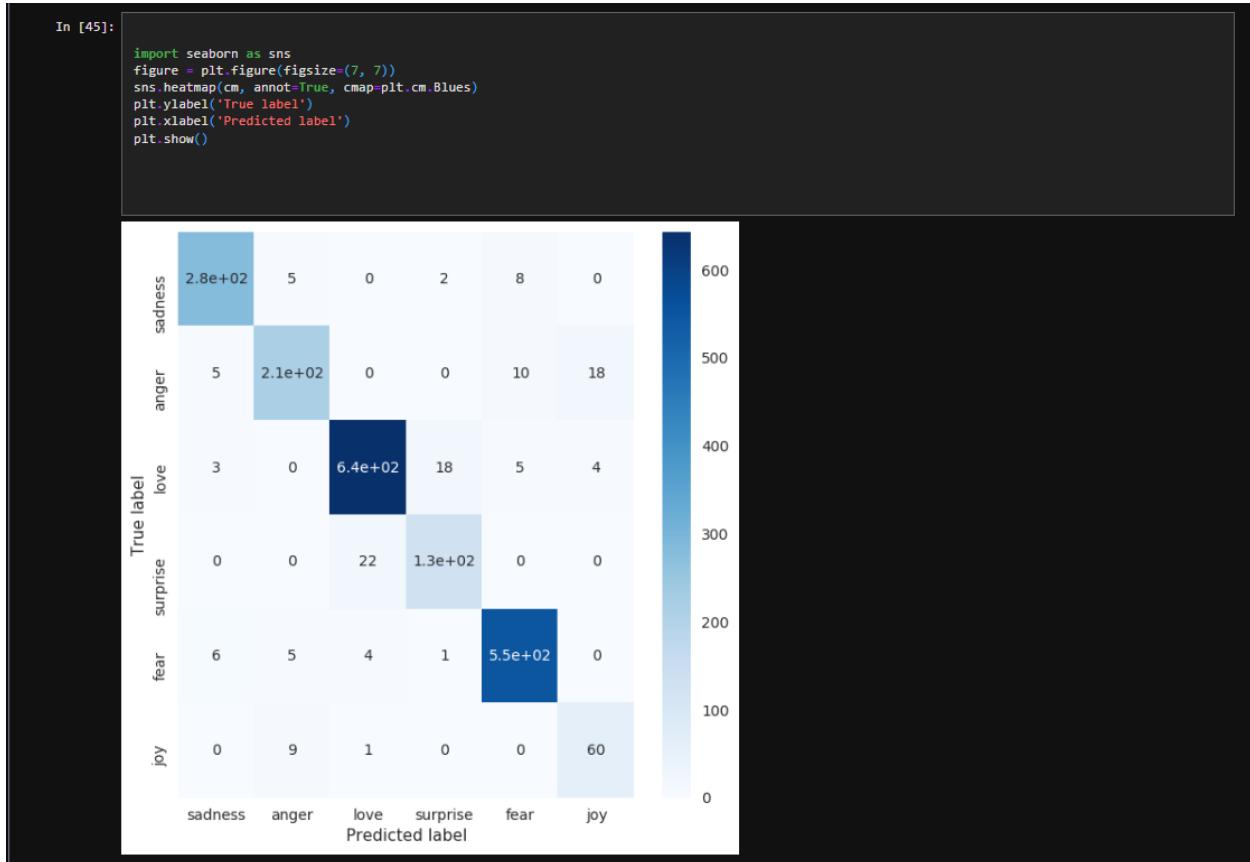
The screenshot shows a Jupyter notebook interface in Amazon SageMaker Studio. The code cell displays a confusion matrix for emotion classification. The matrix shows counts for five emotions: sadness, anger, love, surprise, and fear. The x-axis represents the predicted label and the y-axis represents the true label. The matrix is annotated with numerical values.

	sadness	anger	love	surprise	fear	joy
sadness	281	5	0	2	9	0
anger	5	214	0	0	10	18
love	3	0	643	18	5	4
surprise	0	0	22	126	0	0
fear	6	5	4	1	550	0
joy	0	9	1	0	0	60

d) Predicting on Test Data

```
[41]: y_pred = emotion.m.predict(emotion.X_test)
y_pred = np.argmax(y_pred, axis = 1)
y_pred
63/63 [=====] - 2s 21ms/step
[41]: array([4, 0, 5, ..., 0, 2, 2])
```

e) Heatmap Visualization of Predicted labels



13. Comparison of Performance on different Clusters:

RNN being a very powerful Model we were getting results on “**t3-medium**” cluster instances (3 node) itself within seconds for the same dataset so kept.

Hence RNN is a very good Model to implement sentiment analysis on NLP Problems.

2. Implementing ML Model using Spark MLlib

An important task in ML is model selection, or using data to find the best model or parameters for a given task. This is also called tuning. Tuning may be done for individual Estimators such as LogisticRegression, or for entire Pipelines which include multiple algorithms, featurization, and other steps. Users can tune an entire Pipeline at once, rather than tuning each element in the Pipeline separately.

MLlib supports model selection using tools such as CrossValidator and TrainValidationSplit. These tools require the following items:

- **Estimator:** algorithm or Pipeline to tune
- **Set of ParamMaps:** parameters to choose from, sometimes called a “parameter grid” to search over
- **Evaluator:** metric to measure how well a fitted Model does on held-out test data

At a high level, these model selection tools work as follows:

- They split the input data into separate training and test datasets.
- For each (training, test) pair, they iterate through the set of ParamMaps: For each ParamMap, they fit the Estimator using those parameters, get the fitted Model, and evaluate the Model’s performance using the Evaluator.
- They select the Model produced by the best-performing set of parameters.

The Evaluator can be a Regression-Evaluator for regression problems, a Binary-Classification-Evaluator for binary data, a Multiclass-Classification-Evaluator for multiclass problems, a Multilabel Classification-Evaluator for multi-label classifications, or a Ranking-Evaluator for ranking problems.

The default metric used to choose the best ParamMap can be overridden by the set-Metric-Name method in each of these evaluators.

Below we have implemented this part in the following steps -

There are multiple cloud providers available to choose from in order to analyze spark R services so a few are AWS, Azure, Cloudera, data bricks, google cloud, etc. Here we choose AWS EMR using spark to run.

1. Create an EMR Cluster on your AWS Account.

Clusters				
Actions				
Cluster ID				
Name	ID	Status	Creation time (UTC-5)	Actions
Assgn 13	j-2TUK2CJCK18EX	Terminated Auto-terminate	2022-04-25 14:50 (UTC-5)	View details Clone Terminate
Assgn12	j-1CMSRY8LGZGNI	Terminated User request	2022-04-20 20:31 (UTC-5)	View details Clone Terminate
Ass10	j-9OBXKHQMWI9G	Terminated User request	2022-04-01 16:25 (UTC-5)	View details Clone Terminate
Assgn10	j-QWGRYH0CLXEN	Terminated User request	2022-03-31 09:14 (UTC-5)	View details Clone Terminate
Ass9	j-1C7V2I0WC3M2P	Terminated User request	2022-03-24 16:47 (UTC-5)	View details Clone Terminate
Assgn9	j-25R9WKU1YM8F0	Terminated User request	2022-03-24 16:19 (UTC-5)	View details Clone Terminate

2. Go to advanced options

Create Cluster - Quick Options [Go to advanced options](#)

General Configuration

Cluster name

Logging [i](#)

S3 folder [Folder](#)

Launch mode Cluster [i](#) Step execution [i](#)

Software configuration

Release [i](#)

Applications Core Hadoop: Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
 HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Phoenix 4.14.3, and ZooKeeper 3.4.14
 Presto: Presto 0.266 with Hadoop 2.10.1 HDFS and Hive 2.3.9 Metastore
 Spark: Spark 2.4.8 on Hadoop 2.10.1 YARN and Zeppelin 0.10.0

3. In the first step, Configure Software and Steps, please choose the latest EMR release in Software Configuration and select the below options

Software Configuration

Release [i](#)

<input checked="" type="checkbox"/> Hadoop 2.10.1	<input type="checkbox"/> Zeppelin 0.10.0	<input type="checkbox"/> Livy 0.7.1
<input type="checkbox"/> JupyterHub 1.1.0	<input type="checkbox"/> Tez 0.9.2	<input type="checkbox"/> Flink 1.14.2
<input type="checkbox"/> Ganglia 3.7.2	<input type="checkbox"/> HBase 1.4.13	<input type="checkbox"/> Pig 0.17.0
<input checked="" type="checkbox"/> Hive 2.3.9	<input type="checkbox"/> Presto 0.266	<input type="checkbox"/> ZooKeeper 3.4.14
<input type="checkbox"/> JupyterEnterpriseGateway 2.1.0	<input type="checkbox"/> MXNet 1.8.0	<input type="checkbox"/> Sqoop 1.4.7
<input type="checkbox"/> Mahout 0.13.0	<input type="checkbox"/> Hue 4.10.0	<input type="checkbox"/> Phoenix 4.14.3
<input type="checkbox"/> Oozie 5.2.1	<input checked="" type="checkbox"/> Spark 2.4.8	<input type="checkbox"/> HCatalog 2.3.9
<input type="checkbox"/> TensorFlow 2.4.1		

4. Maximize resource allocation for your current EMR cluster using the below configuration

```
{  
    "Classification": "spark",  
    "Properties": {  
        "maximizeResourceAllocation": "true"  
    }  
}
```

Multiple master nodes (optional)

Use multiple master nodes to improve cluster availability. [Learn more](#)

AWS Glue Data Catalog settings (optional)

Use for Hive table metadata [i](#)

Use for Spark table metadata [i](#)

Edit software settings [i](#)

Enter configuration Load JSON from S3

```
[{"Classification": "spark",}
```

Steps (optional)

A step is a unit of work you submit to the cluster. For instance, a step might contain one or more Hadoop or Spark jobs. You can add steps to your cluster at any time.

5. In the second step, configure the Hardware as below

Node type	Instance type	Instance count	Purchasing option
Master Master - 1	m4.2xlarge 8 vCore, 32 GiB memory, EBS only storage EBS Storage: 100 GiB Add configuration settings	1 Instances	<input checked="" type="radio"/> On-demand i <input type="radio"/> Spot i Use on-demand as max price
Core Core - 2	m4.2xlarge 8 vCore, 32 GiB memory, EBS only storage EBS Storage: 100 GiB Add configuration settings	0 Instances	<input checked="" type="radio"/> On-demand i <input type="radio"/> Spot i Use on-demand as max price
Task Task - 3	m4.2xlarge 8 vCore, 32 GiB memory, EBS only storage EBS Storage: 128 GiB Add configuration settings	0 Instances	<input checked="" type="radio"/> On-demand i <input type="radio"/> Spot i Use on-demand as max price

6. Go to third step, to configure general cluster settings so here we configure bootstrap action.

- Next go to amazon blog -

<https://aws.amazon.com/blogs/big-data/running-sparklyr-r-interface-to-spark-on-amazon-emr/>

- and edit the shell script as below and save it as a .sh file in R:

https://aws-bigdata-blog.s3.amazonaws.com/artifacts/aws-blog-emr-rstudio-sparklyr/rstudio_sparklyr_emr5.sh

- Go to aws and open S3, Create a bucket - [sparkbigdataproj](#)

Buckets (2) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

Find buckets by name

Name	AWS Region	Access	Creation date
aws-logs-686071142997-us-east-1	US East (N. Virginia) us-east-1	Objects can be public	March 24, 2022, 16:19:53 (UTC-05:00)
sparkbigdataproj	US East (Ohio) us-east-2	Objects can be public	April 27, 2022, 12:18:14 (UTC-05:00)

7. Upload the sh file into it

Files and folders (1 Total, 6.1 KB)

All files and folders in this table will be uploaded.

	Name	Folder	Type	Size
<input type="checkbox"/>	bootstrapaws.sh	-	text/x-sh	6.1 KB

Destination

Destination
<s3://sparkbigdataproj>

► Destination details

8. Open the object and Copy the S3 URI Path

Owner	S3 URI
c553025e5cd87e8f54ede52a1f6a3f690ca5952b15ae9968a9cc335d5aeedfc4	<input type="checkbox"/> s3://sparkbigdataproj/bootstrapaws.sh
AWS Region	Amazon Resource Name (ARN)
US East (Ohio) us-east-2	<input type="checkbox"/> arn:aws:s3:::sparkbigdataproj/bootstrapaws.sh
Last modified	Entity tag (Etag)
April 27, 2022, 12:21:27 (UTC-05:00)	<input type="checkbox"/> 65b612ddca72b71287b0cb5d38705987
Size	Object URL
6.1 KB	<input type="checkbox"/> https://sparkbigdataproj.s3.us-east-2.amazonaws.com/bootstrapaws.sh
Type	
sh	
Key	

9. Choose Custom Action then Configure and add the URI Path in emr cluster.

Bootstrap Actions

Bootstrap actions are scripts that are executed during setup before Hadoop starts on every cluster node. You can use them to install software and customize your applications. [Learn more](#)

Add bootstrap action

Edit Bootstrap Action

Bootstrap action type	Custom action
Name	Sparklyr
Script location	s3://sparkbigdataproj/bootstrapaws.sh
Optional arguments	--rstudio --rstudio-url https://download2.rstudio.org/server/centos/x86_64/rstudio-server-rhel-1.2.1335-x86_64.rpm

Cancel **Save**

10. Go to last step, Security unselect the cluster visible option and create a cluster

Security Options

EC2 key pair Proceed without an EC2 key pair **i**

Cluster visible to all IAM users in account **i**

Permissions **i**

Default Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR_DefaultRole](#) **i** Use EMR_DefaultRole_V2 **i**

EC2 instance profile [EMR_EC2_DefaultRole](#) **i**

Auto Scaling role [EMR_AutoScaling_DefaultRole](#) **i**

► Security Configuration

► EC2 security groups

11. We will wait cluster to start and then configure the security groups for Master and add the edit inbound rules as below:

Visible to all users: All [Change](#)

Security groups for Master: [sg-0b49b4d6b7dc221bc](#) **i** (ElasticMapReduce-master)

Security groups for Core & [sg-0ed1fc5a25dcb57a6](#) **i** (ElasticMapReduce-slave)
Task:

The screenshot shows the AWS Management Console interface for managing security group inbound rules. A specific security group, sg-0b49b4d6b7dc221bc, is selected. Two rules are listed:

- Rule 1:** Protocol: SSH, Port Range: 22, Source: 64.131.120.16/32. This rule is highlighted with a blue border.
- Rule 2:** Protocol: TCP, Port Range: 8787, Source: My IP.

At the bottom left, there is a button labeled "Add rule".

12. Then, create a cluster, after sometime the cluster status changes to bootstrapping.

Cluster: My cluster Bootstrapping Running bootstrap actions

Summary Application user interfaces Monitoring Hardware Configurations Events Steps Bootstrap actions

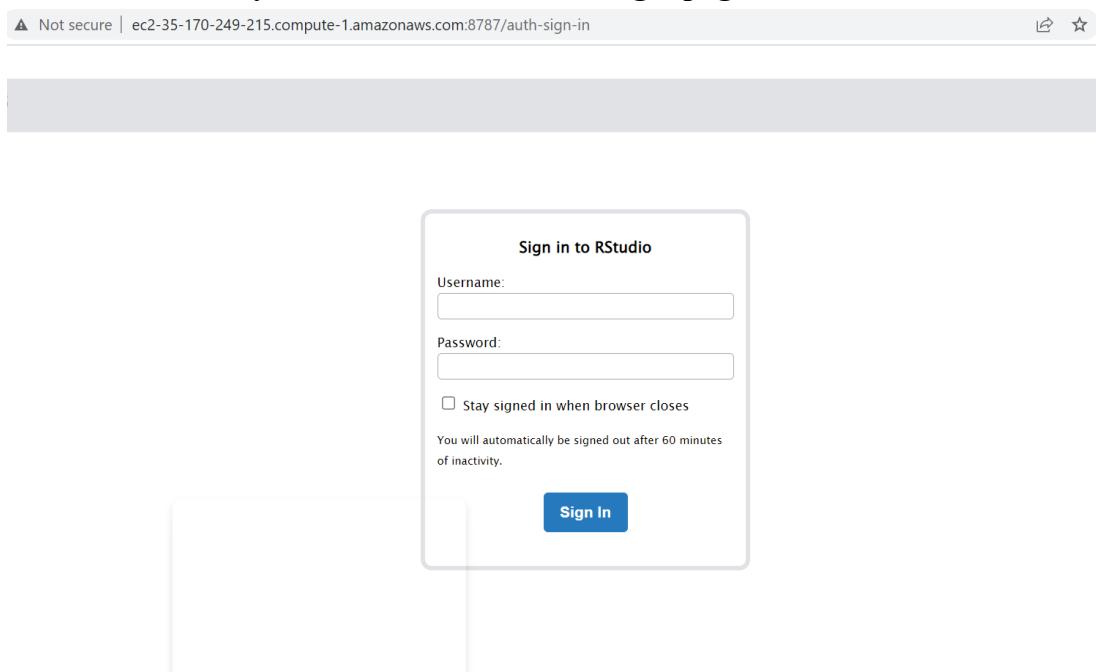
Summary

ID: j-HDJUMIG8TAAH
Creation date: 2022-04-30 15:34 (UTC-5)
Elapsed time: 9 minutes
After last step completes: Cluster waits
Termination protection: On [Change](#)
Tags: -- [View All / Edit](#)
Master public DNS: ec2-44-201-87-56.compute-1.amazonaws.com [Connect to the Master Node Using SSH](#)

Configuration details

Release label: emr-5.35.0
Hadoop distribution: Amazon 2.10.1
Applications: Hive 2.3.9, Spark 2.4.8
Log URI: s3://aws-logs-686071142997-us-east-1/elasticmapreduce/ [Logs](#)

13. Add the 8787 port to the url - <http://ec2-35-170-249-215.compute-1.amazonaws.com:8787/> and open it in a new tab- you will be able to see the login page of R studio.



Now we are ready to implement the Machine Learning Model using the Spark-R configuration we just set up.

What are we implementing in Spark-MLlib

1. Determining Sentiment using **Random Forest**.
2. Determining Sentiment using **Logistic Regression**

14. Installed the required packages

```
> install.packages("dplyr")
also installing the dependencies 'glue', 'cli', 'rlang', 'vctrs'

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/glue_1.6.2.tgz'
Content type 'application/x-gzip' length 155475 bytes (151 KB)
=====
downloaded 151 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/cli_3.3.0.tgz'
Content type 'application/x-gzip' length 1171398 bytes (1.1 MB)
=====
downloaded 1.1 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/rlang_1.0.2.tgz'
Content type 'application/x-gzip' length 1822104 bytes (1.7 MB)
=====
downloaded 1.7 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/vctrs_0.4.1.tgz'
Content type 'application/x-gzip' length 1761486 bytes (1.7 MB)
=====
downloaded 1.7 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/dplyr_1.0.9.tgz'
Content type 'application/x-gzip' length 1327720 bytes (1.3 MB)
=====
downloaded 1.3 MB

The downloaded binary packages are in
  /var/folders/d/_23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8 downloaded_packages
> |
```

```
The downloaded binary packages are in
  /var/folders/d/_23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8 downloaded_packages
> install.packages("data.table")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/data.table_1.14.2.tgz'
Content type 'application/x-gzip' length 2354815 bytes (2.2 MB)
=====
downloaded 2.2 MB

The downloaded binary packages are in
  /var/folders/d/_23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8 downloaded_packages
> install.packages("ggplot2")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/ggplot2_3.3.6.tgz'
Content type 'application/x-gzip' length 4122229 bytes (3.9 MB)
=====
downloaded 3.9 MB

The downloaded binary packages are in
  /var/folders/d/_23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8 downloaded_packages
> |
```

TC Off Campus housing 11:14 AM

Once we have set up the environment, we are now implementing the actual part. Importing necessary libraries. We'll be using a number of Python tools and frameworks. Using the function readFile(filename), we will read the contents of a text file and output the contents of the file as a list containing single words. Using this function, we will read all the CSV files, in which each row represents a text file and the columns contain the counts of each word in that specific text file. Read the CSV files and display data(training testing validation)

1. Training Data Image

2. Validation Data glimpse(df_val)

3. Viewing Glimpse of Dataset and Preprocessing

```
+   select(Class) %>%
+   group_by(Class) %>%
+   summarise(count = n()) %>%
+   glimpse
```

Rows: 2
Columns: 2

Class	count
0	170588
1	295

4. Implement logistic regression in Spark-R

Now we train a basic logistic regression model to classify the sentiment of the reviews. Make sure you do not use the filename as a feature if you previously included it in the Dataframe. Compare the accuracy of this basic model on the training set and the validation set. Are you overfitting? Try changing the parameters of the logistic regression, such as adding a regularization term, to reduce the overfitting.

```

install.packages("dplyr")
df_train <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/
train.csv")
df_test <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/train.csv")
df_val <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/train.csv")

spark_write_parquet(df_train, path="/user/rstudio-user",
mode="overwrite")
spark_write_parquet(df_test, path="/user/rstudio-user", mode="overwrite")
spark_write_parquet(df_val, path="/user/rstudio-user", mode="overwrite")

dim(df_train)
dim(df_val)
dim(df_test)

df_train$Class <- factor(df_train$Class)

train %>%
  select(Class) %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  glimpse

test %>%
  select(Class) %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  glimpse

#Logistic Regression model
Logistic_Model=glm(Class~.,test_data,family=binomial())
summary(Logistic_Model)

```

Logistic Regression output-1

```

> summary(Logistic_Model)

Call:
glm(formula = Class ~ ., family = binomial(), data = df_test)

Deviance Residuals:
    Min      1Q   Median      3Q     Max 
-4.7005 -0.0290 -0.0177 -0.0101  4.1608 

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -8.525e+00 3.399e-01 -25.080 < 2e-16 ***
Time        -4.941e-06 2.860e-06 -1.727 0.084110 .
V1          1.397e-01 5.491e-02  2.544 0.010966 *  
V2          5.574e-02 8.832e-02  0.631 0.527959  
V3         -9.386e-02 6.724e-02 -1.396 0.162740  
V4          7.831e-01 1.153e-01  6.793 1.10e-11 *** 
V5          1.679e-01 9.973e-02  1.683 0.092324 .  
V6         -1.671e-01 9.540e-02 -1.752 0.079845 .  
V7         -9.329e-02 9.375e-02 -0.995 0.319690  
V8         -1.979e-01 3.831e-02 -5.166 2.39e-07 *** 
V9         -5.196e-02 1.761e-01 -0.295 0.767896  
V10         7.846e-01 1.478e-01 -5.310 1.09e-07 *** 
V11         7.781e-02 1.055e-01 -0.738 0.460796  
V12         1.511e-01 1.114e-01  1.356 0.174961  
V13         -3.745e-01 1.058e-01 -3.540 0.000400 *** 
V14         -6.649e-01 8.045e-02 -8.265 < 2e-16 *** 
V15         -1.838e-01 1.084e-01  1.804e-01 -1.696 0.089825 . 
V16         1.898e-02 2.211e-01  0.086 0.931600  
V17         -2.416e-02 9.246e-02 -0.261 0.793863  
V18         2.615e-01 2.121e-01 -0.752 0.452002  
V19         2.615e-01 1.470e-01  1.778 0.075333 .  
V20         -3.901e-01 1.128e-01 -3.457 0.000545 *** 
V21         4.734e-01 8.580e-02  5.517 3.45e-08 *** 
V22         8.765e-01 1.803e-01  4.860 1.17e-06 *** 
V23         -1.337e-01 7.490e-02 -1.785 0.074310 . 
V24         1.602e-01 1.848e-01  0.867 0.385821  
V25         3.093e-02 1.659e-01  0.186 0.852130  
V26         -1.228e-01 2.433e-01 -0.505 0.613668  
V27         -9.467e-01 1.409e-01 -6.718 1.85e-11 *** 
V28         -4.860e-01 1.389e-01 -3.498 0.000469 *** 
Amount       1.151e-03 4.907e-04  2.346 0.019001 *  

```

Logistic Regression output-2

V10	-7.846e-01	1.478e-01	-5.310	1.09e-07	***
V11	-7.781e-02	1.055e-01	-0.738	0.460796	
V12	1.511e-01	1.114e-01	1.356	0.174961	
V13	-3.745e-01	1.058e-01	-3.540	0.000400	***
V14	-6.649e-01	8.045e-02	-8.265	< 2e-16	***
V15	-1.838e-01	1.084e-01	-1.696	0.089825	.
V16	1.898e-02	2.211e-01	0.086	0.931600	
V17	-2.416e-02	9.246e-02	-0.261	0.793863	
V18	2.615e-01	2.121e-01	-0.752	0.452002	
V19	2.615e-01	1.470e-01	1.778	0.075333	.
V20	-3.901e-01	1.128e-01	-3.457	0.000545	***
V21	4.734e-01	8.580e-02	5.517	3.45e-08	***
V22	8.765e-01	1.803e-01	4.860	1.17e-06	***
V23	-1.337e-01	7.490e-02	-1.785	0.074310	.
V24	1.602e-01	1.848e-01	0.867	0.385821	
V25	3.093e-02	1.659e-01	0.186	0.852130	
V26	-1.228e-01	2.433e-01	-0.505	0.613668	
V27	-9.467e-01	1.409e-01	-6.718	1.85e-11	***
V28	-4.860e-01	1.389e-01	-3.498	0.000469	***
Amount	1.151e-03	4.907e-04	2.346	0.019001	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					
(Dispersion parameter for binomial family taken to be 1)					
Null deviance: 4342.9 on 170882 degrees of freedom					
Residual deviance: 1375.6 on 170852 degrees of freedom					
AIC: 1437.6					
Number of Fisher Scoring iterations: 12					

4. Implement Random Forest in Spark-R

A Random Forest classifier random forest classifier. It is a type of ensemble learning method, where many decision trees (hence forest) make different predictions, and the majority outcome is chosen as the prediction. Random forest methods are also known for their resistance to over-fitting and are a widely used technique for classification problems. It prevents overfitting by using a series of weaker trees that cannot themselves overfit the training set and takes a majority vote from them to classify. Training a Classifier on the same dataset trains weak learners sequentially, so they focus on the points that are hard to classify, so we made sure to limit each individual tree so it is individually weak.

```
test$predictedTrim10 <- predict(rfModelTrim10, test)
F1_10 <- F1_Score(test$Class, test$predictedTrim10)
F1_10

# build data frame of number of variables and scores
numVariables <- c(1,2,3,4,5,10,17)
F1_Score <- c(F1_1, F1_2, F1_3, F1_4, F1_5, F1_10, F1_all)
variablePerf <- data.frame(numVariables, F1_Score)

# plot score performance against number of variables
options(repr.plot.width=4, repr.plot.height=3)
ggplot(variablePerf, aes(numVariables, F1_Score)) + geom_point() + labs(x = "Number of Variables", y = "F1 Score", title = "F1 Score Performance")

# build random forest model using every variable|
rf.fit <- randomForest(class ~ ., data=df_train, ntree=1000,
                        keep.forest=FALSE, importance=TRUE)

Training acc: 0.935714285714
Validation acc: 0.817857142857
```

Closing AWS Services and Releasing Resource Utilized

1. Deleting Cluster-Instances after completing project

The screenshot shows the Amazon SageMaker RStudio interface. On the left, a sidebar lists various options like 'Getting started', 'Control panel', and 'SageMaker dashboard'. The main area is titled 'User Details' and shows a table of applications. One application, 'datascience-1-0-ml-m5-xlarge-372beb14237fd194a4e6424fd19', is in the process of being deleted. Another application, 'default', is listed as ready. On the right, a detailed view of the user profile 'csp554bigdatabert' is shown, including its name, execution role, status, ID, and creation/modification details.

2. Removing Roles after completing Project

The screenshot shows the AWS Identity and Access Management (IAM) service. The left sidebar has 'Identity and Access Management (IAM)' selected. Under 'Access management', 'Roles' is also selected. The main table lists 14 IAM roles. Most of them have checkboxes checked, indicating they are selected for deletion. A message at the top right says 'Activate Windows' and 'Go to Settings to activate Windows.' The bottom of the screen shows the Windows taskbar with various icons and the date/time '20-Nov-22 07:23 PM'.

IAM > Roles

Roles (6) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForAmazonSageMakerNotebooks	AWS Service: sagemaker (Service-Linked Role)	1 hour ago
AWSServiceRoleForEMRCleanup	AWS Service: elasticmapreduce (Service-Linked Role)	12 days ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
EMR_DefaultRole	AWS Service: elasticmapreduce	12 days ago
EMR_EC2_DefaultRole	AWS Service: ec2	12 days ago

3. Removing Notebook Instances after Completing Project

Notebook instances

Name	Instance	Creation time	Status	Actions
CSP554BigData	ml.t3.large	Nov 20, 2022 05:34 UTC	Deleting	-

4. Removing AWS S3 Buckets Created during Project

Buckets (3) Info

Name	AWS Region	Access	Creation date
sagemaker-studio-2f90mv34lt7	US East (N. Virginia) us-east-1	Objects can be public	November 20, 2022, 16:53:45 (UTC-06:00)
sagemaker-studio-39442337255-0qkb9d39h12e	US East (N. Virginia) us-east-1	Objects can be public	November 19, 2022, 23:36:46 (UTC-06:00)
sagemaker-us-east-1-39442337255	US East (N. Virginia) us-east-1	Objects can be public	November 20, 2022, 16:58:45 (UTC-06:00)

CONCLUSION

This project aimed to explore, analyze, and build a machine learning algorithm using Python, R, SageMaker, Spark-MLLib, SparkR, TensorFlow. We implemented various ML models to perform Sentiment Analysis of Emotions on a Text Dataset containing 20000 records. We implemented a few Deep Learning classifier models using BERT, RoBERTa and RNN on AWS Sagemaker and SparkMLlib.

The Model with BERT provided us with an accuracy of 97% while the RoBERTa provided 93.75%, and RNN classifier with 97.79% accuracy. The Loss parameters were similar and small for all of these models. Hence, the RNN model appears to be a better option for such data. Although RoBERTa is a BERT model without NSP objective & improved dynamic mask generation, we found BERT worked better for our data. Additionally, we compared performance of these models on different Cluster instances with distinct nodes. We found better CPU/GPU configuration definitely impacts execution speed in AWS.

One of the drawbacks of the data however is that it is past data. We might use a recent data frame that has been updated in future. These new data points can then be used to make predictions or train new models for more accurate results. With larger dataset tuning parameters, we may try to improve the model's accuracy.

REFERENCES

1. Emotions dataset for NLP - <https://www.kaggle.com/datasets/praveengovi/emotionsdataset-for-nlp>
2. Practical Data Science on AWS Cloud - <https://github.com/Ashleshk/Practical-Data-Scienceon-the-AWSCloud-Specialization>
3. Machine Learning using Spark MLlib - <https://www.youtube.com/watch?v=BxOhyCQ008&t=2s>
4. Tensorflow in AWS - <https://docs.aws.amazon.com/deep-learningcontainers/latest/devguide/deep-learningcontainers-ecs-tutorials-training.html#deeplearning-containers-ecs-tutorials-training-tf>
5. SageMaker Case Study - https://sagemakerexamples.readthedocs.io/en/latest/introduction_to_applying_machine_learning/breast_cancer_prediction/Breast%20Cancer%20Prediction.html
6. SageMaker Studio Notebook Architecture - <https://aws.amazon.com/blogs/machinelearning/dive-deep-into-amazon-sagemaker-studio-notebook-architecture/>
7. GloVe Word Mapping - <https://en.wikipedia.org/wiki/GloVe>