**Objective:**

To train machine learning or deep learning model and classify upcoming news on the fly with good accuracy by building end to end machine learning pipeline.

To make containerized application, which is scalable, robust, fault tolerant.

**Planning:**

We are using agile methodology to build the project. Task level details are mentioned in below Gantt chart.

#Sprint: 4

#People: 2

Sprint:1

Table

Description automatically generated

Sprint:2

Graphical user interface, table

Description automatically generated

Sprint:3

**Graphical user interface

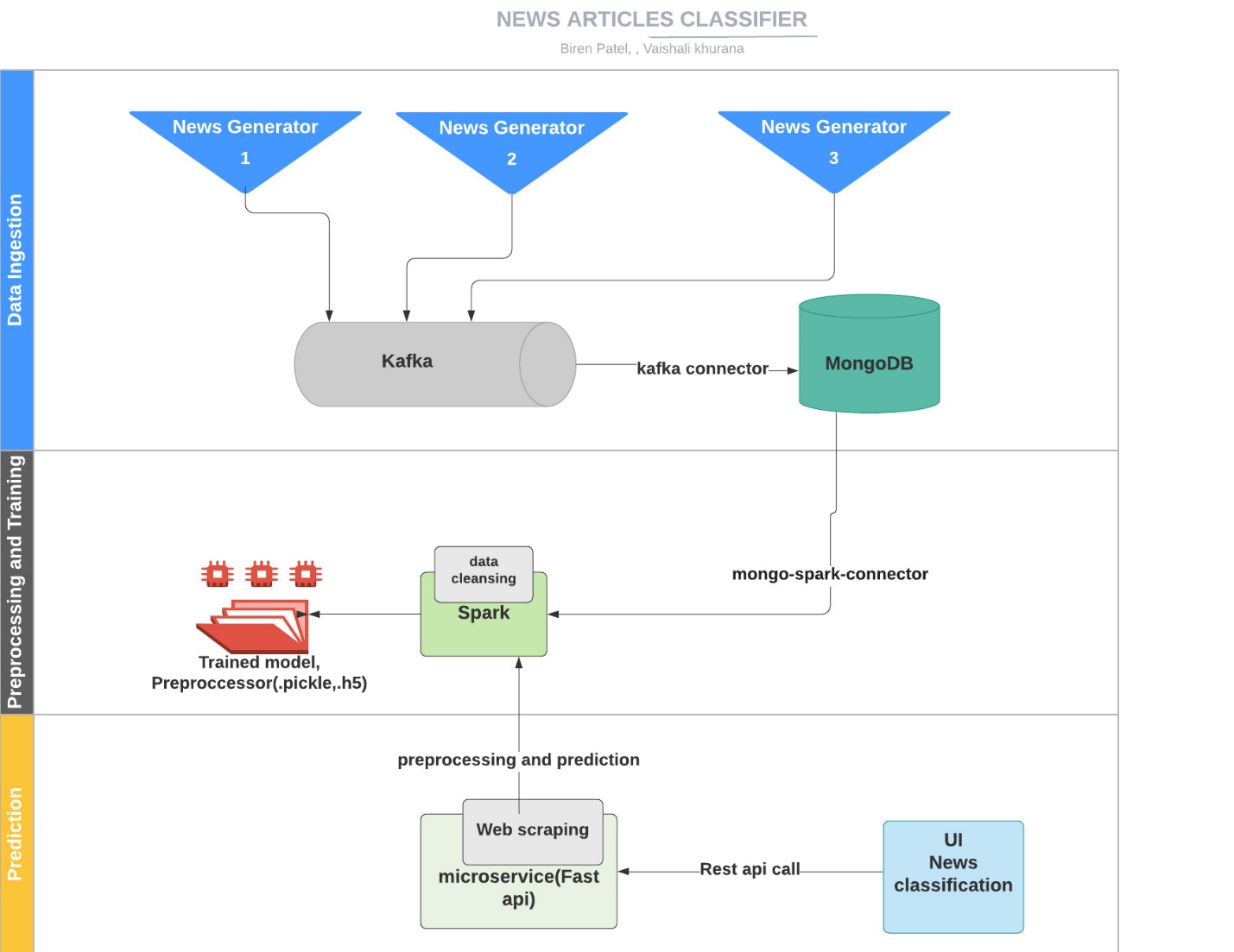
Description automatically generated with medium confidence**

Sprint:4

**Graphical user interface

Description automatically generated with medium confidence**

**Architecture:**

****

Bert

|  |
| --- |
|  |

**Bert model (Transformer) Architecture:**

Diagram

Description automatically generated

**Components and Description:**

|  |  |
| --- | --- |
| Component | Description |
| Data Ingestion | Multithreaded service that is collecting the data from web using rapid api and custom news generator, passing it to kafka queue. Kafka consumer is used to sync the data between kafka and database. Finally, data is dumped into MongoDb database |
| Preprocessor and Trainer | Reading data from mongo db to spark session using mongo-spark-connector.  As a part of feature selection we are using category and summary columns.  Data cleansing including stop word removal, tokenization, tf-idf vectorization using pyspark.  This cleaned data is used for training bert model and achieved training accuracy of 87.8% and test accuracy of 82.9%. |
| Predictor and UI | Predicting news category based on the news summary entered by user in UI. Clicking “Predict” button calls backend rest API which firstly processes the data using the trained tokenizer and then predict using the trained Bert model. |

**Environment Details:**

Docker environment with kafka broker running on 9092 port, zookeeper running on 2181, mongo-db running on 27017, mongo-express running on 8082, spark-master running on 8080, spark-worker running on 8081, predictor service running on 8888 port.  
Producer is connecting to kafka and producing news records to kafka queue, consumer is consuming news records from kafka and dumping to mongo db, preprocessor\_trainer is cleansing the news data and training machine learning model on spark node using pyspark and predictor service has UI integration with Fast API in backend.

* **Hardware Details:** 
  + NVIDIA GPU
  + Memory requirements: Minimum 4GB RAM dedicated for docker
* **Tools/libraries used**:
  + Docker
  + Pycharm
  + Kafka
  + Zookeeper
  + MongoDB
  + MongoExpress
  + Spark
  + Pyspark
  + colab notebook
  + Tensorflow : 2.0.0
  + Seaborn
  + mlflow
  + Pretrained Bert model(Transformer): uncased\_L-8\_H-512\_A-8
  + Fast API
  + HTML
  + CSS
  + Jinja2Templates

**Data Ingestion:**

* **What goes in as an input:**

We have used rapid api and custom news generator as data sources.

***Graphical user interface, text

Description automatically generated***

* **How the input is being processed:**

Multithreaded application that is collecting the data from web using rapid api and custom news generator, passing it to kafka queue. Kafka consumer is dumping the data into mongo db.

**Graphical user interface, text

Description automatically generated**

* **What comes out as an output:**

News records saved in news\_collection under news database MongoDb.

Graphical user interface, application

Description automatically generated

**Data Cleansing:**

* **What goes in as an input:**

News records saved in news\_collection under news database MongoDb

Graphical user interface, application

Description automatically generated

* **How the input is being processed:**

Records from mongo db collection is processed using pyspark over spark node which is connected through mongo\_spark\_connector.

As a part of feature selection we are using category and summary columns.

Data cleansing including stop word removal, tokenization, tf-idf vectorization using pyspark

Text

Description automatically generated

**After feature selection and tf-idf vectorization:**

Text

Description automatically generated

* **What comes out as an output:**

Processed and cleaned data, saved the preprocessed pipeline

Graphical user interface, text

Description automatically generated

**Model Training:**

* **What goes in as an input:**

Spark dataframe having cleaned data.

Chart, bar chart

Description automatically generated

* **How the input is being processed:**

After the records from mongo db collection is processed and cleaned, these records are further used for training using pyspark over spark node which is connected through mongo\_spark\_connector.

Input data is divided into 80% training and 20% test data.

Pretrained Bert model(uncased\_L-8\_H-512\_A-8) is also used for training for 15 epochs with training accuracy of 87.8% and test accuracy of 82.9%, with following parameters:

**Table

Description automatically generated**

* **What comes out as an output:**

The trained model and tokenizer is saved in registry.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

**Table

Description automatically generated**

**Model Prediction:**

* **What goes in as an input:**

News Summary text entered by user in UI

**Graphical user interface, text, application, email

Description automatically generated**

* **How the input is being processed:**

Clicking “Predict” button calls backend rest API which firstly processes the data using the trained tokenizer and then predict using the trained Bert model and returned response is displayed on UI.

Graphical user interface, text, application, email, website

Description automatically generated

* **What comes out as an output:**

Predicted news category is returned to the UI.

Graphical user interface, text, application, email

Description automatically generated

**Challenges encountered and how we tackled them:**

We faced below mentioned challenges:

* Finding better legal data sources
  + We have used Rapid API and custom dataset.
* Rapid API rate limiting makes the pipeline slow
  + For training, we have used the dataset dumped into mongodb
* Data Labeling
  + Annotating correct labels to custom dataset was done manually
* Connecting MongoDb and spark
  + Tried different ways of connecting spark to mongodb then implemented it with mongo-spark-connector
* Authorization issue in MongoDb connection
  + Configured appropriate parameters to resolve it
* Loading model from HDFS
  + Configured HDFS in spark node.
* Python-Tensorflow version compatibility
  + Installed compatible libraries versions
* Slow prediction because of Hardware limitation
  + Increased docker memory allocation and processor cores
* Loading Bert tokenizer in predictor service
  + Used save and load methods of pytorch
* Responsive UI Design and Integration for mobile support
  + Used Jinja2Templates

**Future Scope:**

* Further scale optimizations
* Implementing re-training mechanism using feedback feature
* Once we have large volume of labeled data, we will train our own model in place of transfer learning
* Adding multi language support

**Github link:**

https://github.com/biren162/Capstone