# Report M3: Model Experimentation and Packaging

[Report M3: Model Experimentation and Packaging 1](#_Toc187091315)

[Model Details 1](#_Toc187091316)

[Dataset 1](#_Toc187091317)

[Model 2](#_Toc187091318)

[Hyperparameter Tuning 3](#_Toc187091319)

[On Demand Training: 3](#_Toc187091320)

[Hyperparameter Tuning Results 4](#_Toc187091321)

[Model Packaging 4](#_Toc187091322)

[Docker file 5](#_Toc187091323)

[Running the docker to create the image 7](#_Toc187091324)

[Build 7](#_Toc187091325)

[Run Locally 7](#_Toc187091326)

[Push 8](#_Toc187091327)

Objective: Train a machine learning model, perform hyperparameter tuning, and package the model for deployment.

## Model Details

The model we have developed is a sentiment prediction on 2019 Indian Election Tweets.

### Dataset

The original dataset has more than 160000 records. The dataset contains 9 columns. This first column is the tweet, and the second column is the corresponding label with three classes. For this assignment we chose this dataset because of its size, it helps to demonstrate the DVC use cases required in **“M2: Process and Tooling”.**

* 1 – Positive
* 0 – Neutral
* -1 – Negative.

A screenshot of a phone

Description automatically generated

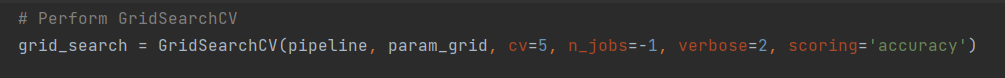
### Model

* First, we apply a set of NLP preprocessing techniques such as converting to lower case, removal of special characters, removal of URLs, removal stop words, lemmatization etc. The data is appended with another column “**cleaner\_tweet**” after this preprocessing.
* Then the training records are split into 80-20 basis for training and testing.
* Then a sklearn pipeline is created as shown below.

A screen shot of a computer program

Description automatically generated

* + The pipeline has two components.
    - TFIDFVectorizer takes the cleaned tweet and converts into TF-IDF Vector of features.
    - The logistic regression algorithm takes the vectorized features and predicts the output.
* Then a grid search is defined as shown below to find the best model.



* In the method for training the parameter grid is passed from the caller. This helps us to write unit tests for this grid search with fewer parameters and train the model on lots of parameters. This point is helpful to write testing in the GitHub Actions requirement of “**M1: MLOps Foundations.**”
* Once the best model is found by the grid search, the best model is stored as a pickle file as shown below.



The pkl\_file\_path is passed as an environment variable in local testing.

A screenshot of a computer

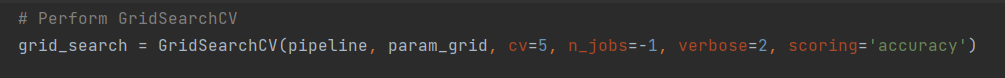
Description automatically generated

Also, the environment variables are mentioned in the docker file below.

A computer screen shot of a black screen

Description automatically generated

## Hyperparameter Tuning

* As mentioned above, the training method takes the parameter from the caller who is invoking the training. This will help to customize the training parameter without modifying the training code. 
* The parameters passed for the actual training are shown below.

A computer screen with white text

Description automatically generated

There are four parameters.

* + **Tfidf\_\_min\_df**: This parameter defines the minimum number of documents a word should appear in order to be considered as eligible for vectorizing. This parameter is really important to reduce the dimensionality of the feature set by avoiding really rare words that appear in just one or two tweets.
    - We have given two possible values for this 0.001 (0.01%) of the documents or 0.01 (1%) of the documents.
  + **Tfidf\_\_max\_features:** The maximum number of features after vectorizing. This also helps to keep the dimensionality under control. 1000, 1500, 10000 are the three possibilities given.
  + **Clf\_\_penalty:** L1 regularization or L2 regularization are the two options.
  + **Clf\_\_max\_iter:** Maximum iterations if the model fails to converge. There are three possible values 100, 500, 1000.

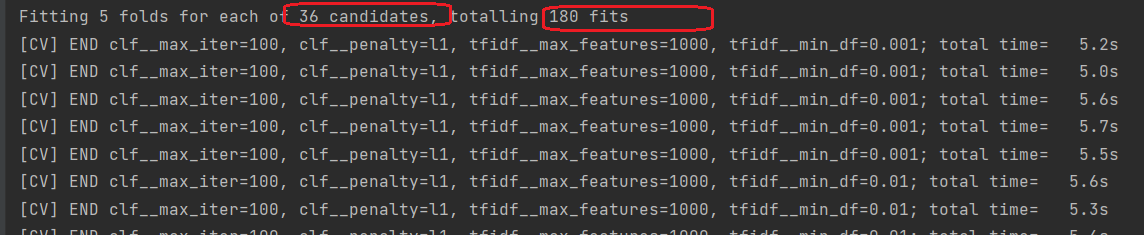
As shown above the grid search has a possibility of 2\*3\*2\*3 = 36 combinations.

### On Demand Training:

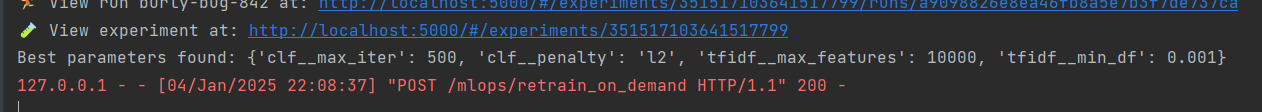
We have provided an option and exposed an API for invoking the training. The URL is: “**<<hostname:port>>/mlops/retrain\_on\_demand**”. The API invokes the GridSearchCV using the model described above and searches all the thirty-six combinations to produce the best model.

#### Hyperparameter Tuning Results

Response of the training is shown in the screenshot. The best model found on the full dataset of 160000 records is **max iteration 500, l2 regularization, with max features of 10000 and min df of 0.001**. It took 265 seconds to complete all the 36 combinations.



As we can see it totally did 180 fits, because of 5-fold cross validations, i.e. 5 fits for each of the 36 candidates and found the best fit. This process is quite exhaustive and may not be best approach for larger models, so we might prefer some sort of advanced mechanisms, such as Multi-fidelity hyperparameter optimization or Bayesian optimizations in large models.



The evaluation scores are also printed by the API, with all the four critical metrics at ~85%

A screenshot of a computer

Description automatically generated

## Model Packaging

As seen above we have produced the best model and saved it as a pickle file.



We have mentioned above that the training was invoked on demand using API exposed. This API was exposed in a flask application.

Similarly, we have exposed the best model for prediction using the flask application also.

A screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

The code for prediction is shown above.

As seen, earlier we had created a SkLearn pipeline to TF-IDF Vectorize and then do the regression. That is helpful here as during prediction we can simply send the text to the model for prediction and the SkLearn pipeline takes care of vectorizing the natural language text before passing to the algorithm. We do not have to explicitly vectorize the tweet before prediction.

While the code snippet is shared above, the entire file can be seen at <https://github.com/mnazaar/MLOPS_GROUP_13_Assignement1/blob/dev/Twitter_Sentiment_Indian_Election_2019/src/main/app.py>

### Docker file

The above model is packaged in a docker app using the docker file.

A screenshot of a computer program

Description automatically generated

The above file has the environment variable for the app to read the data file for retraining and where to store or read the trained model.

The file can be verified at <https://github.com/mnazaar/MLOPS_GROUP_13_Assignement1/blob/deploy/Dockerfile>

Also, the requirement.txt has the dependencies needed to run this model without any issues consistently.

<https://github.com/mnazaar/MLOPS_GROUP_13_Assignement1/blob/deploy/requirements.txt>

A screenshot of a computer program

Description automatically generated

Also, it exposes the application via port 5001 in the container.

**Docker file at github** <https://github.com/mnazaar/MLOPS_GROUP_13_Assignement1/blob/main/Dockerfile>

### Running the docker to create the image

For documenting the image creation, the version 1.0.5 of the application is packaged using the command.

#### Build

docker build -t nazaarblue/mlops\_twitter\_senti:1.0.5 .

A black screen with blue and white lines

Description automatically generated

Once built we can see the image staged in the local docker repository.

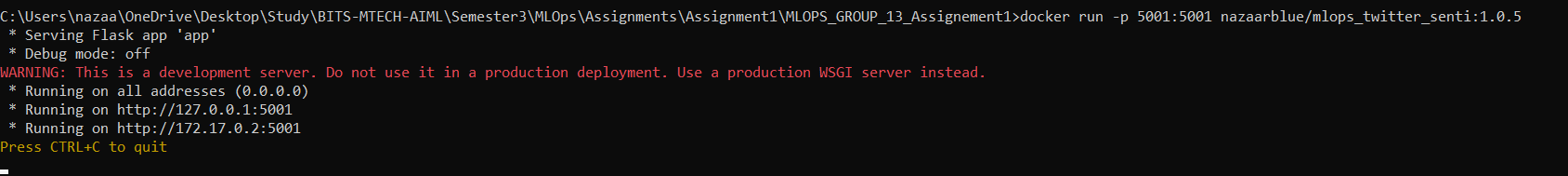
A screenshot of a computer

Description automatically generated

#### Run Locally

Once the model is built, we can run the docker image locally to validate the application.

docker run -p 5001:5001 nazaarblue/mlops\_twitter\_senti:1.0.5



The application has started, and we call use curl or postman for prediction.

A screenshot of a computer

Description automatically generated

#### Push

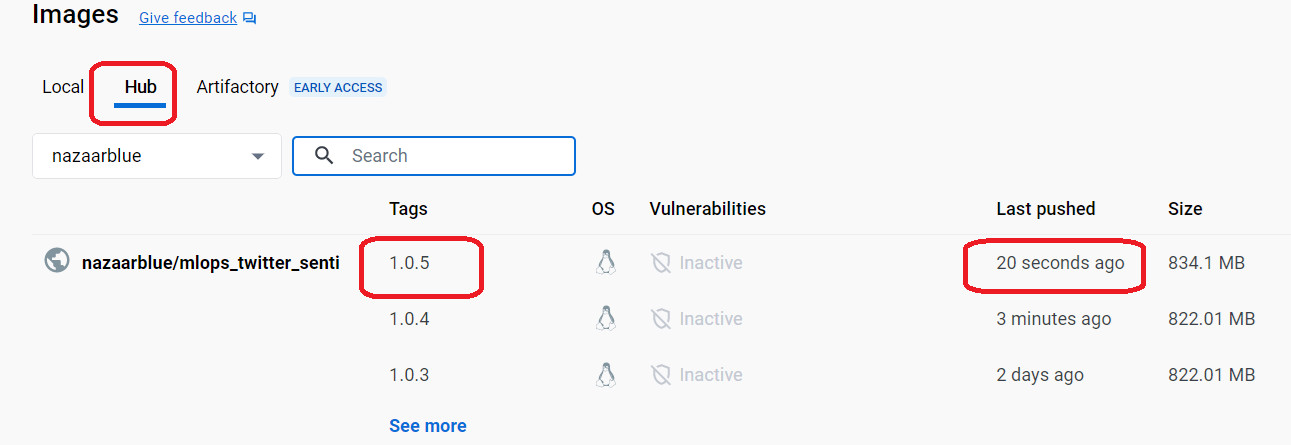
Now we can push the image to the docker hub repository using the command.

docker push nazaarblue/mlops\_twitter\_senti:1.0.5

A black rectangular frame with white and blue spots

Description automatically generated with medium confidence

Once pushed the model is available in the hub as well.



In “**M4: Model Deployment & Orchestration**”, we will deploy the model built above to a Kubernetes cluster on Google Cloud using deployment.yaml and helm charts.