Detection of Fake News

Introduction

"Fake News" is misinformation communicated through traditional media channels like print, and television as well as non-traditional media channels like social media. A made-up story with an intention to deceive. News articles that are intentionally and verifiably false and could mislead readers. Fake news is a type of yellow journalism or propaganda that consists of deliberate misinformation or hoaxes spread via traditional print and broadcast news media or online social media.

Problem in defining a Fake news

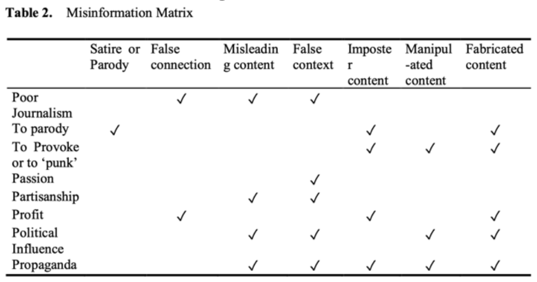


Table 2. Misinformation Matrix

	Satire or Parody	False connection	Misleading content	False context	Imposter content	Manipul-ated content	Fabricated content
Poor Journalism		✓	✓	✓			
To parody	✓				✓		✓
To Provoke or to 'punk'					✓	✓	✓
Passion				✓			
Partisanship			✓	✓			
Profit		✓			✓		✓
Political Influence			✓	✓		✓	✓
Propaganda			✓	✓	✓	✓	✓

Detecting fake news is a challenging task to accomplish as it requires models to summarize the news and compare it to the actual news to classify it as fake. Moreover, the task of comparing proposed news with the original news itself is a daunting task as it's highly subjective and opinionated. The World Wide Web contains data in diverse formats such as documents, videos, and audios. News published online in an unstructured format (such as news, articles, videos, and audios) is relatively difficult to detect and classify as this strictly requires human expertise.

So how will you detect the fake news? We will be evaluating multiple machine learning method to classify the news article scrapped/gathered from multiple source, as fake or real.

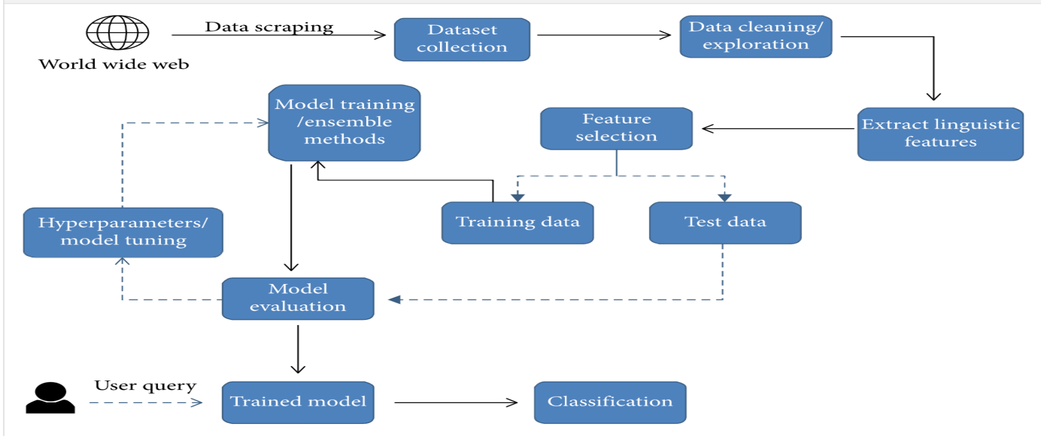
Data Collection

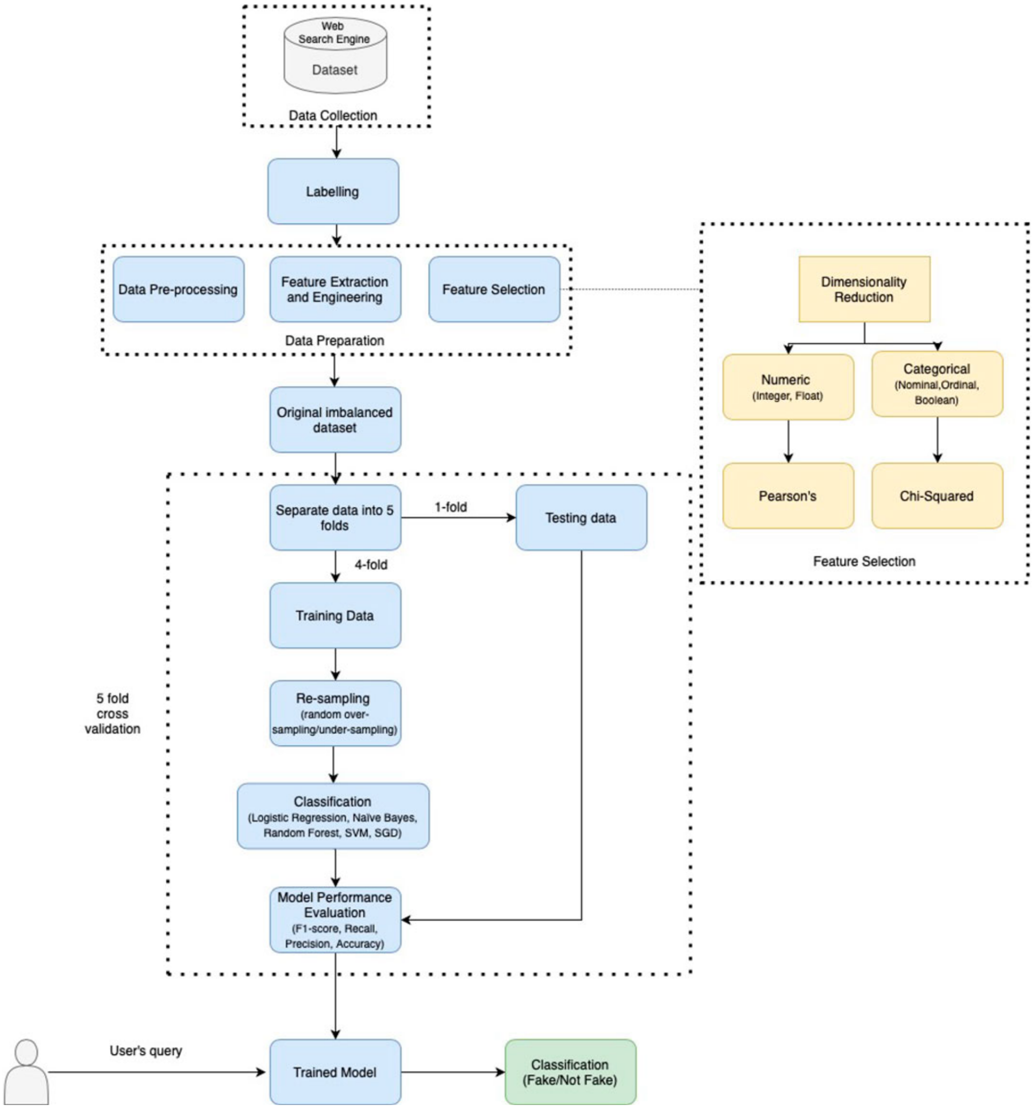
1. Kaggle fake news data set
   a. True.csv – 21192 real unique records
   b. Fake.csv – 22851 fake unique records
2. Research Articles
   a. 249 research articles
3. Data (Some news data)
   a. 72103 unique fake and real news
4. News data set
   a. 166355 unique fake and real news
5. Politifacts data set
   a. 625 real unique records
   b. 433 fake unique records
6. Train and test data set
   a. 146373 unique fake and real news

The final dataset consisted of a collection of approximately 430170 news (fake/misleading and trusted/"real"), gathered from different sources. The above datasets has text(news) and a column labels denoting whether the news is REAL or FAKE. The data is pretty much balanced.

Proposed workflow

Workflow for training algorithms and classification of news articles.



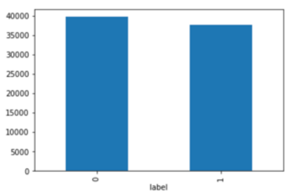

Web Search Engine
Dataset
Data Collection
Labelling
Data Pre-processing
Feature Extraction and Engineering
Feature Selection
Data Preparation
Dimensionality Reduction
Numeric (Integer, Float)
Categorical (Nominal,Ordinal, Boolean)
Pearson's
Chi-Squared
Feature Selection
Original imbalanced dataset
Separate data into 5 folds
1-fold
Testing data
4-fold
Training Data
Re-sampling (random over-sampling/under-sampling)
Classification (Logistic Regression, Naive Bayes, Random Forest, SVM, SGD)
Model Performance Evaluation (F1-score, Recall, Precision, Accuracy)
5 fold cross validation
User's query
Trained Model
Classification (Fake/Not Fake)

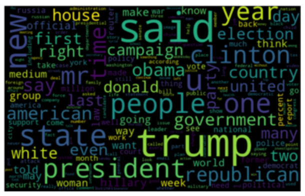
Data Pre-Processing and Feature Engineering

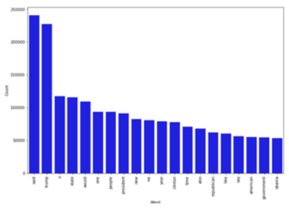
To observe the most meaningful context words and to improve the performance of the classifiers, in the data pre-processing stage, we removed all parts that were irrelevant, redundant, and not related to the content: punctuation (with a few exceptions of symbols, such as exclamation marks, question marks, and quotation marks) and extra delimiters; symbols; dashes from both titles and descriptions; and stop words. In the feature engineering stage, which typically includes feature creation, transformation, extraction, and selection, we used pre-training algorithms, such as bag-of-words (BoW) and term frequency–inverse document frequency (TF-IDF) for mapping cleaned texts (titles and descriptions) into numeric representations.

- Data cleaning
    - Clean empty and null values
    - Case Insensitive
    - Remove Stop words
    - Remove Punctuations
    - Lemmatization/Stemming
    - POS - parts-of-speech - TODO
- Feature Engineering/extraction
    - Word counts and cloud
    - Frequency Distributions
    - Relevancy
    - Sentiment Analysis : TODO

Visualization of combined data after clean up





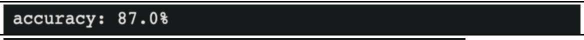
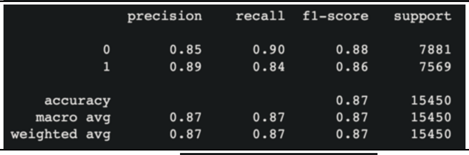
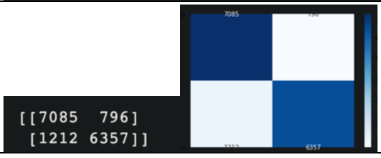


Algorithms

We used the following learning algorithms to evaluate the performance of fake news detection classifiers.

Logistic Regression

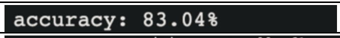
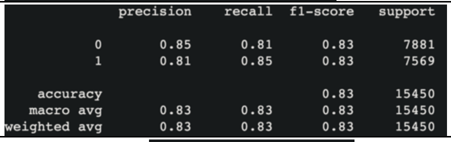
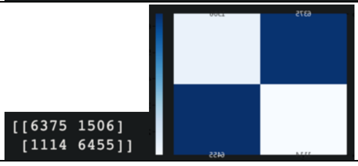
Logistic regression uses a sigmoid function to transform the output to a probability value; the objective is to minimize the cost function to achieve an optimal probability. As we are classifying text based on a wide feature set, with a binary output (true/false or Real article/fake article), a logistic regression (LR) model is used, since it provides the intuitive equation to classify problems into binary or multiple classes.

Accuracy	 `accuracy: 87.0%`
Classification Report	
Confusion Metrics and Heat map	 `[[7085  796]` ` [1212 6357]]`

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      7881
           1       0.89      0.84      0.86      7569

    accuracy                           0.87     15450
   macro avg       0.87      0.87      0.87     15450
weighted avg       0.87      0.87      0.87     15450
```

Decision Tree Classification

A decision tree is a graphical representation of all possible solutions to a decision based on certain conditions. On each step or node of a decision tree, used for classification, we try to form a condition on the features to separate all the labels or classes contained in the dataset to the fullest purity. Let's see how the idea works on Fake News

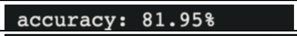
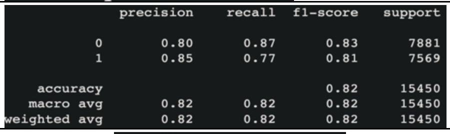
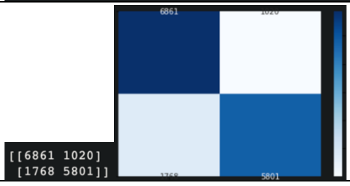
Accuracy	 accuracy: 83.04%
Classification Report	
Confusion Metrics and Heat map	 [[6375 1506] [1114 6455]]

```
             precision    recall  f1-score   support

           0       0.85      0.81      0.83      7881
           1       0.81      0.85      0.83      7569

    accuracy                           0.83     15450
   macro avg       0.83      0.83      0.83     15450
weighted avg       0.83      0.83      0.83     15450
```

```
[[6375 1506]
 [1114 6455]]
```

Random Forest (RF)

Random forest (RF) is an advanced form of decision trees (DT) which is also a supervised learning model. RF consists of large number of decision trees working individually to predict an outcome of a class where the final prediction is based on a class that received majority votes. The error rate is low in random forest as compared to other models, due to low correlation among trees.

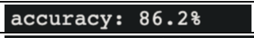
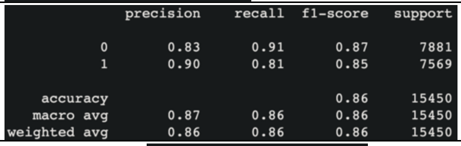
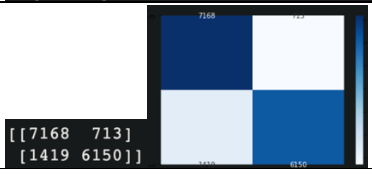
Accuracy	 accuracy: 81.95%
Classification Report	
Confusion Metrics and Heat map	 [[6861 1020] [1768 5801]]

```
             precision    recall  f1-score   support

           0       0.80      0.87      0.83      7881
           1       0.85      0.77      0.81      7569

    accuracy                           0.82     15450
   macro avg       0.82      0.82      0.82     15450
weighted avg       0.82      0.82      0.82     15450
```

```
[[6861 1020]
 [1768 5801]]
```

Stochastic Gradient Descent

SGDClassifier is a Linear classifiers (SVM, logistic regression, a.o.) with SGD training. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, see the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance. This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).

Accuracy	 accuracy: 86.2%
Classification Report	
Confusion Metrics and Heat map	 [[7168 713] [1419 6150]]

```
             precision    recall  f1-score   support

           0       0.83      0.91      0.87      7881
           1       0.90      0.81      0.85      7569

    accuracy                           0.86     15450
   macro avg       0.87      0.86      0.86     15450
weighted avg       0.86      0.86      0.86     15450
```

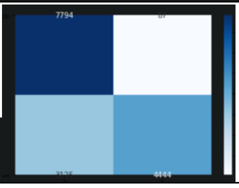
```
[[7168  713]
 [1419 6150]]
```

Boosting Ensemble Classifiers

Boosting is another widely used ensemble method to train weak models to become strong learners. For that purpose, a forest of randomized trees is trained, and the final prediction is based on the majority vote outcome from each tree. This method allows weak learners to correctly classify data points in an incremental approach that are usually misclassified. Initially equal weighted coefficients are used for all data points to classify a given problem. In the successive rounds, the weighted coefficients are decreased for data points that are correctly classified and are increased for data points that are misclassified. Each subsequent tree formed in each round learns to reduce the errors from the preceding round and to increase the overall accuracy by correctly classifying data points that were misclassified in previous rounds. One major problem with boosting ensemble is that it might overfit to the training data which may lead to incorrect predictions for unseen instances.

GradientBoostingClassifier

Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model to minimize the error.

Accuracy	`accuracy: 79.21%`
Classification Report	`precision recall f1-score support` `0 0.71 0.99 0.83 7881` `1 0.98 0.59 0.73 7569` `accuracy 0.79 15450` `macro avg 0.85 0.79 0.78 15450` `weighted avg 0.84 0.79 0.78 15450`
Confusion Metrics and Heat map	`[[7794 87]` `[3125 4444]]` 

e**X**treme **G**radient **B**oosting : TODO

The XGBoost stands for eXtreme Gradient Boosting, which is a boosting algorithm based on gradient boosted decision trees algorithm. XGBoost applies a better regularization technique to reduce overfitting, and it is one of the differences from the gradient boosting. The 'xgboost' is an open-source library that provides machine learning algorithms under the gradient boosting methods. The xgboost.XGBClassifier is a scikit-learn API compatible class for classification.

Multinomial Naive Bayes Classifier

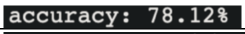
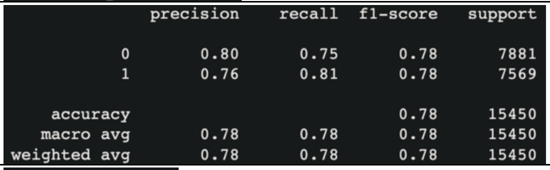
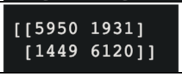
Multinomial Naive Bayes algorithm is a probabilistic learning method that is mostly used in Natural Language Processing (NLP). The algorithm is based on the Bayes theorem and predicts the tag of a text such as a piece of email or newspaper article. It calculates the probability of each tag for a given sample and then gives the tag with the highest probability as output.
Naive Bayes classifier is a collection of many algorithms where all the algorithms share one common principle, and that is each feature being classified is not related to any other feature. The presence or absence of a feature does not affect the presence or absence of the other feature.

Accuracy	`accuracy: 80.14%`
Classification Report	`precision recall f1-score support` `0 0.77 0.87 0.82 7881` `1 0.85 0.73 0.78 7569` `accuracy 0.80 15450` `macro avg 0.81 0.80 0.80 15450` `weighted avg 0.81 0.80 0.80 15450`
Confusion Metrics and Heat map	`[[6893 988]` `[2081 5488]]`

Bernoulli Naive Bayes Classifier

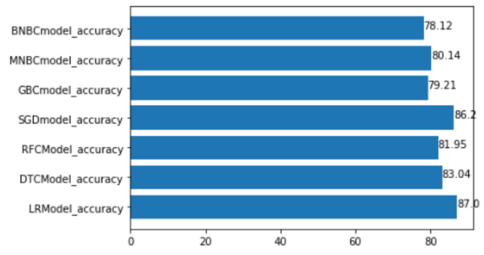
Bernoulli Naive Bayes is a variant of Naive Bayes. So, let us first talk about Naive Bayes in brief. Naive Bayes is a classification algorithm of Machine Learning based on Bayes theorem which gives the likelihood of occurrence of the event. Naive Bayes classifier is a probabilistic classifier which means that given an input, it predicts the probability of the input being classified for all the classes. It is also called conditional probability.
Two important assumptions made for Naive Bayes Classifier:

- First is that the attributes are independent of each other and does not affect each others performance, this is the reason it is called 'naive'.
- Second is that all the features are given equal importance. For example if there are 10 features, knowing only 5 features will not give us accurate outcome. All features are necessary to predict outcome and are given equal importance.

Accuracy	 `accuracy: 78.12%`
Classification Report	
Confusion Metrics and Heat map	 `[[5950 1931]` ` [1449 6120]]`

```
              precision    recall  f1-score   support

           0       0.80      0.75      0.78      7881
           1       0.76      0.81      0.78      7569

    accuracy                           0.78     15450
   macro avg       0.78      0.78      0.78     15450
weighted avg       0.78      0.78      0.78     15450
```

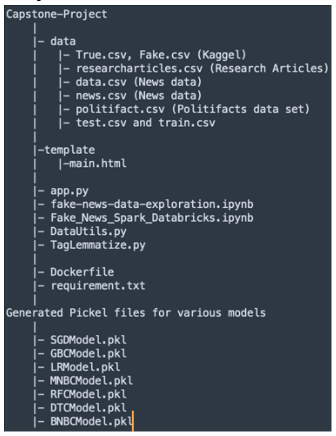
Model Accuracy comparison



Deployment.

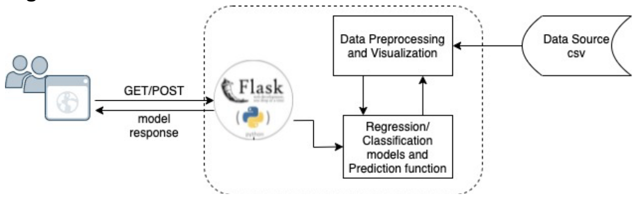
Using jupyter notebook we explored/analyzed data and apply various regression and classification algorithms on it, we finally know by accuracy score which model is suitable and has highest accuracy. Now we need to create a web based application where user should be able to verify a new is fake using our model. For this we will create a web based application bundling our model pkl file with UI html and will deploy it on a flask server.

Project structure



```
Capstone-Project
    |
    |- data
    |   |- True.csv, Fake.csv (Kaggel)
    |   |- researcharticles.csv (Research Articles)
    |   |- data.csv (News data)
    |   |- news.csv (News data)
    |   |- politifact.csv (Politifacts data set)
    |   |- test.csv and train.csv
    |
    |-template
    |   |-main.html
    |
    |- app.py
    |- fake-news-data-exploration.ipynb
    |- Fake_News_Spark_Databricks.ipynb
    |- DataUtils.py
    |- TagLemmatize.py
    |
    |- Dockerfile
    |- requirement.txt
    |
Generated Pickel files for various models
    |
    |- SGDModel.pkl
    |- GBCModel.pkl
    |- LRModel.pkl
    |- MNBCModel.pkl
    |- RFCModel.pkl
    |- DTCModel.pkl
    |- BNBCModel.pkl
```

High Level flow

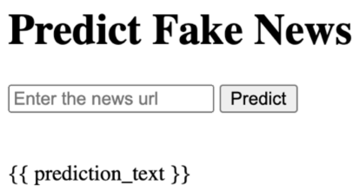

GET/POST
model response
Flask
Data Preprocessing and Visualization
Data Source csv
Regression/ Classification models and Prediction function

Local deployment and testing

Ensure that you are in the project home directory.

1. Create the machine learning model by running below command - python fake_news_detection.py
2. This would create a serialized version of our model into a file model.pkl
3. Run app.py using below command to start Flask API
   a. python app.py

By default, flask will run on http://127.0.0.1:5000



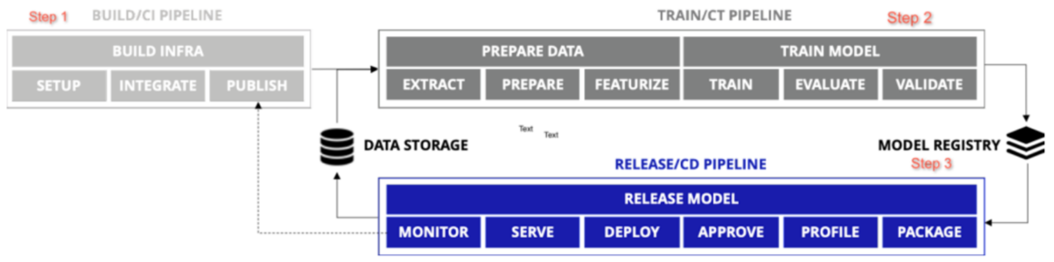
Predict Fake News

{{ prediction_text }}

Cloud deployment

Once the Flask application is properly tested locally, we can now proceed to create a Docker image and deploy it on the cloud. But before that, we need to answer some basic question

- Major components of the system? What are the inputs and outputs?



- What cloud solution we will use? Google cloud
- Where and how will the data be stored?

  Data will be stored on Google Drive and in csv format.
- How will data get from one component of the system to another?

  A script will download the data from the various links and store it on a Google Drive

- What is the lifecycle of your ML/DL model? How frequently do you need to retrain your model? Is it at fixed intervals when you collect a certain amount of new data or when some other conditions are met?

  We will periodically measure the Model drift and based on the accuracy result will train the Model with a new set of data. So conditional based.

  What is Model Drift : Model's predictive performance degrading over time due to a change in the environment that violates the model's assumptions. The most accurate way to detect model drift is by comparing the predicted values from a given machine learning model to the actual values. The accuracy of a model worsens as the predicted values deviate farther and farther from the actual values.Once a model is deployed, we will log and monitor two types of data: model performance metrics and model quality metrics.

Model performance metrics refer to technical aspects of the model, such as inference latency or memory footprint. These metrics can be logged and monitored easily when a model is deployed on Databricks. Model quality metrics depend on the actual labels. Once the labels are logged, you can compare predicted and actual labels to compute quality metrics and detect drift in the predictive quality of the model.

- What kind of data do you need for retraining? How will you store and manage it?
  We will need the latest Fake and Real news data to train the model, we will be logging and collecting the question and its prediction with accuracy score. We will use this data to train the model. Apart from user input data we will be using new sets of fake and real news.

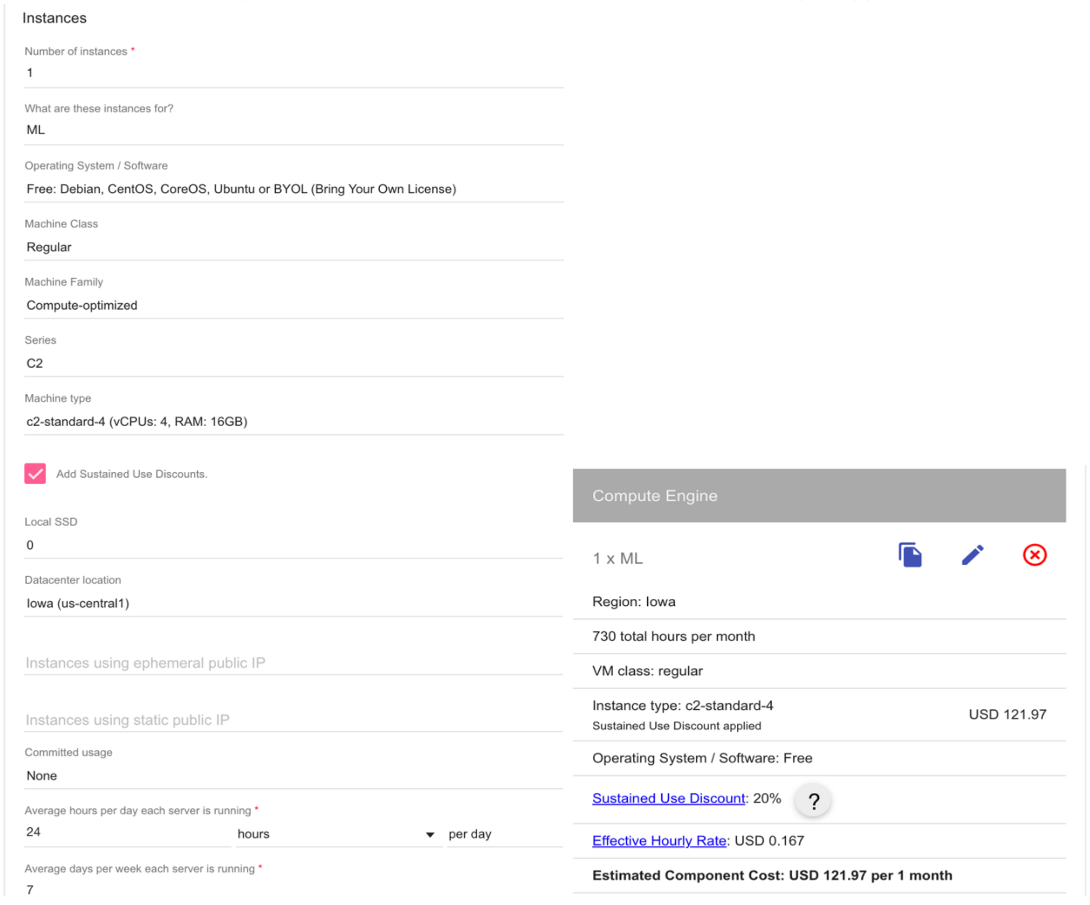
- How do you know if the retrained model is good enough to deploy? Based on the accuracy score

- How will the retrained model be deployed?
  Even though we have retrained our model and the performance metrics look great, there's still a big risk of the updated model performing poorer than the previous model even after retraining. We will leave the old model for a specific window or until the model has served a particular number of requests. Then we can serve the new model the same data and get the predictions. This way, we can compare both models' predictions to understand which of them is performing better. If we're satisfied with the new model's performance, we can start using it confidently. This is a typical example of A/B testing, this would ensure that our model is validated on upstream data. We need to automate how to deploy our models after retraining. We can deploy our machine learning models to the production environment using kubernetes.

- How will the system be monitored? How will you debug it if there are problems? https://cloud.google.com/ai-platform/prediction/docs/monitor-prediction

- How will your system respond to unexpected errors or outages? Notifications via email or text messages.

- What is the estimated implementation cost in terms of resources, time, and money as applicable?



Instances

Number of instances *	1
What are these instances for?	ML
Operating System / Software	Free: Debian, CentOS, CoreOS, Ubuntu or BYOL (Bring Your Own License)
Machine Class	Regular
Machine Family	Compute-optimized
Series	C2
Machine type	c2-standard-4 (vCPUs: 4, RAM: 16GB)
Add Sustained Use Discounts.	✓
Local SSD	0
Datacenter location	Iowa (us-central1)
Instances using ephemeral public IP	
Instances using static public IP	
Committed usage	None
Average hours per day each server is running *	24 hours per day
Average days per week each server is running *	7

Compute Engine

1 x ML

Region: Iowa

730 total hours per month

VM class: regular

Instance type: c2-standard-4 USD 121.97
Sustained Use Discount applied

Operating System / Software: Free

Sustained Use Discount: 20%

Effective Hourly Rate: USD 0.167

Estimated Component Cost: USD 121.97 per 1 month

1. Package with Docker
   We will create our **image**, before deploying it to Google cloud. 3 steps to create an image
   - Get Docker
   - Setup our Dockerfile
   - Build our image

2. Creating a requirement.txt
   flask
   flask-restful
   gunicorn

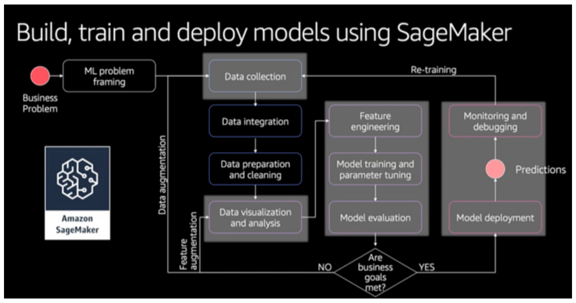
3. Creating a Dockerfile
```
FROM python:3.7-slim-buster
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 app:app
```

4. Build a docker image
   docker build -t fake-news-api .

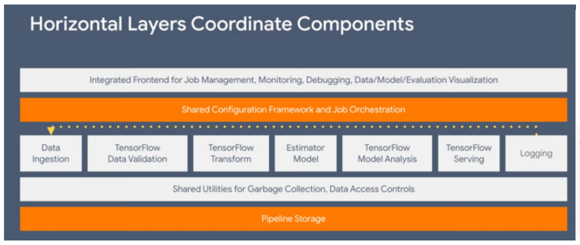
5. Deploy to Google cloud
   The below link has the detail approach which I follow to deploy my app to Google cloud
   https://towardsdatascience.com/deploy-apis-with-python-and-docker-4ec5e7986224

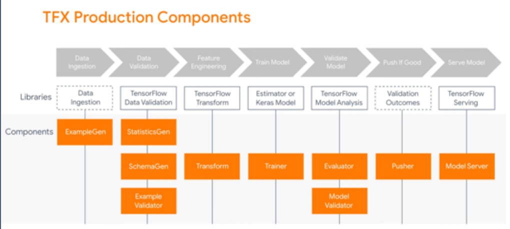
Cloud deployment options

Amazon Sagemaker (Build, train, and deploy a machine learning model with Amazon SageMaker)
https://aws.amazon.com/getting-started/hands-on/build-train-deploy-machine-learning-model-sagemaker/



Google TFX
TFX is a Google-production-scale machine learning (ML) platform based on TensorFlow. It provides a configuration framework and shared libraries to integrate common components needed to define, launch, and monitor your machine learning system.
https://cloud.google.com/ai-platform/prediction/docs/deploying-models
https://www.tensorflow.org/tfx/guide


Horizontal Layers Coordinate Components
Integrated Frontend for Job Management, Monitoring, Debugging, Data/Model/Evaluation Visualization
Shared Configuration Framework and Job Orchestration
Data Ingestion
TensorFlow Data Validation
TensorFlow Transform
Estimator Model
TensorFlow Model Analysis
TensorFlow Serving
Logging
Shared Utilities for Garbage Collection, Data Access Controls
Pipeline Storage


TFX Production Components
Data Ingestion
Data Validation
Feature Engineering
Train Model
Validate Model
Push If Good
Serve Model
Libraries
Data Ingestion
TensorFlow Data Validation
TensorFlow Transform
Estimator or Keras Model
TensorFlow Model Analysis
Validation Outcomes
TensorFlow Serving
Components
ExampleGen
StatisticsGen
SchemaGen
Example Validator
Transform
Trainer
Evaluator
Model Validator
Pusher
Model Server