Automatic Labeling of News Articles

A Natural Language Processing Case Study

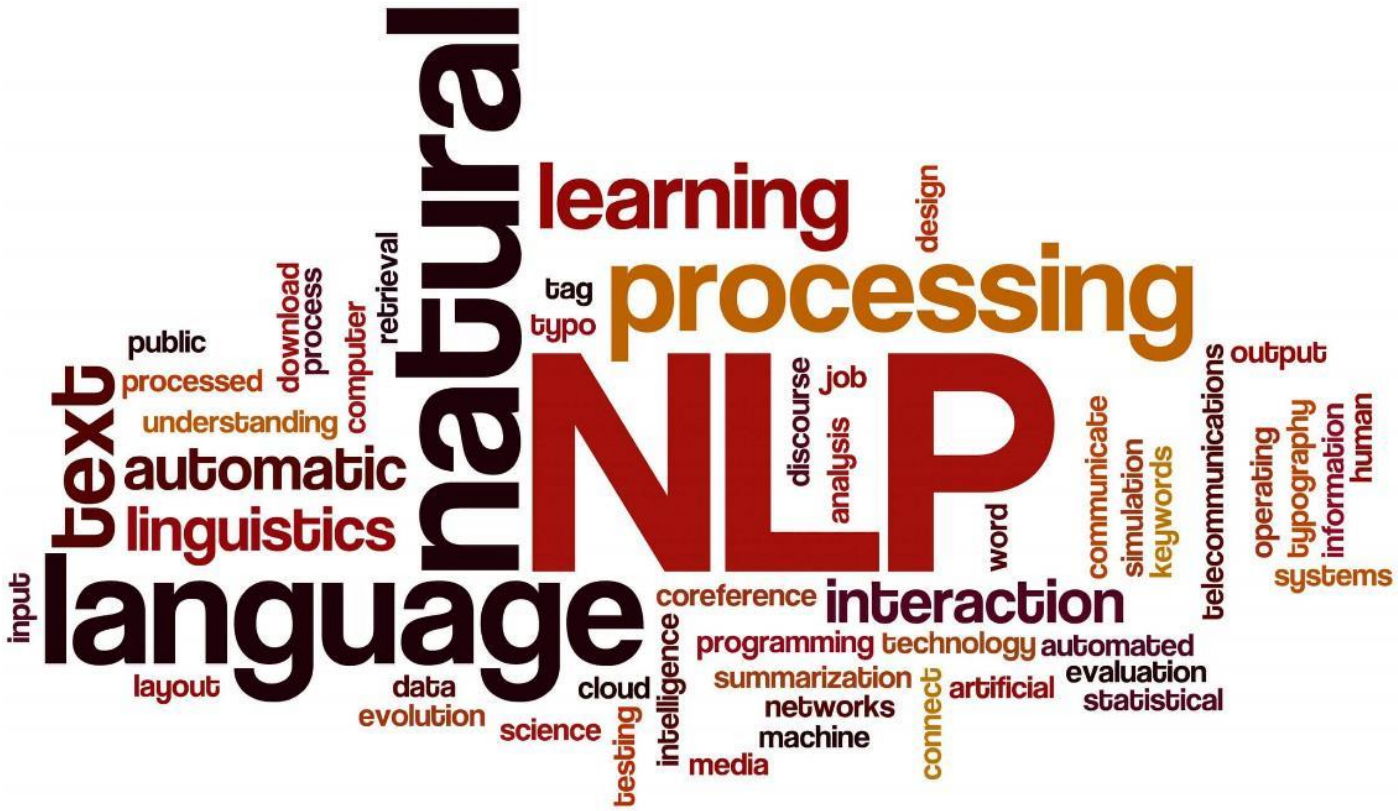
Rasoul Asaee

Motivation

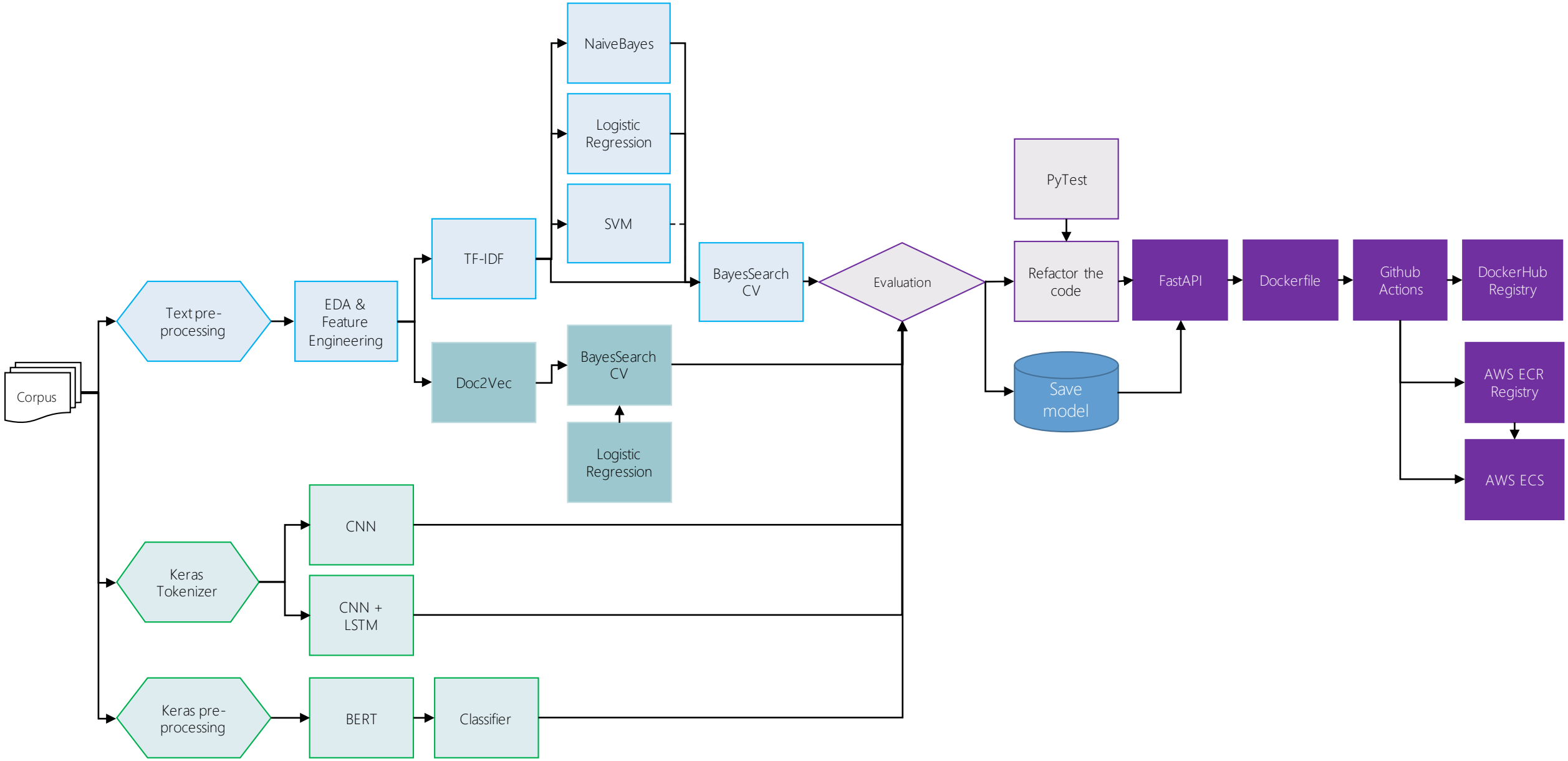
- Text is one of the fastest growing data types
- Generally a subject matter expert label the text data
- People use cellphones to quickly access the latest news
- Automatic labeling enable:
  - Journalists to further promote their articles
  - Audience to quickly access the content

Methodology

- Large corpus of news articles
- Text pre-processing
- Natural language processing
- Model evaluation
- Containerization and deployment
- Maintenance



Methodology



Data

- Covid-19 Public Media Dataset by Anacode
- Non-medical impacts of Covid-19, especially in terms of the social, political, economic and technological dimensions
- Over 350,000 online articles with full texts

Data – collection

- Scraped from online media in the time span January 1 - December 31, 2020
- Data collection strategy
  - Articles from English-language high-impact blogs and news websites
  - Scraping the 65 websites continuously every day,
  - Filtered them on Covid19-related keywords to keep only relevant articles in the dataset

Data – attributes

- Author
- Date
- Domain
- Title
- URL
- Content
- Topic_area

Text pre-processing

- Noise Cleaning: remove html tags, accented text, and special characters
- Spell Checking: correct misspelled words, and remove meaningless string
- Contraction Mapping
- Lemmatization
- ‘Stop Words’ Identification
- Case Conversion

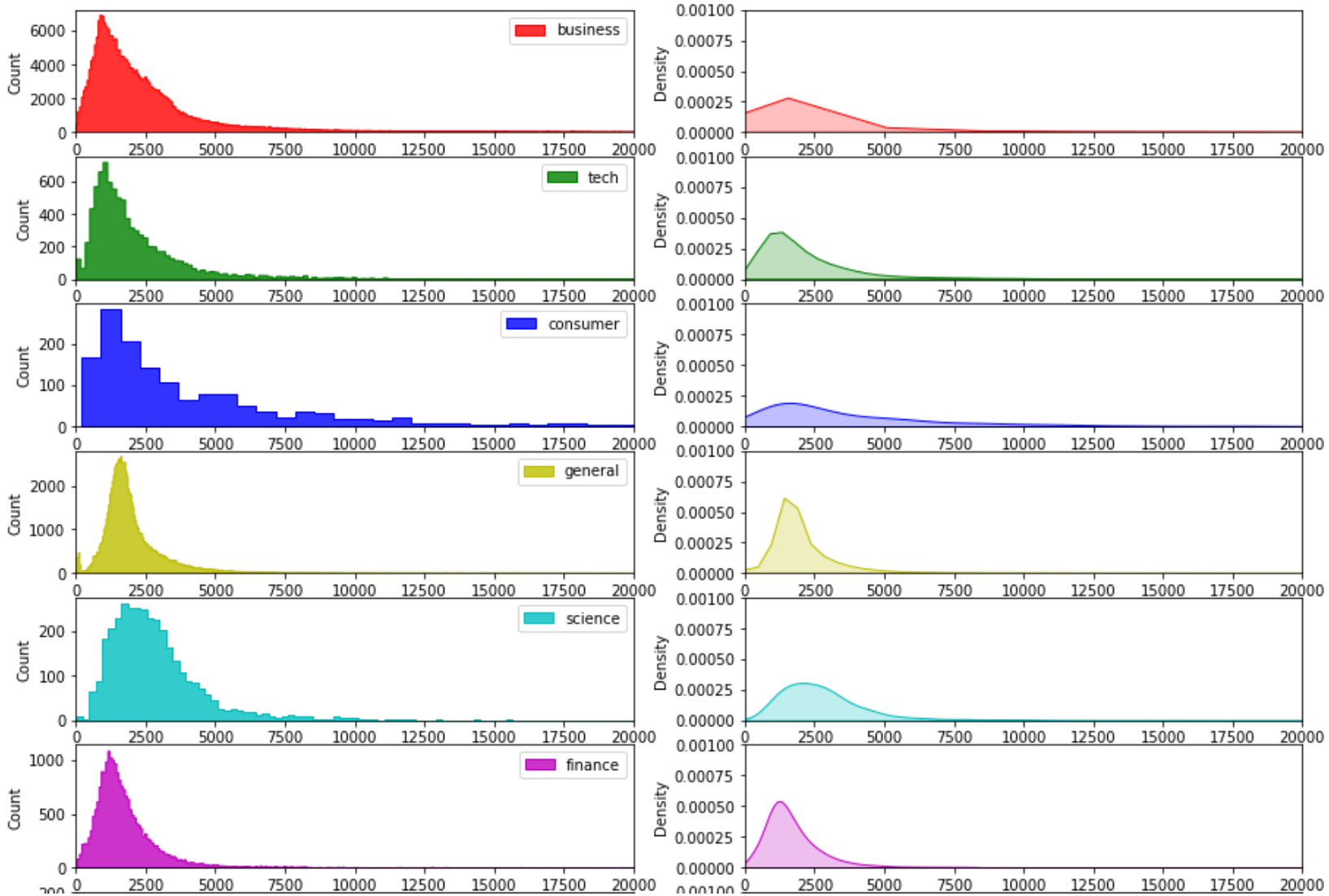
EDA and Feature Engineering – length analysis

- Word count: counts the number of tokens in the text (separated by a space)
- Character count: sum the number of characters of each token
- Sentence count: count the number of sentences (separated by a period)
- Average word length: sum of words length divided by the number of words (character count/word count)
- Average sentence length: sum of sentences length divided by the number of sentences (word count/sentence count)

	title	topic_area	content	word_count	char_count	avg_word_length
0	Three Industrial Giants You Should Own In 2020	business	end year around corner past time think come ea...	564	3156	5.595745
1	Labor Stocks Are Going To Break Out In 2020	business	labor market one closely watch segment economy...	623	3539	5.680578
2	Tesla (TSLA) Breaks Shipment Record, Beats Est...	business	could forgive might think little big pile pie ...	603	3424	5.678275
3	On the road to AI adoption, execs grapple with...	tech	kick ai item watch competition agenda adoption...	160	965	6.031250
4	Red Carpet Sustainability After Coronavirus Sh...	consumer	pandemic life return normal celebrity walk red...	738	4144	5.615176

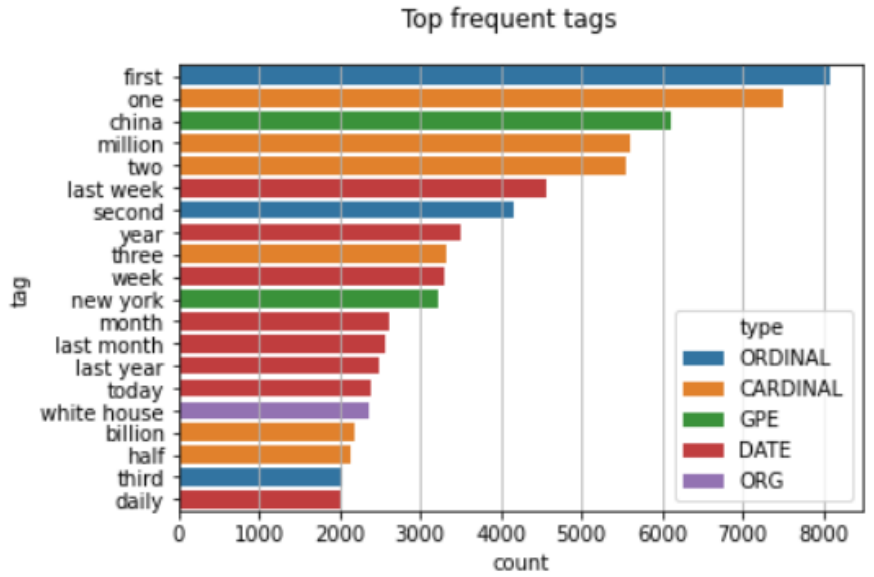
EDA and Feature Engineering – length analysis

- What's the distribution of those new variables with respect to the target?
  - If the distributions are different with regard to the label then the new variable is predictive
  - The upper limit for the length of articles



EDA and Feature Engineering – NER

- Named-Entity Recognition (NER) is the process to tag named entities mentioned in unstructured text with pre-defined categories
- One of the best open source NER tools is SpaCy

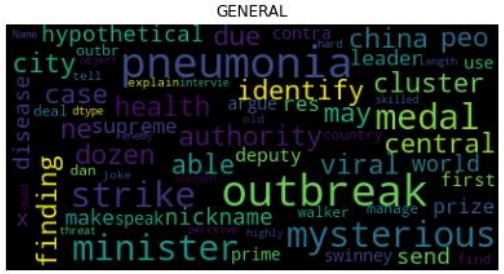


PERSON: People, including fictional.
NORP: Nationalities or religious or political groups.
FAC: Buildings, airports, highways, bridges, etc.
ORG: Companies, agencies, institutions, etc.
GPE: Countries, cities, states.
LOC: Non-GPE locations, mountain ranges, bodies of water.
PRODUCT: Objects, vehicles, foods, etc. (Not services.)
EVENT: Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART: Titles of books, songs, etc.

LAW: Named documents made into laws.
LANGUAGE: Any named language.
DATE: Absolute or relative dates or periods.
TIME: Times smaller than a day.
PERCENT: Percentage, including "%".
MONEY: Monetary values, including unit.
QUANTITY: Measurements, as of weight or distance.
ORDINAL: "first", "second", etc.
CARDINAL: Numerals that do not fall under another type.

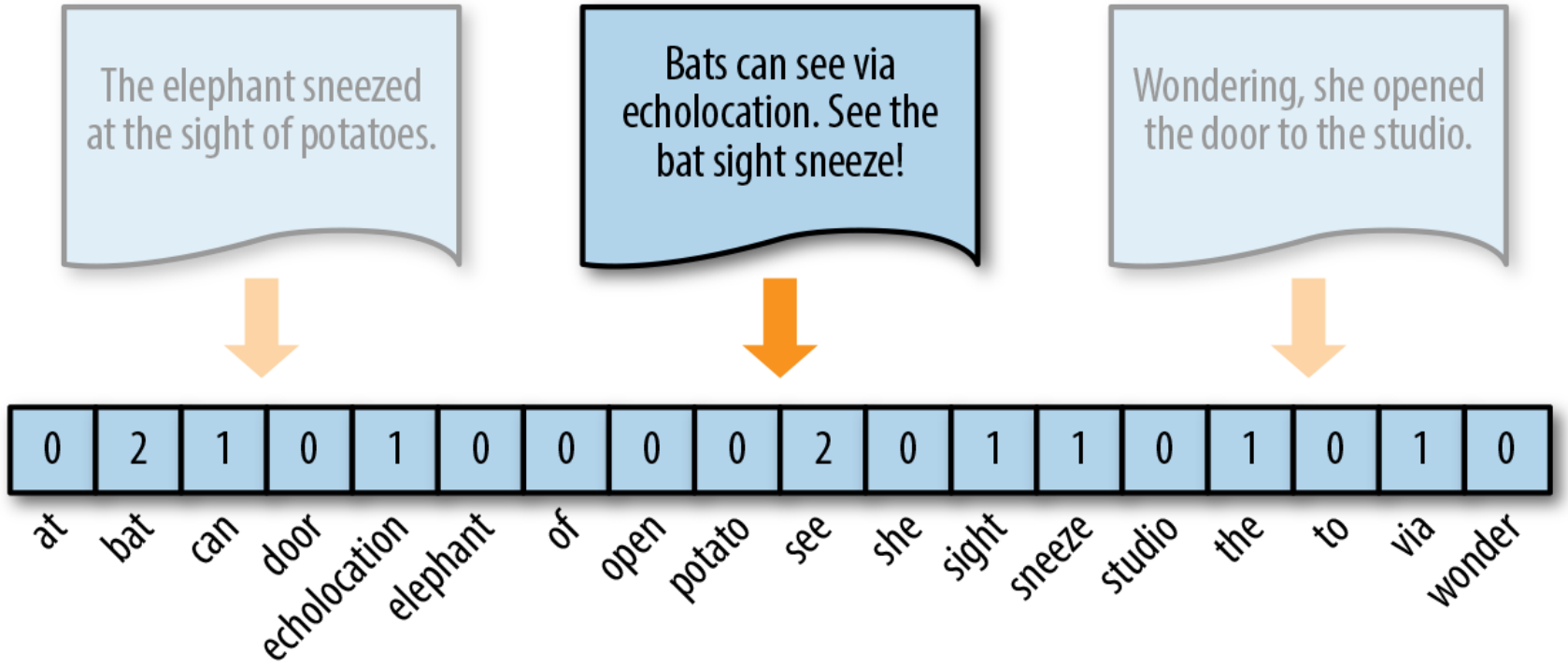
EDA and Feature Engineering – word cloud

- A nice way to visualize the text information is with a word cloud
- The frequency of each tag is shown with:
  - font size
  - color
- Using the wordcloud module



Text Vectorization

- Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features.



Reference: https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html

Text Vectorization

Vectorization	Function	Good For	Considerations
Frequency	Counts term frequencies	Bayesian models	Most frequent words not always most informative
One-Hot Encoding	Binarizes term occurrence (0, 1)	Neural networks	All words equidistant, so normalization extra important
TF–IDF	Normalizes term frequencies across documents	General purpose	Moderately frequent terms may not be representative of document topics
Distributed Representations	Context-based, continuous term similarity encoding	Modeling more complex relationships	Performance intensive; difficult to scale without additional tools (e.g., Tensorflow)

Reference: https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html

Text Vectorization – TF-IDF

- Normalizes the frequency of tokens in a document with respect to the rest of the corpus.
- Generally both the term frequency and inverse document frequency are scaled logarithmically to prevent bias of longer documents or terms that appear much more frequently relative to other terms

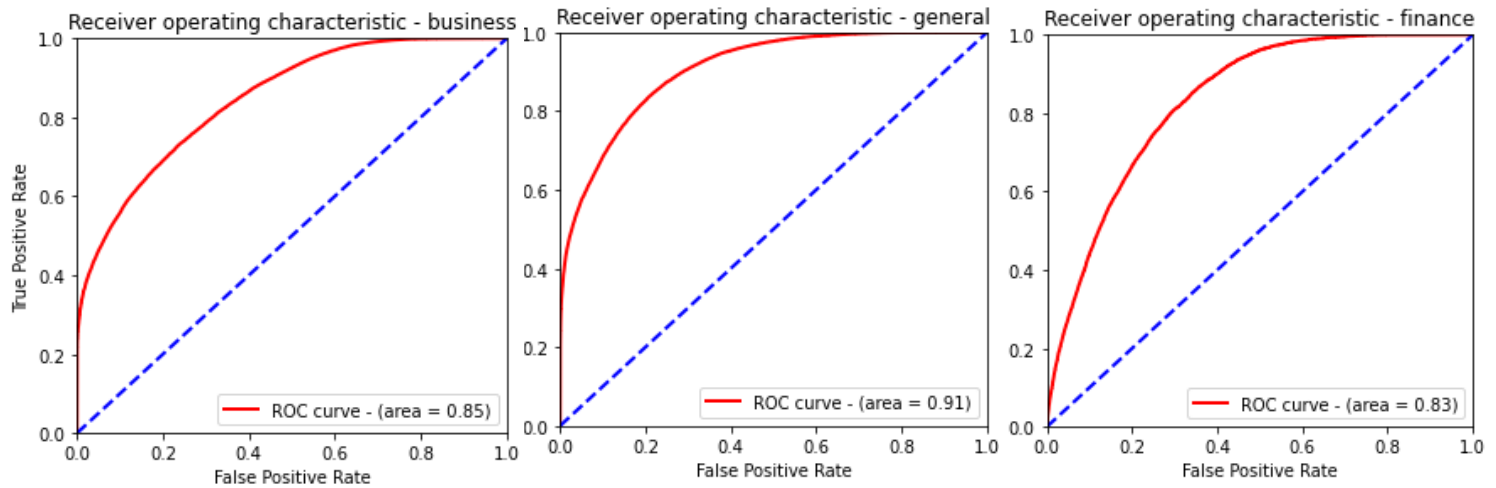
	title	topic_area	ability	able	accelerate	access	accord	accordance	account	accounting	...	worker	world	worth	would	write	year	yet	yield	york	young
0	Three Industrial Giants You Should Own In 2020	business	0.00	0.00	0.0	0.0	0.00	0.0	0.0	0.0	...	0.00	0.00	0.0	0.03	0.02	0.12	0.03	0.11	0.00	0.0
1	Labor Stocks Are Going To Break Out In 2020	business	0.03	0.00	0.0	0.0	0.04	0.0	0.0	0.0	...	0.06	0.00	0.0	0.02	0.03	0.17	0.00	0.12	0.00	0.0
2	Tesla (TSLA) Breaks Shipment Record, Beats Est...	business	0.00	0.02	0.0	0.0	0.02	0.0	0.0	0.0	...	0.00	0.00	0.0	0.11	0.02	0.05	0.08	0.00	0.00	0.0
3	On the road to AI adoption, execs grapple with...	tech	0.00	0.00	0.0	0.0	0.00	0.0	0.0	0.0	...	0.09	0.06	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.0
4	Red Carpet Sustainability After Coronavirus Sh...	consumer	0.00	0.00	0.0	0.0	0.03	0.0	0.0	0.0	...	0.03	0.05	0.0	0.04	0.00	0.05	0.10	0.00	0.03	0.0

Classification - NaiveBayes

- Since the number of articles per topic area are not balanced, the tags with low frequency were not included in the prototype.
- The data is divided to %70 for train and %30 for test
- The multinomial Naive Bayes classifier is used.
- The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

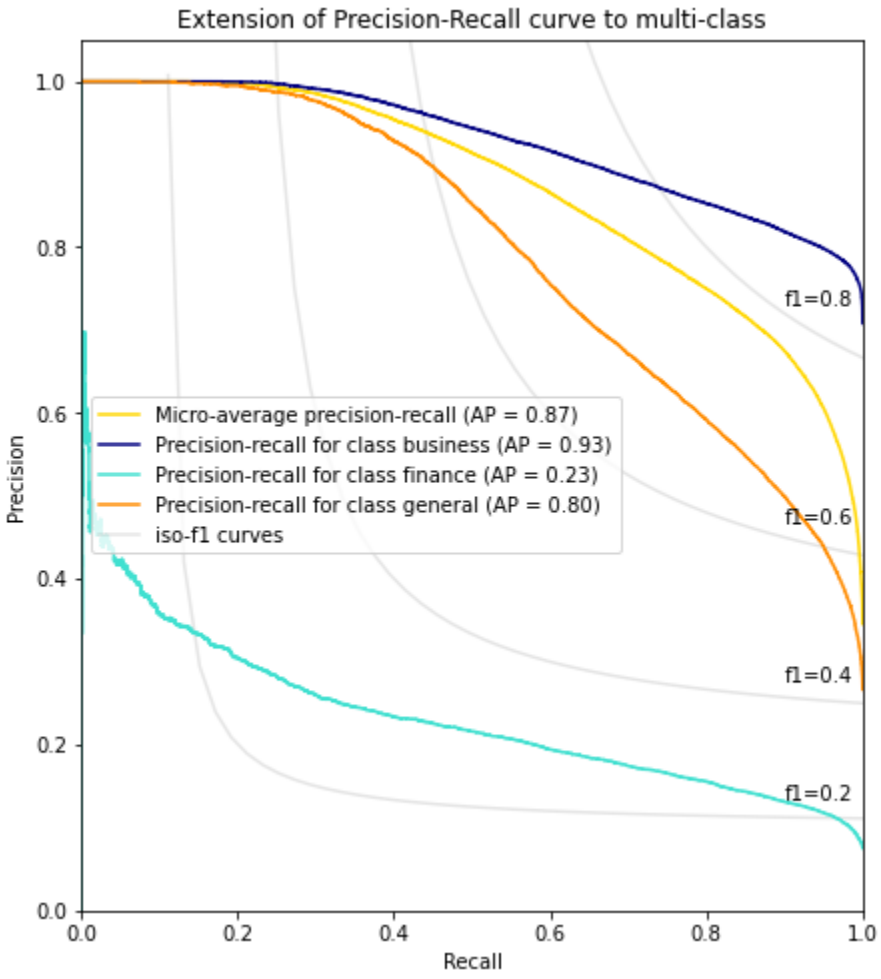
Label	Count
business	244422
general	84921
finance	22386
tech	8676
science	3478
consumer	1451
healthcare	1340
automotive	925
environment	777
construction	305
ai	239

- ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis.
- This means that the top left corner of the plot is the "ideal" point - a false positive rate of zero, and a true positive rate of one.
- A larger area under the curve (AUC) is usually better.



Classification - NaiveBayes

- The results indicate that the model struggles to classify the under-represented class.
- Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples.
- The recall score of 0.00 for finance class indicates that almost none of the finance articles could be correctly classified.



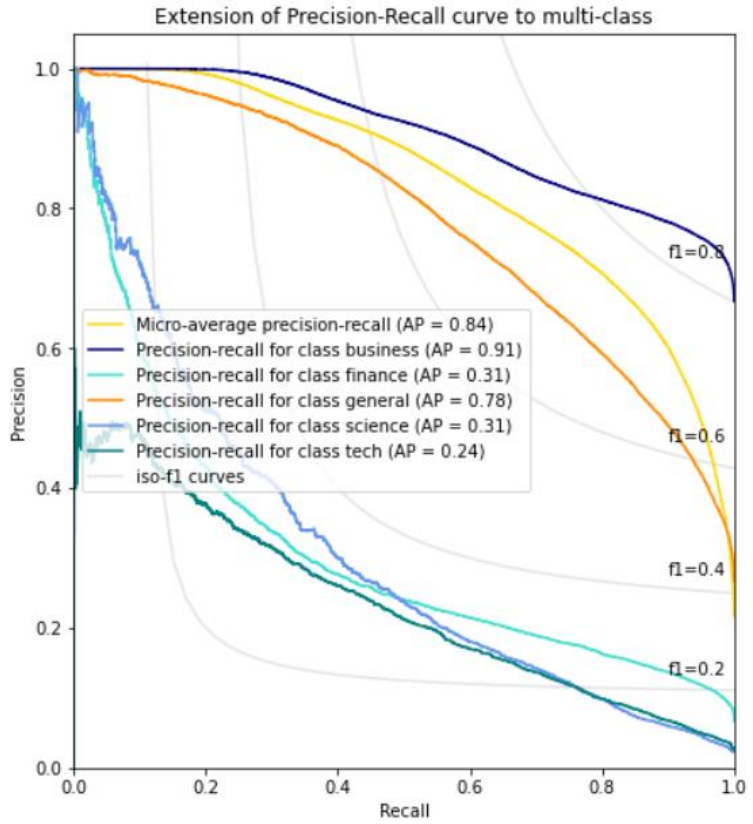
	precision	recall	f1-score	support
business	0.84	0.83	0.84	73389
finance	0.67	0.00	0.01	6628
general	0.61	0.78	0.68	25502
accuracy			0.77	105519
macro avg	0.70	0.54	0.51	105519
weighted avg	0.77	0.77	0.75	105519

Classification - NaiveBayes

- Hyper-parameter Tuning with BayesSearchCV
  - In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings (n_iter) is sampled from the specified distributions.
  - alpha: Additive (Laplace) smoothing parameter (0 for no smoothing).
- Since the prototype analysis indicated that the models had difficulty with under-represented classes, the topic areas with low samples were merged to create fewer classes with higher number of samples.
- A wide range of values for vectorization and NaiveBayes model were used, the model struggles to classify the classes with lower number of data points. It is proven that the low performance of NaiveBayes in this project is associated with the imbalance in the dataset.

```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB())
])
parameters:
{'clf__alpha': (0.0001, 1.0, 'log-uniform'),
 'tfidf__use_idf': (True, False),
 'vect__max_df': (0.5, 0.75, 1.0),
 'vect__max_features': [10000, 30000, None]}
```

	precision	recall	f1-score	support
business	0.80	0.85	0.82	73696
finance	1.00	0.00	0.00	6716
general	0.62	0.77	0.69	25912
science	0.90	0.01	0.01	1678
tech	0.00	0.00	0.00	2674
accuracy			0.75	110676
macro avg	0.66	0.32	0.30	110676
weighted avg	0.75	0.75	0.77	110676

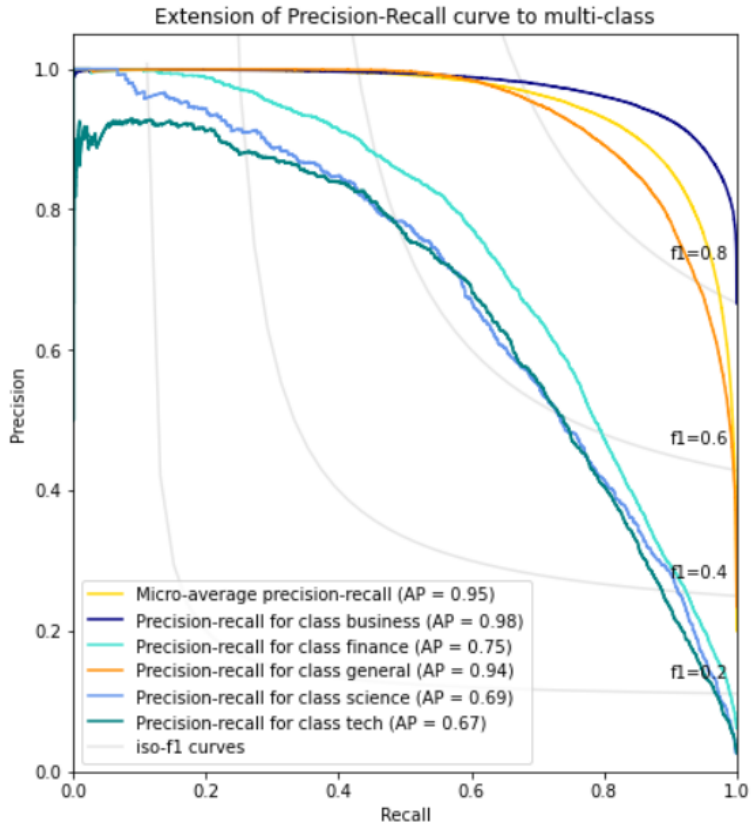


Classification – Logistic Regression

- Hyper-parameter Tuning with BayesSearchCV
  - Regularization with C: Inverse of regularization strength; smaller values specify stronger regularization.
- Liblinear solver was used for the logistic regression to avoid an error associated with the Scikit-Learn trying to decode an already decoded string.
- Observations
  - The recall score for the "finance" class has improved, which means the model could correctly classify over %50 third of finance articles.
  - The f1-score for every labels have improved as well.
  - The precision and recall score for the artificially formed classes are comparable with the scores of the finance class.
  - I believe the current results are the among the best scores that can be achieved with classical ML algorithms for the existing dataset.

```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', LogisticRegression(solver='liblinear')
)
])
parameters:
{'clf__C': (1e-05, 100.0, 'log-uniform'),
 'clf__dual': (True, False),
 'clf__max_iter': [100, 110, 120, 130, 140],
 'tfidf__use_idf': (True, False),
 'vect__max_df': (0.5, 0.75, 1.0),
 'vect__max_features': [10000, 30000, None]}
```

	precision	recall	f1-score	support
business	0.89	0.95	0.92	73696
finance	0.80	0.55	0.65	6716
general	0.87	0.83	0.85	25912
science	0.74	0.51	0.60	1678
tech	0.75	0.52	0.61	2674
accuracy			0.88	110676
macro avg	0.81	0.67	0.73	110676
weighted avg	0.87	0.88	0.87	110676



Classification - Support Vector Machine

- SVC and NuSVC implement the "one-versus-one" approach for multi-class classification.
- In total, n_classes * (n_classes - 1) / 2 classifiers are constructed and each one trains data from two classes.
- The SVM had a relatively high performance for the classification of the articles with the subset of the data.
- While the results are encouraging the model too a long time for the training phase. Since, I was working with a subset of the articles and feature space, it would be very challenging to extend this method to the whole dataset.

	precision	recall	f1-score	support
business	0.90	0.97	0.94	73389
finance	0.91	0.50	0.64	6628
general	0.92	0.81	0.86	25502
accuracy			0.90	105519
macro avg	0.91	0.76	0.81	105519
weighted avg	0.91	0.90	0.90	105519

Text Vectorization – distributed representation

- Frequency models:
  - Encode similarities between documents in the context of that same vector space
  - Fail to compare documents that don't share items even if they are semantically similar

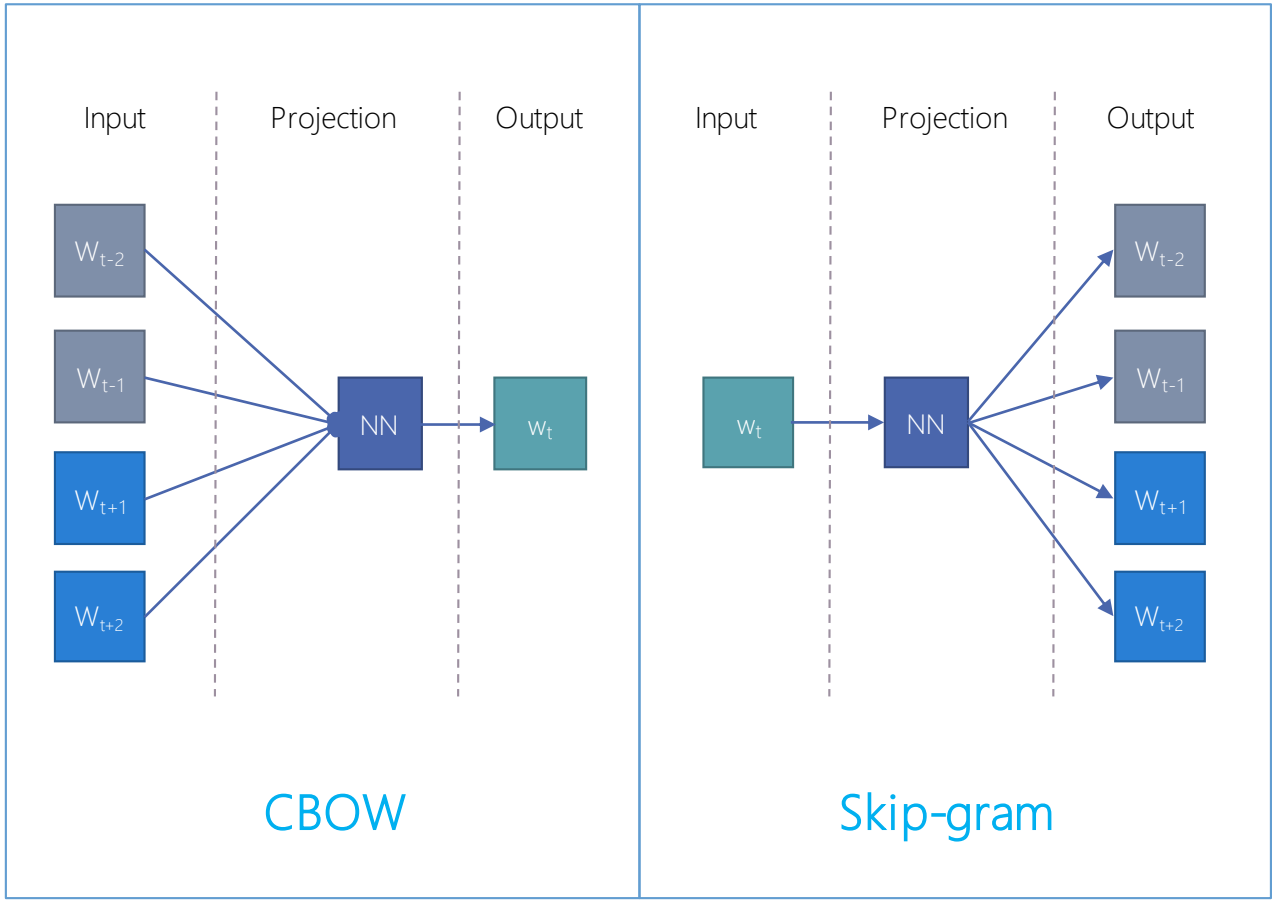
- Distributed representation:
  - Encode text along a continuous scale with a distributed representation
  - The document is represented in a feature space that has been embedded to represent word similarity.

Text Vectorization – Word2Vec

Project words into a latent space

- Continuous Bag of Words (CBOW)
  - The distributed representations of context (or surrounding words) are combined to predict the word in the middle.

- Skip-gram
  - The distributed representation of the input word is used to predict the context

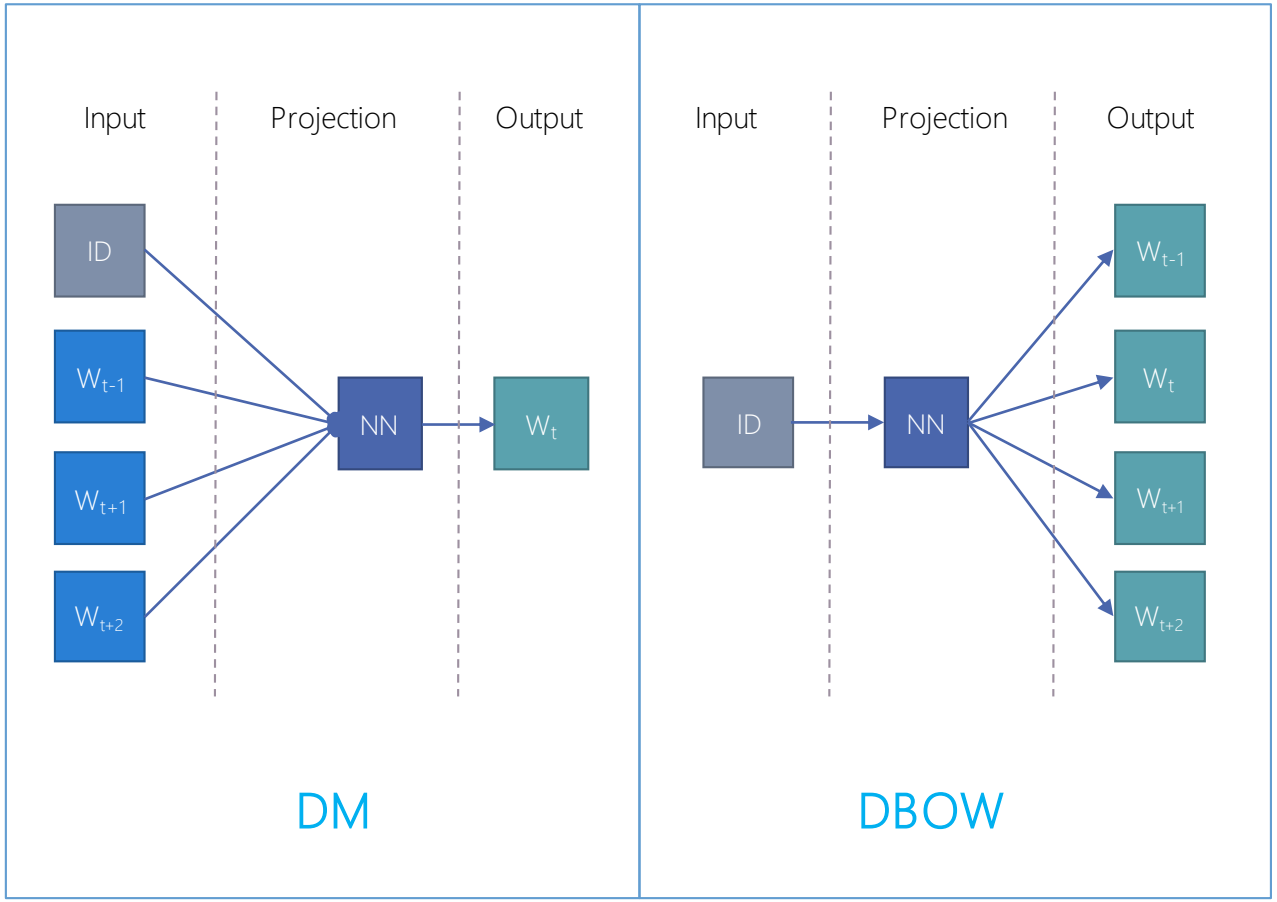


Text Vectorization – Doc2Vec

Project a document into a latent space

- Distributed Memory
  - Randomly sample consecutive words from a paragraph and predict a center word from the randomly sampled set of words by taking as input — the context words and a paragraph id

- Distributed Bag of Words (DBOW)
  - Ignores the context words in the input, but force the model to predict words randomly sampled from the paragraph in the output.



Text Vectorization – Doc2Vec

- Splitting the data in train and test groups:
  - Used the Scikit-Learn train-test split model with a ratio of 30% test data.
- Tokenizing and tagging the articles:
  - Gensim has a tokenizer to which create word tokens.
  - Article index as the tag for each entry. Tags are unique IDs for each article used to look-up the learned vectors after training.
- Initializing the model:
  - Training a Doc2Vec model is a memory intensive process. I had to adopt measures to fit the data in memory. A tool for managing the size of the model is the *vector_size* which indicates the dimensionality of the feature vectors. I set it to 100 for the base case.
  - The default number of iterations (epochs) over the corpus is 10 for Doc2Vec. Typical iteration counts in the published Paragraph Vector paper results, using 10s-of-thousands to millions of docs, are 10-20. More iterations take more time and eventually reach a point of diminishing returns. I used 15 epochs for the base case
  - Tested different values with Bayes search.
- Building the vocabulary dictionary:
  - The vocabulary is a list of all of the unique words extracted from the training corpus.
- Training the Doc2Vec NN:
  - Use the *model.train* to fit the create embedding for each article.
- Infer vectors:
  - Use the trained model to infer a vector for any piece of text by passing a list of words to the *model.infer_vector* method. This vector can then be compared with other vectors via cosine similarity.

Classification - Logistic Regression

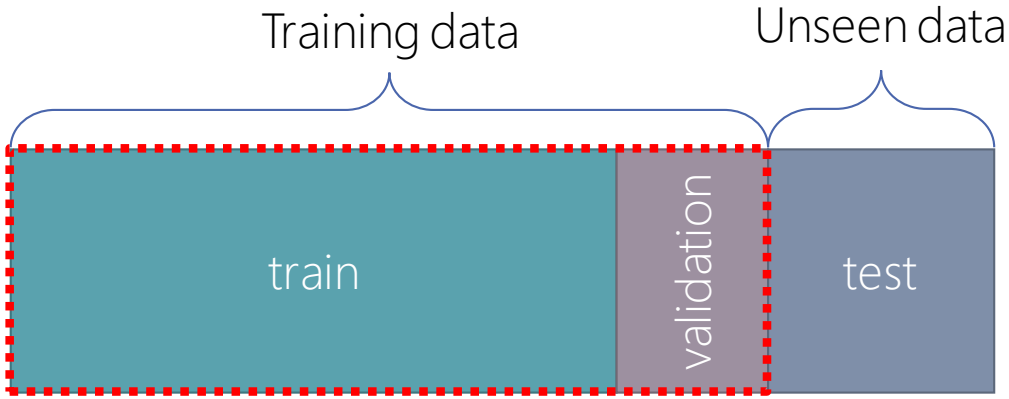
- In order to use the BayesSearchCV
  - Created a wrapper around the Gensim's Doc2Vec model that uses a similar structure as native Scikit-Learn classes.
  - Passed the Doc2Vec parameters to __init__
  - Defined the tagging, tokenization, building vocabulary, and training in the fit method.
  - The transform method returns the infer_vector for train and test data.
- Doc2Vec hyper-parameters
  - window – The maximum distance between the current and predicted word within a sentence.
  - dm - Defines the training algorithm. If dm=1, 'distributed memory' (PV-DM) is used. Otherwise, distributed bag of words (PV-DBOW) is employed.
  - vector_size – Dimensionality of the feature vectors.
  - min_count – Ignores all words with total frequency lower than this.
  - epochs – Number of iterations (epochs) over the corpus. Defaults to 10 for Doc2Vec.
- Running a BayesSearchCV is very time intensive step. The results of BayesSearch do not improve the classification results. So, I will not explore a more focused hyper-parameter space and will move to another model to investigate performance of DL for the given dataset.

```
pipeline = Pipeline([
        ('vect', Doc2VecModel()),
         ('clf', LogisticRegression(solver='liblinear'))])
parameters:
{'vect__window': list(range(5)),
 'vect__dm': [0,1],
 'vect__vector_size': list(range(100,400)),
 'vect__min_count': list(range(100)),
 'vect__epochs': [10,15,20,25,30],
 'clf__dual': (True,False),
 'clf__max_iter': [100,110,120,130,140],
 'clf__C': (1e-5, 1e2, "log-uniform")}
```

	precision	recall	f1-score	support
business	0.70	0.96	0.81	73674
finance	0.00	0.00	0.00	6763
general	0.62	0.22	0.33	25951
science	0.00	0.00	0.00	1614
tech	0.00	0.00	0.00	2674
accuracy			0.69	110676
macro avg	0.26	0.24	0.23	110676
weighted avg	0.61	0.69	0.62	110676

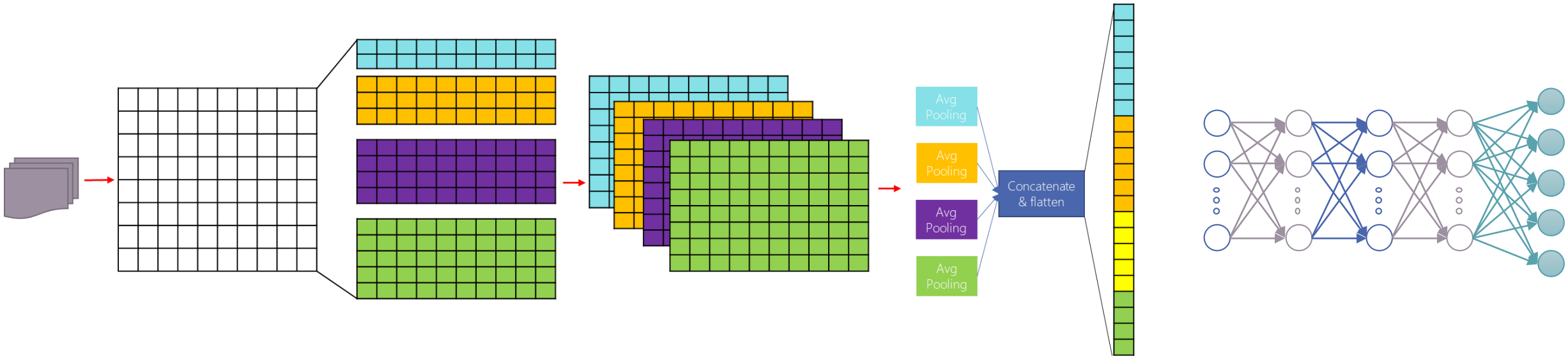
Deep Learning – text processing

- Merge labels to create five classes
- Data split strategy:
  - %30 for test
  - %70 for training
    - %20 for validation
    - %80 for training
- Text vectorization with Keras:
  - The Tokenizer class vectorizes a text corpus into a list of integers. Each integer maps to a value in a dictionary that encodes the entire corpus, with the keys in the dictionary being the vocabulary terms themselves.
  - The pad_sequences class adds padding to the end of each article to ensure all entries are the same length




```
{...
 "give": 702,
 "post": 703,
 "cause": 704,
 "rules": 705,
 "capacity": 706,
 "forecast": 707,
 "systems": 708,
 "received": 709,
 "items": 710,
 ...}
```

Deep Learning - CNN

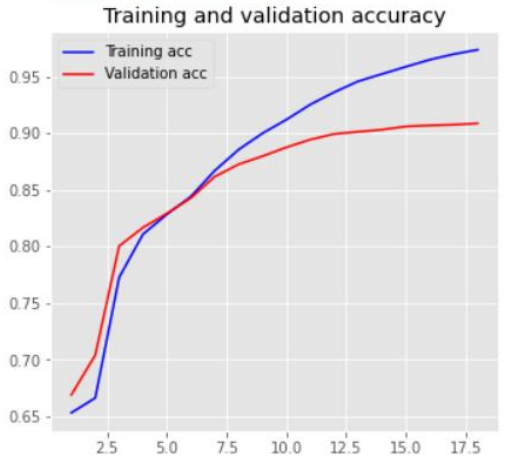


Corpus | Tokenizer | Convolution layers | Average pooling + concatenate and flatten | Dense layer + Dropout | Output

Deep Learning - CNN

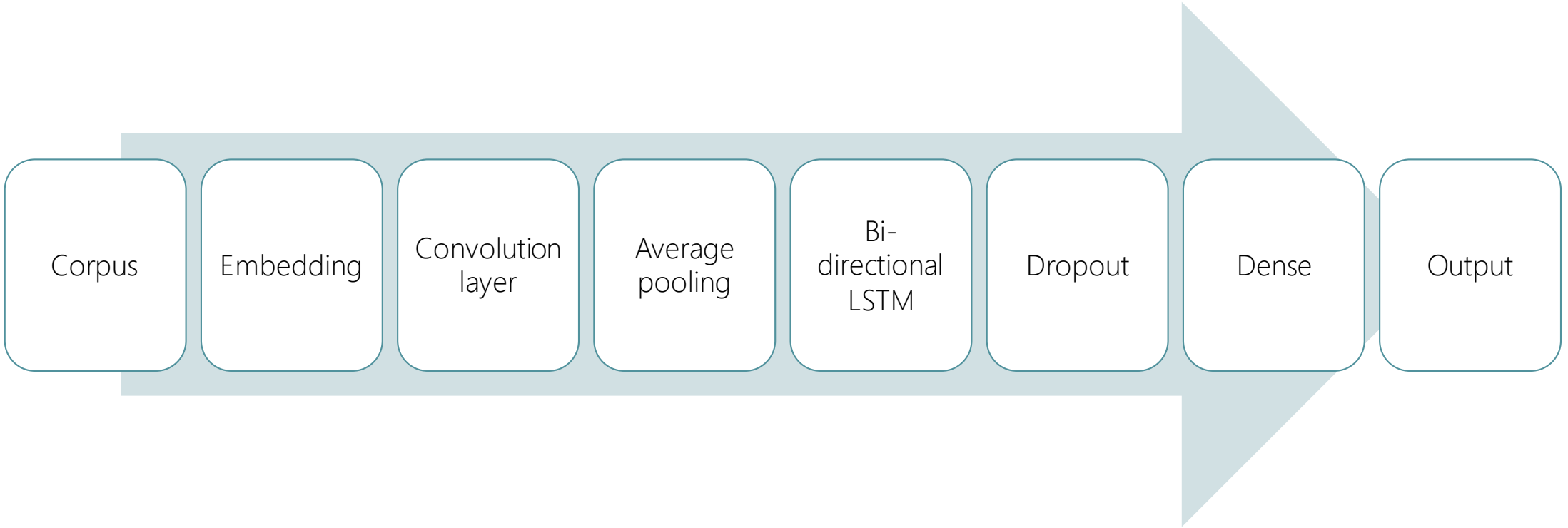
- The model tend to overfit to the training data. To prevent this issue
  - Added dropout layers
  - Added early stopping
- The CNN model outperform the previous models including logistic regression and Doc2Vec
- The CNN model is the candidate for the deployment
- Saved the tokenizer, model architecture, and weights for the deployment

```
Training Accuracy: 0.9816
Validation Accuracy:  0.9088
```



	precision	recall	f1-score	support
business	0.92	0.97	0.94	73935
finance	0.89	0.74	0.81	6788
general	0.91	0.87	0.89	25646
science	0.47	0.09	0.15	1690
tech	0.52	0.52	0.52	2617
accuracy			0.91	110676
macro avg	0.74	0.64	0.66	110676
weighted avg	0.90	0.91	0.90	110676

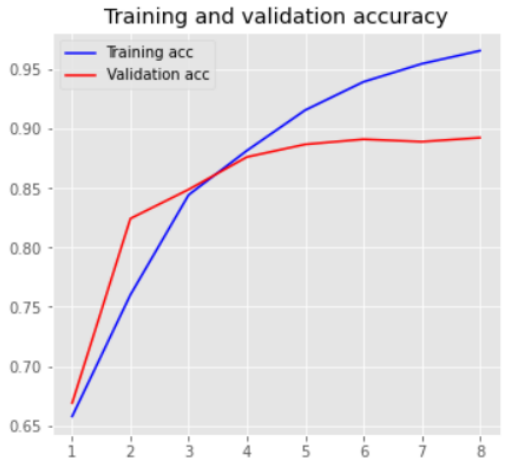
Deep Learning – CNN + LSTM

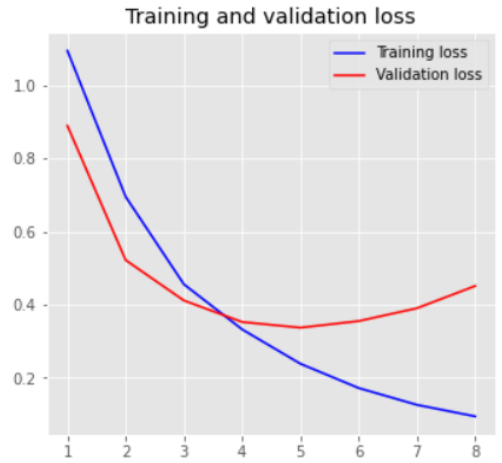


Deep Learning – CNN + LSTM

- The model tend to overfit to the training data. To prevent this issue
  - Added dropout layers
  - Added early stopping
- The LSTM model tend to overfit very quickly
- The recall for under represented labels are not as high as the results of the CNN model

```
Training Accuracy: 0.9743
Validation Accuracy:  0.8926
```

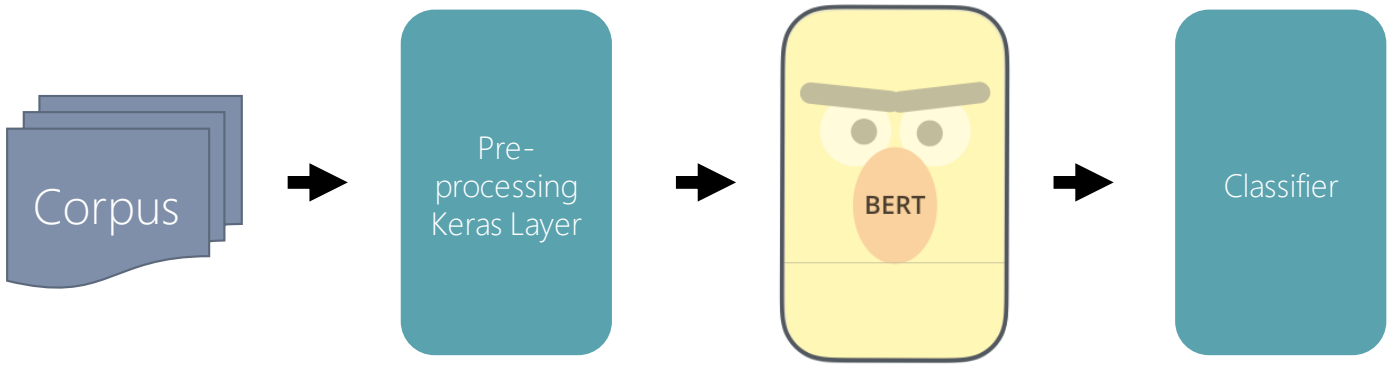




	precision	recall	f1-score	support
business	0.93	0.95	0.94	73935
finance	0.70	0.74	0.72	6788
general	0.89	0.87	0.88	25646
science	0.00	0.00	0.00	1690
tech	0.37	0.43	0.40	2617
accuracy			0.89	110676
macro avg	0.58	0.60	0.59	110676
weighted avg	0.88	0.89	0.89	110676

Deep Learning – BERT

- Small BERTs have the same general architecture but fewer and/or smaller Transformer blocks, which lets you explore tradeoffs between speed, size and quality.
- Text inputs need to be transformed to numeric token ids and arranged in several Tensors before being input to BERT. TensorFlow Hub provides a matching preprocessing model for each of the BERT models



Deep Learning – BERT

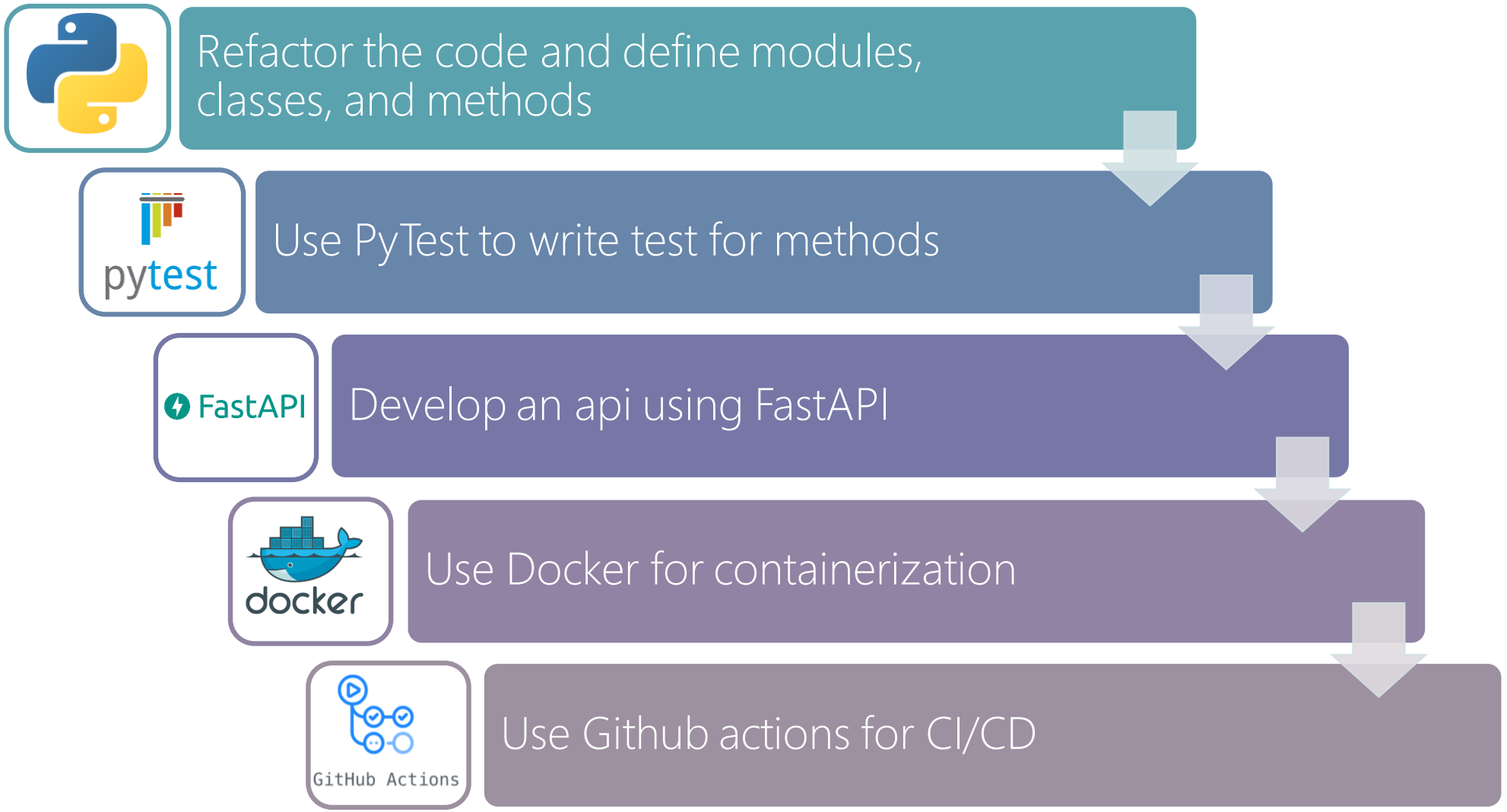
- Assembled all pieces required in my BERT model including the preprocessing module, BERT encoder, data, and classifier.

- Loss function: SparseCategoricalCrossentropy
- Optimizer: Adam

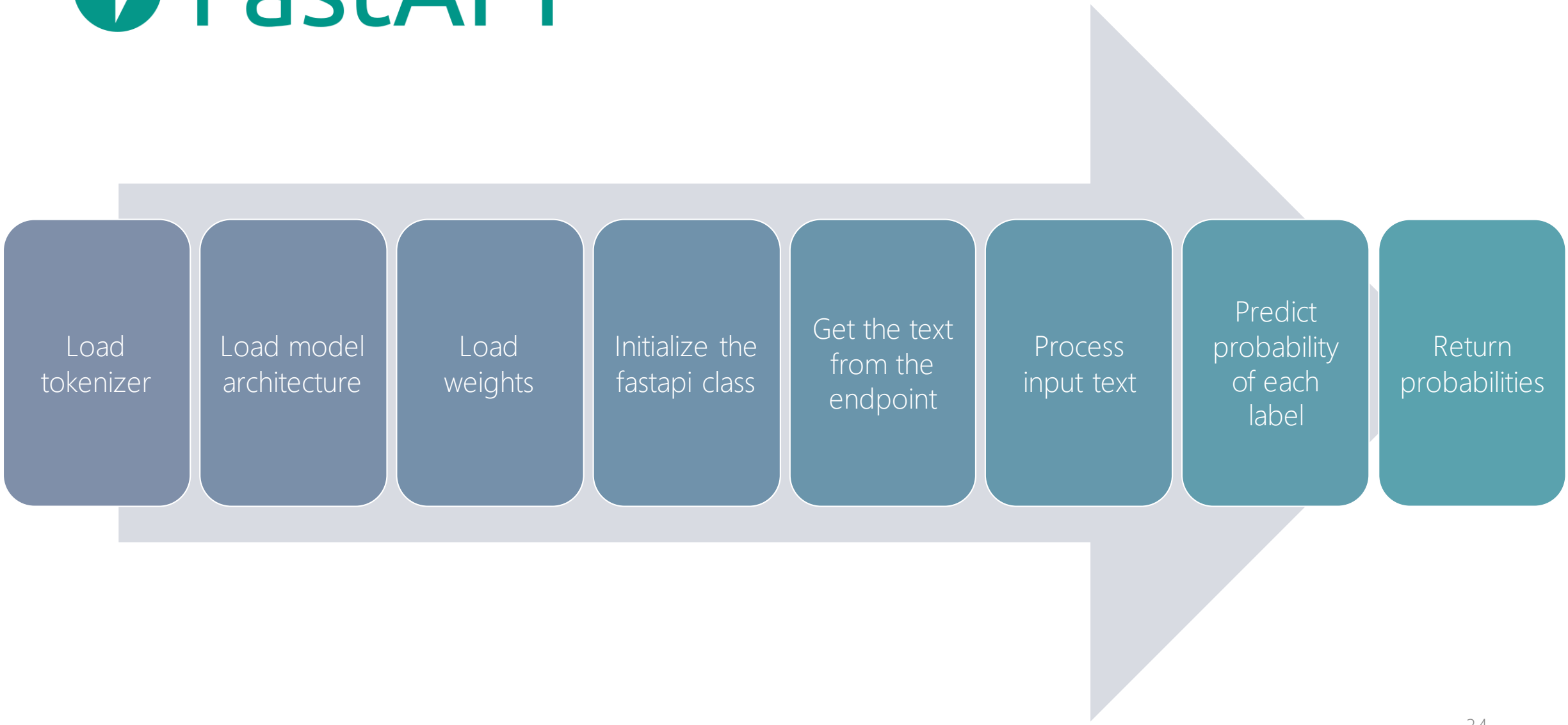
- The current configuration do not show a significant accuracy gain for the present dataset. I will use the CNN model for use in the next step.

	precision	recall	f1-score	support
business	0.95	0.92	0.93	73935
finance	0.71	0.81	0.76	6788
general	0.83	0.89	0.86	25646
science	0.84	0.60	0.70	1690
tech	0.68	0.67	0.68	2617
accuracy			0.89	110676
macro avg	0.80	0.78	0.79	110676
weighted avg	0.90	0.89	0.89	110676

Deployment Strategy

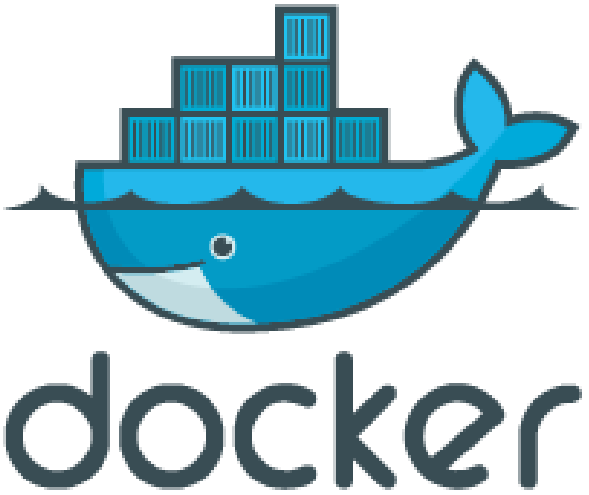


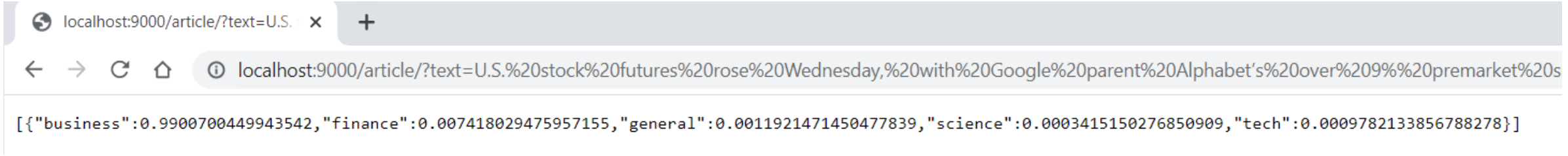
FastAPI


Load tokenizer
Load model architecture
Load weights
Initialize the fastapi class
Get the text from the endpoint
Process input text
Predict probability of each label
Return probabilities

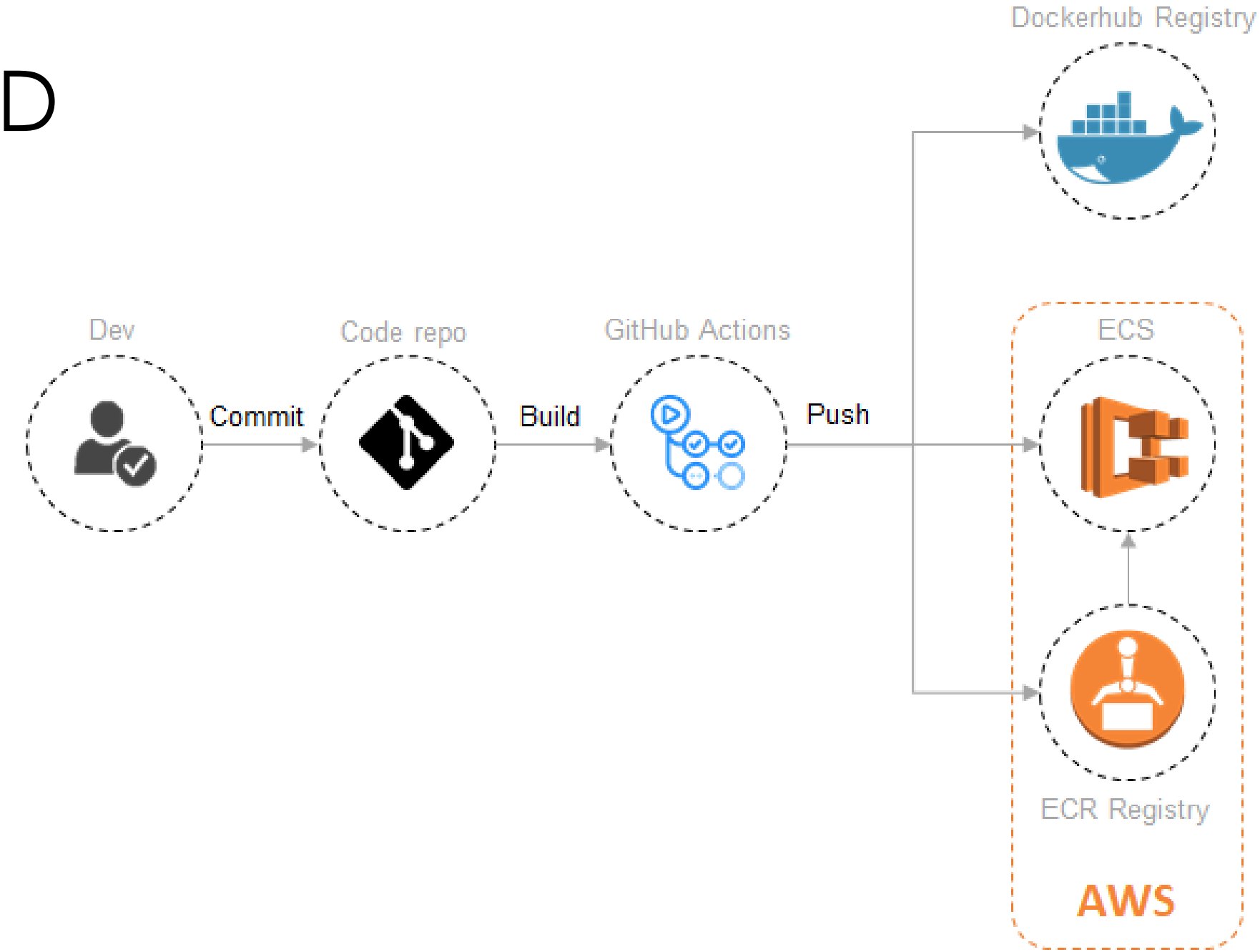
Containerization

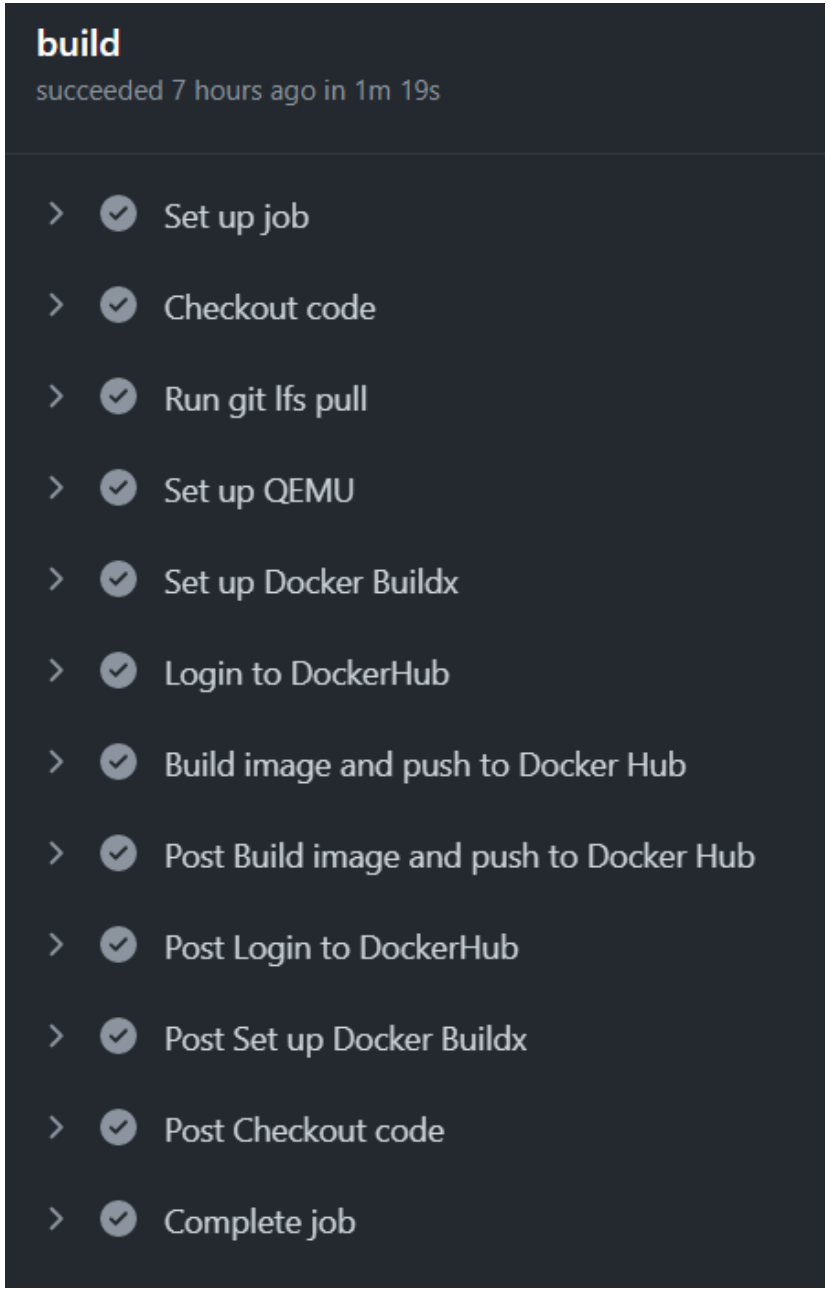
- TensorFlow used as the base image
- Copy the saved model, api, and requirements to the Docker image
- Install the requirements
- Exposed the target port
- Run the api with uvicorn





CI/CD


Dockerhub Registry
Dev
Commit
Code repo
Build
GitHub Actions
Push
ECS
ECR Registry
AWS

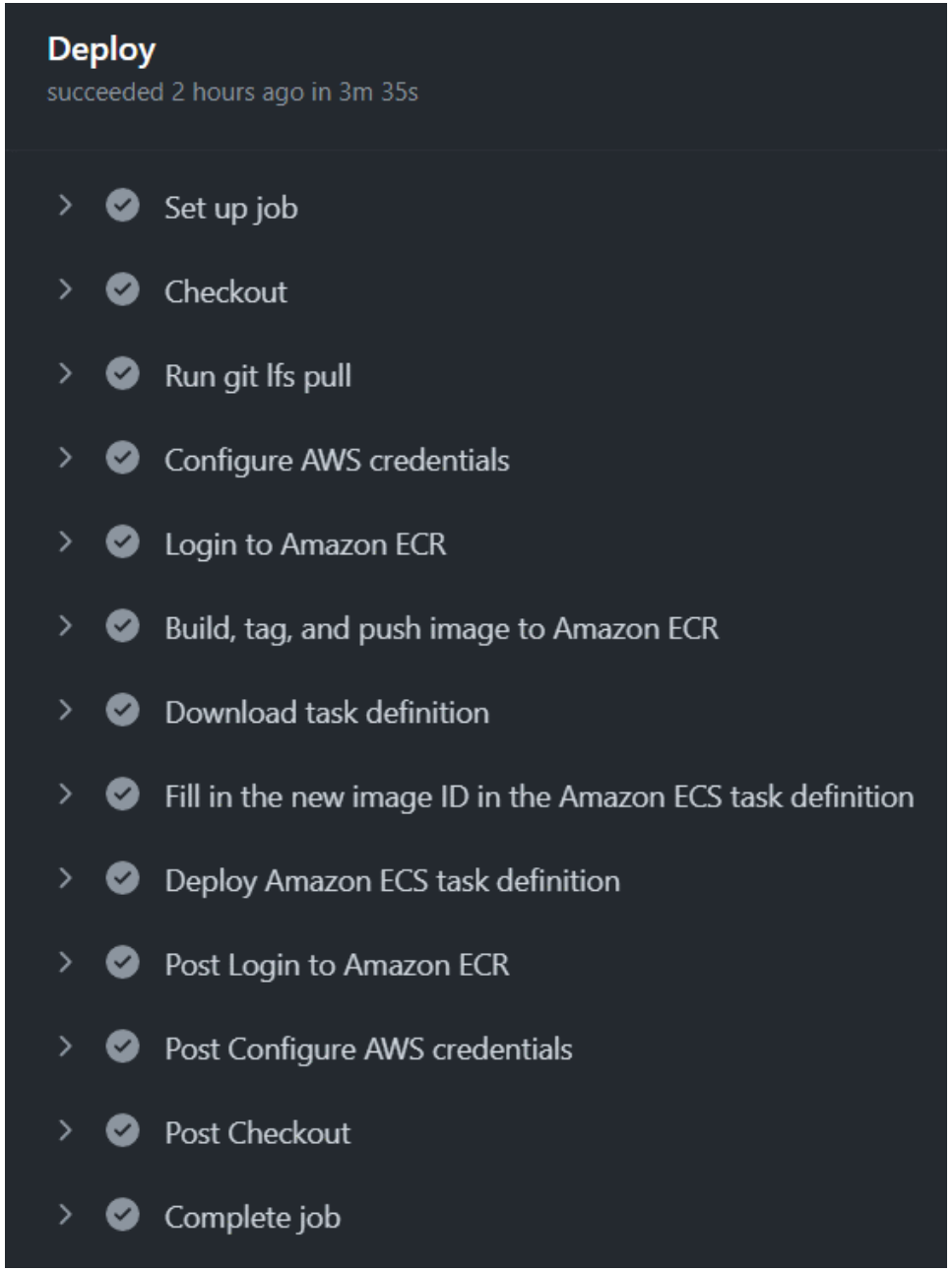


CI/CD

- Used Docker image workflow By GitHub Actions
  - For each new push the actions automatically build the image and push that to Docker Hub

CI/CD

- Used the Deploy to Amazon ECS workflow to push the container to AWS ECS
  - For each new push the actions automatically build the image and push that to AWS ECR
  - Downloads the task definition from ECS, update the image info and deploy to the ECS




THANK YOU