

SAP IES AICOE TAKE-HOME INTERVIEW TEST

Technical Assessment

Author

BRYAN TAN NING XUAN

Date

3 MAY 2023

Agenda

01 Problem Definition

02 Code Refactoring

03 Machine Learning

04 Deployment

Problem Definition

Our task is to analyse a dataset of contracts. We review methods to search the dataset, build a machine learning model to analyse the contracts, and deploy this model on our local machine

01

CODE REFACTORING

Optimise code to make it more efficient and readable

02

MACHINE LEARNING

Predict the label of the contract given the Provision

03

DEPLOYMENT

Run ML model on locally via an API and dockerise deployment

Code Refactoring: Overview

For Task B, the **RecordSearch** class comprises two functions to search for entries in the dataset.

1. **where()**: searches the dataset for entries that matches the search criteria
2. **findby()**: searches the dataset to see if there are any entries that matches the search criteria

Please refer to 01_code_refactoring_test.ipynb for a full description of the class and functions

Code Refactoring: where()

```
def where (options={}):  
    csv = pd.read_csv("../data/contract_dataset_v20220109.csv")  
  
    query = " & ".join([f'{k}=="{v}"' for k, v in options.items()])  
    csv_data = csv.query(query)  
  
    output = []  
    for index, row in csv_data.iterrows():  
        mapped = row.to_dict()  
        output.append(mapped)  
  
    return output
```

Please refer to 01_code_refactoring_test.ipynb for the annotated function

Code Refactoring: where()

ISSUES WITH ORIGINAL CODE

1. Incorrect results when there are multiple criteria in the search query
2. Inefficient boilerplate when searching for matching columns. Cumbersome to extend if new columns are added
3. Inefficient method for converting matching entries to dictionary. Again, cumbersome to extend if new columns are added

SOLUTION

1. Join queries before searching database
2. Use **query()** method to search dataframes for matching entries
3. Use **to_dict()** method to convert matching entries to dictionary

Code Refactoring: find_by()

```
def find_by(options):  
    csv = pd.read_csv("../data/contract_dataset_v20220109.csv")  
  
    for index, row in csv_data.iterrows():  
        if (("provision" in options and row["provision"] == options["provision"]) or  
            ("label" in options and row["label"] == options["label"]) or  
            ("source" in options and row["source"] == options["source"])):  
            mapped = row.to_dict()  
            return mapped  
  
    raise RecordNotFound
```

Please refer to 01_code_refactoring_test.ipynb for the annotated function

Code Refactoring: `find_by()`

ISSUES WITH ORIGINAL CODE

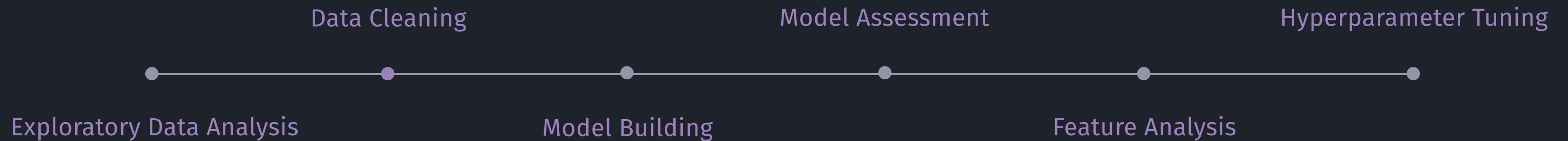
1. Inefficient boilerplate when searching for matching columns
2. Inefficient method for converting matching entries to dictionary. Cumbersome to extend if new columns are added

SOLUTION

1. Combine search criteria into single conditional when searching for matching row. Also makes code more readable
2. Use `to_dict()` method to convert matching entries to dictionary

Machine Learning: Overview

Term-Frequency Inverse Document Frequency (TF-IDF) and Logistic Regression was used as our classifier. Our final model achieved 96.8% out-of-sample accuracy.



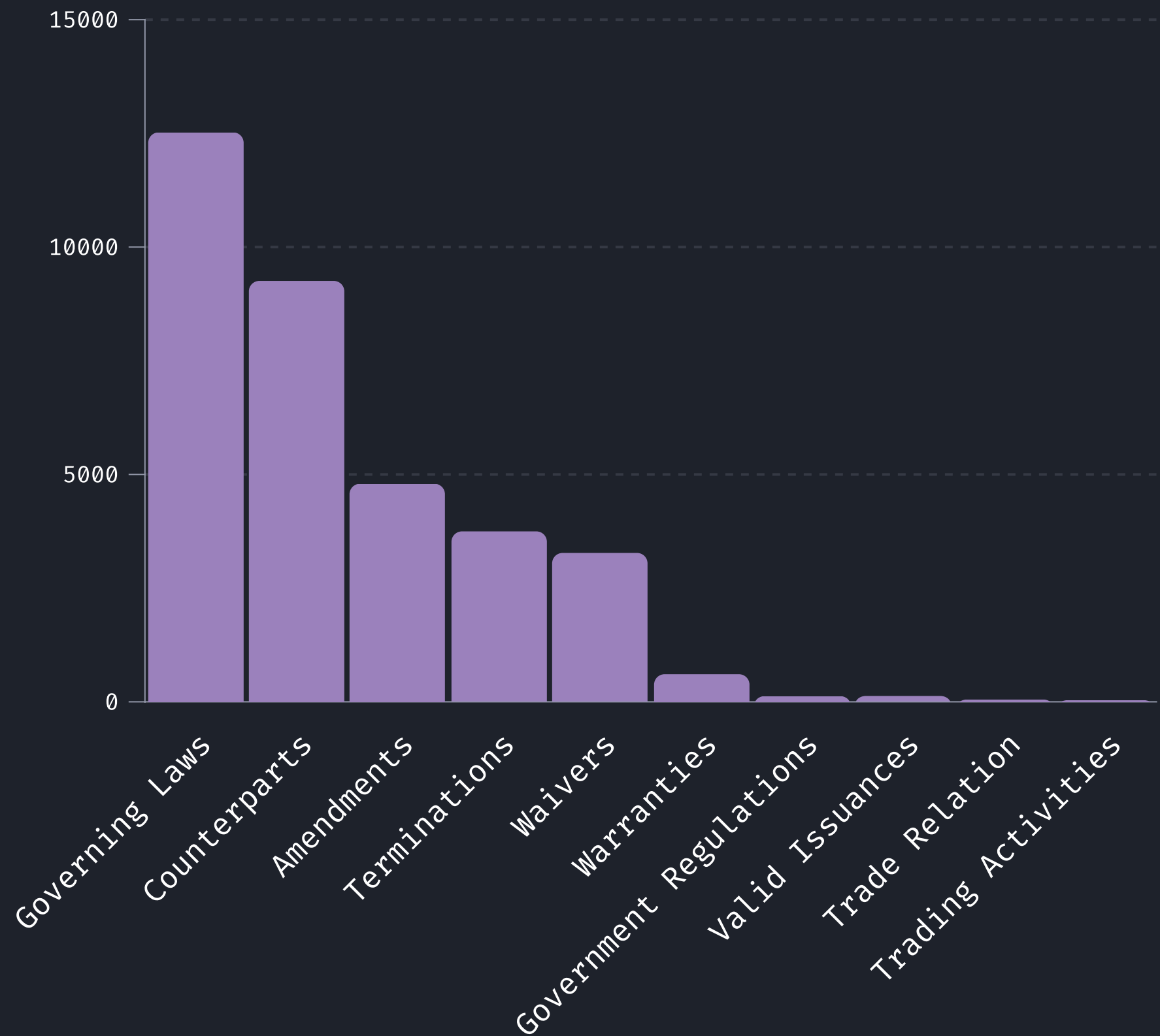
EDA and Data Cleaning

EDA INSIGHT

The dataset is not imbalanced, so we do not need oversampling, class-weighting, or threshold finetuning. However, if minority classes are of interest, we might want to finetune the model for them.

DATA CLEANING CONSIDERATIONS

Invalid characters may arise when texts are formatted and may confuse the classifier. Punctuations also do not add value to the classification and will become noise. We therefore remove these characters.



Model Building

TF-IDF

A classic NLP technique that uses the frequency of words to determine their importance as features. It vectorises the input based on these the most important features, and then uses this vector for classification. TF-IDF also allows us to visualise which words are more important for each class, which provides us insight into whether our model makes intuitive sense.

OTHER NLP TECHNIQUES

- Spellchecker and tokenising: not that useful because contractual provisions should already be formal and accurate. Acronyms might also be useful features, so spellchecker might be counter-productive
- Lemmatising and stemming: not that useful because the various grammatical forms of words might be important features. Different grammatical forms of key features implicitly "oversample" these keywords and helps improve classification accuracy, so we want to retain them
- Bag-of-words, ngrams: useful to incorporate

Please refer to 02_machine_learning_test.ipynb for the code

Model Assessment

The model performed very well at the aggregate level and the class level.

99.95%

ROC AUC SCORE

Measures efficiency of the classifier

96.34%

OUT-OF-SAMPLE ACCURACY

Measures the generalisation error of model. Considered the go-to benchmark of classification models

98.6-100%

CLASS-SPECIFIC ACCURACY

Classification accuracy for each class

98%

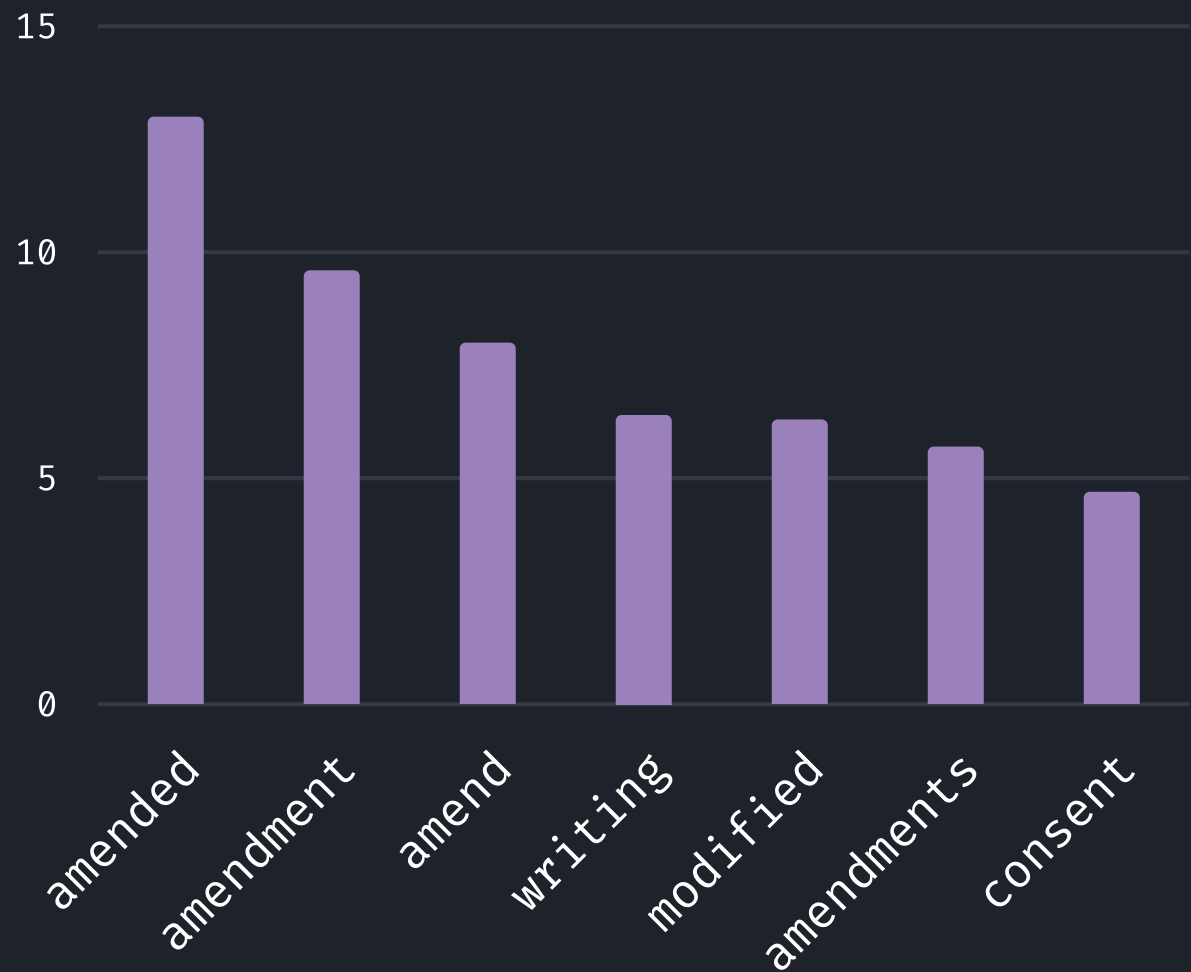
UNWEIGHTED F1-SCORE

Accuracy considering both True Negatives and False Positives across all classes

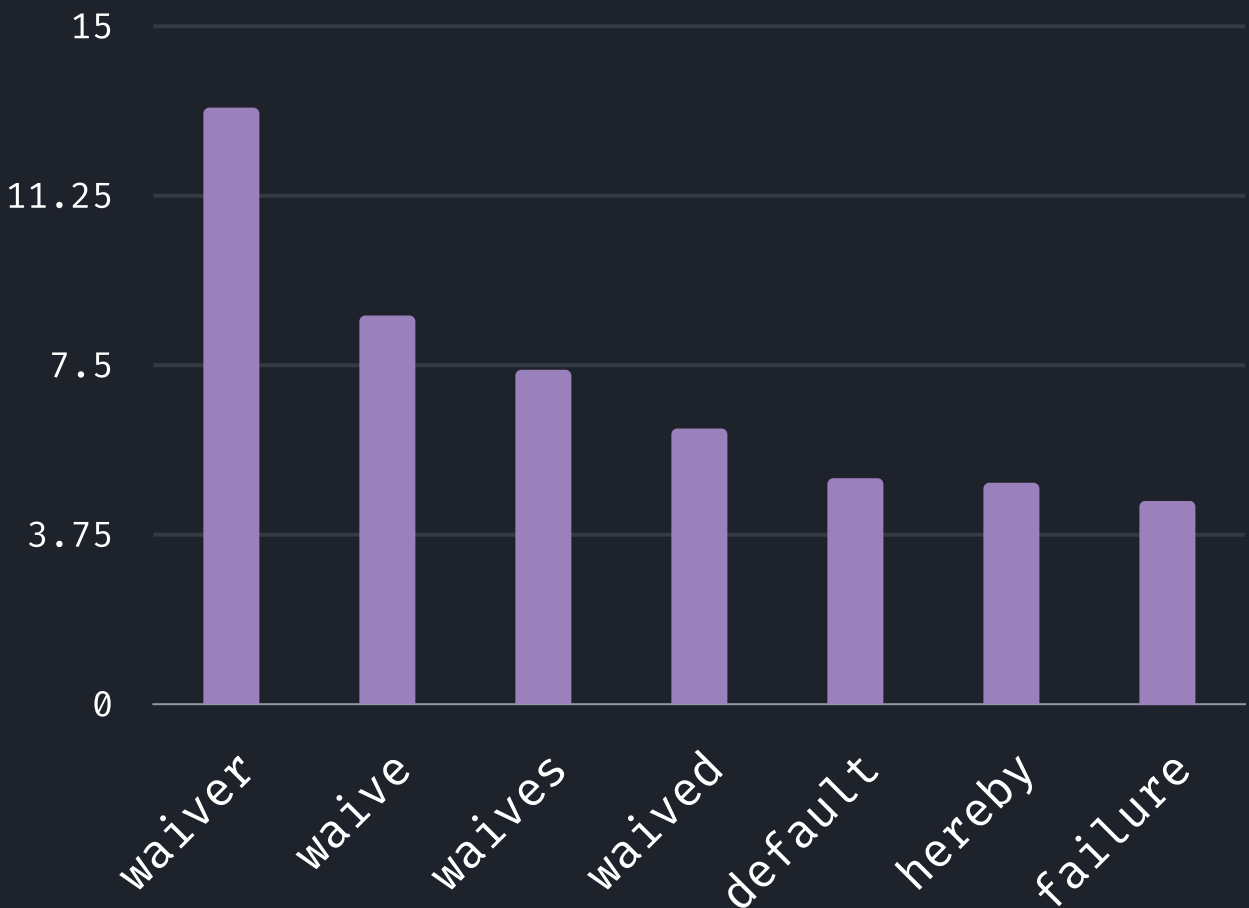
Feature Analysis

We can gain insight into the classifier by studying the features that ranked highly on importance. The features below tell us which keywords are important to each class and show that the classification algorithm is sensible.

AMENDMENTS

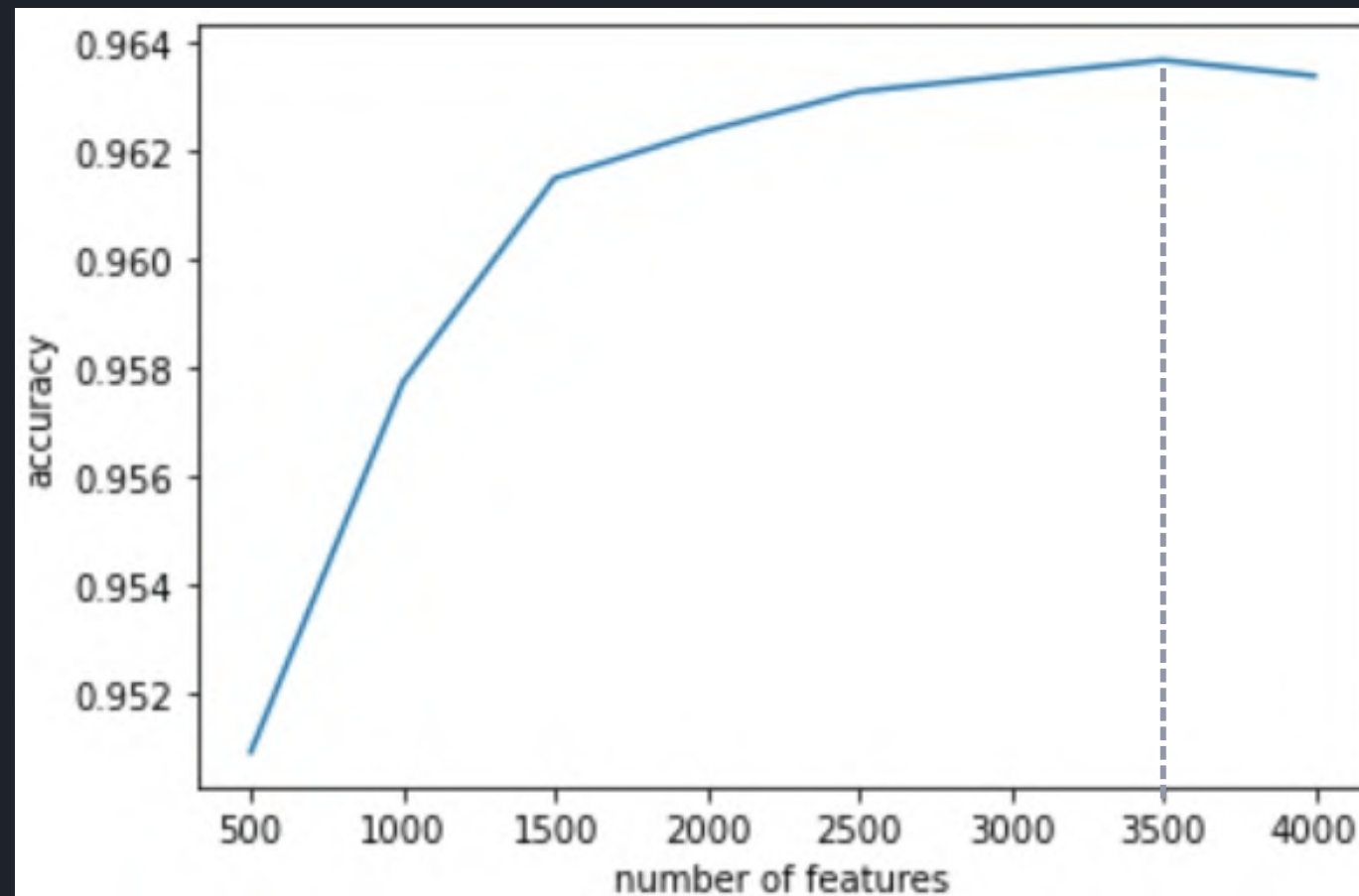


WAIVERS



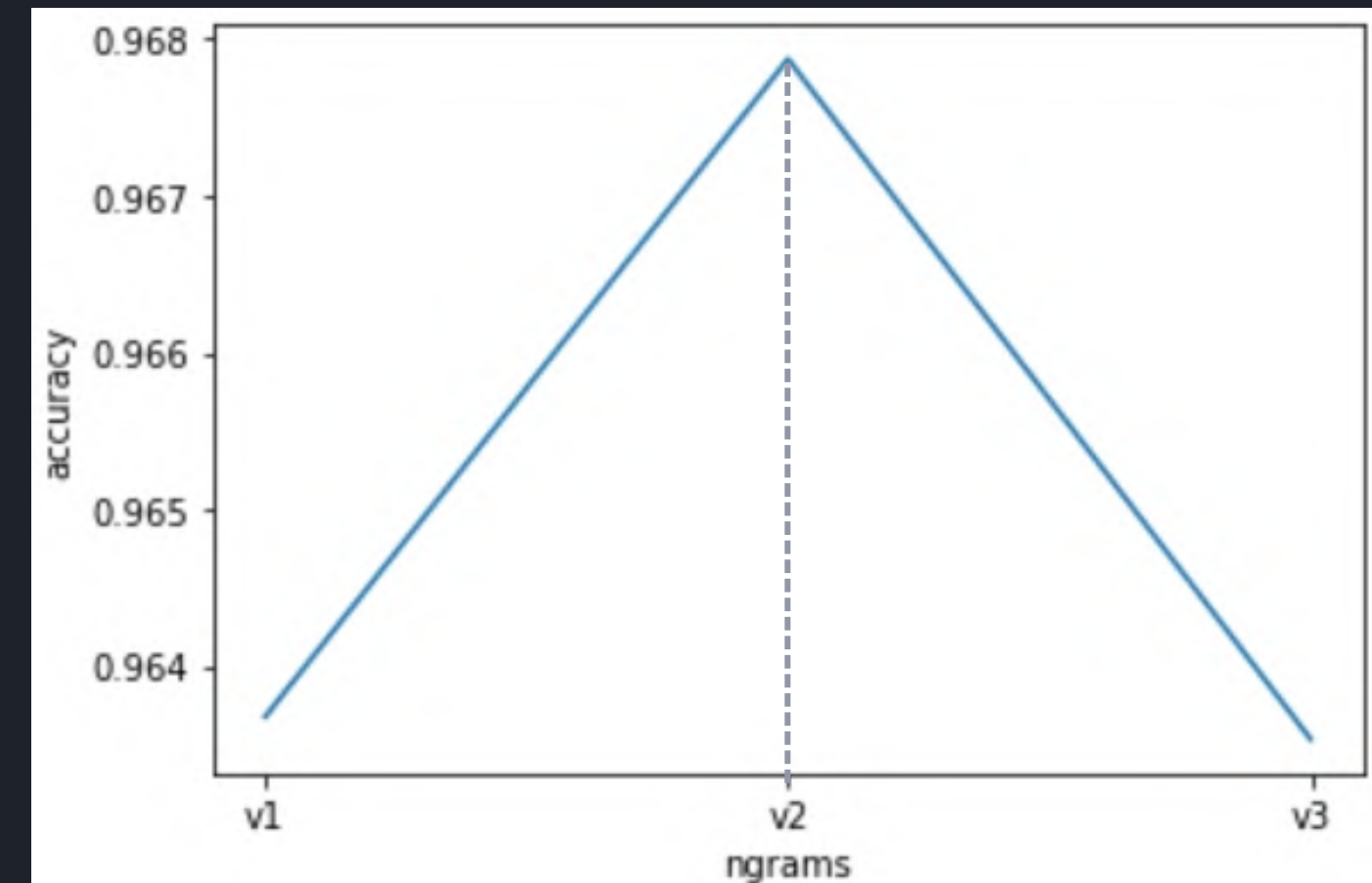
Hyperparameter Tuning

NUMBER OF FEATURES



Increasing the maximum number of features can improve the classification accuracy because we account for more keywords. However, using too much overfits the model

TYPES OF NGRAMS



Using permutations of bigrams (and trigrams) can improve the classification accuracy because phrases can be important differentiators. Using too many however, adds more noise

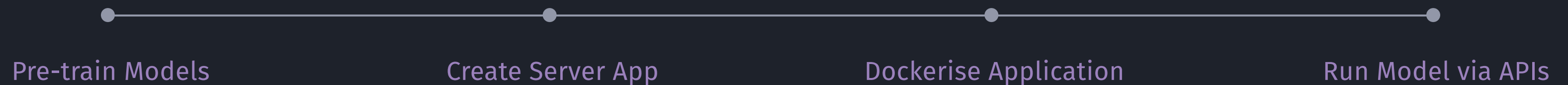
Machine Learning Future Work

Despite the good performance, there are several areas for the NLP analysis to be improved:

1. **Hyperparameters:** there are many hyperparameters left to tune. A rigorous approach would do a grid search of the $n \times n$ permutations of the n hyperparameters to find the optimal configuration
2. **Memory and runtime constraints:** this analysis primarily considers accuracy as our main KPI. However, in practice, memory and runtime considerations should also be accounted for
3. **Model selection:** It is customary to explore a range of classifiers before selecting the most optimal one. Here, we only consider the logistic model, but Naive Bayes has also been known to handle text processing well
4. **LLMs:** The hype of ChatGPT and BERT have made Large Language Models more prominent, and they have proven very powerful. These models operate on the concept of embeddings, which are known to account for the semantic meaning of words far better than NLP approaches. As this exercise was designed to test my NLP aptitude, LLM approaches were not adopted. However, the author has some experience in this area and will be happy to discuss it further.

Deployment: Overview

We deploy the ML model we built earlier on a local server with Flask, which is known to be one of the simplest Python servers. We dockerise the application, run the application from the dockerfile, and send the input data to this server via an API. We then receive the classification output from our terminal.



Pre-Train Models and Create Server Application

01 MODEL PRE-TRAINING

We pre-fitted the TF-IDF vectoriser and pre-trained the classifier. We used the **joblib** package to save our vectoriser and classifier.

03 DOCKERISE APPLICATION

We create a docker file that incorporates the dependencies and pre-trained models. We build the docker image and run the container.

02 SERVER APPLICATION

We load the pre-saved files and write the code to receive a JSON input. The code runs the predict function on the input and returns the predicted class.

04 RUN MODEL

We send the input data to the local server via an API. The API returns the predicted label of the contract.

Please refer to 03_deployment_test.ipynb for the pre-training code and server.py for the application code

Try Pitch Method references <https://medium.com/swlh/machine-learning-model-deployment-in-docker-using-flask-d77f6cb551d6>

Model Building and Code Execution

```
Anaconda Prompt (anaconda3)
(base) C:\Users\Tan Ning Xuan\Desktop\Careers\SAP\sap_ies_aicoe_take_home_test_230203\deployment_ningxuan>docker build -t mldeploy .
[+] Building 1.9s (11/11) FINISHED
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 288B 0.0s
=> [internal] load metadata for docker.io/library/python:3.7 1.7s
=> [1/6] FROM docker.io/library/python:3.7@sha256:741bcc4e5900df61be4d3c1061ec1623074b50cce1a72269ffa87b94574063 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 2.29MB 0.1s
=> CACHED [2/6] WORKDIR /app 0.0s
=> CACHED [3/6] RUN pip install pandas numpy scikit-learn flask gunicorn 0.0s
=> CACHED [4/6] ADD server.py server.py 0.0s
=> [5/6] ADD clf_model.pkl clf_model.pkl 0.0s
=> [6/6] ADD vec.pkl vec.pkl 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:54d88d1621928899a42e39cc667264538102ab1c10635ae0b63cf48c74655a3c 0.0s
=> => naming to docker.io/library/mldeploy 0.0s

(base) C:\Users\Tan Ning Xuan\Desktop\Careers\SAP\sap_ies_aicoe_take_home_test_230203\deployment_ningxuan>docker run -dp 3000:3000 mldeploy
126d691e0a80ccd28841056702ed12f6c9083acd10df94bc30fe48869fb2fb3a
```

dockerise and run application

```
(base) C:\Users\Tan Ning Xuan\Desktop\Careers\SAP\sap_ies_aicoe_take_home_test_230203\deployment_ningxuan>curl --location --request POST "http://localhost:3000" --header "Content-Type: application/json" --data-raw "{ \"provision\": \"Borrower and any endorsers or guarantors hereof severally waive presentment and demand for payment, notice of intent to accelerate maturity, protest or notice of protest and non-payment, bringing of suit and diligence in taking any action to collect any sums owing hereunder or in proceeding against any of the rights and properties securing payment hereunder, and expressly agree that this Note, or any payment hereunder, may be extended from time to time, and consent to the acceptance of further security or the release of any security for this Note, all without in any way affecting the liability of Borrower and any endorsers or guarantors hereof. No extension of time for the payment of this Note, or any installment thereof, made by agreement by Lender with any person now or hereafter liable for the payment of this Note, shall affect the original liability under this Note of the undersigned, even if the undersigned is not a party to such agreement.\" }"
{"Predicted Label":"[Waivers]"}

```

receive contract label

send API input




Deployment Analysis

ADVANTAGES

- Design of stack is simple to implement and straightforward for the client to use
- Data is managed efficiently and automatically optimised by Docker. Can manage multiple services concurrently

DISADVANTAGES

- The memory of the image is surprisingly big for a relatively simple application. If images in the container need to be processed, this will be an issue that might need to be resolved by caching

Name ↑	Tag	Status	Created	Size	Actions
mldeploy 54d88d162192 	latest	In use	about 10 hours	1.35 GB	 

- Challenging to scale application or add additional functions at this level. It is also difficult for another user to understand pre-trained model behaviour
- Could add a User Interface for the local server to show the status and history of the app



Thank you