Predicting
Supreme Court Decision

with Neural Network

BA865 Team Project

Eunjin Jeong, Yesol Lee,
Yongxian Lun, Ji Qi

Meet Team

Yesol Lee
(Sally)

Ji Qi

Yongxin Lun
(Caroline)

Eunjin Jeong

01.

Introduction

Background
Problem Statement

02.

Dataset Overview

Data structure & source

03.

Dense Layer Model

Simple Neural Network Model
With Dense laysers

04.

1D CNN Model

Applying 1D Convolutional
Network for Text Data

05.

Text Vectorization Layer

Simple Neural Network Model
With Dense laysers

06.

RNN Model

Simple Neural Network Model
With Dense laysers

01.

INTRODUCTION

1. Introduction - Background & Problem statement

Background

- The Court is the highest tribunal for all cases and interpretation of the Constitution or the laws in the United States. Supreme court decisions **impact parties in each case, stakeholders, government and society**. Supreme court decisions **regulate individuals' life, rights and obligations**.

Problem Statement

- To predict the winning party(petitioner, respondent) of each supreme court case using neural network model

2.

Data overview & Preprocessing

Data Source & Features

Source

Oyez Project

Free law project from Cornell's Legal Information Institute, Justia, and Chicago-Kent College of Law to archive Supreme Court data.

Features

- Index ID
- Name (same as first/second party)
- href
- First party(Petitioner)
- Second party(Respondent)
- Facts
- Winning party
- Winner index(0,1)

Predictors

Labels

Feature engineering

```
name_pet = []
name_rep = []
for i in range(df.shape[0]):
  fact = df["facts"][i]
  petitioner = df["first_party"][i]
  respondent = df["second_party"][i]
  p = True
  r = True
  for _ in petitioner.split():
    if _ in fact:
      p = True
      break
    else:
      p = False
  if p == False:
    #name_pet.append("Petitioner name not found in {}".format(i))
    name_pet.append(i)
  for _ in respondent.split():
    if _ in fact:
      r = True
      break
    else:
      r = False
  if r == False:
    #name_rep.append("Respondent name not found in {}".format(i))
    name_rep.append(i)
```

Combine 'fact' with 'First_party'+' '+'second_party'

- **13.05%** of facts don't contain the first party name
- **17.18%** of facts don't contain the second party name
- **1.93%** of facts don't contain both first party the second party names

Addressing data imbalance

Original Data

Imbalance winner index column
0: 2114 data points (61.03%)
1: 1350 data points (38.97%)
Total 3464 cases

Balancing

Train-Test Split 80:20

Upsample minor class (winner index=1) using Sklearn resample

Final Train Data

0: 1689 data points (50%)
1: 1689 data points (50%)
3378 train data/693 Test data

3.

Dense Layer Model

Process

- **Step1**: Vectorize Facts column by using Doc2Vec method
    - Vectorize Paragraphs instead of word

- **Step2**: Train Dense Layer model using training data and cross validation
    - Combine dense layers and several overfitting delaying techniques

- **Step3**: Evaluate the model with the test data

STEP 1. Doc2Vec

Prepare data for Doc2Vec model

- Tokenize text data
- Tagged Document

```
doc2vec_model =
Doc2Vec(vector_size=50, min_count=2,
epochs=40, dm=1, seed=865, windows=5)
```

Defined Doc2Vec Model

Vectorize the text data

- Train the model with Train data
- Infer vector by the trained model

	0	1	2	3	4	5	6	7	8	9	...
1748	1.178729	2.310049	-0.280424	0.817469	1.397772	-1.025686	-0.570710	-0.653005	1.457297	-0.381615	...
1936	-0.823386	2.223216	0.705624	2.241970	1.392194	-0.646291	-0.497492	-0.565977	0.280135	-1.400482	...
2386	2.652067	-0.049756	0.998193	0.415585	0.192079	0.214855	-0.541054	0.663737	0.153164	-2.467853	...
1662	1.583976	-0.548037	-0.756846	0.812545	0.707028	-0.267583	0.227733	0.661317	0.802523	-0.324713	...
184	1.909457	0.594758	-1.135511	0.245674	1.244574	0.413633	-0.720574	1.773448	0.945724	0.029906	...

5 rows × 50 columns

Preview of vectorized data

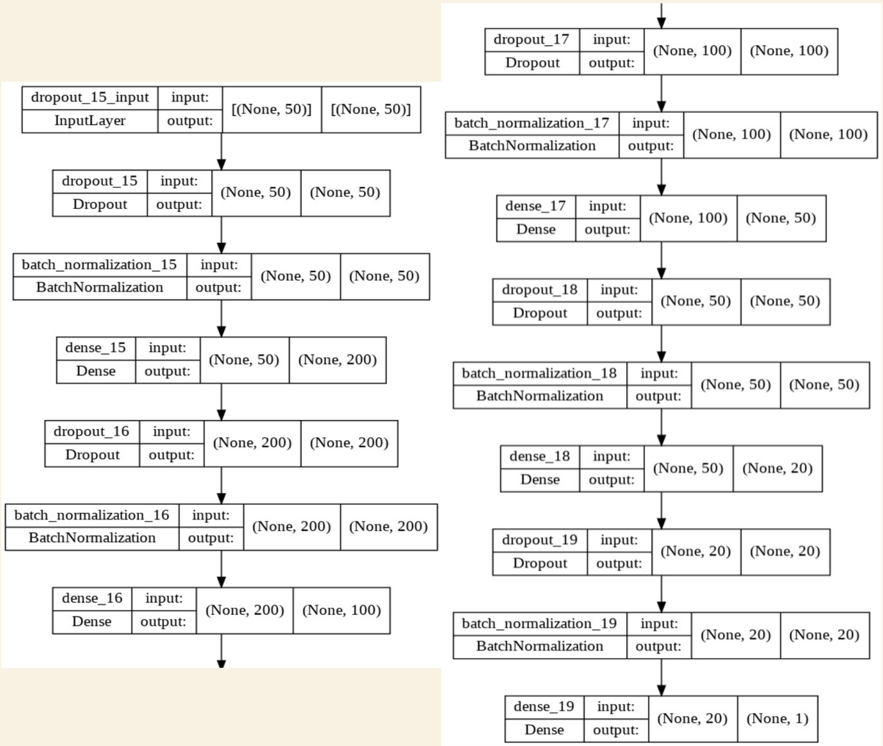
STEP 2. Modeling

Key information about the Model

- Layer type: Dense Layer
- Output activation function: Sigmoid
- Loss function: binary_crossentropy
- Optimizer: Adam

Prevent Overfitting

- Drop out
- Batch Normalization
- Kernel Regularizer

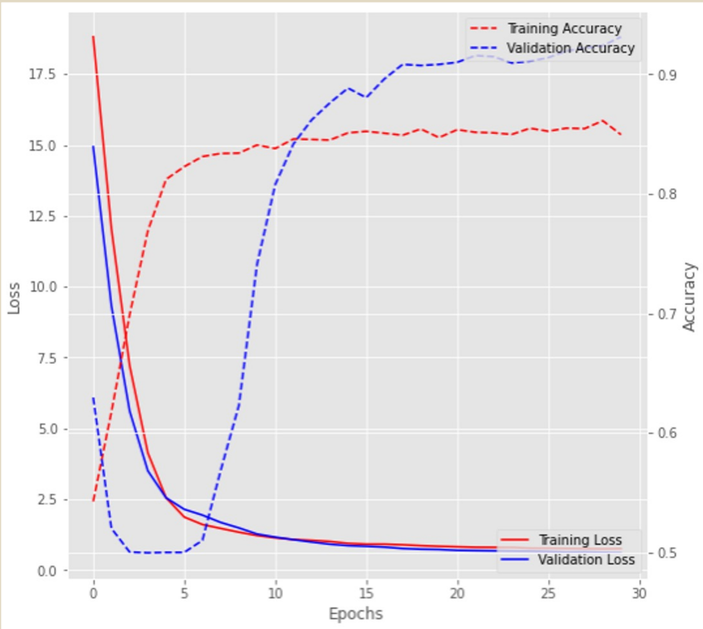


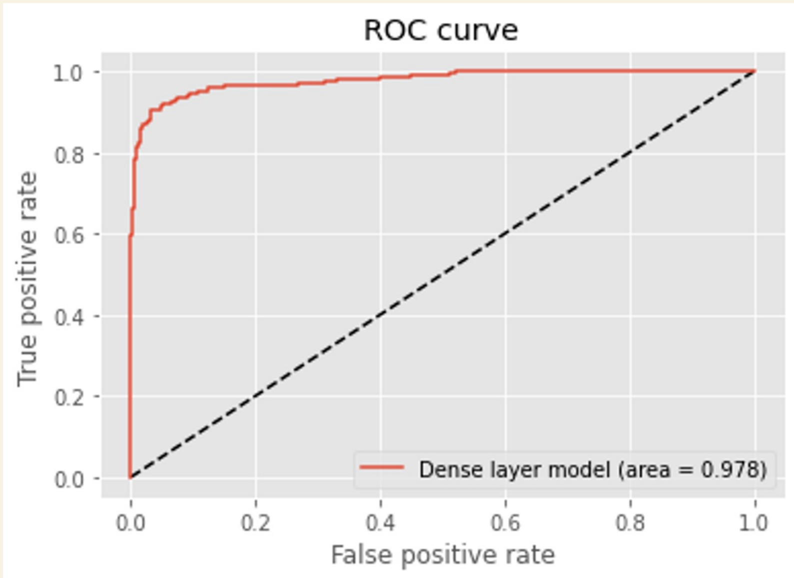
STEP 3. Evaluate the Model

Validation Accuracy: 0.7847

Test Accuracy: 0.9033

Area Under the Curve = 0.978





4.

1D Convolutional Network

Process

- **Step1**: Vectorize Facts data by using TextVectorization Layer
    - Vectorize based on unigram
- **Step2**: Train 1D CNN model using training data and cross validation
    - CNN related layers, Embedding layer, Dense layer
- **Step3**: Evaluate the model with the test data

STEP 1. Text Vectorization

Define Text Vectorization layer

- No ngram
- Standardize output length
- Output integer indices

Train Data Tensor shape

after processing

= (3382, 500)

```
text_vectorization = keras.layers.TextVectorization(
    max_tokens=2000, output_mode="int",output_sequence_length = 500)

text_vectorization.adapt(X_train)
```

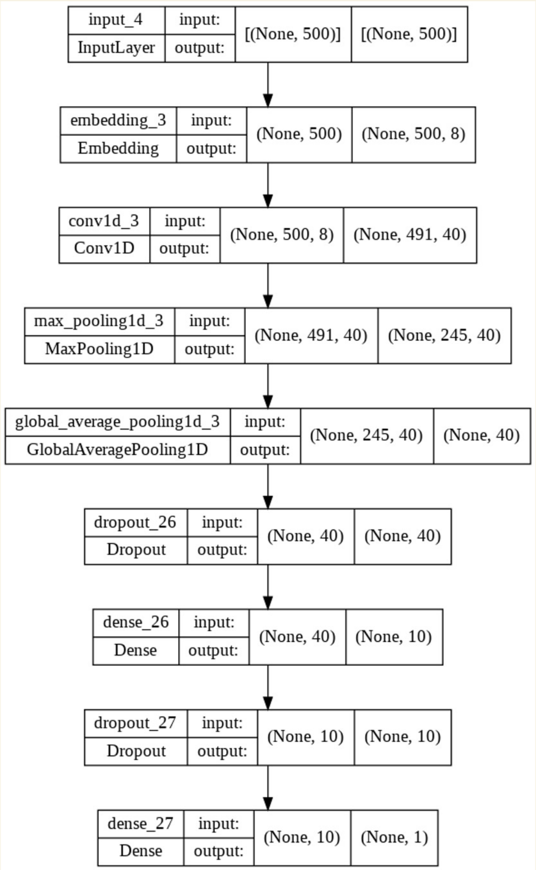
STEP 2. Modeling

Key information about the Model

- Layer type: Conv1D, MaxPool1D, GlobalAveragePooling1D, Embedding, Dense layer
- Output activation function: Sigmoid
- Loss function: binary_crossentropy
- Optimizer: Adam

Prevent Overfitting

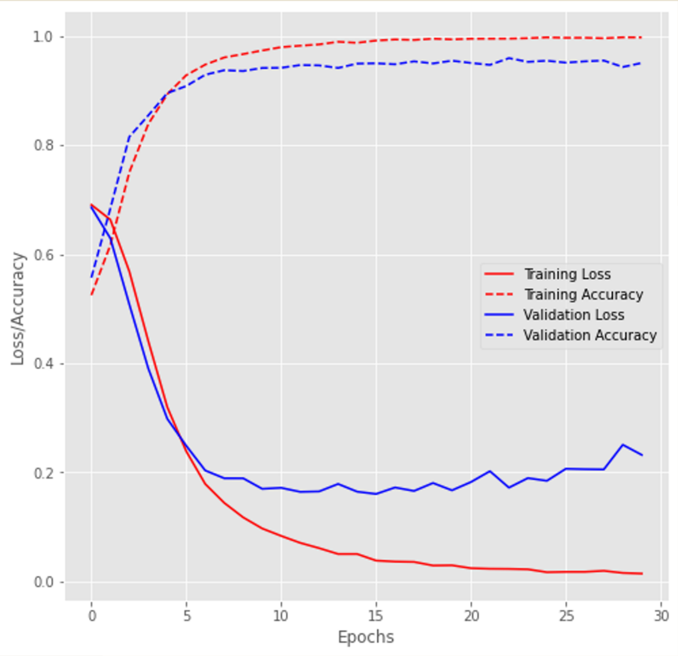
- Drop out



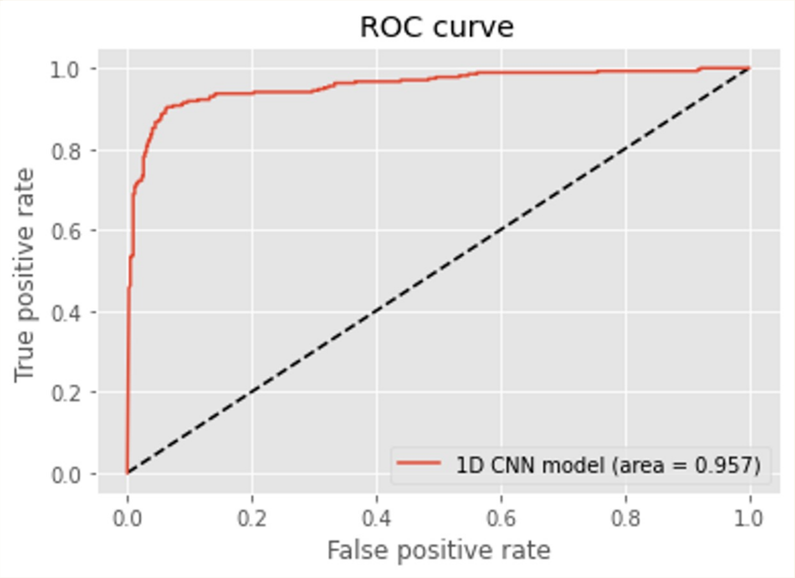
STEP 3. Evaluate the Model

Validation Accuracy: 0.9155

Test Accuracy: 0.9120



Area Under the Curve = 0.957



5.

Text Vectorization Layer

STEP 1. Text Vectorization

Text Vectorization Layer

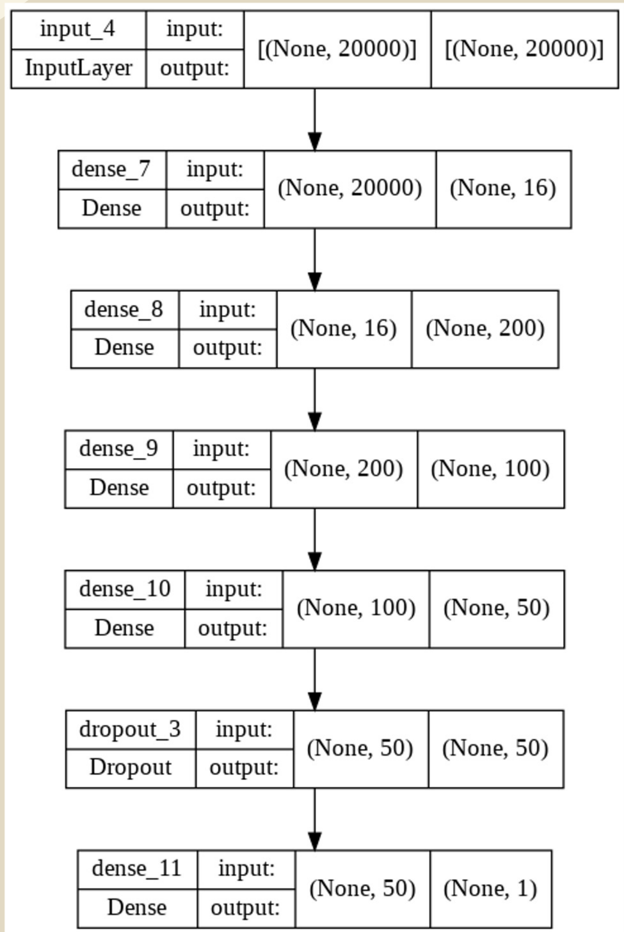
```
keras.layers.TextVectorization
(
  ngrams=2,
  max_tokens=20000,
  output_mode = "tf_idf"
)
```

- Ngrams = 2
- Max_tokens = 20000
- Output_mode = “tf_idf”

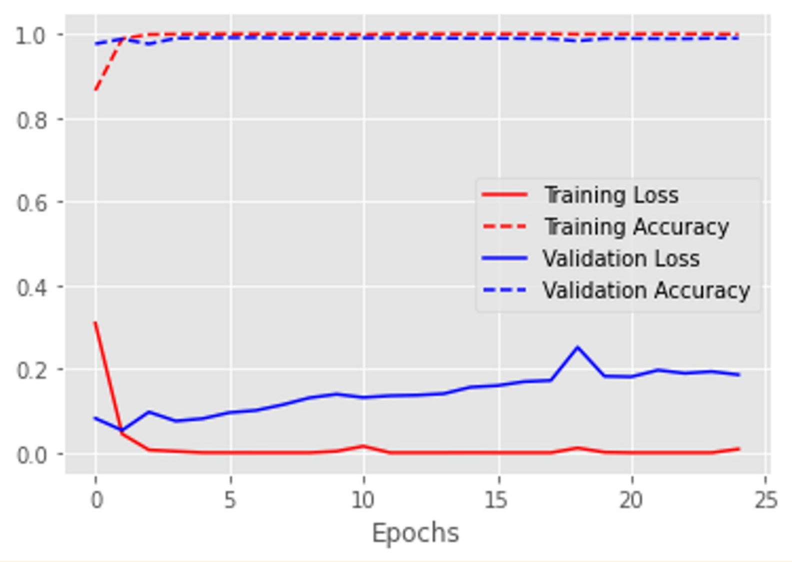
STEP 2. Model

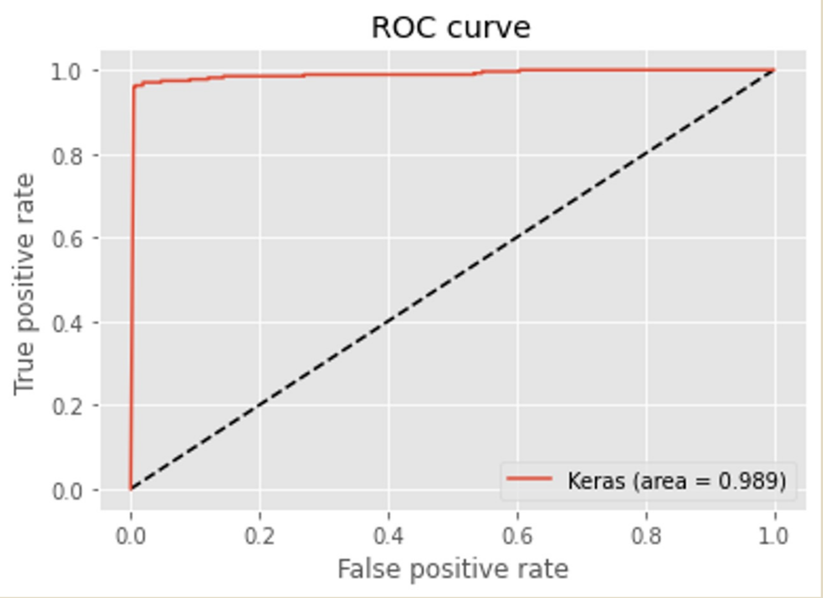
TF-IDF Bigram Model

- Several dense layers and dropout method
- Output activation function: sigmoid
- Metrics: accuracy
- Cross validation



STEP 3. Model Results





Validation Accuracy: 0.9883

Test Accuracy: 0.9885

Area = 0.989

6.

RNN
Model

Process (RNN without Textual Embeddings)

- **Step1**: Vectorize Facts column by using TextVectorization layer and one_hot encoding layer
- **Step2**: Add Bidirectional LSTM Layer to capture more information
- **Step3**: Train RNN model using training data and cross validation and Combine dense layers and several overfitting delaying techniques
- **Step4**: Evaluate the model with the test data

STEP 1 & 2. TextVectorization layer and RNN layer

TextVectorization layer

```
text_vectorization = keras.layers.TextVectorization(
    max_tokens=1000,
    output_mode="int",
    ngrams =1
)
```

Bidirectional LSTM layer

```
layers.Bidirectional(layers.LSTM(16))
```

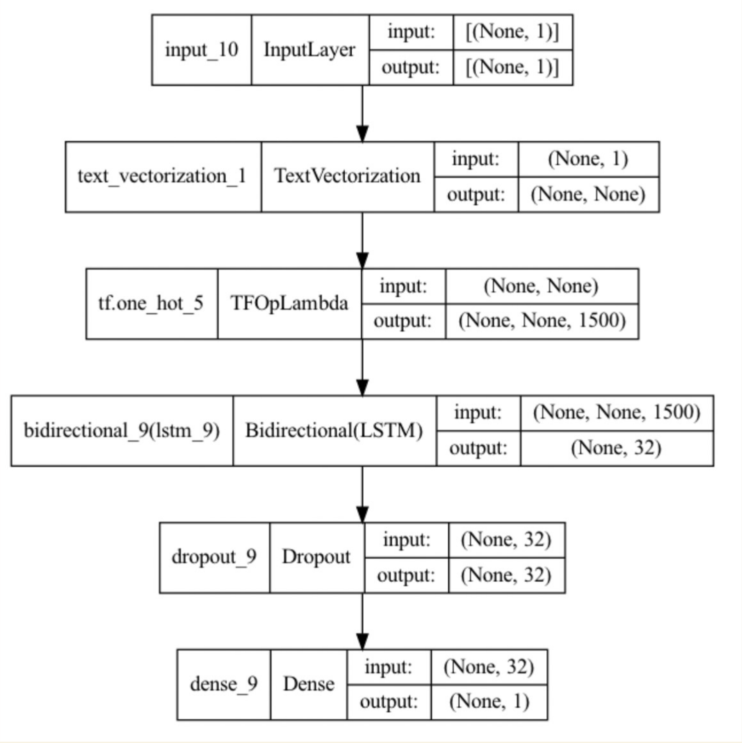
STEP 3. Modeling

Key information about the Model

- Layer type: Bi-LSTM + Dense Layer
- Output activation function: Sigmoid
- Loss function: binary_crossentropy
- Optimizer : RMS Props

Prevent Overfitting

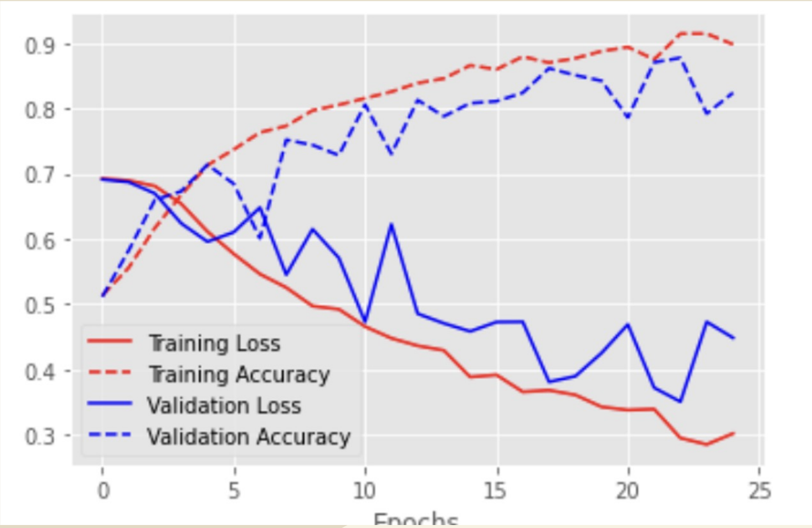
- Drop-out layer



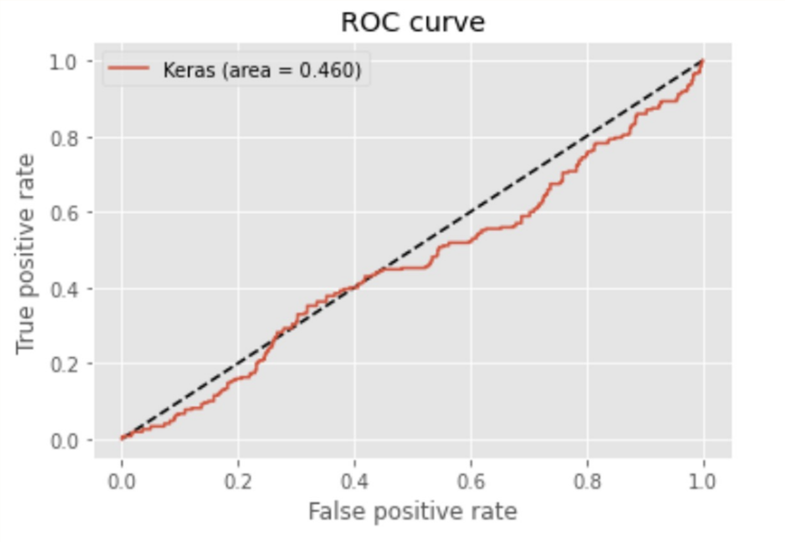
STEP 3. Evaluate the Model

Validation Accuracy: 0.75

Test Accuracy: 0.6104



Area Under the Curve = 0.460



Process (Textual Embeddings)

- **Step1**: Vectorize Facts column by using TextVectorization layer and one_hot encoding layer
- **Step2**: Add Bidirectional LSTM Layer to capture more information
- **Step3**: Add Textual Embedding Layer ()
- **Step4**: Train RNN model using training data and cross validation and Combine dense layers and several overfitting delaying techniques
- **Step5**: Evaluate the model with the test data

STEP 1 & 2. TextVectorization layer and RNN layer

TextVectorization layer

```
text_vectorization = keras.layers.TextVectorization(
    max_tokens=1000,
    output_mode="int",
    ngrams =1
)
```

Bidirectional LSTM layer

```
layers.Bidirectional(layers.LSTM(16))
```

STEP 3. Textual Embedding layer

```
x = layers.Embedding(input_dim=1000,output_dim=8,input_length=500, mask_zero=True)
```

- We are setting 1000 as the vocabulary size, as we will be encoding numbers 0 to 999.
- We want the length of the word vector to be 8, hence output_dim is set to 8.
- The length of the input sequence to embedding layer will be 500.
- Truncates after 500 tokens, and pads up to 500 tokens for shorter facts.
- Mask zero means it will skip 0 tokens and will not pass them on.

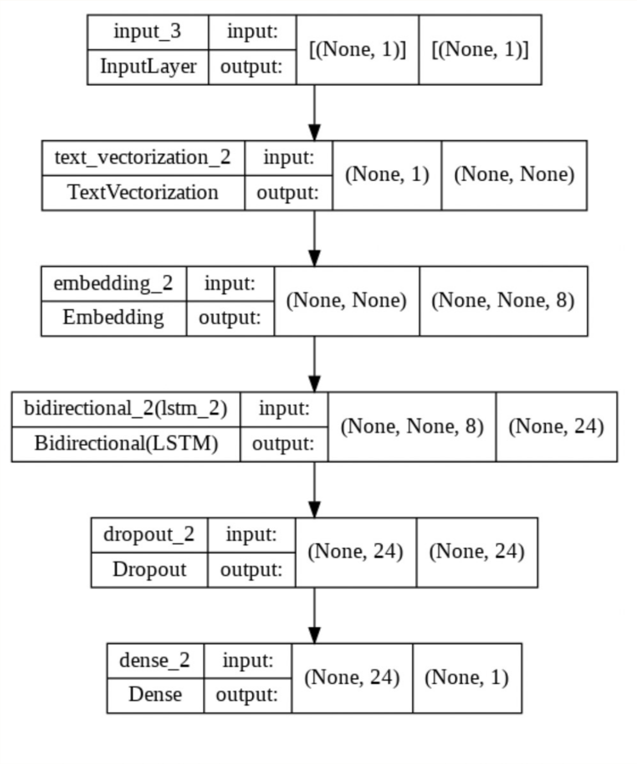
STEP 4. Modeling

Key information about the Model

- Layer type: Textual Embeddings + Bi-LSTM + Dense Layer
- Output activation function: Sigmoid
- Loss function: binary_crossentropy
- Optimizer : RMS Props

Prevent Overfitting

- Drop-out layer

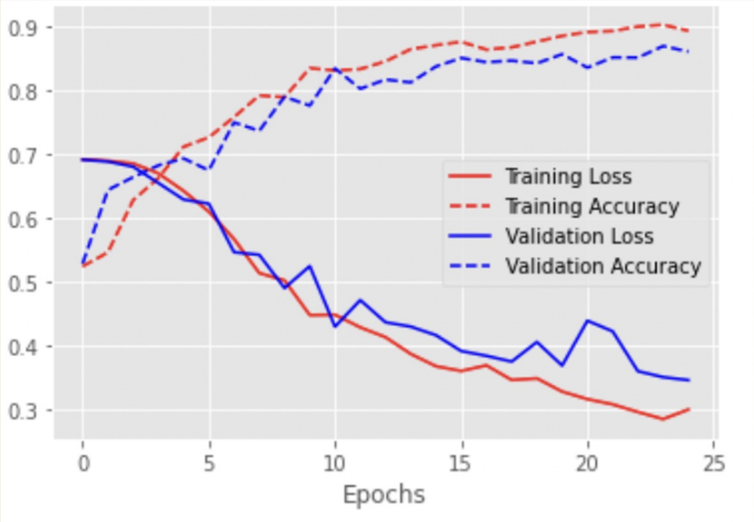


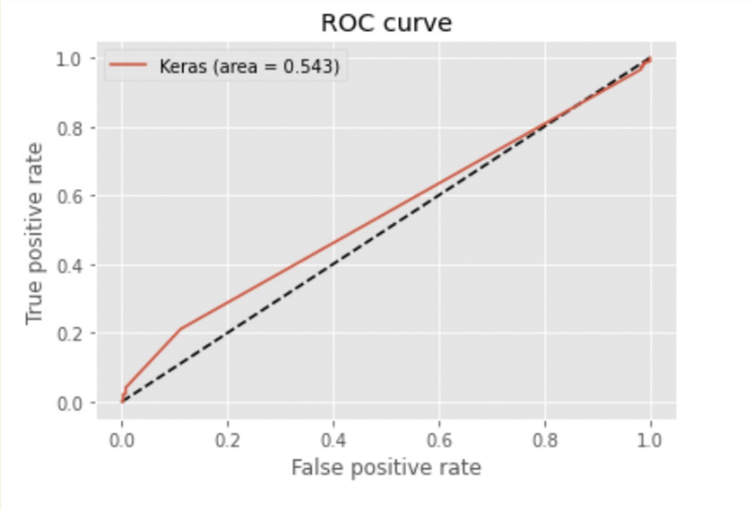
STEP 5. Evaluate the Model

Validation Accuracy: 0.7904

Test Accuracy: 0.6103

Area Under the Curve = 0.543





7.

Conclusion

Model interpretation - LIME

Model	Accuracy	AUC
Dense + Doc2Vec	0.9033	0.978
1D convolution	0.9120	0.957
Dense + Text vectorization	0.9885	0.989
Bidirectional LSTM	0.6104	0.460
Textual Embedding	0.6104	0.543
Pretrained Embedding	0.8297	0.922

Limitation & Suggestion

Cross-Validation & Upsample

Upsampling in each folds

Gather more features

Oyez database: advocate, location, lower court and date

Domain specific model

Domain specific pretrain model: legal corpuses
ex) leGloVe

THANK YOU!

Do you have any questions?