BA865 Final Project: Predicting Supreme Court Decisions

Team: Eunjin Jeong, Ji Qi, Yesol Lee, Yongxian Lun

1. Introduction

- **Case background**: The Supreme 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. Therefore, predicting supreme court decision is critical that it helps stakeholder decision making.
- **Problem statement**: This project aims to predict whether petitioner will win or respondent will win in each case using multiple neural network models. This is a binary classification problem given winner index, party names, and case facts. Winner index indicates whether petitioner won or respondent won.
- **Dataset**: The source of this data is the Oyez project. Oyez project is a free law project from Cornell's Legal Information Institute, Justia, and Chicago-Kent College of Law to archive Supreme Court data. We used dataset(task1_data.pkl) gathered by Mohammed Alsayed et all, in https://github.com/smitp415/CSCI_544_Final_Project.git (https://github.com/smitp415/CSCI_544_Final_Project.git)

2. Statistics of dataset

Load the dataset

In [2]:
```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.calibration import CalibratedClassifierCV

import nltk
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import RegexpTokenizer

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

import seaborn as sns
```

In [3]:
```
# For displaying facts
pd.set_option('display.max_colwidth', None)
```

In [4]:
```
# Load dataset as dataframe
df = pd.read_pickle('https://github.com/yesol-ba/portfolio/blob/main/Data/ba865_supreme%20court%20data_task1_

#df = pd.read_pickle('/content/task1_data.pkl') # Sally's Path
df.rename(columns={'Facts': 'facts'}, inplace=True)
df.drop(columns=['index'], inplace=True)
df.reset_index(inplace=True)

print(f'There are {len(df)} cases.')
```

```
There are 3464 cases.
```

In [8]:
```
# Looking at the dataset
df.head(3)
```

Out[8]:

	index	ID	name	href	first_party	second_party	winning_party	winner_index	facts
0	0	50606	Roe v. Wade	https://api.oyez.org/cases/1971/70-18	Jane Roe	Henry Wade	Jane Roe	0	In 1970, Jane Roe (a fictional name used in court documents to protect the plaintiff's identity) filed a lawsuit against Henry Wade, the district attorney of Dallas County, Texas, where she resided, challenging a Texas law making abortion illegal except by a doctor's orders to save a woman's life. In her lawsuit, Roe alleged that the state laws were unconstitutionally vague and abridged her right of personal privacy, protected by the First, Fourth, Fifth, Ninth, and Fourteenth Amendments.
1	1	50613	Stanley v. Illinois	https://api.oyez.org/cases/1971/70-5014	Peter Stanley, Sr.	Illinois	Stanley	0	Joan Stanley had three children with Peter Stanley. The Stanleys never married, but lived together off and on for 18 years. When Joan died, the State of Illinois took the children. Under Illinois law, unwed fathers were presumed unfit parents regardless of their actual fitness and their children became wards of the state. Peter appealed the decision, arguing that the Illinois law violated the Equal Protection Clause of the Fourteenth Amendment because unwed mothers were not deprived of their children without a showing that they were actually unfit parents. The Illinois Supreme Court rejected Stanley's Equal Protection claim, holding that his actual fitness as a parent was irrelevant because he and the children's mother were unmarried.

	index	ID	name	href	first_party	second_party	winning_party	winner_index	facts
2	2	50623	Giglio v. United States	https://api.oyez.org/cases/1971/70-29	John Giglio	United States	Giglio	0	John Giglio was convicted of passing forged money orders. While his appeal to the you.S. Court of Appeals for the Second Circuit was pending, Giglio's counsel discovered new evidence. The evidence indicated that the prosecution failed to disclose that it promised a key witness immunity from prosecution in exchange for testimony against Giglio. The district court denied Giglio's motion for a new trial, finding that the error did not affect the verdict. The Court of Appeals affirmed.

In [6]:
```
# There are 3 numerical columns and 6 object columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3464 entries, 0 to 3463
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   index          3464 non-null   int64 
 1   ID             3464 non-null   int64 
 2   name           3464 non-null   object
 3   href           3464 non-null   object
 4   first_party    3464 non-null   object
 5   second_party   3464 non-null   object
 6   winning_party  3464 non-null   object
 7   winner_index   3464 non-null   int64 
 8   facts          3464 non-null   object
dtypes: int64(3), object(6)
memory usage: 243.7+ KB
```

In [7]:
```
# There isn't any missing values in this dataset
df.isna().sum()
```

Out[7]:
```
index            0
ID               0
name             0
href             0
first_party      0
second_party     0
winning_party    0
winner_index     0
facts            0
dtype: int64
```

Descriptive statistics

In [9]:
```
avg_char = df['facts'].apply(lambda x: len(str(x))).mean()
print(f'Average facts character length: {avg_char:.0f}')

avg_word = df['facts'].apply(lambda x: len(str(x).split())).mean()
print(f'Average facts word length: {avg_word:.0f}')

del avg_char, avg_word
```

```
Average facts character length: 1179
Average facts word length: 189
```

In [10]:
```
print(f'There are {len(df)} cases.')
print(f'There are {len(df[df["winner_index"]==0])} rows for class 0.')
print(f'There are {len(df[df["winner_index"]==1])} rows for class 1.')
```

```
There are 3464 cases.
There are 2114 rows for class 0.
There are 1350 rows for class 1.
```

In [11]:
```
# Facts character stats
df['facts'].apply(lambda x: len(str(x))).describe()
```

Out[11]:
```
count    3464.000000
mean     1179.302252
std       556.335680
min        95.000000
25%       784.000000
50%      1112.500000
75%      1496.000000
max      6108.000000
Name: facts, dtype: float64
```

In [12]:
```
# Facts word stats
df['facts'].apply(lambda x: len(str(x).split())).describe()
```

Out[12]:
```
count    3464.000000
mean      188.618938
std        91.496982
min        13.000000
25%       125.000000
50%       176.000000
75%       239.000000
max       974.000000
Name: facts, dtype: float64
```

In [13]:
```
# Seqence Model Check (Not Pass)
text_vectorization = keras.layers.TextVectorization(
    max_tokens=1000, # adding more tokens to allow for increase due to bigrams.
    output_mode="multi_hot", # This is requesting integer encodings (which means we'll have a sequence of int
)
text_vectorization.adapt(df['facts'])
vectorized_facts = text_vectorization(df['facts'])


lengths = [len(x) for x in vectorized_facts]

print(f'The average fact in our data has {np.mean(lengths):.0f} words, and we have {len(df)} samples.\n')

print(f'The ratio of samples to average sample length is {(len(df)/np.mean(lengths)):.0f}. We are nowhere clo

print(f'We need a larger dataset containing at least {(np.mean(lengths)*1500):.0f} samples.')
```

```
Metal device set to: Apple M1

2024-02-07 13:08:02.118679: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:3
05] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built
with NUMA support.
2024-02-07 13:08:02.119044: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:2
71] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical P
luggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
2024-02-07 13:08:02.171462: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU fr
equency: 0 Hz
2024-02-07 13:08:02.215477: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] P
lugin optimizer for device_type GPU is enabled.

The average fact in our data has 1000 words, and we have 3464 samples.

The ratio of samples to average sample length is 3. We are nowhere close to 1500.

We need a larger dataset containing at least 1500000 samples.
```

3. Data preprocessing

- **'winner_index' as Label**: 0 means first party(petitioner) wins and 1 means second party(respondent) wins. There are imbalances, so we will upsample minor class.
- **'first_party', 'second_party', 'facts' as Predictors**: We will use these features as predictors but do feature engineering to combine it.
- **'name', 'winning_party'**: ‘name’ consists of first party name and second party name, so we don’t use this feature as we already included party names in ‘facts’. ‘winning party’ is represented by ‘winner_index’, which is target variable.
- **'ID', 'href**': ‘ID’ was generated as an identifier when gathering data. It doesn’t add a lot of values, and new IDs in test set that didn’t appear in train set might produce errors. ‘href’ is reference number grated after case, so we won’t use it as well.

Feature engineering

- Checking whether party names are included in facts
  - 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
- Therefore, we decided to merge 'facts', 'first_party', and 'second_party' to preserve party information.
- Then, we will only use merged 'facts' as a predictor.

In [14]:
```
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)
```

In [15]:
```
perc_miss_pet = len(name_pet) / len(df) * 100
print('{:.2f}% of facts don\'t contain the first party name'.format(perc_miss_pet))

perc_miss_rep = len(name_rep) / len(df) * 100
print('{:.2f}% of facts don\'t contain the second party name'.format(perc_miss_rep))

perc_miss_both = len(set(set(name_pet) & set(name_rep))) / len(df) * 100
print('{:.2f}% of facts don\'t contain both first party the second party names'.format(perc_miss_both))
```

```
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
```

In [16]:
```
# Combining first party and second party with facts
df['facts'] = df['first_party']+' '+df['second_party']+' '+df['facts']
```

In [116]:
```
df['facts'][2]
```

Out[116]:
```
'John Giglio  United States John Giglio was convicted of passing forged money orders. While his appeal to t
he you.S. Court of Appeals for the Second Circuit was pending, Giglio’s counsel discovered new evidence. Th
e evidence indicated that the prosecution failed to disclose that it promised a key witness immunity from p
rosecution in exchange for testimony against Giglio. The district court denied Giglio’s motion for a new tr
ial, finding that the error did not affect the verdict. The Court of Appeals affirmed.'
```

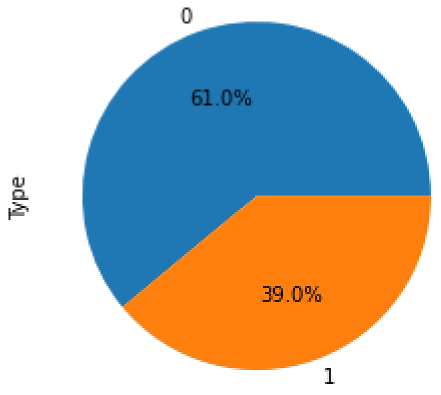
Imbalance in Label class

- winner_index

In [18]:
```
print(df["winner_index"].value_counts())

df.groupby('winner_index').size().plot(kind='pie',
                                       y = "winner_index",
                                       label = "Type",
                                       autopct='%1.1f%%')
```

```
0    2114
1    1350
Name: winner_index, dtype: int64
```

Out[18]: <AxesSubplot:ylabel='Type'>



Train-Test split

- We split train-test before upsampling to avoid duplicated rows in each set

In [19]:
```
# Perform an 80-20 split for training and testing data
X_train, X_test, \
y_train, y_test = train_test_split(
    df[['winner_index', 'facts']],
    df['winner_index'],
    test_size=0.2,
    stratify=df['winner_index'],
    random_state=865
)
```

In [20]:
```
petitioner = X_train[X_train["winner_index"] == 0]
respondent = X_train[X_train["winner_index"] == 1]
print(petitioner.shape)
print(respondent.shape)
```

```
(1691, 2)
(1080, 2)
```

Upsampling train data

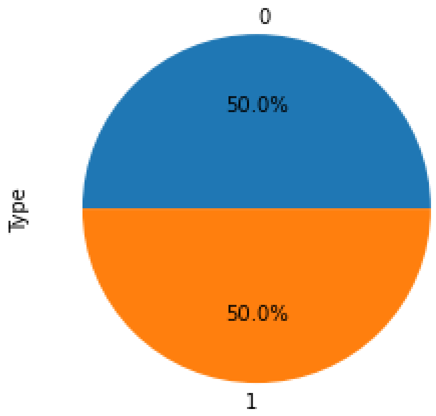
- We upsampled minor class, which is winner index 0 (respondent winning) using sklearn resample.
- Eventually got 1689 cases in each class and shuffled the rows.

In [21]:
```
from sklearn.utils import resample
upsample_respondent = resample(respondent,
             replace=True,
             n_samples=len(petitioner),
             random_state=865)
```

In [22]:
```
upsample_train = pd.concat([upsample_respondent, petitioner])

print(upsample_train["winner_index"].value_counts())

upsample_train.groupby('winner_index').size().plot(kind='pie',
                                        y = "winner_index",
                                        label = "Type",
                                        autopct='%1.1f%%')
```

```
1    1691
0    1691
Name: winner_index, dtype: int64
```

Out[22]: <AxesSubplot:ylabel='Type'>



In [23]:
```
# Let's shuffle things...
shuffled_indices= np.arange(upsample_train.shape[0])
np.random.shuffle(shuffled_indices)
```

In [24]:
```
shuffled_train = upsample_train.iloc[shuffled_indices,:]

X_train= shuffled_train['facts']

y_train = shuffled_train['winner_index']
```

In [25]:
```
# Dropping winner_index in X_test set
X_test = X_test['facts']
```

6. Dense layer with Text Vectorization layer

2-grams + TD-IDF

In [96]:
```
text_vectorization_bi_tfidf = keras.layers.TextVectorization(
    ngrams=2,
    max_tokens=20000,
    output_mode = "tf_idf"
#     standardize=custom_standardization_fn,
#     split=custom_split_fn
)
```

In [102]:
```
text_vectorization_bi_tfidf.adapt(tf.data.Dataset.from_tensor_slices(X_train.values))
```

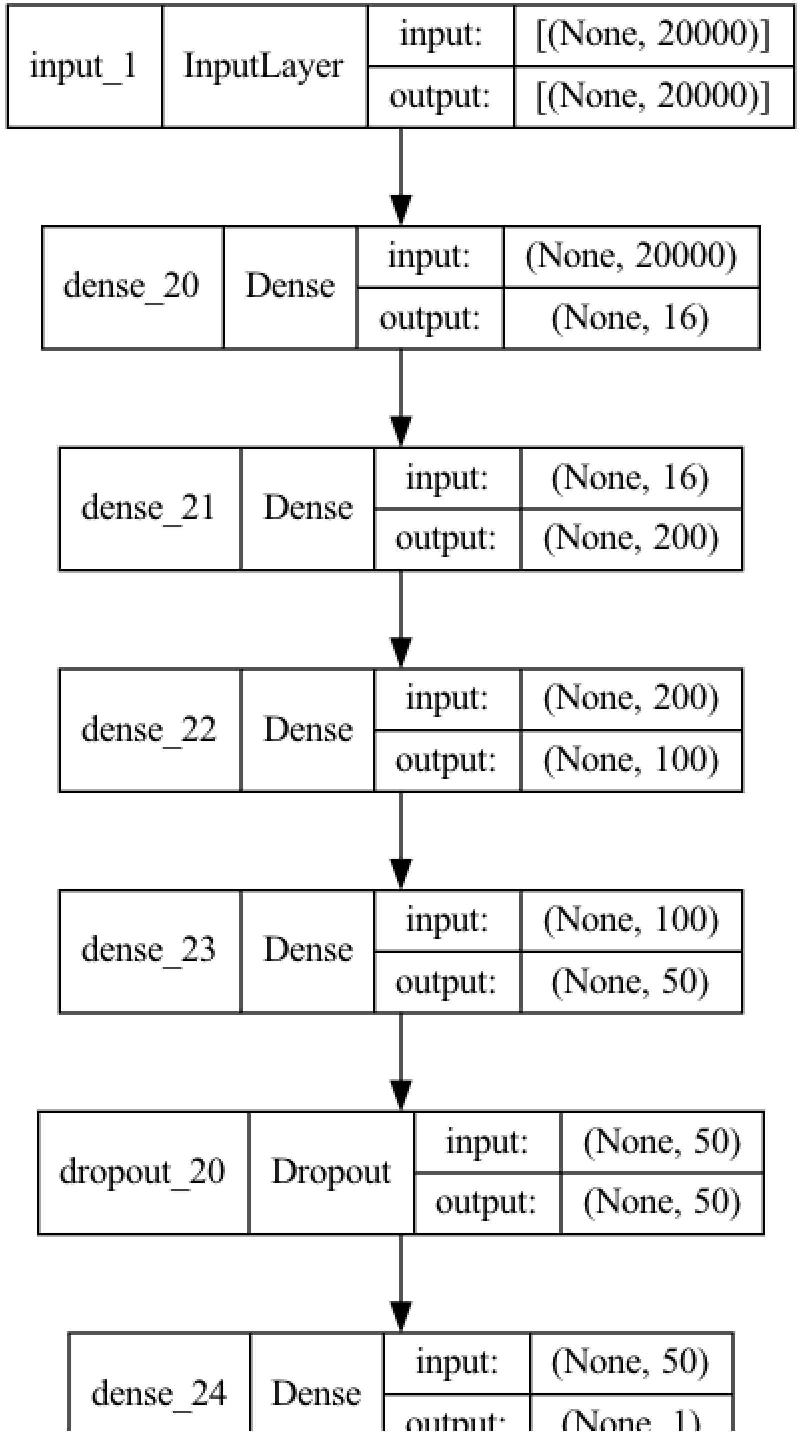
```
2024-02-07 13:41:09.708249: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] P
lugin optimizer for device_type GPU is enabled.
```

In [104]:
```
binary_2gram_tfidf_text = text_vectorization_bi_tfidf(X_train)
binary_2gram_tfidf_text
```

Out[104]:
```
<tf.Tensor: shape=(3382, 20000), dtype=float32, numpy=
array([[ 514.46844  ,   15.966226 ,    4.1964893, ...,    0.       ,
           0.       ,    0.       ],
       [ 608.96265  ,    9.024388 ,    1.3988298, ...,    0.       ,
           0.       ,    0.       ],
       [ 755.9537   ,    9.718572 ,    4.1964893, ...,    0.       ,
           0.       ,    0.       ],
       ...,
       [ 467.22134  ,   10.412756 ,    3.4970746, ...,    0.       ,
           0.       ,    0.       ],
       [1028.9369   ,   19.437143 ,    8.392979 , ...,    0.       ,
           0.       ,    0.       ],
       [ 367.47748  ,    6.2476535,    2.7976596, ...,    0.       ,
           0.       ,    0.       ]], dtype=float32)>
```

In [105]:
```
max_tokens=20000
hidden_dim=16

def td_idf_model():
    inputs = keras.Input(shape=(max_tokens,))
    x = keras.layers.Dense(hidden_dim, activation="relu")(inputs)
    x = layers.Dense(200, activation="relu")(x)
    x = layers.Dense(100, activation="relu")(x)
    x = layers.Dense(50, activation="tanh")(x)
    x = keras.layers.Dropout(0.5)(x)
    outputs = keras.layers.Dense(1, activation="sigmoid")(x)
    model = keras.Model(inputs, outputs)
    model.compile(optimizer="rmsprop",
                  loss="binary_crossentropy",
                  metrics=["accuracy"])


    return model

model_bi_tfidf = td_idf_model()
keras.utils.plot_model(model_bi_tfidf,show_shapes=True)
```

Out[105]:


input_1
InputLayer
input:
[(None, 20000)]
output:
[(None, 20000)]
dense_20
Dense
input:
(None, 20000)
output:
(None, 16)
dense_21
Dense
input:
(None, 16)
output:
(None, 200)
dense_22
Dense
input:
(None, 200)
output:
(None, 100)
dense_23
Dense
input:
(None, 100)
output:
(None, 50)
dropout_20
Dropout
input:
(None, 50)
output:
(None, 50)
dense_24
Dense
input:
(None, 50)

		output:	(None, 1)

In [106]:
```
k = 4
num_validation_samples = len(X_train) // k
num_epochs = 25
batch_sizes = 50
all_loss_histories = []
all_val_loss_histories = []
all_acc_histories = []
all_val_acc_histories = []

# For each validation fold, we will train a full set of epochs, and store the history.
for fold in range(k):
    validation_data = binary_2gram_tfidf_text[num_validation_samples * fold:
                           num_validation_samples * (fold + 1)]
    validation_targets = y_train[num_validation_samples * fold:
                           num_validation_samples * (fold + 1)]
    training_data = np.concatenate([
        binary_2gram_tfidf_text[:num_validation_samples * fold],
        binary_2gram_tfidf_text[num_validation_samples * (fold + 1):]])
    training_targets = np.concatenate([
        y_train[:num_validation_samples * fold],
        y_train[num_validation_samples * (fold + 1):]])

    model_bi_tfidf = td_idf_model()
    callbacks = [keras.callbacks.ModelCheckpoint("tfidf_2gram.keras",
                                    save_best_only=True)
      ]
    history = model_bi_tfidf.fit(training_data, training_targets,
                    validation_data = (validation_data,validation_targets),
                    epochs=num_epochs, batch_size=batch_sizes,callbacks=callbacks)
    #model = keras.models.load_model("tfidf_2gram.keras")

    val_loss_history = history.history['val_loss']
    val_acc_history = history.history['val_accuracy']
    loss_history = history.history['loss']
    acc_history = history.history['accuracy']
    all_val_loss_histories.append(val_loss_history)
    all_loss_histories.append(loss_history)
    all_val_acc_histories.append(val_acc_history)
    all_acc_histories.append(acc_history)

average_loss_history = [np.mean([x[i] for x in all_loss_histories]) for i in range(num_epochs)]
```

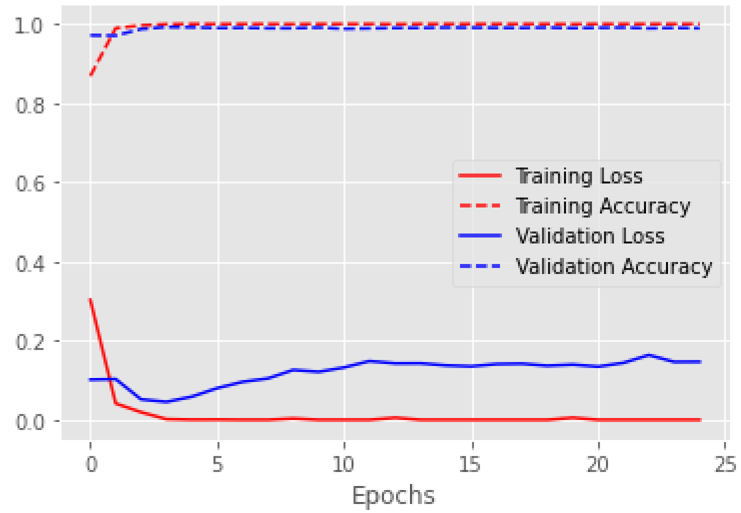
```
average_val_loss_history = [np.mean([x[i] for x in all_val_loss_histories]) for i in range(num_epochs)]
average_acc_history = [np.mean([x[i] for x in all_acc_histories]) for i in range(num_epochs)]
average_val_acc_history = [np.mean([x[i] for x in all_val_acc_histories]) for i in range(num_epochs)]
```

```
Epoch 1/25

2024-02-07 13:42:33.741880: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] P
lugin optimizer for device_type GPU is enabled.

51/51 [==============================] - ETA: 0s - loss: 0.2980 - accuracy: 0.8762

2024-02-07 13:42:37.077380: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] P
lugin optimizer for device_type GPU is enabled.

51/51 [==============================] - 5s 28ms/step - loss: 0.2980 - accuracy: 0.8762 - val_loss: 0.0864
- val_accuracy: 0.9775
Epoch 2/25
51/51 [==============================] - 1s 13ms/step - loss: 0.0482 - accuracy: 0.9874 - val_loss: 0.0762
- val_accuracy: 0.9787
Epoch 3/25
51/51 [==============================] - 1s 13ms/step - loss: 0.0176 - accuracy: 0.9953 - val_loss: 0.0382
- val_accuracy: 0.9905
Epoch 4/25
51/51 [==============================] - 1s 12ms/step - loss: 0.0035 - accuracy: 0.9984 - val_loss: 0.0414
```

In [107]:
```
np.mean(average_val_acc_history)
```

Out[107]:
```
0.9885798817873002
```

In [108]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

plt.plot(average_loss_history,c='r')
plt.plot(average_acc_history,c="r",linestyle="dashed")
plt.plot(average_val_loss_history,c='b')
plt.plot(average_val_acc_history,c='b',linestyle="dashed")
plt.xlabel("Epochs")
plt.legend(['Training Loss','Training Accuracy','Validation Loss','Validation Accuracy'])
plt.show()
```



In [109]:
```
binary_2gram_tf_test = text_vectorization_bi_tfidf(X_test)
binary_2gram_tf_test
```

Out[109]:
```
<tf.Tensor: shape=(693, 20000), dtype=float32, numpy=
array([[ 409.47488  ,    6.2476535,    2.0982447, ...,    0.       ,
           0.       ,    0.       ],
       [ 341.22906  ,    4.165102 ,    4.1964893, ...,    0.       ,
           0.       ,    0.       ],
       [ 797.95105  ,   20.131327 ,    4.8959045, ...,    0.       ,
           0.       ,    0.       ],
       ...,
       [1065.6847   ,   13.18949  ,    5.5953193, ...,    0.       ,
           0.       ,    0.       ],
       [ 157.49034  ,   20.825512 ,    7.693564 , ...,    0.       ,
           0.       ,    0.       ],
       [ 388.4762   ,   11.801123 ,    3.4970746, ...,    0.       ,
           0.       ,    0.       ]], dtype=float32)>
```

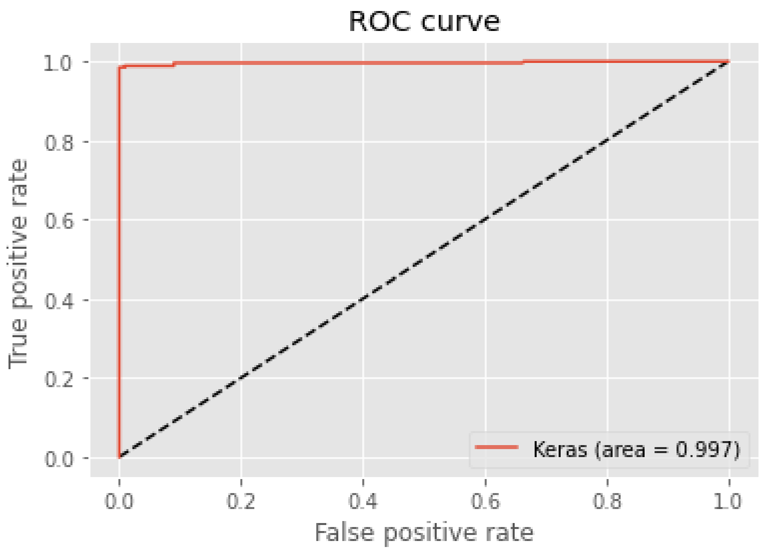
In [110]:
```
model_bi_tfidf.evaluate(binary_2gram_tf_test, y_test)
```

```
22/22 [==============================] - 0s 9ms/step - loss: 0.1162 - accuracy: 0.9913
```

Out[110]:
```
[0.1162339448928833, 0.9913420081138611]
```

In [111]:
```
from sklearn.metrics import roc_curve
y_pred_keras = model_bi_tfidf.predict(binary_2gram_tf_test).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)

from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
2024-02-07 13:43:59.478761: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] P
lugin optimizer for device_type GPU is enabled.
```



6. Conclusion

Model Selection & Interpretation

- **Best model**: Dense layer with text-vectorizatoin(bigram, TD-IDF) performed best(AUC) among our models
  - **Sigmoid/Binary-crossentrophy**: Since our prediction problem was binary classification, we used sigmoid output activation function that it returns values between 0 and 1, which can be treated as probabilities of a data point belonging to binary class. Likewise, we used binary-crossentrophy as loss function.
  - **Test accuracy/AUC**: We measured test accuracy for each model. To choose best model, we generated AUC.
- **LIME**: We used LIMe to explain our model and to see what words in text contributed to the prediction

In [178]:
```
# Create lime explainer
try:
  import lime
  from lime.lime_text import LimeTextExplainer
except ImportError as error:
  !pip install lime
  import lime
  from lime.lime_text import LimeTextExplainer

X_train_array = X_train.to_numpy()
X_test_array = X_test.to_numpy()
y_train_array = y_train.to_numpy()
y_test_array = y_test.to_numpy()

class_names=['petitioner_winning','respondent_winning']
explainer=LimeTextExplainer(class_names=class_names)

def new_predict(text):
  vectorized = text_vectorization_bi_tfidf(text)
  padded = keras.preprocessing.sequence.pad_sequences(vectorized, maxlen=20000,padding='post')
  pred=model_bi_tfidf.predict(padded)
  pos_neg_preds = []
  for i in pred:
    temp=i[0]
    pos_neg_preds.append(np.array([1-temp,temp])) #I would recommend rounding temp and 1-temp off to 2 places
  return np.array(pos_neg_preds)
```

Input your new case

One case only

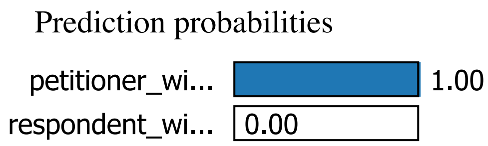
In [187]:
```
petitioner = 'Carol Howes, Warden'

respondent = ' Randall Lee Fields'

facts = ' A jury found Randall Fields guilty of two counts of third-degree criminal sexual conduct for the se
```

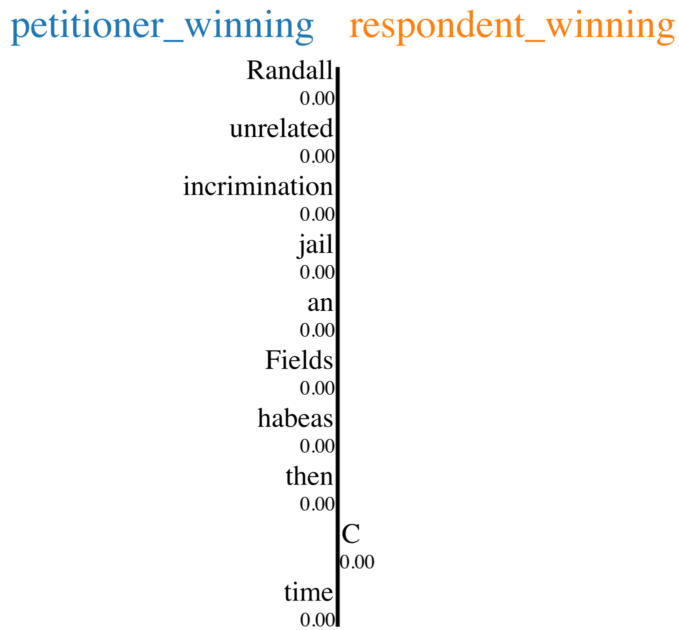
In [188]:
```
expt = petitioner + respondent + facts
```

In [189]:
```
expt
```

Out[189]:
```
'Carol Howes, Warden Randall Lee Fields A jury found Randall Fields guilty of two counts of third-degree cr
iminal sexual conduct for the sexual abuse of a thirteen-year-old child. Fields was in jail on a disorderly
charge when Lenawee County, Michigan deputies questioned him about allegations of sex with a minor. The sex
case was unrelated to the one Fields was in jail for at the time. Fields filed an appeal of right in the Mi
chigan Court of Appeals claiming that his statements were inadmissible because he had not been given his Mi
randa warnings before questioning. The state court reasoned that because Fields was free to return to the j
ail and was questioned on a matter unrelated to his incarceration, there was no obligation to provide him w
arnings under Miranda. Fields then filed a petition for a writ of habeas corpus under 28 U.S.C. § 2254 clai
ming that his Fifth Amendment right against self-incrimination was violated, and the U.S. District Court ag
reed. The United States Court of Appeals for the Sixth Circuit affirmed.'
```

In [190]:
```
explainer.explain_instance(expt,new_predict).show_in_notebook(text=True)
```





Text with highlighted words

Carol Howes, Warden Randall Lee Fields A jury found Randall Fields guilty of two counts of third-degree criminal sexual conduct for the sexual abuse of a thirteen-year-old child. Fields was in jail on a disorderly charge when Lenawee County, Michigan deputies questioned him about allegations of sex with a minor. The sex case was unrelated to the one Fields was in jail for at the time. Fields filed an appeal of right in the Michigan Court of Appeals claiming that his statements were inadmissible because he had not been given his Miranda warnings before questioning. The state court reasoned that because Fields was free to return to the jail and was questioned on a matter unrelated to his

More cases

In [203]:
```
vectorized = text_vectorization_bi_tfidf([expt])
padded = keras.preprocessing.sequence.pad_sequences(vectorized, maxlen=20000,padding='post')
```

In [205]:
```
model_bi_tfidf.predict(padded).round(2)
```

Out[205]:
```
array([[0.]], dtype=float32)
```

In [ ]:

Suggestion

- **Cross-validation with upsampled data**: For better measurement, we could have done upsampling manually in each cross validation folds. However, since our goal was exploring multiple NN models, upsampling in each folds hurted runtime efficiency and code-reuse. We decided to upsample train set first. As we kept test set aside, we obtained a valid measure of model performance on test set.
- **Domain specific pretrained model**: We could further work using domain specific pretrained model. We found https://github.com/ashkonf/LeGloVe (https://github.com/ashkonf/LeGloVe), which is python implementation of GloVe word vectors for legal domain-specific corpuses.
- **Gather more features**: In Oyez database, we could find more information such as advocate, location, lower court and date. Gathering this information as new features might be able to improve our model performance.