



MATCH COMMENTARY SUMMARIZATION

KAASHAN PANJWANI - J035

SIDH SATAM – J042



TEXT SUMMARIZATION

Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document.



WHY TEXT SUMMARIZATION?

Propelled by the modern technological innovations, data is to this century what oil was to the previous one. Today, our world is parachuted by the gathering and dissemination of huge amounts of data.

In fact, the International Data Corporation (IDC) projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025. That's a lot of data!

With such a big amount of data circulating in the digital space, there is need to develop machine learning algorithms that can automatically shorten longer texts and deliver accurate summaries that can fluently pass the intended messages.

Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.



TWO MAIN APPROACHES FOR TEXT SUMMARIZATION

Extraction-based summarization:

The extractive text summarization technique involves pulling key phrases from the source document and combining them to make a summary. The extraction is made according to the defined metric without making any changes to the texts.

e.g - Source text: Joseph and Mary rode on a donkey to attend the annual event in Jerusalem. In the city, Mary gave birth to a child named Jesus.

Extractive summary: **Joseph and Mary attend event Jerusalem. Mary birth Jesus.**

As you can see above, the words in bold have been extracted and joined to create a summary — although sometimes the summary can be grammatically strange.



Abstraction-based summarization:

The abstraction technique entails paraphrasing and shortening parts of the source document. When abstraction is applied for text summarization in deep learning problems, it can overcome the grammar inconsistencies of the extractive method.

The abstractive text summarization algorithms create new phrases and sentences that relay the most useful information from the original text — just like humans do.

Therefore, abstraction performs better than extraction. However, the text summarization algorithms required to do abstraction are more difficult to develop; that's why the use of extraction is still popular.

Source text: Joseph and Mary rode on a donkey to attend the annual event in Jerusalem. In the city, Mary gave birth to a child named Jesus.

Abstractive summary: Joseph and Mary came to Jerusalem where Jesus was born.



DATASET

<https://www.kaggle.com/secareanualin/football-events>

The central element is the text commentary.

The dataset provides a granular view of 9,074 games, totaling 941,009 events from the biggest 5 European football (soccer) leagues: England, Spain, Germany, Italy, France from 2011/2012 season to 2016/2017 season as of 25.01.2017.

The dataset is organized in 3 files:

- events.csv contains event data about each game.
- ginf.csv - contains metadata and market odds about each game
- dictionary.txt contains a dictionary with the textual description of each categorical variable coded with integers



FEATURE EXTRACTION

Word Embeddings

It replaces each word in your dict with a list of numbers.

We use Glove300 embeddings.



Models available

Cornerstone model

To implement this task, researchers use a deep learning model that consists of 2 parts, an encoder, that understands the input, and represent it in an internal representation, and feed it to another part of the network which is the decoder.

The main deep learning network that is used for these 2 parts in a LSTM, which stands for long short term memory, which is a modification on the rnn

In the encoder we mainly use a multi-layer bidirectional LSTM, while in the decoder we use an attention mechanism with beam search.



Pointer Generator

This is actually a neural network that is trained to learn when to generate novel words, and when to copy words from the original sentence.

It is called a pointer generator network as we use a pointer to point out to the word would be copied from the original sentence.

But researchers found 2 main problems with the above which is

- 1) The inability of the network to copy Facts (like names , and match scores) as it doesn't copy words , it generates them , so it sometimes incapable of generating facts correctly
- 2) Repetition of words



Using Reinforcement learning with deep learning

It is about combining two fields together, it actually uses the pointer generator in its work, and uses the same prepossessed version of the data .

It is used to fix 2 main problems with the cornerstone implementation which are -

- 1) The decoder in the training, uses the output from the encoder, the actual summary and then uses its current output for the next action, while in testing it doesn't have a ground truth, as it is actually needed to be generated, so it only uses output from the encoder and then uses its current output for the next action. This causes an Exposure Problem
- 2) The training of the network relies on a metric for measuring the loss , which is different from the metric used in testing, as the metric used in training is the cross entropy loss, while the metric for the testing is non-differentiable measures such as BLEU and ROUGE

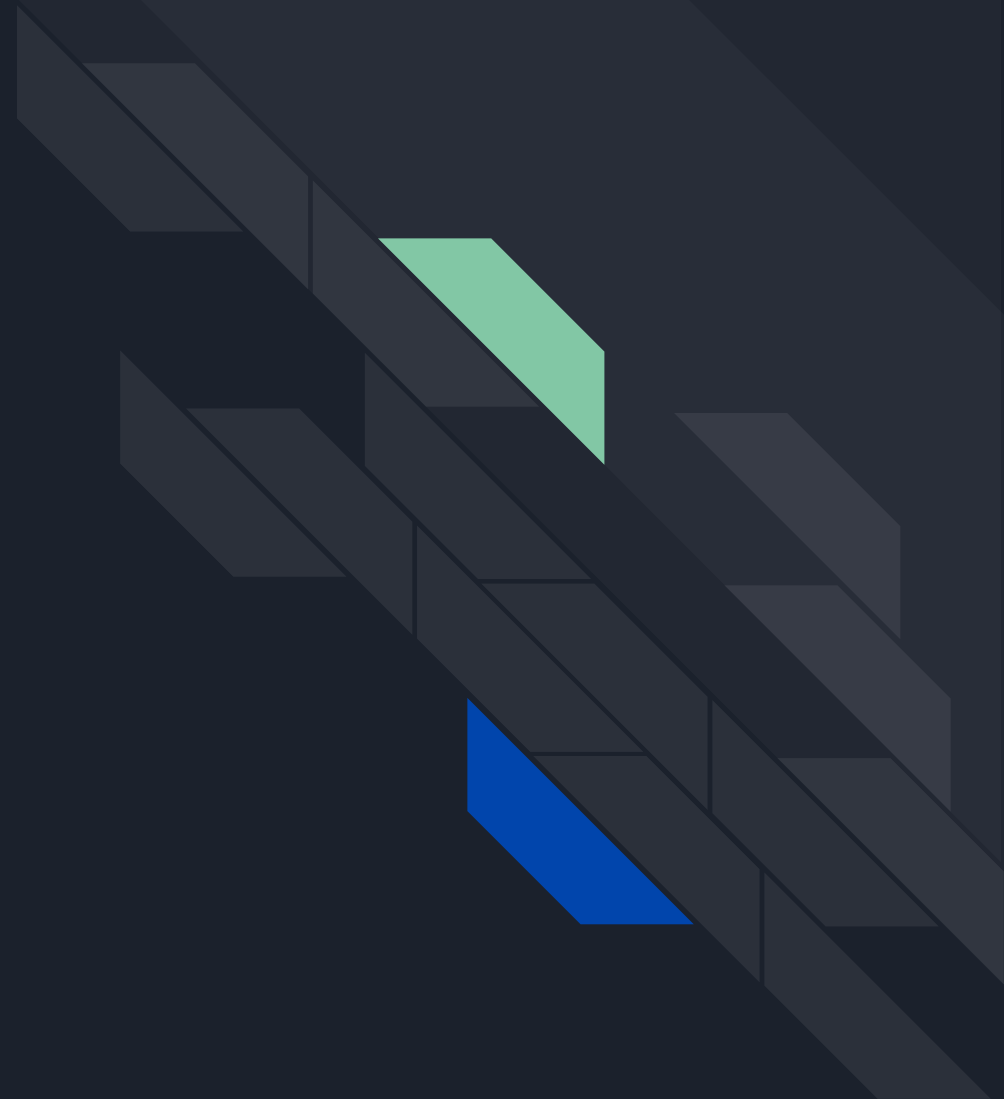


MODEL SELECTION

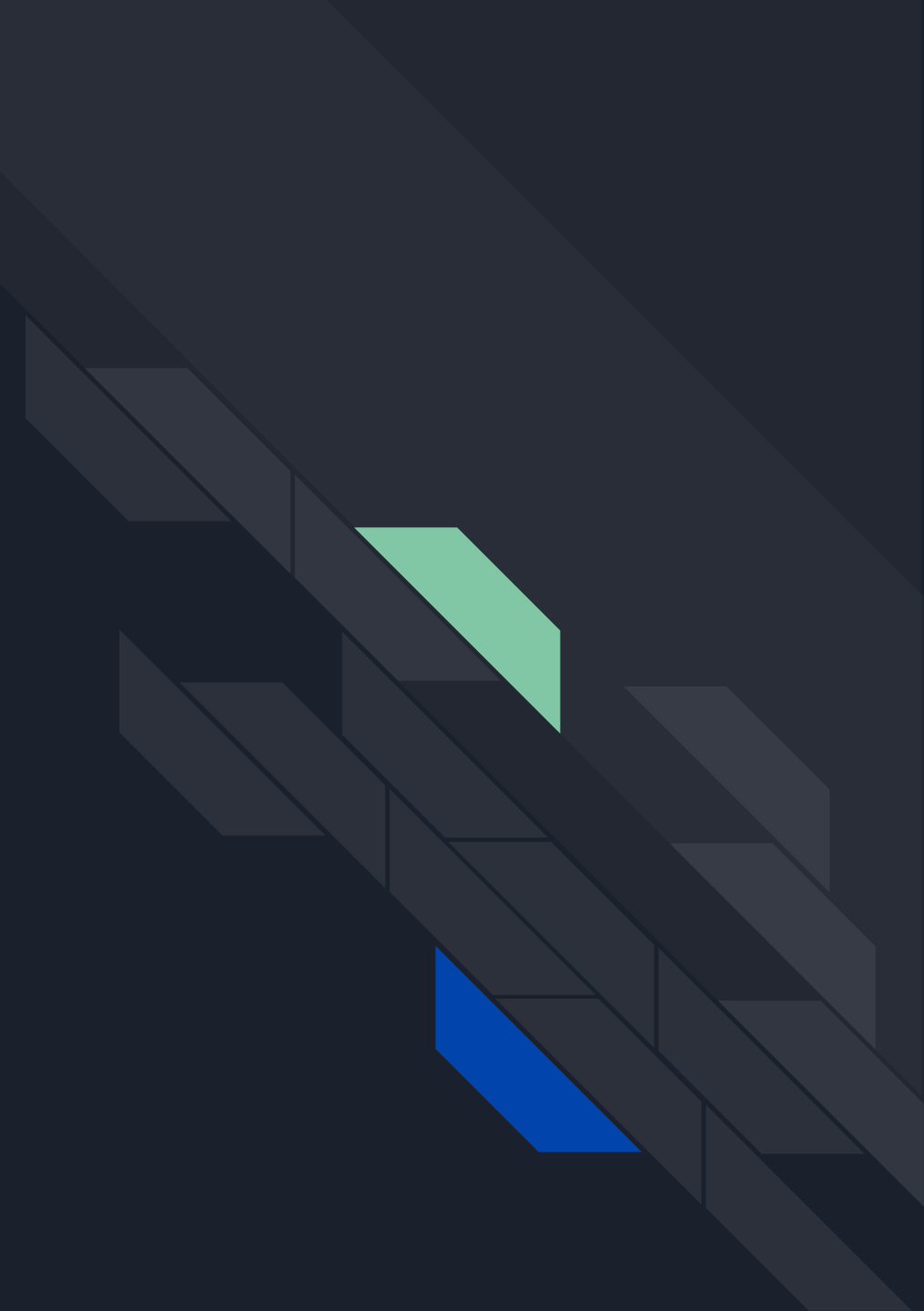
Our approach is creating 2 models:

- **DEEP LEARNING:** Cornerstone Model
As we use the output of BERT extractive summarizer to generate ground truth, our reference summaries is actually extractive summary itself which is why we choose the Cornerstone Model which is purely abstractive
- **MACHINE LEARNING:** Ridge Classifier and Decision Tree Classifier
For Expected Goal Analysis, to prevent overfitting from occurring, we add a bias to the model (Ridge Regularisation) in order to get a good testing accuracy as opposed to just a simple Logistic classifier.
For Match Outcome Prediction we use a Decision Tree Classifier to predict Match outcome based on Odds and Difference of Odds calculated

Machine Learning



Expected Goal Analysis





REQUIRED LIBRARIES -

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split as tts
    from sklearn.linear_model import LogisticRegression, RidgeClassifier
    from sklearn.metrics import confusion_matrix
    import matplotlib.pyplot as plt
    import matplotlib
    import seaborn as sns
```



DATA PREPROCESSING

```
[ ] location = {  
    1 : 'Attacking half',  
    2 : 'Defensive half',  
    3 : 'Centre of the box',  
    4 : 'Left wing',  
    5 : 'Right wing',  
    6 : 'Difficult angle and long range',  
    7 : 'Difficult angle on the left',  
    8 : 'Difficult angle on the right',  
    9 : 'Left side of the box',  
    10 : 'Left side of the six yard box',  
    11 : 'Right side of the box',  
    12 : 'Right side of the six yard box',  
    13 : 'Very close range',  
    14 : 'Penalty spot',  
    15 : 'Outside the box',  
    16 : 'Long range',  
    17 : 'More than 35 yards',  
    18 : 'More than 40 yards',  
    19 : 'Not recorded'  
}
```

```
[ ] bodypart = {  
    1 : 'right foot',  
    2 : 'left foot',  
    3 : 'head'  
}
```

```
[ ] assist_method = {  
    0 : 'None',  
    1 : 'Pass',  
    2 : 'Cross',  
    3 : 'Headed pass',  
    4 : 'Through ball'  
}
```

```
[ ] situation = {  
    1 : 'Open play',  
    2 : 'Set piece',  
    3 : 'Corner',  
    4 : 'Free kick'  
}
```

DATA PREPROCESSING (CONTD.)

Preprocessing for Outcome Prediction and Expected Goals Analysis

```
[ ] attempts.replace({'location':location, 'bodypart':bodypart, 'assist_method': assist_method, 'situation':situation}, inplace=True)
```

```
[ ] attempts.head()
```



	is_goal	location	bodypart	assist_method	situation	fast_break
0	0	Left side of the box	left foot	Pass	Open play	0
11	0	Outside the box	right foot	Pass	Open play	0
13	1	Left side of the box	left foot	Pass	Open play	0
14	0	Outside the box	right foot	None	Open play	0
17	0	Outside the box	right foot	None	Open play	0

```
[ ] y = attempts.iloc[:,0]
X = pd.get_dummies(attempts.iloc[:,1:], columns=['location', 'bodypart', 'assist_method', 'situation'])
```




RIDGE CLASSIFIER FOR XG ANALYSIS

- We used Ridge Regression in this.
- We fit the model on our training set and then predict for our testing set, and then check for the classification report, which gives us the accuracy of the model.

```
[ ] ridge = RidgeClassifier()  
    ridge.fit(X_train, y_train)
```

```
👤 RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,  
                  max_iter=None, normalize=False, random_state=None,  
                  solver='auto', tol=0.001)
```

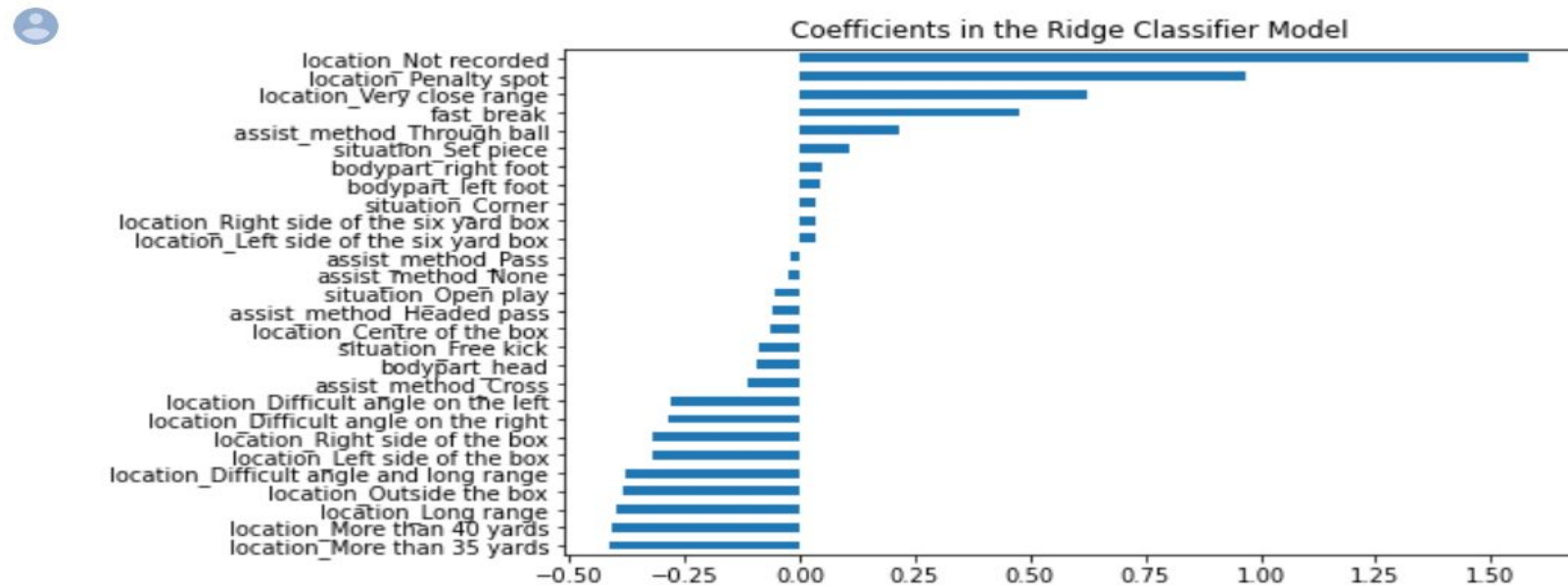
```
[ ] ridge.score(X_test, y_test)
```

```
👤 0.9096172589866309
```

Analysis of Factors affecting Expected Goal

```
[ ] coef = pd.Series(ridge.coef_[0], index = X_train.columns).sort_values()

matplotlib.rcParams['figure.figsize'] = (8.0, 5.0)
coef.plot(kind = "barh")
plt.title("Coefficients in the Ridge Classifier Model")
plt.show()
```



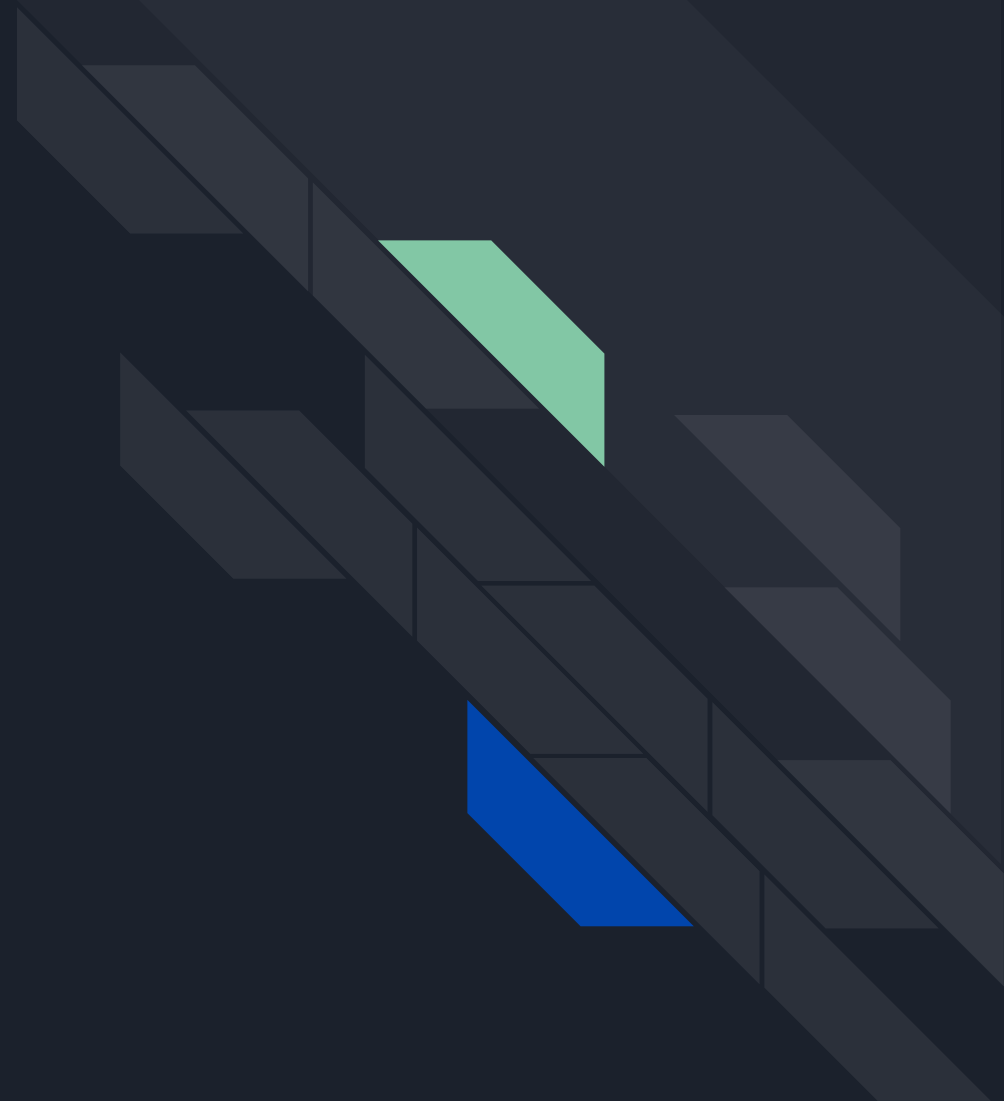
Confusion Matrix of Ridge Regression Model

```
[ ] cm = confusion_matrix(y_test, ridge.predict(X_test))
df_cm = pd.DataFrame(cm, index = ['Not Goal', 'Goal'], columns=['Not Goal', 'Goal'])
plt.figure(figsize=(7,5))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size

plt.show()
```



Match Outcome Prediction





REQUIRED LIBRARIES

We use Decision Tree Classifier for this approach

```
[ ] from sklearn.model_selection import train_test_split as tts  
    from sklearn.tree import DecisionTreeClassifier  
    from sklearn.metrics import accuracy_score, confusion_matrix
```



Preprocessing


```
[ ] def actual_result_encode(fthg,ftag):  
    if fthg>ftag:  
        return (1)  
    elif fthg==ftag:  
        return (2)  
    elif fthg<ftag:  
        return (3)
```

```
[ ] outcome_decode={1: 'Home Wins',  
                     2: 'Draw',  
                     3: 'Away Wins'  
                     }
```

Creating variables for outcome prediction

```
[ ] result = []
    for i in range(1, ginf.shape[0]+1):
        result.append(actual_result_encode(ginf[i-1:i]["fthg"].item(), ginf[i-1:i]["ftag"].item()))
    y = ginf[['id_odsp']]
    y['outcome'] = result
    y.set_index("id_odsp", inplace=True)

    X1 = ginf.iloc[:, [0, 9, 10, 11, 12, 13]]
    X1.set_index("id_odsp", inplace=True)
    X1["diff_h_d"] = abs(X1["odd_h"] - X1["odd_d"])
    X1["diff_d_a"] = abs(X1["odd_d"] - X1["odd_a"])
    X1["diff_h_a"] = abs(X1["odd_h"] - X1["odd_a"])
    X1 = X1.drop(["fthg", "ftag"], axis=1)
    X1.head()
```



	odd_h	odd_d	odd_a	diff_h_d	diff_d_a	diff_h_a
id_odsp						
UFot0hit/	1.56	4.41	7.42	2.85	3.01	5.86
Aw5DfILH/	2.36	3.60	3.40	1.24	0.20	1.04
bkjpaC6n/	1.83	4.20	4.80	2.37	0.60	2.97
CzPV312a/	1.55	4.50	9.40	2.95	4.90	7.85
GUOdmrtl/	2.50	3.40	3.45	0.90	0.05	0.95

Decision Tree Classifier for Outcome Prediction

```
[ ] X_train, X_test, y_train, y_test = tts(X1, y, test_size=0.3, random_state=42)

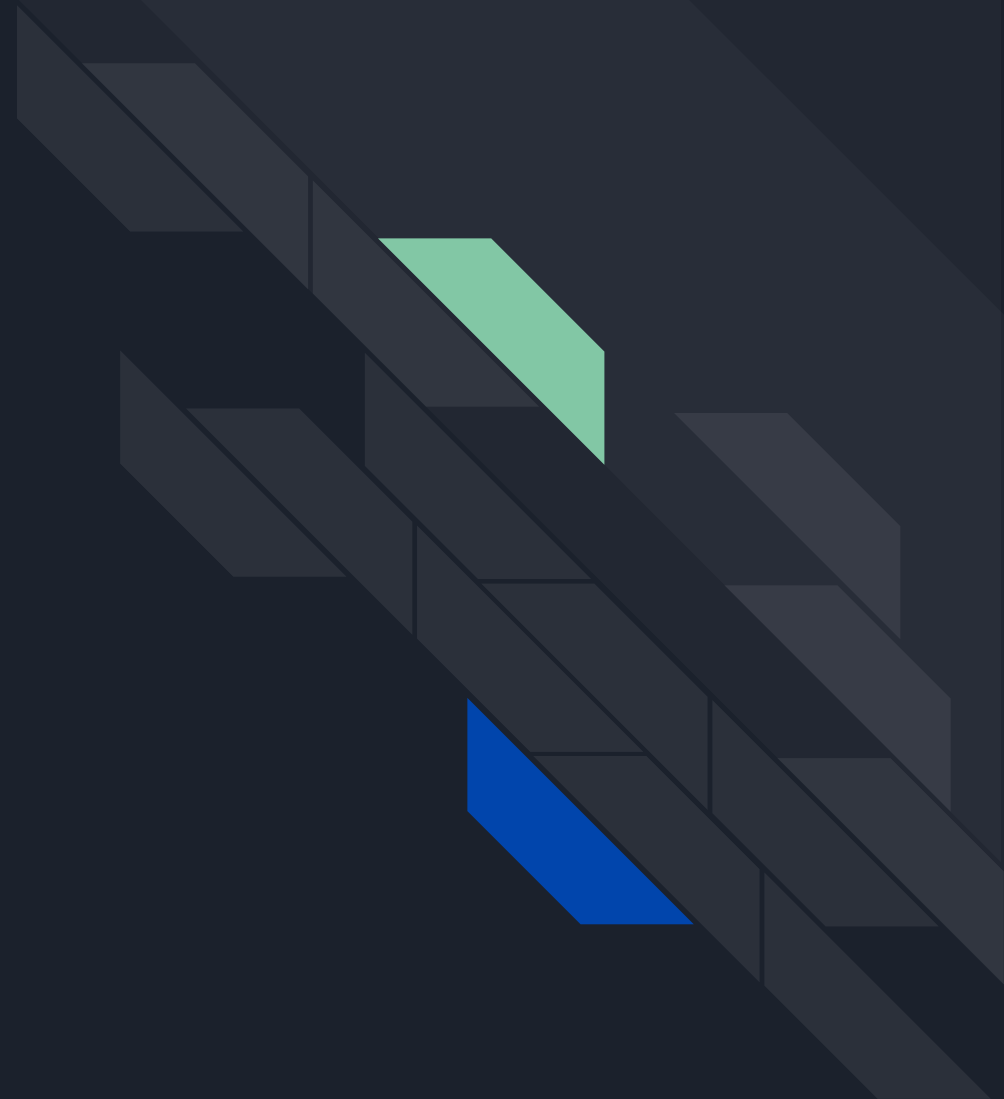
[ ] dt = DecisionTreeClassifier()
    dt.fit(X_train, y_train)
    print("Decision Tree Classifier Accuracy: " + str(accuracy_score(y_train, dt.predict(X_train))))
    cm = confusion_matrix(y_train, dt.predict(X_train))
    cm = pd.DataFrame(cm)
    cm.columns = ["Predicted Home Win", "Predicted Draw", "Predicted Away Win"]
    cm.index = ["Actual Home Win", "Actual Draw", "Actual Away Win"]
    cm
```



Decision Tree Classifier Accuracy: 0.979513987001978

	Predicted Home Win	Predicted Draw	Predicted Away Win
Actual Home Win	3228	4	1
Actual Draw	68	1758	0
Actual Away Win	44	28	1947

Deep Learning





DEEP LEARNING

- Getting Data ready
- Create the Model
- Fit the Model
- Optimize the Model

Since the models were taking too long to run, we ran them on Google Cloud Platform.



GETTING DATA READY

- For getting our ground truths, we must first run a BERT extractive summariser. Thus, we call the summariser and run it on each of the text cells in our dataframe column.
- We create a function to get all our clean texts and thus, get a list of all the texts.
- We then go on to creating a dictionary in order to converting our texts in numbers and ultimately creating our finally dataframe to work with.



Creating the model

Following the Cornerstone approach, we use multi-layer bidirectional LSTM cells to create our Encoder and LSTM with attention (in training and testing) and beam search (in testing) cells to create our Decoder



FITTING THE MODEL

- Model is fit on entire dataset of 5009 match commentaries for 10 epochs



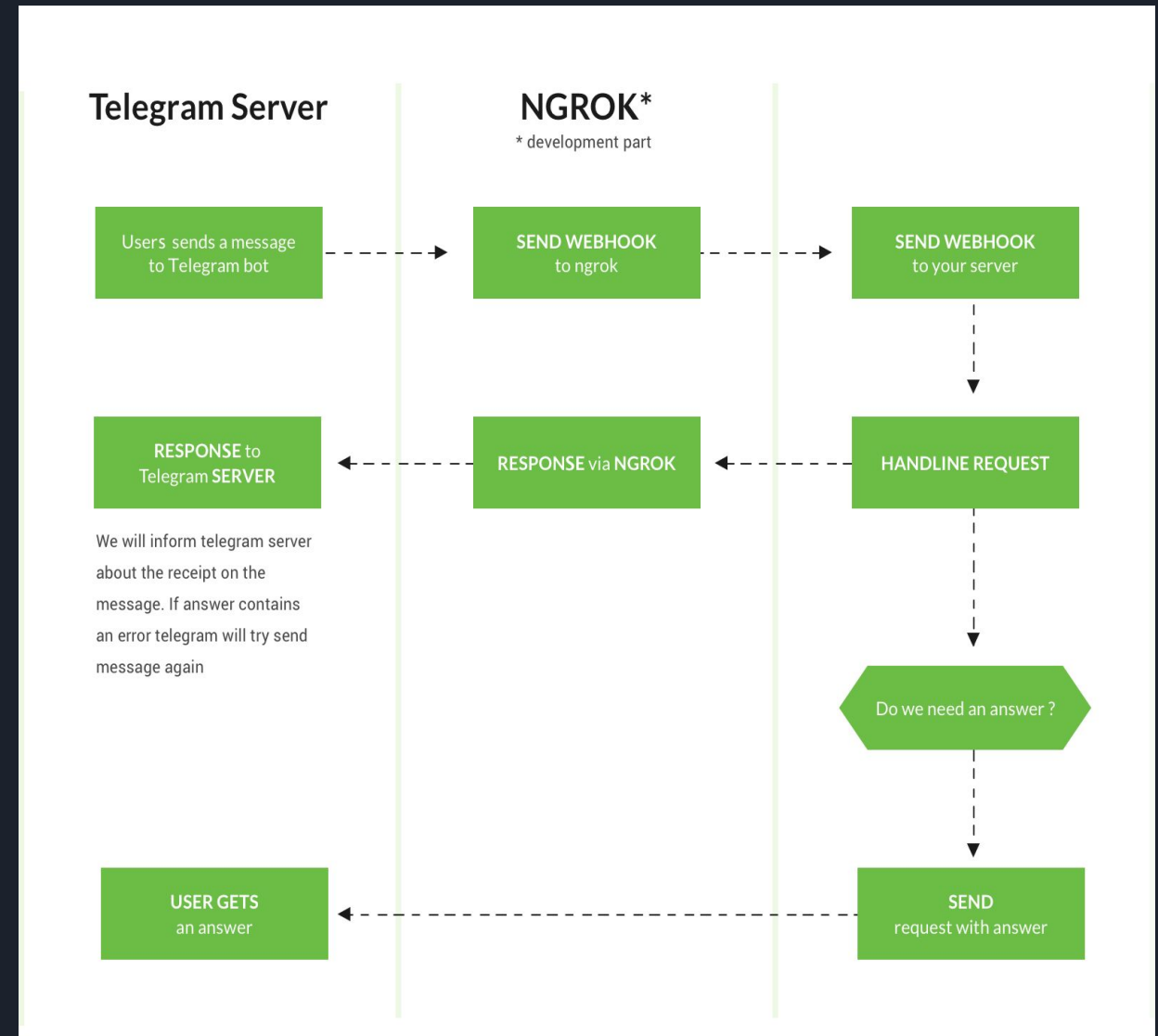
Generate Abstractive Summaries

Using the trained model we generate summaries on the entire dataset and store it to a dataframe

Deployment

Deployed on telegram bot

1. Create new bot with @BotFather
2. Create and run bot.py
3. Start ngrok
4. Set WebHook



Match Summarizer Telegram Bot

Shows list of matches, Outcome and Summary for each match and XG Analysis

