**Movie Title Based On Input Plot Description**

**Participants:**

1. Hari Kiran Keerthipati - Writes code for the project, as well as take care of the deployment of the application on to the internet. Will develop a simple web interface where users can input the plot, actors and other details and the website will output the movie name based on the given input details.
2. Sasidhar Yalamanchili - Takes care of the documentation, report preparation, final project presentation and other details of the project.
3. Aashish Eada - Takes care of the wireframes and other design details required for the project.

* We will have a meeting every Wednesday in Willis Library for 1hour 30minutes or by zoom and discuss the technical and functional details of the project.
* Discuss the project for 30 minutes every Monday after class.

**Abstract:**

This is a user-centric project designed to streamline the process of retrieving movie titles from a dataset based on user-provided plot descriptions and other input details that may be entered by the user like actor's names, genre, and release year. By using the power of Natural Language Processing and a movie database containing approximately 35,000 movies, this project empowers the users to accurately discover movie titles.

This project serves as a valuable resource for a wide range of users, including movie enthusiasts, and individuals in need of identifying movie titles from plot descriptions. Whether users are exploring film databases, conducting research, or simply seeking to retrieve movie titles efficiently, our project provides a convenient solution.

**Data Abstraction:**

|  |  |
| --- | --- |
| **Feature** | **Datatype** |
| Release year | Int |
| Title | String |
| Origin/ethnicity | String |
| Director | String |
| Cast | String |
| Genre | String |
| Wiki page | String |
| Plot | String |

Release year: Year in which the movie was released.

Title: Name of the movie

Origin/ethnicity: Place where the movie was made and to which region the specific movie belongs to.

Director: The person who is responsible for the creative and technical aspects of the film.

Cast: The actors who portray different characters in a movie

Genre: It refers to the category that groups the films based on the theme, narration, and features.

Wiki page: A Wikipedia link to the movie.

Plot: It is the story and sequence of events and actions that makeup the storyline and structure of the film. It is the theme around which the entire movie revolves around.

**Project Design:**

**Technologies, IDEs and Packages**:

1. Jupyter Notebook - We will extensively use jupyter notebook to write all the code required for the project.
2. Django - We will use Django to create a web application for the project. Django is a backend framework which is used to develop the backend of the web applications.
3. HTML/CSS/JS - We will use these frontend technologies to design the frontend functionality of the web application.
4. PyCharm - We will use the PyCharm IDE to write the code that runs the web application.
5. **Packages**: Numpy, Pandas, Tensorflow, Pytorch, Seaborn, Matplotlib, Keras, Natural Language Toolkit, Spacy, TextBlob.

**Flow Chart:**A screenshot of a computer screen

Description automatically generated

**State of the art:**

* There are only one or two sources on the internet which will figure out the movie name given the movie plot.
* There are several projects which predict the movie genre given the plot summary.

1. Movie Title Prediction from Plot using BART:

* Bart uses a standard seq2seq/machine translation architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT).
* Takes in the movie plot and produces the movie name as output.
* It has achieved a training loss of 3.02.
* Drawbacks: It used only the plot summary of the movie to make a prediction. **We intend to use plot summaries, actor names, genres, Origin and Director as input when building our application. This way our output predictions will be much more accurate compared to just using the movie plot.**

1. Movie title predictor using LSTM.

* Achieved an accuracy of 80% on the test set.
* Uses only the plot summary of the movie to make the prediction.
* Uses RMSprop as loss function.
* Drawbacks: Uses only the movie plot as an input.

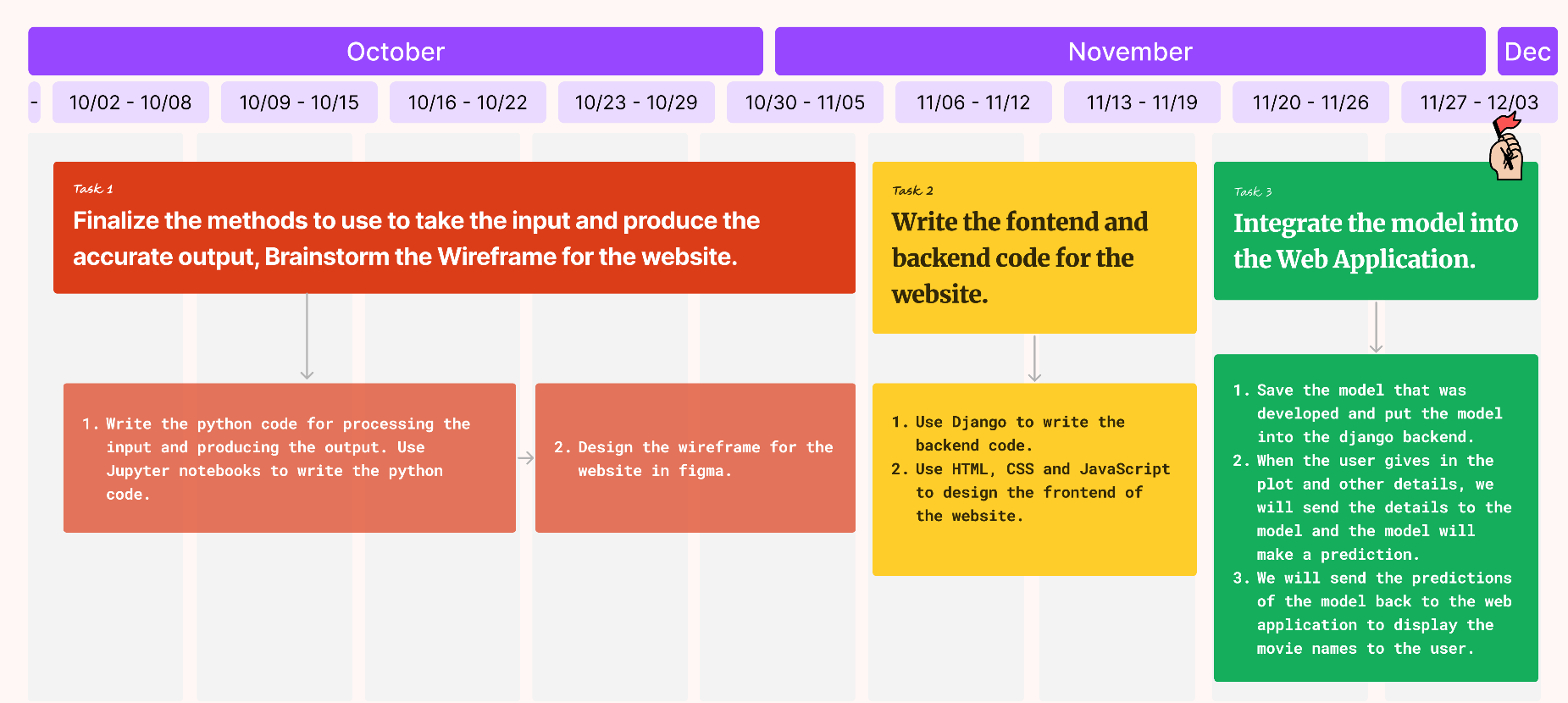
1. Movie title predictor Doc2Vec

* Uses Doc2Vec model to predict the movie name given the movie plot.
* Drawbacks: Uses only the movie plot as an input.

There are not many sources which are available. We can clearly see that this project is considerably unique and not a lot of people have worked on this area of fetching the movie name given the plot.

**What We Will Improve:**

* **We will improve upon those models using multiple features from the input instead of restricting ourselves to just the plot of the movie.**
* **We will also develop a user-friendly website so that any individual can go to our website and find out the movie name if they have forgotten about it.**

**Project Milestones:**

**Project Expected Result:**

1. Produce an output movie name given the movie plot.
2. Take in the user input from a website, process the user input and display the movie results back to the user.

**Project Functions:**

1. We have designed three different models for our project.
   1. Multinomial Naïve Bayes model
   2. TF-IDF model
   3. Doc2Vec model

* **Multinomial Naïve Bayes model:**
  + We built this model using the MultinomialNB classifier available in sklearn library.
  + Data was preprocessed and then fed to the model and the accuracy of the model is calculated.
  + **A screenshot of a computer

    Description automatically generated**
* **TF-IDF Model:**
  + The data was preprocessed.
  + Using the TfidfVectorizer class in sklearn, the inputs and the plots were converted to vectors.
  + The cosine\_similarity between the plot vectors and the title vectors is calculated.
  + Accuracy is calculated using the different comparisons of the top n similarity scores.
  + **A screenshot of a computer program

    Description automatically generated**
  + **A screenshot of a computer program

    Description automatically generated**
* **Doc2Vec Model:**
  + The data is preprocessed using the gensim library.
  + All the input documents are tagged using the TaggedDocument class in gensim.
  + Doc2Vec Model is built, and the accuracy is calculated.
  + The model is saved for later use.
  + **A screenshot of a computer program

    Description automatically generated**
* **Project Result:**
  + For Naïve bayes, we have achieved an accuracy of 0.3% when trained on the movies data. Since this accuracy is well below the benchmarks, this model is rejected and thrown away.
  + For TF-IDF, we got an accuracy of 97.45% when we compare the true label with 300 top movies. Although this may seem good enough, the user is tasked with searching through 300 movies to find the movie that he/she is looking for. So, clearly this model is not good enough for users. So, this model is also discarded.
  + For Doc2Vec model, which captures the semantic meaning of entire documents or paragraphs, which can be very helpful for similarity searching, we have achieved a 100% accuracy on the test set even if we have paraphrased the plot using online tools.
* **Result Evaluation:**
  + **Naïve Bayes:**

|  |  |
| --- | --- |
| **Model** | **Accuracy (%)** |
| Multinomial Naïve Bayes | 0.3 |

* **TF-IDF:**

|  |  |  |
| --- | --- | --- |
| **Model** | **N** | **Accuracy (%)** |
| TF-IDF | 5 | 1.43 |
| TF-IDF | 10 | 2.86 |
| TF-IDF | 50 | 20.06 |
| TF-IDF | 100 | 40.12 |
| TF-IDF | 150 | 55.88 |
| TF-IDF | 200 | 73.08 |
| TF-IDF | 250 | 88.85 |
| TF-IDF | 300 | 97.45 |

* **Doc2Vec Model:**

|  |  |
| --- | --- |
| Model | Accuracy (%) |
| Doc2Vec | 100 |

**Project Code:**

**Bayes:**

movies\_df = pd.read\_csv("wiki\_movie\_plots\_deduped.csv")

#%%

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english', max\_features=10000)

X = tfidf\_vectorizer.fit\_transform(movies\_df['Plot']) # Fitting and transforming the input using tf-idf vectorizers.

y = movies\_df['Title']

#%%

# Splitting the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#%%

multnomial\_classifier = MultinomialNB()

multnomial\_classifier.fit(X\_train, y\_train)

#%%

y\_pred = multnomial\_classifier.predict(X\_test)

#%%

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100 :.2f}%")

**tf-idf:**

movies\_df = pd.read\_csv("wiki\_movie\_plots\_deduped.csv")

#%%

plots = movies\_df["Plot"].tolist()

titles = movies\_df["Title"].tolist()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(plots, titles, test\_size=0.2, random\_state=42)

#%%

def remove\_the\_punc\_and\_spaces(text):

tmp = re.sub(r'[^\w\s]', '', text)

tmp = re.sub(r'\s+', ' ', tmp)

return tmp

def preprocess\_text(text):

text = text.lower()

return remove\_the\_punc\_and\_spaces(text)

preprocessed\_plots = [preprocess\_text(plot) for plot in X\_train]

preprocessed\_titles = [preprocess\_text(title) for title in y\_train]

#%%

vectorizer\_tf\_idf = TfidfVectorizer()

plot\_vectors = vectorizer\_tf\_idf.fit\_transform(preprocessed\_plots)

title\_vectors = vectorizer\_tf\_idf.transform(preprocessed\_titles)

#%%

cosine\_similarity\_matrix = cosine\_similarity(plot\_vectors, title\_vectors)

#%%

def retrieve\_movie\_titles(input\_plot, num\_top=5):

pre\_processed\_plot = preprocess\_text(input\_plot)

plot\_vector = vectorizer\_tf\_idf.transform([pre\_processed\_plot])

# find the similarity between plot description and all movie titles.

similarity\_scores = cosine\_similarity(plot\_vector, title\_vectors)[0]

sorted\_titles\_and\_scores = sorted(zip(titles, similarity\_scores), key=lambda x: x[1], reverse=True)

top\_5\_movies = [title for title, score in sorted\_titles\_and\_scores[:num\_top]]

return top\_5\_movies

#%%

def evaluate\_model(num\_top=5):

true\_labels = y\_test

prediction\_labels = []

for plot in X\_test:

prediction\_labels.append(retrieve\_movie\_titles(plot, num\_top=num\_top))

correct\_predictions = 0

for true\_label, predicted\_labels in zip(true\_labels, prediction\_labels):

if true\_label in predicted\_labels:

correct\_predictions += 1

return correct\_predictions / len(true\_labels)

#%%

n = 5

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 10

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 15

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 20

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 30

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 50

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 75

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 100

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 120

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 150

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 200

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 250

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

#%%

n = 300

print(f"Percentage of correct predictions based on top {n} similarity scores: {evaluate\_model(n) \* 100}%")

**Doc2Vec:**

movies\_df = pd.read\_csv("/kaggle/input/wikipedia-movie-plots/wiki\_movie\_plots\_deduped.csv", sep=",")

#%%

movies\_df.info()

#%% md

# Code to create the Doc2Vec Model

#%%

plot\_text = movies\_df["Plot"].values

text\_processed = preprocess\_documents(plot\_text)

#%% md

# Tagging the documents with TaggedDocument Class

#%%

tagged\_docs = [TaggedDocument(d, [i]) for i, d in enumerate(text\_processed)]

#%% md

# Building the doc2vec model

#%%

doc2vec\_model = Doc2Vec(tagged\_docs, dm=0, vector\_size=200, window=2, min\_count=1, epochs=100, hs=1)

#%% md

# Saving the model.

#%%

doc2vec\_model.save('doc2vec\_whole\_data\_set.model')

df = pd.read\_csv("wiki\_movie\_plots\_deduped.csv", sep=",")

test\_df = pd.read\_csv("Test\_Data\_Doc2vec.csv", sep=",")

#%%

def evaluate\_model():

correct\_predictions = 0

for true\_label, plot in zip(test\_df['Title'].values, test\_df['Plot'].values):

predicted\_labels = []

plt = gensim.parsing.preprocessing.preprocess\_string(plot)

test\_doc\_vector = model.infer\_vector(plt)

sims = model.docvecs.most\_similar(positive=[test\_doc\_vector])

for s in sims:

predicted\_labels.append(df['Title'].iloc[s[0]])

if true\_label in predicted\_labels:

correct\_predictions += 1

return round(correct\_predictions / len(test\_df['Plot'].values) \* 100)

#%%

print(f"The model has an accuracy of {evaluate\_model()}%")

**Website Screenshots:**

**Website Screenshots:**

**Asking input from the user:**

A screenshot of a computer

Description automatically generated

**Entering the user input:**

A screenshot of a computer

Description automatically generated

**Top 10 Predictions on display:**

A screenshot of a computer

Description automatically generated

**References:**

1. Rupawala, A., Pujara, D., Shikalgar, M., & Ukey, E. (2020, March). Movie Genre Prediction from Plot Summaries by Comparing Various Classification Algorithms. IRJET, 7(3)
2. Balraj, A. (2020). Movie Title Prediction from Plot using BART. Kaggle. <https://www.kaggle.com/code/balraj98/movie-title-prediction-from-plot-using-bart>
3. Bhanuprakash. (2020). Movie Title Predictor. Kaggle. <https://www.kaggle.com/code/bhanuprakash06/movie-title-predictor>
4. “The Internet Movie Script Database (IMSDb).” [Online]. Available: <https://imsdb.com/>
5. M. J. Cafarella and O. Etzioni, “A search engine for natural language applications,” in Proceedings of the 14th international conference on World Wide Web, 2005, pp. 442–452.
6. R. Dhir and A. Raj, “Movie Success Prediction using Machine Learning Algorithms and their Comparison,” in 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC). IEEE, 2018, pp. 385–390
7. L. Michelbacher, “Multi-word tokenization for natural language processing,” 2013
8. https://dev.to/dcodeyt/creating-beautiful-html-tables-with-css-428l
9. https://codepen.io/green-plastic/pen/JjKjON
10. https://www.kaggle.com/datasets/jrobischon/wikipedia-movie-plots/code