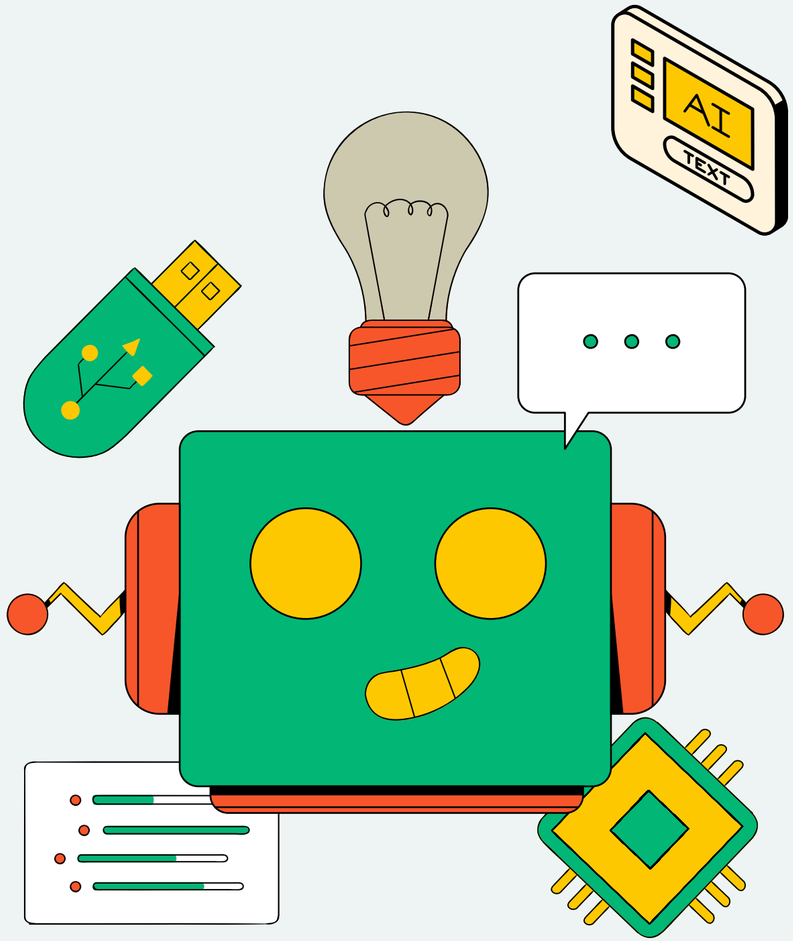
CS688 – WEB ANALYTICS
AND MINING



BOOSTING TEST SCORES WITH AI: AUTOMATING CORRECT STATEMENT IDENTIFICATION

TERM PROJECT

PRESENTED BY:

AMINA BAUYRZHAN

PROBLEM: HOW TO CHEAT ON CS688?

Which of the following statements are correct?

Selected Answers:

a. The text transforms available within “tm” package are applied with the function "tm_map()".

b. The R package “quanteda” has been designed for quantitative text analysis.

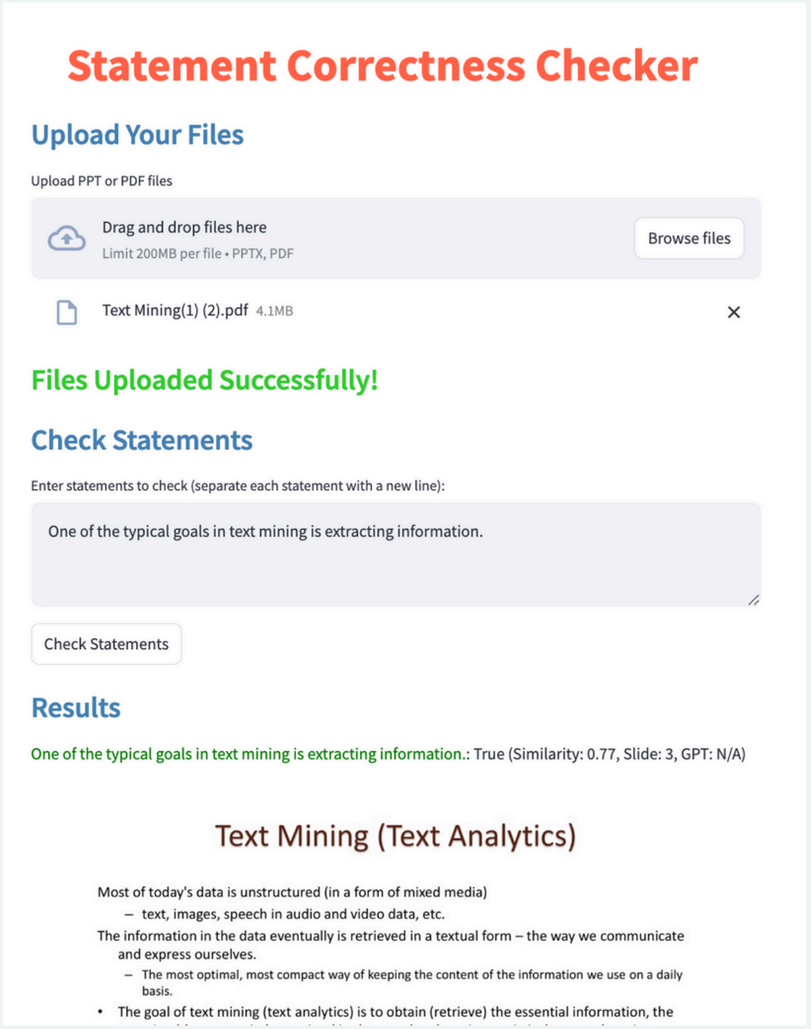
c. The R package “spacyr” adds the powerful functionality of the advanced NLP library “spaCy” to R.

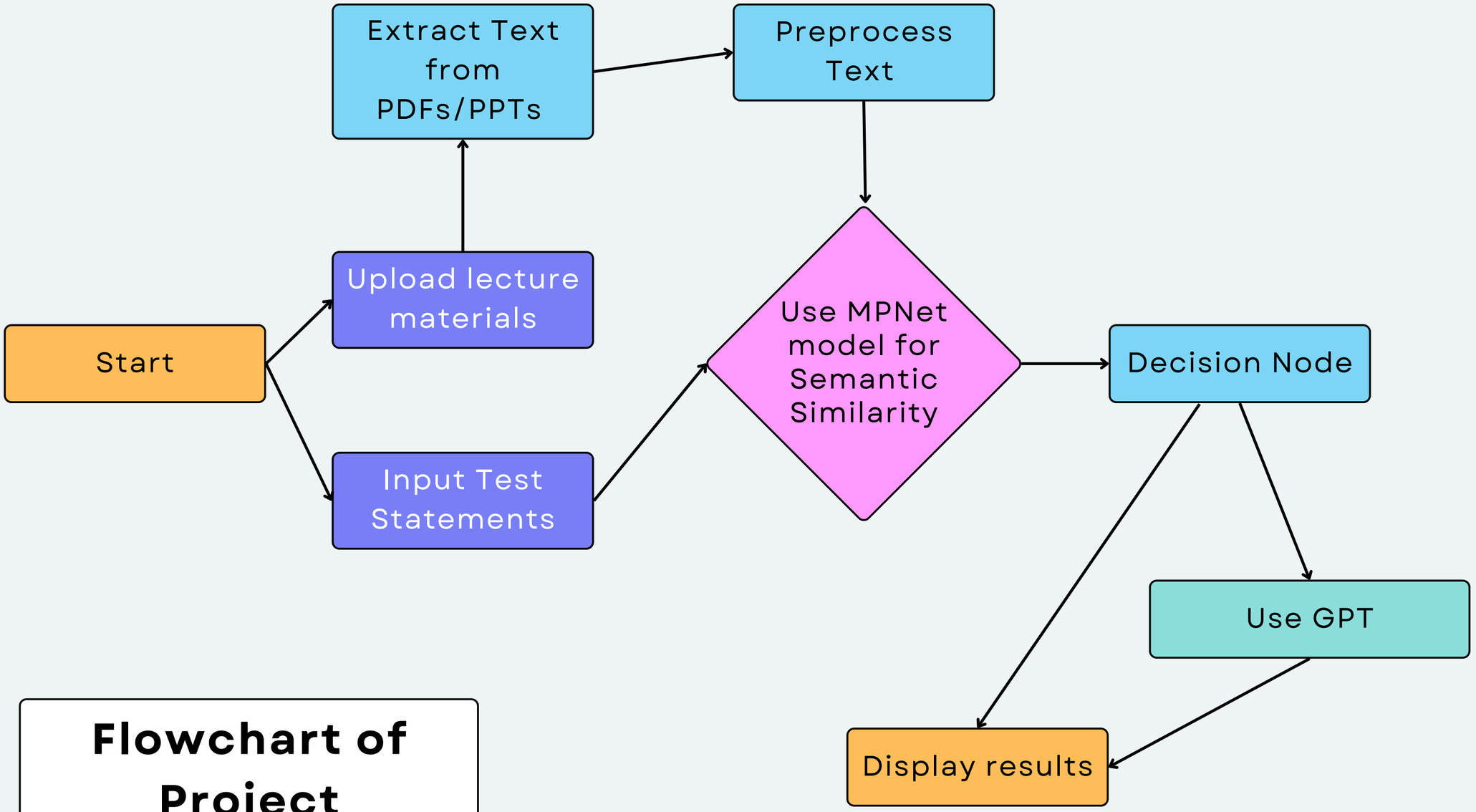
d. The R package “textclean” is used to clean and process text.

In the CS688 course, students are required to identify and validate correct statements during tests based on extensive lecture materials provided in PDF and PowerPoint formats. These materials often contain vast amounts of information, making it challenging for students to quickly and accurately find the correct statements under time constraints.

OBJECTIVE

The goal is to develop an application that automates the process of identifying and validating correct statements from lecture materials using advanced Natural Language Processing (NLP) techniques, including MPNet(from BERT family) and GPT models.





Flowchart of Project

KEY COMPONENTS

- Text Extraction
- Preprocessing
- Semantic Analysis with MPNet model
- GPT Integration
- Streamlit Interface

TEXT EXTRACTION

Purpose:

- Convert lecture materials (PDFs, PPTs) into text for analysis.

PDF Extraction:

- Used pdfminer to extract and segment text by pages.

PPT Extraction:

- Used python-pptx to extract text from slides.

Output:

- Structured text ready for preprocessing

```python
def extract_text_from_ppt(file):
    prs = Presentation(file)
    slides_text = []
    for i, slide in enumerate(prs.slides):
        slide_text = []
        for shape in slide.shapes:
            if hasattr(shape, "text"):
                slide_text.append(shape.text)
        slides_text.append({"slide_number": i + 1, "text": " ".join(slide_text)})
    return slides_text

def extract_text_from_pdf(file):
    pdf_text = extract_text(file)
    pages = pdf_text.split("\f")
    pages_text = []
    for i, page in enumerate(pages):
        pages_text.append({"slide_number": i + 1, "text": page})
    return pages_text


def merge_pdfs(uploaded_files):
    merger = PdfMerger()
    merged_file = BytesIO()
    for file in uploaded_files:
        merger.append(file)
    merger.write(merged_file)
    merger.close()
    merged_file.seek(0)
    return merged_file
```

PREPROCESSING

Purpose:

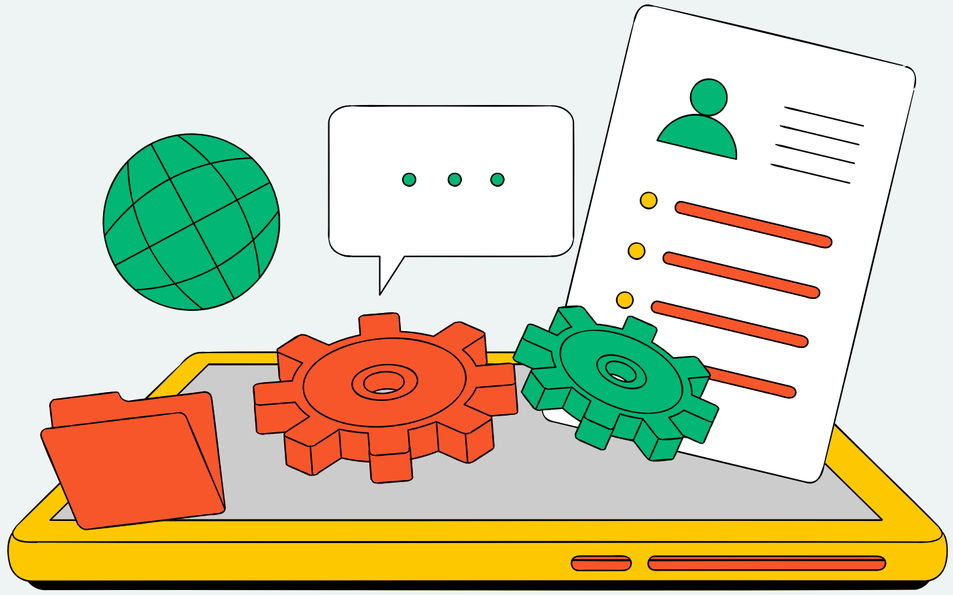
- Prepare extracted text for semantic analysis by cleaning

Text Cleaning:

- Remove unwanted characters, symbols, and formatting
- Convert text to lowercase

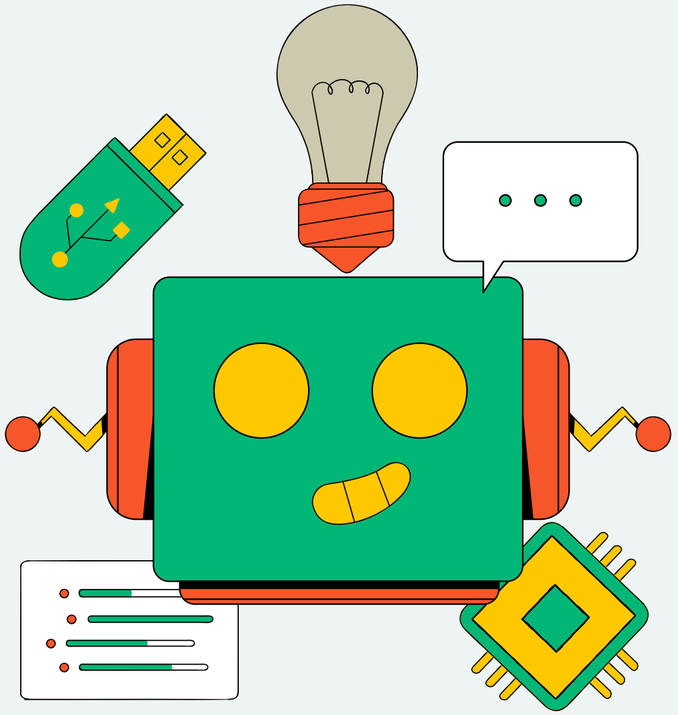
Output:

- The result is a clean, standardized, and concise text, ready for accurate semantic analysis



```
def preprocess_text(text):
    text = re.sub(r'[\-\-\•]\s*\n', ' ', text)
    text = re.sub(r'^\s*[\•\-\–]\s*', '', text, flags=re.MULTILINE)
    text = text.lower()
    text = re.sub(r'[^\w\s\-\.\,\'\"]', '', text)
    return text
```

SEMANTIC ANALYSIS



Sliding Window Technique:

- Text from lecture slides is divided into smaller, overlapping segments using a sliding window approach.
- Each segment (window) is analyzed independently to capture all possible contexts related to the test statements.

Embedding Generation:

- For each window of text, embeddings are generated using the MPNet model.
- Embeddings represent the semantic meaning of the text, allowing for a meaningful comparison with test statements.

```python
# Load SentenceTransformer model
@st.cache_resource
def load_model():
    return SentenceTransformer('paraphrase-mpnet-base-v2')

def sliding_window(text, window_size, step_size):
    words = text.split()
    for i in range(0, len(words) - window_size + 1, step_size):
        yield " ".join(words[i:i + window_size])

def get_embeddings(text, model):
    text = preprocess_text(text)
    return model.encode(text, convert_to_tensor=True)
```

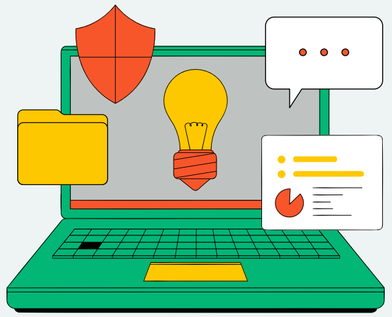
MPNet (Masked and Permuted Network)

- Architecture: MPNet is a transformer-based model, similar to BERT, but with enhancements. It introduces a more complex training objective that combines the strengths of BERT's masked language modeling and XLNet's permuted language modeling. This combination allows MPNet to better capture the dependencies between tokens, leading to improved performance on various NLP tasks.
- Model Size: The base version of MPNet has 12 layers, 768 hidden dimensions, and 12 attention heads, similar to BERT-base.
- Training: MPNet was trained on a large corpus of text data using self-supervised learning techniques. It was fine-tuned for the specific task of generating sentence embeddings, which makes it suitable for tasks like semantic search, clustering, and paraphrase identification.

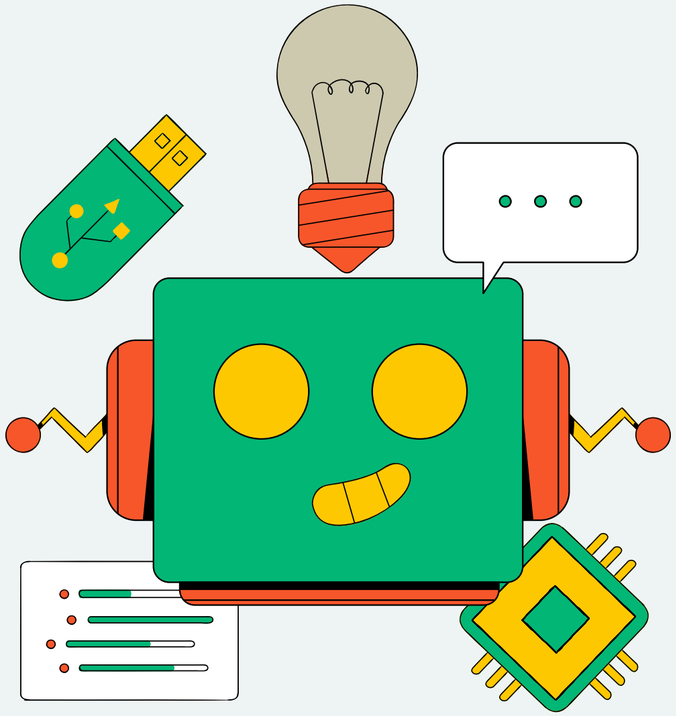
paraphrase-mpnet-base-v2

- Purpose: This specific model is fine-tuned to capture semantic similarities between sentences, making it ideal for tasks where you need to compare how closely related different pieces of text are.
- Use Case: It's commonly used in applications like:
  - Semantic search: Finding the most relevant documents or sentences from a database.
  - Clustering: Grouping similar sentences or documents together.
  - Paraphrase detection: Identifying sentences that have the same meaning but are phrased differently.
- Performance: The model has been fine-tuned on a large dataset of paraphrase pairs and related sentence pairs, giving it a strong ability to understand the underlying meaning of sentences rather than just matching keywords.
- Output: When you input a sentence or a document into this model, it generates a dense vector (embedding) that represents the semantic meaning of the text. These embeddings can then be compared using cosine similarity to measure how similar two sentences are.

In summary, paraphrase-mpnet-base-v2 is a highly effective model for tasks that require understanding and comparing the meanings of different sentences or pieces of text. It builds upon and enhances the foundational ideas from BERT while leveraging MPNet's advanced architecture for better performance in specific NLP tasks.



SEMANTIC ANALYSIS



Similarity Calculation:

- Cosine similarity is calculated between the embeddings of the statement and each text window.
- This similarity score determines how closely the statement matches the content of the lecture slides.

Decision Making:

- Statements are classified as True or False based on predefined similarity thresholds.

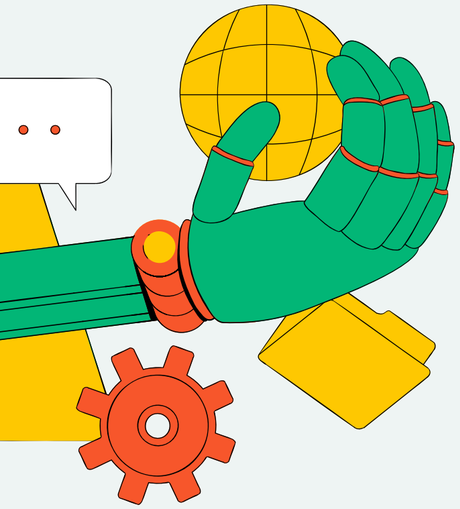
```python
def check_statements(slides, statements, model, window_size=30, step_size=10):
    results = {}
    similarity_threshold_true = 0.6
    similarity_threshold_negation = 0.55

    for statement in statements:
        logging.info(f"Checking statement: {statement}")
        negated = is_negated(statement)
        best_match = {"slide_number": None, "similarity": 0, "window_text": ""}
        statement_embedding = get_embeddings(statement, model)

        for slide in slides:
            slide_number = slide["slide_number"]
            for window_text in sliding_window(slide["text"], window_size, step_size):
                window_embedding = get_embeddings(window_text, model)
                similarity = util.pytorch_cos_sim(window_embedding, statement_embedding)
                similarity_value = similarity.item()
                logging.info(f"Comparing with window text: {window_text}, similarity: {similarity_value}")

                if similarity_value > best_match["similarity"]:
                    best_match = {"slide_number": slide_number, "similarity": similarity_value, "window_text": window_text}

        if negated:
            if best_match["similarity"] >= similarity_threshold_negation:
                results[statement] = {"result": 'False', "similarity": best_match["similarity"], "slide_number": best_match["s
            else:
                results[statement] = {"result": 'True', "similarity": best_match["similarity"], "slide_number": best_match["sl
        else:
            if best_match["similarity"] >= similarity_threshold_true:
                results[statement] = {"result": 'True', "similarity": best_match["similarity"], "slide_number": best_match["sl
            else:
```



GPT INTEGRATION

```python
def check_statement_with_gpt(statement):
    prompt = f"Check if the following statement is true or false:\n\n{statement}"
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt},
        ],
        max_tokens=50,
        temperature=0.0,
    )
    return response['choices'][0]['message']['content'].strip()
```

Purpose:

- Handle cases where mpnet's similarity score is ambiguous or inconclusive

When GPT is Used:

- GPT is invoked when mpnet’s confidence is below or above a certain threshold, indicating potential ambiguity

GPT’s Role:

- GPT (Generative Pre-trained Transformer) re-analyzes these flagged statements
- Evaluates the context and meaning more deeply, providing a final True/False classification

Process:

- mpnet identifies ambiguous statements → GPT analyzes these cases → Adjusted True/False classification is provided

STREAMLIT INTERFACE

Statement Correctness Checker

Upload Your Files

Upload PPT or PDF files

Drag and drop files here
Limit 200MB per file • PPTX, PDF
Browse files

Text Mining(1) (2).pdf 4.1MB

Files Uploaded Successfully!

Check Statements

Enter statements to check (separate each statement with a new line):

Edit-distance is another type of algorithm for fuzzy string-matching, in which is based on the minimum operations needed to transform one string into another.
Fuzzy string-matching do not use any linear algebra concepts.
One of the typical goals in text mining is extracting information.

Set Ambiguity Thresholds

0.40 0.60

0.00 1.00

Use GPT for ambiguous cases

Check Statements

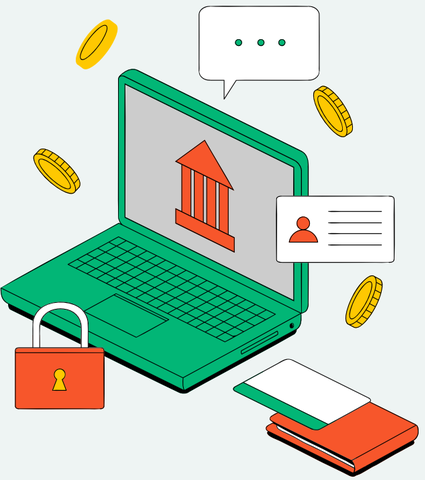
STREAMLIT INTERFACE

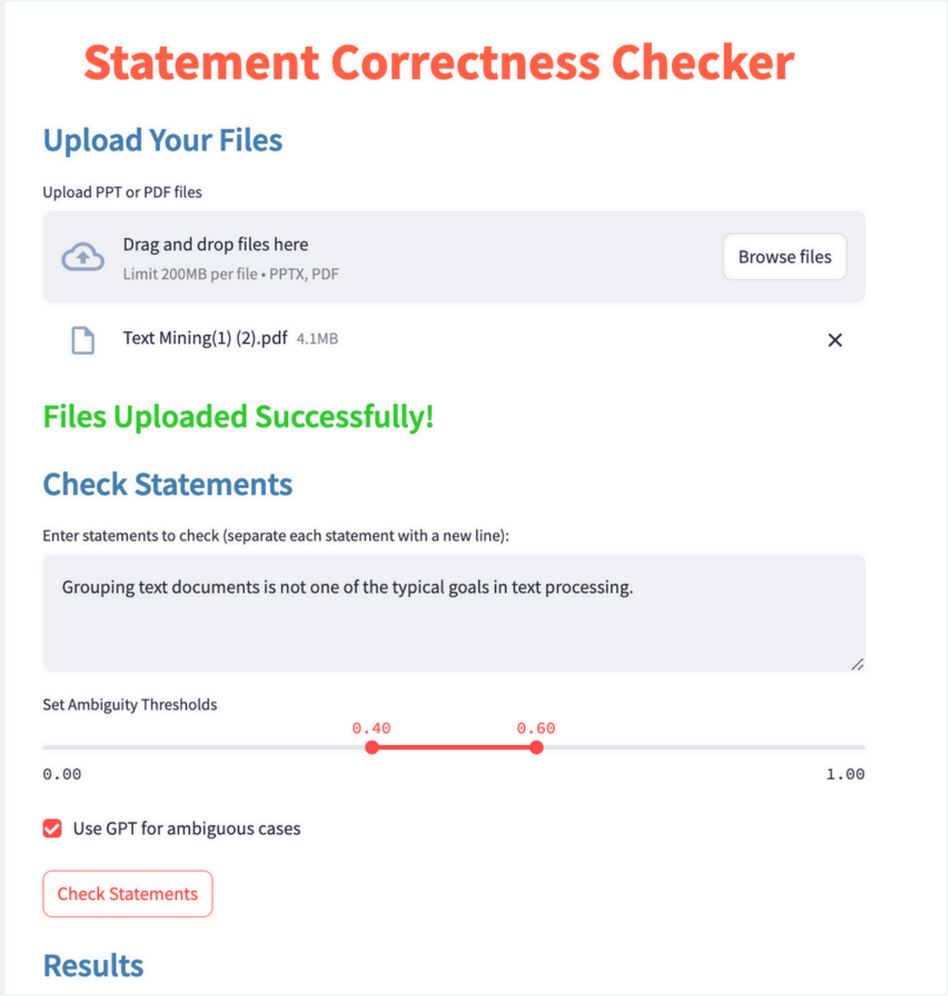
File Uploader Section:

- Supports uploading multiple PDFs and PPTs
- Drag-and-drop functionality for easy file upload
- Feedback provided once files are successfully uploaded

Statement Input and Analysis:

- Text area for entering multiple test statements
- Button to initiate analysis

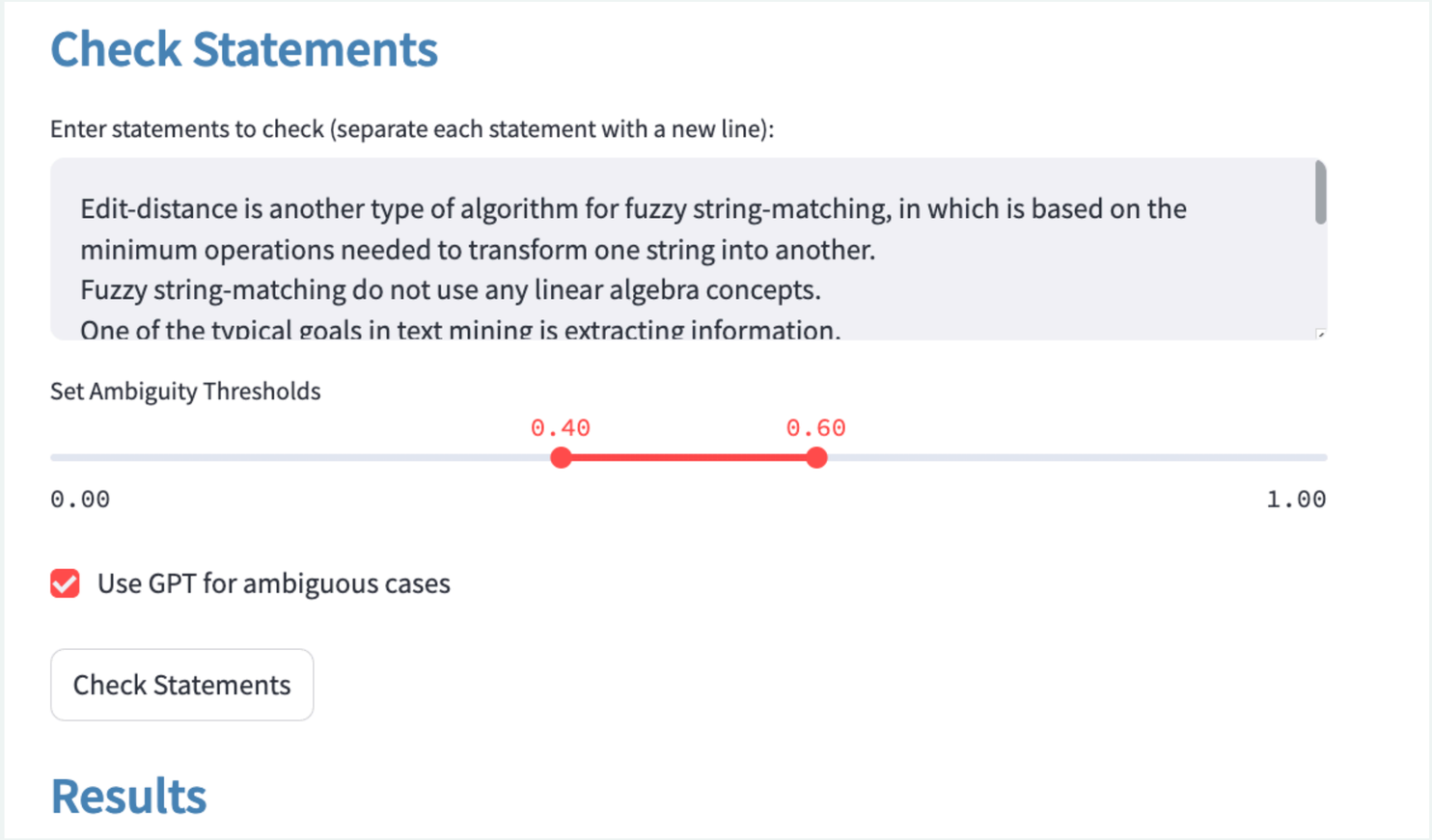




STREAMLIT INTERFACE

Set Ambiguity Tresholds:

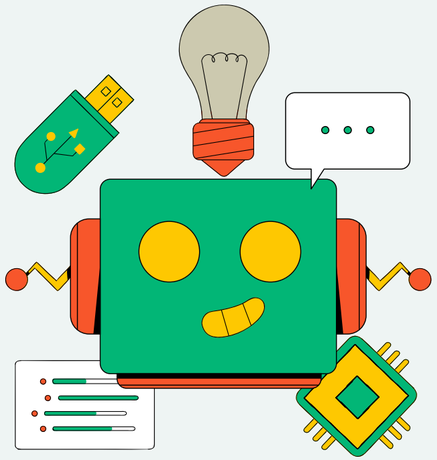
- Adjustable confidence threshold for mpnet model's similarity scores
- Option to enable or disable GPT for handling ambiguous cases



STREAMLIT INTERFACE

Results Display:

- Results displayed with True/False classifications, color-coded (green for True, red for False)
- Shows similarity scores, slide/page references, and optional GPT analysis details
- Displays relevant content or images from the lecture material

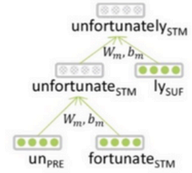


Results

A TFIDF word embedding represents synonyms (such as hotel and motel) by vectors that are close to each other in vector space.: False (Similarity: 0.61, Slide: 19, GPT: False)

Word Embeddings

- Represent the meaning of the word as a higher dimensional vector space. Expectation is that words with similar meaning will cluster together.
- Words made of **morphems** that we represent as vectors.
  - prefix stem suffix
  - un interest ed
- Vocabulary – localistic representation – vector [0, 0, 0, 0, 1,0,0, 0, 0]
  - Does not give relationship between words (i.e. motel vs hotel)
    - motel = [0,0,0,0,0,1,0,0,0]
    - hotel = [0,0,1,0,0,0,0,0,0]
    - These two vectors are orthogonal (their dot product is zero) – not that good!
- We would like a representation that translates the semantic ***similarity*** as perceived by humans to ***proximity*** in a vector space.



In contrast to WordNet, Word2Vec learns the word representations and clusters the embeddings of similar words together.: True (Similarity: 0.73, Slide: 24, GPT: True)

Word Embedding - Word2vec Model

Vord2Vec algorithm is based on distribution similarity – looks at the context of the surrounding words w_c for each center word w_t.

- First proposed by Bengio et al. in 2003 (no deep learning yet).
- A sliding window is implemented over the entire corpus. For each center word w_t, a probability of a context word w_c is defined.
- More recently - feed this objective to deep learning to get the word embeddings

Distribution similarity – look at the context of the surrounding words (**word embeddings**) – represent words as vectors

The algorithm has these 3 steps:

1. $p(w_c|w_t)$ - Define probability of a context word w_c for a given center word w_t (up to some window size).
2. $J = 1 - p(w_c|w_t)$- Define loss function J.
3. Keep adjusting the vector representations of words to minimize the loss function.

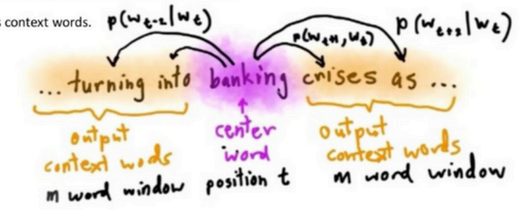
Vord2Vec – A prediction for each word what are its context words.

It has two algorithms:

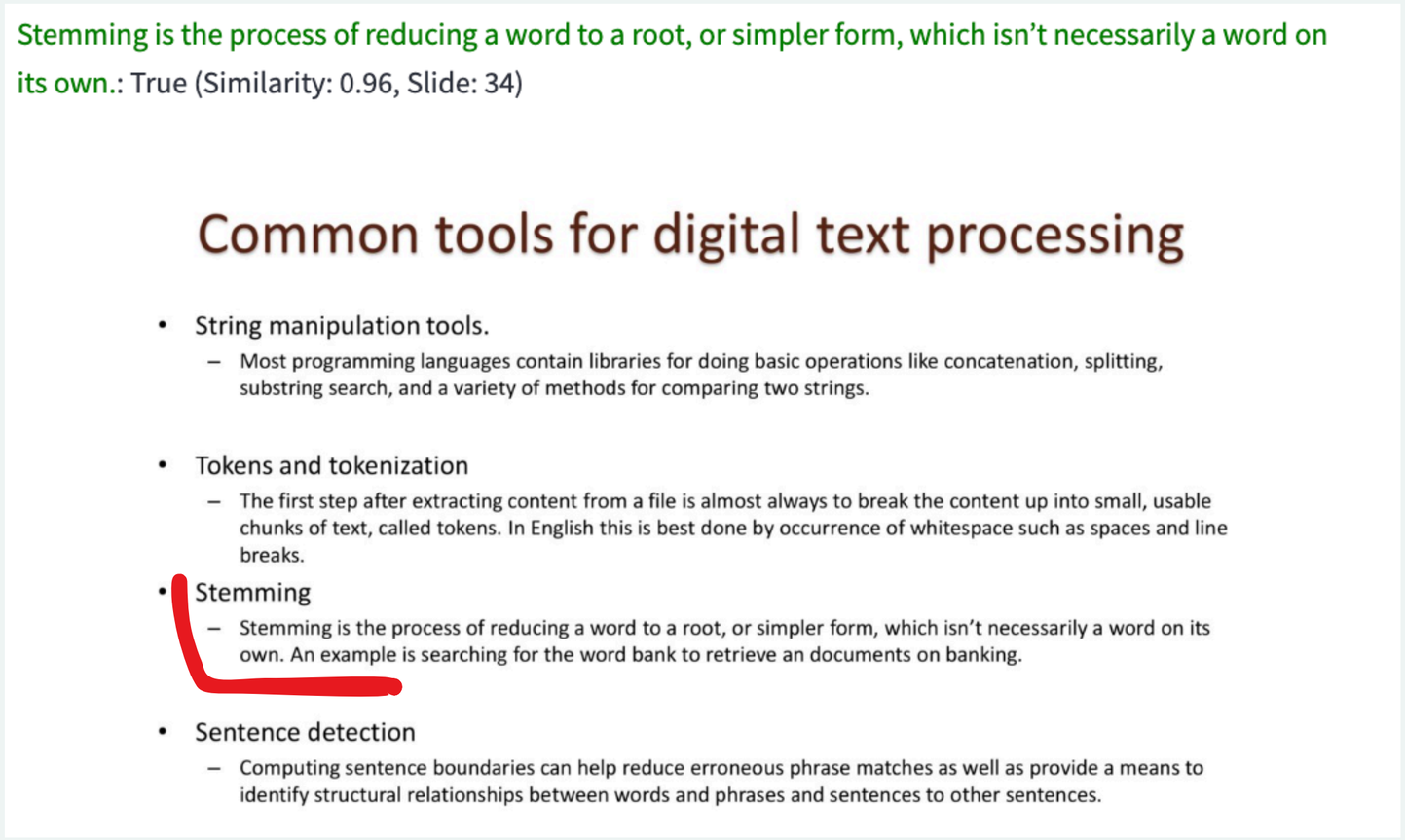
1. **Skip-gram** – probability of surrounding words
2. **Bag Of Words** – just add vectors in a sentence

It has two training algorithms:

1. Hierarchical Softmax
2. Negative sampling

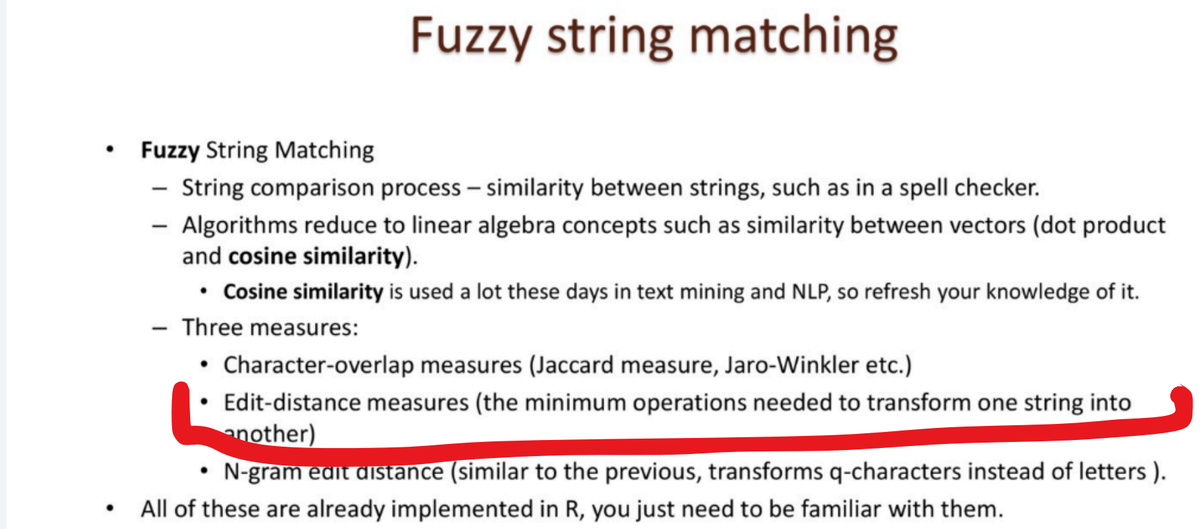


Stemming is the process of reducing a word to a root, or simpler form, which isn't necessarily a word on its own.: True (Similarity: 0.96, Slide: 34)



Results

Edit-distance is another type of algorithm for fuzzy string-matching, in which is based on the minimum operations needed to transform one string into another.: True (Similarity: 0.69, Slide: 16)



Results

BERT models may represent the same word by different embeddings, dependent on the word context.:
True (Similarity: 0.74, Slide: 38)

Zero-shot classification with BERT

- Zero-shot classification is a technique that allows us to associate an appropriate label with a piece of text.
  - This association is irrespective of the text domain and the aspect.
  - It can be a topic, emotion, or event described by the label.
  - To perform zero-shot classification, we need a zero-shot model capable of generating contextualized embeddings (such as model="facebook/bart-large-mnli").
    - This model is different from previous NLP models such as word2vec, which tends to have a fixed embedding vector for sentences that appear somewhat similar.
    - For example, these two sentences have similar words but different contexts, so a BERT model will generate different embedding for the two sentences.
      - "She didn't receive fair treatment"
      - "There is a fun fair in NYC this summer"
- Code:
  1. Initialize model
  2. Initialize pipeline
  3. Define input text
  4. Define Classes (Label Candidates)
  5. Pass text & label candidates to pipeline to get the result
  6. Print similarity scores

```
from transformers import pipeline

# Initializing pipeline & save the result in a variable called classifier_pipeline
classifier_pipeline = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
print("1) Easy Example")
input_sequence = "I love traveling"
label_candidate = ['travel', 'cooking', 'entertainment', 'dancing', 'technology']
result = classifier_pipeline(input_sequence, label_candidate)
print_result(result)
```

The BERT model is a transformer model that learns the context of the word from both ends of a sentence.:
True (Similarity: 0.74, Slide: 18)

The Core Ideas Behind BERT

It is related to language modeling and the problems related to it that goes under the name “fill in the blank”

"The Eiffel Tower is located in _____, the capital of France.”

- In the pre-BERT world, a language model would have looked at this text sequence in one-direction (for example left to right) trying to predict the next word until the sentence is completed.

- BERT introduced **bidirectionality** as its key technical innovation.
  - Instead of predicting the next word in a sequence, BERT makes use of a novel technique called **Masked LM** (MLM) that randomly masks words in the sentence and then it tries to predict them.
  - In order to predict the masked word, the BERT Model uses the full context of the sentence by looking both at the left and right surroundings of the masked word.
  - It takes both the previous and next tokens into account at the **same time** (which can be processed in parallel).
  - The previous LSTM models were processing the text sequentially (as time series data) so their operation could not have been parallelized.
  - Unlike LSTM, BERT is based on the Google’s **Transformer** model architecture introduced in 2017, a novel neural network architecture based on a self-attention mechanism that replaced the RNN (recurrent neural networks) previously used for language modeling.

BERT does not belong to the category of Masked Language Models.: False (Similarity: 0.48, GPT: False. BERT (Bidirectional Encoder Representations from Transformers) does belong to the category of Masked Language Models., Slide: 15)

General about the BERT Model

BERT is a **bidirectional transformer** pre-trained model that represents (**embeds**, vectorizes) text as numbers.

- BERT model achieves efficient text **embedding** (representation) in which similar words have a similar encoding, through an unsupervised (not specified by hand) training procedure.
- Each word is represented as a 4-dimensional vector of floating-point values.

The core part of BERT is the stacked bidirectional **encoders** from the transformer model

- During pre-training, a **masked language modeling** and **next sentence prediction**, a few extra layers are added onto BERT that can be used to generate a specific output.
- The raw output of BERT is the output from the stacked Bi-directional encoders. This fact is especially important as it allows you to essentially do any NLP task with BERT, including translation.
  - Google Translate was using NN with 7 layers

Many of the NLP tasks that BERT can provide are provided on Huggingface website (https://huggingface.co/).

The AI community building the future.

Build, train and deploy state of the art models powered by the reference open source in natural language processing.

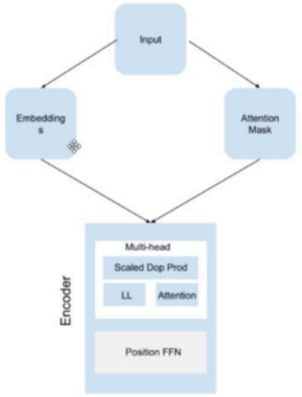
For each word, BERT models create three embeddings.: True (Similarity: 0.75, Slide: 24)

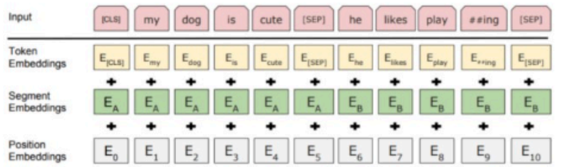
BERT Model in a Nutshell

The input text goes into the embedding and as well attention function.

The Encoder functions operate on the Embeddings and the Attention Mask using a dot product operation.

- The Embedding is the first layer in BERT that takes the input and creates a lookup table. Once the learning process is over the embeddings will cluster similar words together. It creates three embeddings for
  1. Token
  2. Segments
  3. Position
- The Attention Mask converts the [PAD] to 1 and elsewhere 0.
- The Encoder finds the representations and the patterns from the input and attention mask. It has two main components:
  - Multi-head Attention
  - Position-wise feed-forward network.
- The Position FFN (Position-Wise Feed Forward Network)





THANK YOU FOR ATTENTION!
ANY QUESTIONS?