Design a ML system design which predicts whether in each sentence 'jaguar' refers to car or an animal without LLMs

Example

1. The jaguar is a solitary predator native to the rainforest. 0
2. I test drove a jaguar F-type at the dealership. 1
3. Jaguars are often confused with leopards... 0
4. The Jaguar sedan offers a luxurious ride. 1

→ Edge Case example → "jaguar" name of a team

→ wikipedia, Books, wildlife forum, ...
→ wikipedia, car review, dealership

2) Preprocessing

→ Tokenization

→ The jaguar is a solitary predator native to the rainforest.

```
['The', 'jaguar', 'is', 'a', 'solitary', 'predator', 'native', 'to', 'the',
'rainforest', '.']
```

[NLTK, Spacy]

→ Lowercase

→ Removing of punctuations (.,!,@...) & Stop words ('the', 'as', 'a', 'an')

['jaguar', 'solitary', 'predator', 'native', 'rainforest']

0
1

3) Feature engineering

① [TF-IDF] / (Bag of words)

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1, 2))
X_train_tfidf = tfidf.fit_transform(train_sentences)
```

② POS tagging
↳ identify gramatical roles

The → 'DET'
Jag → 'Noun'
'is' → "Auxiliary"
'solitary' → "adj"
] OHE

Noun Adj
1 0

③ Dependency parsing

```
The ← det → jaguar
jaguar ← nsubj → is
is ← ROOT → is
a ← det → predator
solitary ← amod → predator
predator ← attr → is
native ← amod → predator
to ← prep → native
the ← det → rainforest
rainforest ← pobj → to
. ← punct → is
```

TF-IDF → sparse numerical representation of word importance

POS tagging → grammatical role of the words

Dependency tree → relationships b/w words

TF-IDF Vector (Vocabulary: [deer, hunts, jaguar]): [0.5, 0.7, 0.3]
POS Tags (One-Hot Encoded): [0, 1, 0, 1, ...] (e.g., 1 for "NOUN" at "jaguar").
Combined Feature Vector: [0.5, 0.7, 0.3, 0, 1, 0, 1, ...]

Approach 1

↓

Modeling

→ simple model (logistic Regression, RF)

Generation of embeddings using modern techniques

1) Static embeddings → GLOVE

2) Contextual embedding → BERT

```
from gensim.models import KeyedVectors

# Load pre-trained GloVe embeddings (50 dimensions)
glove_model = KeyedVectors.load_word2vec_format("glove.6B.50d.txt", binary=False)

# Get embedding for "jaguar"
jaguar_glove = glove_model['jaguar']
print("GloVe Embedding Shape:", jaguar_glove.shape)
print("First 5 dimensions:", jaguar_glove[:5])
```

→ GLOVE

```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Example 1: Animal context
sentence_animal = "The jaguar hunts in the rainforest."
inputs_animal = tokenizer(sentence_animal, return_tensors="pt", padding=True, truncation=True)
outputs_animal = model(**inputs_animal)

# Extract embedding for "jaguar" (token position 2)
jaguar_embedding_animal = outputs_animal.last_hidden_state[0, 2, :]

# Example 2: Car context
sentence_car = "The Jaguar sedan has a powerful engine."
inputs_car = tokenizer(sentence_car, return_tensors="pt", padding=True, truncation=True)
outputs_car = model(**inputs_car)

# Extract embedding for "Jaguar" (token position 2)
jaguar_embedding_car = outputs_car.last_hidden_state[0, 2, :]

print("Animal Embedding Shape:", jaguar_embedding_animal.shape)
print("Car Embedding Shape:", jaguar_embedding_car.shape)
```

BERT

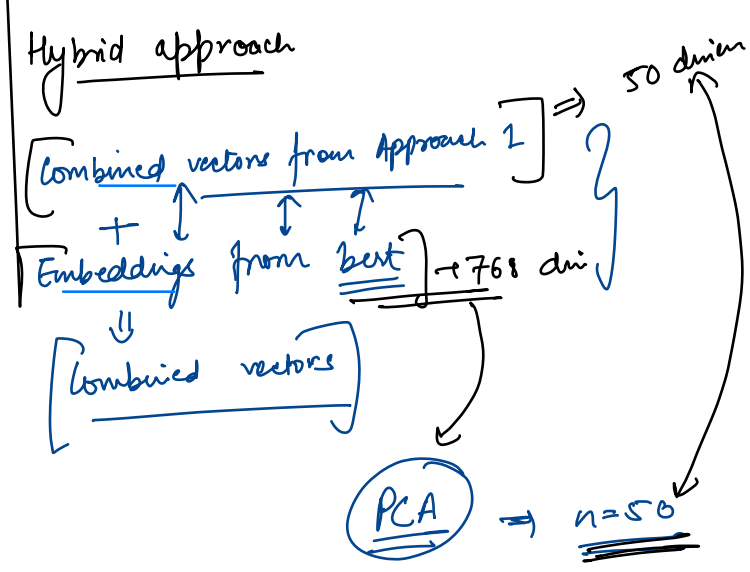
```
similarity = torch.cosine_similarity(jaguar_embedding_animal, jaguar_embedding_car, dim=0)
print(similarity.item())  # Might be ~0.7 (similar but distinct)
```

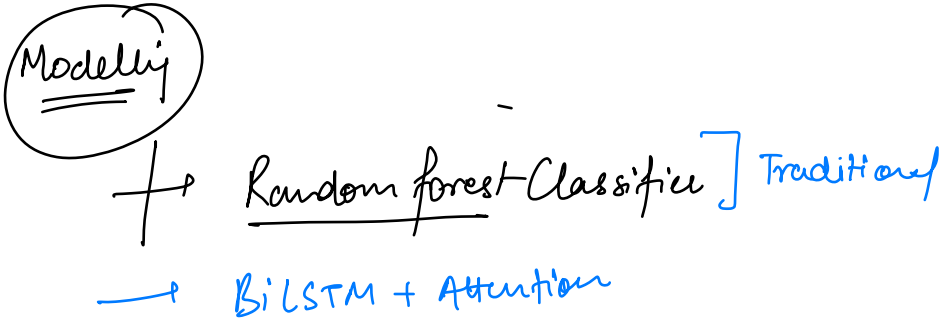
Deep learning

feed directly

→ [768 dim] → tf.sequential

input dim [768, 64]






```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=10000, output_dim=128),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Attention(),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

Summary

Preprocess Text: Tokenize, clean, remove stopwords.
Extract Features:
TF-IDF, POS tags
BERT/GloVe embeddings
Combine Features: Concatenate or use separately.
Train Model: Logistic Regression, Random Forest, or Neural Network.

```
# Pseudocode
sentences = ["The jaguar hunts...", "The Jaguar sedan..."]
labels = [0, 1]

# Step 4: Traditional Features
tfidf_features = TfidfVectorizer().fit_transform(sentences)
pos_features = OneHotEncoder().fit_transform(extract_pos_tags(sentences))
# Dependency parsing -
# Step 5: Embeddings
bert_embeddings = BertModel().encode(sentences)
# glove

# Combine
combined_features = concatenate([tfidf_features, pos_features, bert_embeddings])

# Train
model = RandomForestClassifier().fit(combined_features, labels)
```

class_weights : balanced

Evaluation

- f1-score
- confusion matrix

{BertViz} → words BERT focusing on

Deployment

- Inference pipeline

→ format as that of train

Model (Big / Small)

- optimization → Quantization

+ ONNX Runtime → Converting model in Onnx for faster CPU inference

API deployment

```python
from fastapi import FastAPI
app = FastAPI()

@app.post("/predict")
async def predict_jaguar(sentence: str):
    return predict(sentence)
```

Data Collection: Scrape & annotate sentences.
Preprocessing: Tokenize, clean, and filter.
Feature Extraction: Generate BERT embeddings + TF-IDF.
Model Training: Fine-tune BERT or train hybrid model.
Evaluation: Validate with F1-score and error analysis.
Deployment: Optimize and serve via FastAPI.

+ BestModel or flask

+ Spacy, NLTK
+ Hugging → BERT
+ Pytorch / Tensorflow / Keras
+ Deployment → FastAPI, ONNX, Docker

```python
from transformers import BertForSequenceClassification, Trainer, TrainingArguments
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
```

```
training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=16,
    num_train_epochs=3,
    learning_rate=2e-5,
    evaluation_strategy="epoch",
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,  # Hugging Face Dataset object
    eval_dataset=val_dataset,
)
trainer.train()
```

Questions

Imagine you were working on iPhone. Everytime users open their phones, you want to suggest one app they are most likely to open first with 90% accuracy. How would you do that?

An e-commerce company is trying to minimize the time it takes customers to purchase their selected items. As a machine learning engineer, what can you do to help them?

T-T-P