1. Data Acquisition & Exploration

1. **Obtain the Dataset**
   - **Download** the Quora Question Pairs CSV from Kaggle.
   - **Store** a copy of the raw CSV in version-controlled storage (e.g. an S3 "raw/" folder).
2. **Initial Inspection**
   - Load the first 1,000 rows in a Jupyter notebook.
   - Check for missing values in `question1`, `question2`, and `is_duplicate`.

Compute basic class balance:

```
df['is_duplicate'].value_counts(normalize=True)
```

3. **Data Profiling**
   - **Length distributions**: plot histograms of question-length (in tokens and chars).
   - **Vocabulary size**: count unique tokens after simple whitespace splitting.
   - **Sample edge cases**: very short (<3 words) vs. very long (>50 words) questions.

---

2. Text Preparation & Cleaning

1. **Strip HTML/Markdown**
   - Use `BeautifulSoup(text, "lxml").get_text()` to remove tags.
2. **Normalize Case & Whitespace**
   - `text = text.lower().strip()`
   - Collapse multi-spaces: `re.sub(r'\s+', ' ', text)`.
3. **Punctuation & Special Characters**

Option A: Remove all non-alphanumeric except spaces:

```
text = re.sub(r'[^a-z0-9 ]+', '', text)
```

Option B: Keep question marks/exclamation if you want to retain sentiment cues.

4. **Tokenization**

Use SpaCy's tokenizer for robust handling:

```
import spacy
nlp = spacy.load("en_core_web_sm", disable=["parser","ner"])
tokens = [tok.text for tok in nlp(text)]
```

-

5. **Stopword Removal** (remove all stopwords)
    - Experiment with/without to see impact on downstream performance.
6. **Stemming vs. Lemmatization**
    - Lemmatize via SpaCy (`tok.lemma_`) to preserve dictionary forms but little bit slow due to searching.
7. **Spell-Correction & Contraction Expansion** (advanced)
    - Use `symspellpy` to correct common typos.
    - Expand "isn't" → "is not" with `contractions.fix(text)`.
8. **Cache Preprocessed Text**
    - Save cleaned token lists (e.g. with Pickle or parquet) so you don't reprocess on every run.

---

3. Feature Engineering

We'll extract both **surface-level** and **semantic** features to feed into our model.

3.1 Lexical Similarity Features

Feature	Description
Jaccard Index	|tokens₁ ∩ tokens₂| / |tokens₁ ∪ tokens₂|
Overlap Coefficient	|tokens₁ ∩ tokens₂| / min(|tokens₁|,|tokens₂|)
Bigram Overlap	Same as Jaccard but on bigrams.
Edit Distance	Levenshtein distance normalized by max length.
Common Word Count	Raw count of shared tokens.

python
CopyEdit
```
from sklearn.metrics import jaccard_score
# For binary indicator vectors over a fixed vocab.
```

3.2 Statistical/Text-Vector Features

1. **TF–IDF Embeddings**
    - Fit a `TfidfVectorizer(ngram_range=(1,2), max_features=50_000)`.

- Transform both questions and compute:
    - **Cosine similarity**
    - **Euclidean distance**
2. **Length & Count Features**
    - `abs(len(q1_tokens) - len(q2_tokens))`
    - `abs(len(q1_chars) - len(q2_chars))`

3.3 Semantic Embeddings

1. **Word2Vec / GloVe Averages**
    - Load pretrained GloVe (e.g. 6B/300d).
    - `q_vec = np.mean([glove[w] for w in tokens if w in glove], axis=0)`.
    - Compute cosine similarity between `q1_vec` and `q2_vec`.
2. **Universal Sentence Encoder (USE)**
    - Use TensorFlow Hub to encode each question to a 512-dim vector.
    - Similarity features as above.
3. **BERT-Style Encodings**

Use `transformers`:
python
CopyEdit

```
from transformers import AutoTokenizer, AutoModel
tok = AutoTokenizer.from_pretrained("distilbert-base-uncased")
mdl = AutoModel.from_pretrained("distilbert-base-uncased")
inputs = tok(q1, q2, return_tensors="pt", truncation=True,
padding=True)
outputs = mdl(**inputs).last_hidden_state[:,0,:]  # [CLS] tokens
```

-
- Use the CLS embedding directly as input to a downstream classifier.

---

4. Model Development & Training

4.1 Baseline Classical Models

1. **Feature Matrix & Labels**
    - Concatenate all lexical + statistical features into `X`.
    - Labels `y = df['is_duplicate']`.
2. **Train/Test Split**

    - `train_test_split(..., stratify=y, test_size=0.2, random_state=42)`.
3. **Baseline Algorithms**
    - **Logistic Regression** (`penalty='l2'`, `C=1.0`).
    - **Random Forest** (`n_estimators=200`, `max_depth=10`).
4. **Cross-Validation**
    - 5-fold CV to tune hyperparameters via `GridSearchCV`.
5. **Metrics**
    - Report **Precision**, **Recall**, **F1**, **ROC-AUC** on held-out test.

4.2 Neural Approaches

1. **Siamese LSTM**
    - Two shared LSTM encoders → dropout → merge (e.g. absolute diff + multiplication) → Dense → Sigmoid.
2. **Fine-Tuned Transformer**
    - Add binary classification head on top of BERT.
    - Use `Trainer` API in HuggingFace with learning rate 2e-5, batch_size 16.
    - Early stop on validation loss.
3. **Training Tricks**
    - **Warmup** for first 10% steps.
    - **Grad-clipping** at 1.0.
    - **Mixed precision** (fp16) to speed up.

---

5. Evaluation & Error Analysis

1. **Confusion Matrix**
    - Identify false positives (non-duplicates flagged) vs. false negatives (missed duplicates).
2. **Error Sampling**
    - Manually inspect ~100 errors to categorize:
        - **Paraphrase misses** (same meaning, very different wording).
        - **Negation flips** (“Does IPL destroy…” vs. “Does IPL not destroy…”).
        - **Named-entity confusion** (“India vs. Australia” vs. “Australia vs. India”).
3. **Feature Importance**
    - For tree-based models, plot top 20 features.
4. **Threshold Tuning**
    - Move decision threshold away from 0.5 to balance precision vs. recall for production needs.

---

6. Deployment & Monitoring

1. **Packaging**
   - Serialize model and vectorizers with `joblib.dump()`.
   - Wrap in a FastAPI app (`app.post("/predict")`).
2. **Containerization**
   - Write a `Dockerfile` based on `python:3.9-slim`, install requirements, copy code.
3. **Infrastructure**
   - Deploy on AWS ECS/EKS or GCP Cloud Run behind a load balancer.
4. **Monitoring**
   - **Latency**: instrument with Prometheus + Grafana.
   - **Error Rates**: track 4xx/5xx, timeouts.
   - **Data Drift**: compare live input distribution of similarity scores to training baseline.
5. **CI/CD**
   - On new commits: run unit tests, build Docker image, push to registry.
   - Auto-deploy to a staging environment for smoke tests before pushing to production.
6. **Retraining Pipeline**
   - Schedule a weekly job to pull newly labeled pairs, retrain, and validate against hold-out.
   - Run A/B test on 10% traffic with the new model for two weeks before full rollout.
