# Duplicate Question Detection using Machine Learning

## 1 Introduction

This project aims to identify whether two given questions are duplicates of each other. The dataset consists of pairs of questions with a label indicating if they are duplicates. We use preprocessing, feature engineering, visualization, and machine learning models (Random Forest, XGBoost) to solve the task.

## 2 Data Loading

We start by importing libraries and loading the dataset.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/content/questions.csv', on_bad_lines='skip')
df.shape
```

Here, we load the dataset and check its dimensions.

## 3 Text Preprocessing

We clean and normalize the text in both questions.

```
import re
from bs4 import BeautifulSoup

def preprocess(q):
    q = str(q).lower().strip()
    q = q.replace('%','␣percent').replace('$','␣dollar')
    q = q.replace('   ','␣rupee').replace('   ','␣euro')
    q = q.replace('@','␣at').replace('[math]','')

    # Replace numbers with short forms
    q = re.sub(r'([0-9]+)000000000', r'\1b', q)
    q = re.sub(r'([0-9]+)000000', r'\1m', q)
    q = re.sub(r'([0-9]+)000', r'\1k', q)

    # Expand contractions
    contractions = {"can't":"can␣not","i'm":"i␣am","it's":"it␣is",...}
    q = '␣'.join([contractions.get(w,w) for w in q.split()])

    # Remove HTML tags and punctuation
    q = BeautifulSoup(q,'html.parser').get_text()
    q = re.sub(r'\W+','␣',q).strip()
    return q

df['question1'] = df['question1'].apply(preprocess)
df['question2'] = df['question2'].apply(preprocess)
```

This function converts text to lowercase, expands contractions, removes HTML tags, and cleans punctuation.

## 4 Token-Based Features

We create features based on common words, stopwords, and tokens.

```
from nltk.corpus import stopwords
STOP_WORDS = stopwords.words("english")

def fetch_token_features(row):
    q1 = row['question1'].split()
    q2 = row['question2'].split()
    if len(q1)==0 or len(q2)==0:
        return [0.0]*8

    q1_words = set([w for w in q1 if w not in STOP_WORDS])
    q2_words = set([w for w in q2 if w not in STOP_WORDS])

    common_word_count = len(q1_words & q2_words)
    common_stop_count = len(set(q1)&set(q2) & set(STOP_WORDS))
    common_token_count = len(set(q1) & set(q2))

    features = [0.0]*8
    features[0] = common_word_count / (min(len(q1_words),len(q2_words))
        +0.0001)
    features[1] = common_word_count / (max(len(q1_words),len(q2_words))
        +0.0001)
    features[2] = common_stop_count / (min(len(q1),len(q2))+0.0001)
    features[3] = common_stop_count / (max(len(q1),len(q2))+0.0001)
    features[4] = common_token_count / (min(len(q1),len(q2))+0.0001)
    features[5] = common_token_count / (max(len(q1),len(q2))+0.0001)
    features[6] = int(q1[-1]==q2[-1])  # last word equal
    features[7] = int(q1[0]==q2[0])    # first word equal
    return features
```

These token-level features measure similarity between question pairs.

## 5 Length-Based Features

We add features related to lengths and longest common substrings.

```
import distance
```

```
def fetch_length_features(row):
    q1, q2 = row['question1'], row['question2']
    q1_tokens, q2_tokens = q1.split(), q2.split()
    if len(q1_tokens)==0 or len(q2_tokens)==0:
        return [0.0]*3

    features = [0.0]*3
    features[0] = abs(len(q1_tokens) - len(q2_tokens))
    features[1] = (len(q1_tokens) + len(q2_tokens)) / 2
    strs = list(distance.lcsubstrings(q1,q2))
    features[2] = len(strs[0]) / (min(len(q1),len(q2))+1)
    return features
```

## 6 Fuzzy Matching Features

We use fuzzy string matching to quantify similarity.

```
from fuzzywuzzy import fuzz

def fetch_fuzzy_features(row):
    q1, q2 = row['question1'], row['question2']
    features = [0.0]*4
    features[0] = fuzz.QRatio(q1,q2)
    features[1] = fuzz.partial_ratio(q1,q2)
    features[2] = fuzz.token_sort_ratio(q1,q2)
    features[3] = fuzz.token_set_ratio(q1,q2)
    return features
```

## 7 Visualization

We visualize relationships between features and the target variable.

```
sns.pairplot(new_df[['cwc_min','cwc_max','csc_min','is_duplicate']], hue='
    is_duplicate')
sns.pairplot(new_df[['fuzz_ratio','token_sort_ratio','is_duplicate']], hue
    ='is_duplicate')
```

These plots show how features separate duplicate vs. non-duplicate pairs.

## 8 Dimensionality Reduction (t-SNE)

We apply t-SNE to reduce high-dimensional features to 2D/3D.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE

X = MinMaxScaler().fit_transform(new_df[feature_cols])
y = new_df['is_duplicate'].values

tsne2d = TSNE(n_components=2,random_state=101).fit_transform(X)
```

## 9 Machine Learning Models

We train Random Forest and XGBoost classifiers.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

x_train,x_test,y_train,y_test=train_test_split(temp_df.iloc[:,:-1],
   temp_df['is_duplicate'], test_size=0.2, random_state=42)

rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
print("Random␣Forest␣Accuracy:", accuracy_score(y_test,y_pred))

xgb = XGBClassifier()
xgb.fit(x_train,y_train)
y_pred = xgb.predict(x_test)
print("XGBoost␣Accuracy:", accuracy_score(y_test,y_pred))
```

Both models are tested, and accuracy is compared.

## 10 Feature Analysis

We create additional features such as question length, word counts, and word overlap.

```
df['q1_len'] = df['question1'].str.len()
df['q2_len'] = df['question2'].str.len()
df['q1_num_words'] = df['question1'].apply(lambda x: len(x.split()))
df['q2_num_words'] = df['question2'].apply(lambda x: len(x.split()))

def common_words(row):
    return len(set(row['question1'].split()) & set(row['question2'].split
       ()))
df['word_common'] = df.apply(common_words, axis=1)
```

These features give more information about similarities in question structure.

## 11 Conclusion

The project preprocesses text, engineers multiple types of features (token, length, fuzzy), visualizes them, and applies machine learning classifiers. Random Forest and XGBoost provide competitive performance in detecting duplicate questions.