1. Import Library

- `tensorflow`: Library untuk membangun dan melatih model neural networks.

- `pandas`: Library untuk melakukan manipulasi dan analisis data.

- `sklearn.preprocessing.LabelEncoder`: Kelas yang digunakan untuk mengkodekan label kategori menjadi nilai numerik.

- `tensorflow.keras.preprocessing.text.Tokenizer`: Kelas untuk melakukan tokenisasi pada teks.

- `tensorflow.keras.preprocessing.sequence.pad_sequences`: Fungsi untuk melakukan padding pada sequence teks.

- `fasttext`: Library FastText untuk melakukan prediksi teks menggunakan model FastText.

- `flask.Flask`, `flask.request`: Kelas dan fungsi untuk membuat web server menggunakan Flask.

2. Inisialisasi Aplikasi Flask

```python
app = Flask('Text Classification')
MODEL_PATH = 'model.bin'
```

Membuat objek aplikasi Flask dengan nama 'Text Classification' dan mengatur path model FastText.

3. Load Dataset

```python
df = pd.read_csv('data.csv')
```

Membaca dataset dari file CSV menggunakan Pandas.

4. Preprocessing

```python
le = LabelEncoder()
df['category'] = le.fit_transform(df['category'])
tokenizer = Tokenizer(num_words=1000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['complaint'])
sequences = tokenizer.texts_to_sequences(df['complaint'])
padded_sequences = pad_sequences(sequences, maxlen=20, padding='post')
```

```

- Melakukan encoding label kategori menggunakan `LabelEncoder`.

- Membuat objek `Tokenizer` untuk mengonversi teks menjadi sequence token.

- Melakukan tokenisasi pada teks complaint.

- Melakukan padding pada sequence token menggunakan `pad_sequences` agar memiliki panjang yang sama.

5. Define RNN Model

```python
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(1000, 16, input_length=20),
    tf.keras.layers.LSTM(16),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Membangun model sequential menggunakan TensorFlow yang terdiri dari layer embedding, LSTM, dan dense.

6. Compile Model

```python
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Mengompilasi model dengan fungsi loss, optimizer, dan metrik yang sesuai.

7. Train Model

```python
model.fit(padded_sequences, df['category'], epochs=140, verbose=1)
```

Melatih model menggunakan data yang telah diproses.

8. Load FastText Model

```python

```
fasttext_model = fasttext.load_model(MODEL_PATH)
```

Meload model FastText yang telah dilatih sebelumnya.

9. Fungsi `replaceResult`

```python
def replaceResult(label):
    label = label.replace('__label_', '')
    label = label.replace('_', ' ')
    label = label.strip()
    label = label.replace('SaranadanPrasarana', 'Sarana dan Prasarana')
    return label
```

Fungsi untuk melakukan manipulasi label hasil prediksi dari FastText.

10. Fungsi add_space

```python
def add_space(label):
    if any(char.isupper() for char in label):
        words = []
        start = 0
        for i, char in enumerate(label):
            if char.isupper():
                words.append(label[start:i])
                start = i
        words.append(label[start:])
        label_with_space = ' '.join(words)
        return label_with_space
    else:
        return label
```

Fungsi `add_space` digunakan untuk menambahkan spasi di antara kata-kata pada label yang menggunakan huruf kapital. Fungsi ini beroperasi sebagai berikut:

1. Memeriksa apakah terdapat huruf kapital pada label menggunakan `any(char.isupper() for char in label)`. Jika tidak ada huruf kapital, fungsi akan langsung mengembalikan label tanpa melakukan perubahan.

2. Jika terdapat huruf kapital pada label, fungsi akan melakukan pemisahan kata berdasarkan huruf kapital. Misalnya, jika label adalah "SaranaPrasaranaKampus", fungsi akan memisahkan kata menjadi ["Sarana", "Prasarana", "Kampus"].

3. Selanjutnya, kata-kata yang telah dipisahkan akan digabungkan kembali dengan menggunakan spasi sebagai pemisah, menggunakan `' '.join(words)`.

4. Hasil akhir dari fungsi `add_space` adalah label yang telah ditambahkan spasi di antara kata-kata yang terpisah.

Fungsi ini berguna untuk memperbaiki format label yang mungkin tidak terpisah dengan baik, sehingga memudahkan dalam pembacaan dan interpretasi hasil prediksi.

Bagian Flask

1. Route `/predict`

```python
@app.route('/predict', methods=['POST'])
def predict():
    text = request.form['text']
    # ...
    return {
        'combined_prob': combined_prob,
        'label': replaceResult(predicted_label)
    }
```

Ini adalah endpoint Flask yang akan digunakan untuk menerima permintaan POST dengan data teks yang akan diprediksi. Data teks diambil dari `request.form['text']`. Selanjutnya, dilakukan prediksi

menggunakan model FastText dan TensorFlow. Hasil prediksi akan dikembalikan dalam format JSON yang berisi probabilitas gabungan dan label prediksi.

2. Route `'/'` (Root)

```python
@app.route('/', methods=['GET'])
def welcome():
    return "Server works!"
```

Ini adalah endpoint Flask yang akan memberikan respons "Server works!" ketika mengakses alamat root server.

3. Menjalankan Aplikasi Flask

```python
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=105)
```

Bagian ini menjalankan aplikasi Flask pada host `'0.0.0.0'` dan port `105`. Aplikasi akan dijalankan ketika file ini dijalankan langsung, bukan diimpor sebagai modul.

Dengan menggunakan Flask, Anda dapat menjalankan aplikasi ini sebagai server web yang menerima permintaan prediksi melalui endpoint `/predict` dan memberikan respons dengan hasil prediksi. Endpoint root `'/'` digunakan untuk memastikan server berjalan dengan baik ketika diakses.