Fake News Classification Projekt

1. Projektüberblick

Dieses Projekt klassifiziert Nachrichten in **Fake News** (label=1) und **echte Nachrichten** (label=0). Der Workflow umfasst:

1. **Einlesen der Daten und Aufbereitung**
2. **Textbereinigung und linguistische Merkmalsextraktion**
3. **Balancieren des Datensatzes**
4. **Feature-Erzeugung (Bag-of-Words)**
5. **Training eines Random-Forest-Klassifikators**
6. **Vorhersage neuer Nachrichten**

---

2. Code und Erklärungen

2.1 Daten einlesen

```
import pandas as pd

# CSV laden, nur relevante Spalten
data = pd.read_csv("news.csv", usecols=['Body', 'Fake'])
data = data.rename(columns={'Body': 'text', 'Fake': 'label'})

data.head()
```

Erklärung:

- Liest die Datei `news.csv` ein.
- Behält nur die Spalten `Body` und `Fake`.
- Bennent Spalten in `text` und `label` um.

2.2 Textaufbereitung (Vorbereitung für NLP)

```
import re
import nltk
import spacy
import pandas as pd
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('german'))

nlp = spacy.load('de_core_news_sm')

def preprocess_text(text):
    text_lower = text.lower()
    tokens = re.findall(r'\b[a-z0-9äöüß]+\b', text_lower)
    tokens = [token for token in tokens if token not in stop_words]
    cleaned_text = ' '.join(tokens)

    doc = nlp(cleaned_text)
    adjective_count = sum(1 for token in doc if token.pos_ == 'ADJ')
    adverb_count    = sum(1 for token in doc if token.pos_ == 'ADV')
    entity_count    = len(doc.ents)

    return cleaned_text, adjective_count, adverb_count, entity_count
```

Erklärung:

- Entfernt unerwünschte Zeichen und Stopwörter.
- Wandelt Text in Kleinschreibung um.
- Zählt mithilfe von **SpaCy** Adjektive, Adverbien und erkannte Entitäten.

2.3. Balancieren (sample1) + Einbindung der Features

```
def sample1(data, n_samples=4627):
    df_real = data[data['label'] == 0]
    df_fake = data[data['label'] == 1]

    df_real_sampled = df_real.sample(n=n_samples, random_state=42)
    df_fake_sampled = df_fake.sample(n=n_samples, random_state=42)

    data_balanced = pd.concat([df_real_sampled, df_fake_sampled])
    data_balanced = data_balanced.sample(frac=1,
random_state=42).reset_index(drop=True)

    data_balanced[['cleaned_text', 'adjective_count', 'adverb_count',
'entity_count']] = \
        data_balanced['text'].apply(lambda x: pd.Series(preprocess_text(x)))

    return data_balanced

data_balanced = sample1(data)
```

```
data_balanced.head()
```

Erklärung:

- Teilt Daten in **echte** und **Fake**-Nachrichten.
- Wählt pro Klasse `n_samples` Zeilen aus (z. B. 4627).
- Mischt und extrahiert die aufbereiteten Texte plus linguistische Zählungen.

2.4. Feature-Erstellung (Bag-of-Words)

```
from sklearn.feature_extraction.text import CountVectorizer

def create_feature_df(data, max_features=None):
    vectorizer = CountVectorizer(
        token_pattern=r'[A-Za-zÄÖÜäöüß]+',
        max_features=max_features
    )

    X_counts = vectorizer.fit_transform(data['cleaned_text'])
    feature_names = vectorizer.get_feature_names_out()

    bow_df = pd.DataFrame(X_counts.toarray(), columns=feature_names)
    bow_df['adjective_count'] = data['adjective_count'].values
    bow_df['adverb_count']    = data['adverb_count'].values
    bow_df['entity_count']    = data['entity_count'].values

    return bow_df

features_df = create_feature_df(data_balanced, max_features=2000)
features_df.tail()
```

Erklärung:

- **CountVectorizer** wandelt die aufbereiteten Texte in numerische Features um (BoW).
- `max_features` begrenzt die Anzahl an Wörtern.
- Die linguistischen Merkmale (Adjektiv-/Adverb-/Entitäts-Zähler) werden angehängt.

2.5. Training und Evaluierung (Random Forest)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X = features_df
y = data_balanced['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Erklärung:

- Aufteilung in **Train** (80%) und **Test** (20%).
- **RandomForestClassifier** wird trainiert und ausgewertet.
- **Accuracy**: misst den Anteil korrekt klassifizierter Nachrichten.

2.6. Test an neuer Nachricht

```
new_message_fake = """Schockierende Enthüllung: Wissenschaftler behaupten, dass
der ...
..."""

# Bereinigung und Feature-Extraktion
cleaned_text, adj_count, adv_count, ent_count = preprocess_text(new_message_fake)

vectorizer = CountVectorizer(max_features=2000, token_pattern=r'[A-Za-
zÄÖÜäöüß]+')
vectorizer.fit(data_balanced['cleaned_text'])

message_counts = vectorizer.transform([cleaned_text])
new_message_df = pd.DataFrame(message_counts.toarray(),
                              columns=vectorizer.get_feature_names_out())

new_message_df['adjective_count'] = adj_count
new_message_df['adverb_count']    = adv_count
new_message_df['entity_count']    = ent_count
```

```
prediction = clf.predict(new_message_df)
print(f"Vorhersage: {prediction} (1 = fake, 0 = real)")
```

Erklärung:

- Verwendet **preprocess_text** auf einen Beispieltext.
- Erstellt die gleichen Features wie beim Training.
- **Vorhersage** durch das trainierte Random-Forest-Modell (Ergebnis: 1 = Fake, 0 = Echt).

---

3. Zusammenfassung

- **DataFrame**: Reduzierung auf Spalten *text*, *label*; Balancieren der Klassen auf jeweils 4627 Datensätze.
- **Vorverarbeitung**: Entfernen von Stopwörtern, Kleinschreibung, Extraktion von Adjektiv-/Adverb-/Entitäts-Zählern.
- **Merkmalsextraktion**: CountVectorizer (Bag-of-Words) + linguistische Features.
- **Modell**: RandomForestClassifier liefert solide Ergebnisse bei der Fake-News-Erkennung (Accuracy, …).
- **Neue Texte** können mit denselben Schritten analysiert und klassifiziert werden.