شناسایی ایمیل‌های جعلی

الف) پیش‌پردازش متن

مراحل پیش‌پردازش متن:

1. **حذف علامت‌گذاری، کلمات توقف، URL ها و HTML:**
   - از re برای عملیات regex برای حذف URL ها و HTML استفاده کنید.
   - از string.punctuation برای حذف علامت های نگارشی و نوشتاری استفاده کنید. شامل کوچک کردن حروف و ... میشود.
   - از nltk.corpus.stopwords برای حذف کلمات توقف استفاده کنید.

پیش‌پردازش اختصارات:

- یک دیکشنری از اختصارات رایج و توسعه‌های آن‌ها ایجاد کنید.
- اختصارات را با توسعه‌های آن‌ها جایگزین کنید.

پیش‌پردازش یا حذف ایموجی‌ها و شکلک‌ها:

- از کتابخانه emoji برای حذف ایموجی‌ها استفاده کنید.
- از رجکس برای حذف شکلک‌های رایج استفاده کنید.

```python
def preprocess_text_spacy(text):
    text = BeautifulSoup(text, "html.parser").get_text()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text.lower())
    text = re.sub(r'[^\x00-\x7F]+', '', text)

    doc = nlp(text)

    tokens = [token.lemma_ for token in doc if token.text not in STOP_WORDS]
    abbreviations = {
        "u": "you",
        "r": "are",
        "ur": "your",
        "b4": "before",
        "gr8": "great",
        "l8r": "later",
    }
    tokens = [abbreviations.get(word, word) for word in tokens]

    return ' '.join(tokens)
```

تأثیر بر عملکرد:

- پیش‌پردازش متن به کاهش نویز و حذف اطلاعات غیرضروری کمک می‌کند و بخشی ضروری است که باعث می‌شود مدل‌های یادگیری ماشین بهتر بتوانند الگوهای موجود در داده‌ها را شناسایی کنند. این مراحل باعث افزایش دقت مدل و کاهش خطاهای classification می‌شوند.

ب) تحلیل داده‌ها

کلمات پر تکرار در ایمیل‌های اسپم و غیر اسپم:

- از CountVectorizer از sklearn برای یافتن کلمات پر تکرار استفاده کنید.

```
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
X_train_vec, X_test_vec
```

```
(<4457x7146 sparse matrix of type '<class 'numpy.int64'>'
        with 33593 stored elements in Compressed Sparse Row format>,
 <1115x7146 sparse matrix of type '<class 'numpy.int64'>'
        with 7324 stored elements in Compressed Sparse Row format>)
```

- مجموعه داده‌ها را به ایمیل‌های اسپم و غیر اسپم تقسیم کنید.

```
spam_df = df[df['Label'] == 'spam']
ham_df = df[df['Label'] == 'ham']
```

کلماتی که احتمال اسپم بودن ایمیل را افزایش می‌دهند:

- کلمات رایج که احتمال اسپم بودن یک ایمیل را افزایش می‌دهند شامل اصطلاحات مرتبط با بازاریابی، فروش و درخواست‌های فوری مانند "offer"، "free"، "buy" و غیره هستند.

تأثیر بر عملکرد:

- تحلیل داده‌ها به شناسایی ویژگی‌های کلیدی که در طبقه‌بندی ایمیل‌ها به عنوان اسپم یا غیر اسپم مؤثر هستند، کمک می‌کند. این کلمات کلیدی می‌توانند به عنوان ویژگی‌های مهم در مدل‌های یادگیری ماشین استفاده شوند که باعث بهبود دقت مدل می‌شوند.

ج) متعادل‌سازی داده‌ها

روش‌های متعادل‌سازی مجموعه داده‌ها:

1. **تصادفی حذف کردن:** حذف برخی نمونه‌ها از کلاس عمده.
2. **تصادفی افزوده کردن:** تکرار برخی نمونه‌ها در کلاس اقلیت.
3. **SMOTE (روش مصنوعی افزوده‌سازی اقلیت) یا البته (Synthetic Minority Over-sampling Technique) :** تولید نمونه‌های مصنوعی برای کلاس اقلیت.

استفاده از SMOTE برای متعادل‌سازی داده‌ها:

```
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train_vec, y_train)
```

```
X_train_res, y_train_res
```

```
(<7718x7146 sparse matrix of type '<class 'numpy.int64'>'
        with 88896 stored elements in Compressed Sparse Row format>,
 0       1
 1       0
 2       0
 3       0
 4       1
        ..
 7713    1
 7714    1
 7715    1
 7716    1
 7717    1
 Name: Label, Length: 7718, dtype: int64)
```

تأثیر بر عملکرد:

- متعادل‌سازی داده‌ها باعث کاهش مشکل عدم توازن کلاس‌ها می‌شود که می‌تواند منجر به بهبود دقت مدل و کاهش نرخ خطای دسته‌بندی کلاس اقلیت شود. استفاده از روش‌هایی

مانند SMOTE به تولید داده‌های مصنوعی برای کلاس اقلیت کمک می‌کند که باعث افزایش تعداد نمونه‌های آموزشی و بهبود عملکرد مدل می‌شود.

د) توکن‌سازی

هدف از توکن‌سازی در روش‌های NLP:

- توکن‌سازی فرایند تقسیم متن به واحدهای کوچکتر به نام توکن‌ها (کلمات، جملات و غیره) است. این به تبدیل داده‌های متنی به فرمتی که می‌توان از آن برای آموزش مدل استفاده کرد، کمک می‌کند.

مثال توکن‌سازی:

```
max_words = 5000
max_len = 100

tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)
```

تأثیر بر عملکرد:

- **توکن‌سازی باعث می‌شود متن به صورت عددی نمایه شود که برای مدل‌های یادگیری ماشین قابل استفاده باشد. این فرآیند به بهبود دقت مدل‌ها در تشخیص الگوهای موجود در داده‌های متنی کمک می‌کند و باعث افزایش کارایی مدل می‌شود.**

ه) پیاده‌سازی مدل‌ها

LSTM یا (Long Short-Term Memory) / (حافظه بلند مدت کوتاه):

- **توضیح: شبکه‌های LSTM نوعی شبکه عصبی بازگشتی (RNN) هستند که قادر به یادگیری وابستگی‌های طولانی مدت هستند. این شبکه‌ها برای داده‌های ترتیبی مناسب هستند و به**

منظور اجتناب از مشکل وابستگی طولانی طراحی شده‌اند. LSTM ها دارای معماری پیچیده‌تری نسبت به RNN های ساده هستند و شامل حالت سلولی و دروازه‌ها (ورودی، فراموشی، خروجی) هستند که جریان اطلاعات را تنظیم می‌کنند.

مثال پیاده‌سازی LSTM:

```
def create_lstm_model():
    model = Sequential()
    model.add(Embedding(max_words, 128, input_length=max_len))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

تأثیر بر عملکرد:

- مدل‌های LSTM به دلیل توانایی در یادگیری وابستگی‌های طولانی مدت و حفظ اطلاعات طولانی مدت، معمولاً عملکرد بهتری در پردازش داده‌های ترتیبی مانند متن دارند. این مدل‌ها می‌توانند الگوهای پیچیده‌تر را شناسایی کنند و دقت طبقه‌بندی را افزایش دهند.

```
y_pred = (lstm_model.predict(X_test_pad) > 0.5).astype("int32")
print(classification_report(y_test, y_pred, target_names=['ham', 'spam']))
```

```
35/35 [==============================] - 1s 29ms/step
              precision    recall  f1-score   support

         ham       0.99      0.94      0.97       966
        spam       0.72      0.94      0.82       149

    accuracy                           0.94      1115
   macro avg       0.86      0.94      0.89      1115
weighted avg       0.95      0.94      0.95      1115
```

RNN (شبکه عصبی بازگشتی):

- توضیح: RNN ها شبکه‌های عصبی هستند که دارای loop هایی هستند که اطلاعات را حفظ می‌کنند. این شبکه‌ها برای داده‌های sequential یا داده‌های سری زمانی time series

استفاده می‌شوند. با این حال، RNN های استاندارد از مشکل ناپدید شدن گرادیان رنج می‌برند که باعث کاهش کارایی آن‌ها برای وابستگی‌های طولانی مدت می‌شود.

مثال پیاده‌سازی RNN:

```
def create_rnn_model():
    model = Sequential()
    model.add(Embedding(max_words, 128, input_length=max_len))
    model.add(SpatialDropout1D(0.2))
    model.add(SimpleRNN(100, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

تأثیر بر عملکرد:

- مدل‌های RNN به دلیل سادگی و توانایی در پردازش داده‌های ترتیبی، برای مسائل ساده‌تر مناسب هستند. اما به دلیل مشکل ناپدید شدن گرادیان، معمولاً عملکرد آن‌ها در یادگیری وابستگی‌های طولانی مدت ضعیف‌تر است.

```
y_pred = (rnn_model.predict(X_test_pad) > 0.5).astype("int32")
print(classification_report(y_test, y_pred, target_names=['ham', 'spam']))
```

```
35/35 [==============================] - 1s 14ms/step
              precision    recall  f1-score   support

         ham       0.99      0.98      0.98       966
        spam       0.88      0.91      0.89       149

    accuracy                           0.97      1115
   macro avg       0.93      0.94      0.94      1115
weighted avg       0.97      0.97      0.97      1115
```

GRU (واحد بازگشتی دروازه‌ای):

- توضیح: GRU ها نوعی از شبکه‌های LSTM هستند اما با معماری ساده‌تر. این شبکه‌ها دروازه ورودی و فراموشی را در یک دروازه به روز رسانی ترکیب می‌کنند که باعث افزایش کارایی محاسباتی آن‌ها می‌شود و همچنان مشکل ناپدید شدن گرادیان را حل می‌کنند.

مثال پیاده‌سازی GRU:

```
def create_gru_model():
    model = Sequential()
    model.add(Embedding(max_words, 128, input_length=max_len))
    model.add(SpatialDropout1D(0.2))
    model.add(GRU(100, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

تأثیر بر عملکرد:

- **مدل‌های GRU به دلیل سادگی بیشتر نسبت به LSTM ها و همچنان توانایی در یادگیری وابستگی‌های طولانی مدت، عملکرد مناسبی دارند. این مدل‌ها می‌توانند کارایی بالاتری داشته باشند و دقت مشابه یا بهتری نسبت به LSTM ها ارائه دهند.**

```
y_pred = (gru_model.predict(X_test_pad) > 0.5).astype("int32")
print(classification_report(y_test, y_pred, target_names=['ham', 'spam']))
```

```
35/35 [==============================] - 1s 31ms/step
              precision    recall  f1-score   support

         ham       0.99      0.96      0.97       966
        spam       0.77      0.96      0.85       149

    accuracy                           0.96      1115
   macro avg       0.88      0.96      0.91      1115
weighted avg       0.96      0.96      0.96      1115
```

مقایسه دقت و نمودارهای خطا:

- **از matplotlib برای رسم نمودارهای دقت و خطا استفاده کنید.**

گزارش دقت، یادآوری و F1-Score:

و) بهینه‌سازی مدل

آزمایش با انواع بهینه‌سازها و نرخ‌های یادگیری:

```
optimizers = ['adam', 'sgd', 'rmsprop']
learning_rates = [0.001, 0.01, 0.1]
```

- **بهینه‌ساز و نرخ یادگیری را در مرحله کامپایل مدل تغییر دهید و عملکرد را ارزیابی کنید.**

تأثیر بر عملکرد:

- **انتخاب بهینه‌ساز مناسب و نرخ یادگیری بهینه می‌تواند تأثیر قابل توجهی بر عملکرد مدل داشته باشد. آزمایش با بهینه‌سازها و نرخ‌های یادگیری مختلف به یافتن بهترین ترکیب برای بهبود دقت و کاهش خطاهای مدل کمک می‌کند.**

```
Epoch 4/5
121/121 - 41s - loss: 0.4214 - accuracy: 0.8062 - val_loss: 0.3591 - val_accuracy: 0.8430 - 41s/epoch - 336ms/step
Epoch 5/5
121/121 - 42s - loss: 0.4169 - accuracy: 0.8103 - val_loss: 0.3929 - val_accuracy: 0.8314 - 42s/epoch - 343ms/step
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Training with optimizer sgd and learning rate 0.001...
Epoch 1/5
121/121 - 46s - loss: 0.6958 - accuracy: 0.3877 - val_loss: 0.6979 - val_accuracy: 0.1982 - 46s/epoch - 384ms/step
Epoch 2/5
121/121 - 39s - loss: 0.6948 - accuracy: 0.4131 - val_loss: 0.6955 - val_accuracy: 0.2861 - 39s/epoch - 326ms/step
Epoch 3/5
121/121 - 40s - loss: 0.6939 - accuracy: 0.4558 - val_loss: 0.6934 - val_accuracy: 0.4807 - 40s/epoch - 330ms/step
Epoch 4/5
121/121 - 40s - loss: 0.6929 - accuracy: 0.5149 - val_loss: 0.6914 - val_accuracy: 0.6242 - 40s/epoch - 331ms/step
Epoch 5/5
121/121 - 41s - loss: 0.6921 - accuracy: 0.5526 - val_loss: 0.6894 - val_accuracy: 0.7076 - 41s/epoch - 337ms/step
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Training with optimizer sgd and learning rate 0.01...
Epoch 1/5
121/121 - 45s - loss: 0.6869 - accuracy: 0.6682 - val_loss: 0.6712 - val_accuracy: 0.8529 - 45s/epoch - 375ms/step
Epoch 2/5
121/121 - 42s - loss: 0.6750 - accuracy: 0.7714 - val_loss: 0.6501 - val_accuracy: 0.8520 - 42s/epoch - 347ms/step
Epoch 3/5
121/121 - 40s - loss: 0.6550 - accuracy: 0.8252 - val_loss: 0.6100 - val_accuracy: 0.8448 - 40s/epoch - 333ms/step
Epoch 4/5
121/121 - 41s - loss: 0.6129 - accuracy: 0.8300 - val_loss: 0.5330 - val_accuracy: 0.8430 - 41s/epoch - 337ms/step
Epoch 5/5
121/121 - 40s - loss: 0.5366 - accuracy: 0.8317 - val_loss: 0.4821 - val_accuracy: 0.8233 - 40s/epoch - 332ms/step
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Training with optimizer sgd and learning rate 0.1...
Epoch 1/5
121/121 - 46s - loss: 0.5677 - accuracy: 0.7600 - val_loss: 0.4575 - val_accuracy: 0.8018 - 46s/epoch - 381ms/step
Epoch 2/5
121/121 - 40s - loss: 0.4091 - accuracy: 0.8343 - val_loss: 0.3297 - val_accuracy: 0.8520 - 40s/epoch - 335ms/step
Epoch 3/5
121/121 - 40s - loss: 0.3994 - accuracy: 0.8354 - val_loss: 0.3480 - val_accuracy: 0.8430 - 40s/epoch - 327ms/step
Epoch 4/5
121/121 - 39s - loss: 0.3814 - accuracy: 0.8399 - val_loss: 0.3540 - val_accuracy: 0.8386 - 39s/epoch - 326ms/step
Epoch 5/5
121/121 - 40s - loss: 0.3620 - accuracy: 0.8497 - val_loss: 0.3245 - val_accuracy: 0.8547 - 40s/epoch - 327ms/step
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Training with optimizer rmsprop and learning rate 0.001...
Epoch 1/5
121/121 - 47s - loss: 0.3611 - accuracy: 0.8453 - val_loss: 0.1930 - val_accuracy: 0.9408 - 47s/epoch - 389ms/step
Epoch 2/5
121/121 - 41s - loss: 0.2209 - accuracy: 0.9155 - val_loss: 0.1103 - val_accuracy: 0.9713 - 41s/epoch - 336ms/step
Epoch 3/5
121/121 - 40s - loss: 0.1743 - accuracy: 0.9379 - val_loss: 0.1395 - val_accuracy: 0.9605 - 40s/epoch - 332ms/step
Epoch 4/5
121/121 - 40s - loss: 0.1466 - accuracy: 0.9495 - val_loss: 0.1178 - val_accuracy: 0.9677 - 40s/epoch - 335ms/step
Epoch 5/5
121/121 - 40s - loss: 0.1236 - accuracy: 0.9593 - val_loss: 0.1423 - val_accuracy: 0.9587 - 40s/epoch - 332ms/step
WARNING:tensorflow:Layer lstm_8 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Training with optimizer rmsprop and learning rate 0.01...
Epoch 1/5
```

استفاده از کدی که در در بخش Optimizing شده شما را به این مرحله میبرد.

در نهایت از همه Optimizer ها و Learning Rate ها استفاده کرده و بهترین Combination را منتخب میکند.

```
Training with optimizer rmsprop and learning rate 0.001...
Epoch 1/5
121/121 - 47s - loss: 0.3611 - accuracy: 0.8453 - val_loss: 0.1930 - val_accuracy: 0.9408 - 47s/epoch - 389ms/step
Epoch 2/5
121/121 - 41s - loss: 0.2209 - accuracy: 0.9155 - val_loss: 0.1103 - val_accuracy: 0.9713 - 41s/epoch - 336ms/step
Epoch 3/5
121/121 - 40s - loss: 0.1743 - accuracy: 0.9379 - val_loss: 0.1395 - val_accuracy: 0.9605 - 40s/epoch - 332ms/step
Epoch 4/5
121/121 - 40s - loss: 0.1466 - accuracy: 0.9495 - val_loss: 0.1178 - val_accuracy: 0.9677 - 40s/epoch - 335ms/step
Epoch 5/5
121/121 - 40s - loss: 0.1236 - accuracy: 0.9593 - val_loss: 0.1423 - val_accuracy: 0.9587 - 40s/epoch - 332ms/step
```

ذخیره کردن مدل:

مدل Train شده را در یک فایل با پسوند h5 ذخیره کنید.

```
best_model.save('best_spam_detection_model.h5')
```

دقت و Precision و Recall و F1 Score مدل را به راحتی با دستور classification_report که از کتابخانه sklearn.metrics خوانده بودیم بررسی میکنیم:

```
print("Test Set Evaluation:\n")
print("Accuracy:", accuracy_score(y_test, y_pred_test))
print("Classification Report:\n", classification_report(y_test, y_pred_test, target_names=['ham', 'spam']))
```

```
Test Set Evaluation:

Accuracy: 0.9587443946188341
Classification Report:
              precision    recall  f1-score   support

         ham       0.99      0.96      0.98       966
        spam       0.79      0.95      0.86       149

    accuracy                           0.96      1115
   macro avg       0.89      0.95      0.92      1115
weighted avg       0.96      0.96      0.96      1115
```

و) استفاده از مدل سیو شده

اگر مایل بودید مدل خاصی که در حال حاضر Train کرده اید را خوانده و از آن استفاده کنید

میتوانید از کتابخانه keras استفاده کنید:

```
loaded_model = tf.keras.models.load_model('best_spam_detection_model.h5')
```

```
loaded_model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_9 (Embedding)     (None, 100, 128)          640000

 spatial_dropout1d_9 (Spati  (None, 100, 128)          0
 alDropout1D)

 lstm_7 (LSTM)               (None, 100)               91600

 dense_9 (Dense)             (None, 1)                 101

=================================================================
Total params: 731701 (2.79 MB)
Trainable params: 731701 (2.79 MB)
Non-trainable params: 0 (0.00 Byte)
```

یا ماتریس سردرگمی Confusion Matrix

از y_pred و y_test