main

March 8, 2025

1 Email Spam Detection Notebook

1.1 Complete Analysis with Logistic Regression

1.2 1. Setup and Data Download

[ ]:
```
%%capture
# Installations (if needed)
# !pip install wordcloud seaborn
```

[ ]:
```
import os
import re
import email
import hashlib
import tarfile
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from urllib.request import urlretrieve
from email import policy
from email.utils import parsedate_tz
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪precision_recall_curve, roc_curve, auc

%matplotlib inline
plt.style.use('seaborn')
sns.set_palette("husl")
```

[ ]:
```
# Configuration
DATA_DIR = 'data'
```

```
HAM_URL = 'https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.
 ↪tar.bz2'
SPAM_URL = 'https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.
 ↪bz2'

# Download and extract data
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR, exist_ok=True)
    print("Downloading datasets...")

    ham_path, _ = urlretrieve(HAM_URL, 'easy_ham.tar.bz2')
    spam_path, _ = urlretrieve(SPAM_URL, 'spam.tar.bz2')

    print("Extracting files...")
    with tarfile.open(ham_path, 'r:bz2') as tar:
        tar.extractall(DATA_DIR)
    with tarfile.open(spam_path, 'r:bz2') as tar:
        tar.extractall(DATA_DIR)
else:
    print("Data directory already exists")
```

1.3 2. Data Loading and Preparation

[ ]:
```
def load_files():
    """Load email filenames"""
    ham_dir = os.path.join(DATA_DIR, 'easy_ham')
    spam_dir = os.path.join(DATA_DIR, 'spam')

    ham_files = [f for f in os.listdir(ham_dir) if not f.startswith('.')]
    spam_files = [f for f in os.listdir(spam_dir) if not f.startswith('.')]

    return ham_files, spam_files

ham_files, spam_files = load_files()
print(f"Loaded {len(ham_files)} ham emails")
print(f"Loaded {len(spam_files)} spam emails")
```

[ ]:
```
# Create DataFrame
df = pd.DataFrame({
    'filename': ham_files + spam_files,
    'is_spam': [0]*len(ham_files) + [1]*len(spam_files)
})

# Generate UUIDs
df['email_id'] = df['filename'].apply(
    lambda x: hashlib.sha256(x.encode()).hexdigest()[:16]
)
```

```
df.head()
```

1.4 3. Feature Extraction

[ ]:
```
def extract_email_content(row):
    """Extract headers and content from raw email"""
    try:
        dir_type = 'easy_ham' if row['is_spam'] == 0 else 'spam'
        path = os.path.join(DATA_DIR, dir_type, row['filename'])

        with open(path, 'rb') as f:
            content = f.read().decode('latin-1')

        msg = email.message_from_string(content, policy=policy.default)

        return {
            'sender': msg.get('From', None),
            'subject': msg.get('Subject', None),
            'date': msg.get('Date', None),
            'content_type': msg.get('Content-Type', None),
            'content': '\n'.join(
                part.get_payload() for part in msg.walk()
                if part.get_content_type() == 'text/plain'
            )
        }
    except Exception as e:
        print(f"Error processing {row['filename']}: {e}")
        return None

# Apply feature extraction
features = df.apply(extract_email_content, axis=1).apply(pd.Series)
df = pd.concat([df, features], axis=1)

# Preview extracted features
df[['email_id', 'sender', 'subject', 'content']].head()
```

1.5 4. Feature Engineering

[ ]:
```
# Time of day extraction
def extract_time(hour_str):
    try:
        parsed = parsedate_tz(hour_str)
        return parsed[3] if parsed else None
    except:
        return None
```

```
df['time_of_day'] = df['date'].apply(extract_time)

# Text lengths
df['content_length'] = df['content'].str.len().clip(upper=df['content'].str.
 ↪len().quantile(0.99))
df['subject_length'] = df['subject'].str.len().clip(upper=df['subject'].str.
 ↪len().quantile(0.99))

# Combined text feature
df['combined_text'] = df[['content', 'subject', 'sender']].fillna('').
 ↪astype(str).agg(' '.join, axis=1)

# Handle missing values
df['time_of_day'] = df['time_of_day'].fillna(df['time_of_day'].median())
df = df.dropna(subset=['content'])

df[['time_of_day', 'content_length', 'subject_length']].describe()
```

1.6 5. Exploratory Data Analysis

[ ]:
```
# Class distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='is_spam', data=df, palette='viridis')
plt.title('Spam vs Ham Distribution')
plt.xlabel('Email Type (0=Ham, 1=Spam)')
plt.ylabel('Count')
plt.show()
```

[ ]:
```
# Time of day analysis
plt.figure(figsize=(10, 6))
sns.kdeplot(data=df, x='time_of_day', hue='is_spam', fill=True, palette='Set2')
plt.title('Email Send Time Distribution')
plt.xlabel('Hour of Day')
plt.xticks(range(0, 24, 2))
plt.xlim(0, 23)
plt.show()
```

[ ]:
```
# Word Cloud Generation
def generate_wordcloud(text, title):
    wc = WordCloud(width=800, height=400,
                   background_color='white',
                   stop_words=ENGLISH_STOP_WORDS,
                   max_words=200).generate(text)

    plt.figure(figsize=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(title)
```

```
    plt.axis('off')
    plt.show()

# Generate word clouds
generate_wordcloud(' '.join(df[df['is_spam'] == 1]['combined_text']), 'Spam␣
 ↪Email Word Cloud')
generate_wordcloud(' '.join(df[df['is_spam'] == 0]['combined_text']), 'Ham␣
 ↪Email Word Cloud')
```

1.7 6. Model Training

[ ]:
```
# Prepare data
X = df[['combined_text', 'time_of_day', 'content_length', 'subject_length']]
y = df['is_spam']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

[ ]:
```
# Build pipeline
preprocessor = ColumnTransformer([
    ('text', TfidfVectorizer(
        stop_words=list(ENGLISH_STOP_WORDS),
        ngram_range=(1, 2),
        max_df=0.9,
        min_df=5
    ), 'combined_text'),
    ('numerical', MinMaxScaler(), ['time_of_day', 'content_length',␣
 ↪'subject_length'])
])

model = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(
        class_weight='balanced',
        max_iter=1000,
        solver='liblinear'
    ))
])

# Train model
model.fit(X_train, y_train)
```

1.8 7. Model Evaluation

[ ]:
```
# Classification report
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=['Ham', 'Spam']))
```

[ ]:
```
# Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True, fmt='d', cmap='Blues',
            xticklabels=['Ham', 'Spam'],
            yticklabels=['Ham', 'Spam'])
plt.title('Confusion Matrix')
plt.show()
```

[ ]:
```
# ROC Curve
y_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

[ ]:
```
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Logistic Regression')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```