TensorFlowとPythonによるスパムメール分類：徹底ガイド

1. はじめに：スパムメール分類とは？なぜ機械学習なのか？

1.1. 問題の定義：スパムメール分類

日々大量に送られてくる電子メールの中には、宣伝広告、フィッシング詐欺、ウイルス感染を狙ったものなど、受け取り手にとって不要、あるいは有害なメール（スパムメール）が数多く含まれています。スパムメール分類とは、受信したメールが「スパム（迷惑メール）」なのか、それとも「ハム（非迷惑メール、通常のメール）」なのかを自動的に識別し、分類するタスクです [1]。これは、コンピュータがデータを特定のカテゴリに振り分ける「分類（Classification）」問題の一種です [2]。

1.2. なぜ機械学習が有効なのか？

従来、スパムメール対策としては、「特定のキーワード（例：「儲かる」「無料」）が含まれていたらスパム」といったルールを手動で設定する方法（ルールベース）がありました。しかし、スパム送信者は巧妙であり、常に新しい手口や文面を考案するため、ルールベースではすぐにすり抜けられてしまいます。人間が考えうる全てのスパムパターンを網羅するルールを作成し、維持することは現実的ではありません [4]。
ここで強力な解決策となるのが、**機械学習（Machine Learning, ML）**です。機械学習は、人間が明示的にルールをプログラムするのではなく、大量のデータからコンピュータ自身がパターンや規則性を学習する技術です [3]。スパムメール分類においては、過去のメールデータ（スパムと判定されたメール、されなかったメール）を学習させることで、モデルはスパムメールに共通する微妙で複雑な特徴（単語の組み合わせ、送信元情報、文構造など）を自動的に捉えることができるようになります [1]。これにより、ルールベースでは対応しきれない未知のスパムメールに対しても、高い精度で分類することが可能になります。実際に、Gmailのような主要なメールサービスでは、機械学習を活用してスパムフィルタの精度を高め、ユーザーの利便性を向上させています [1]。
機械学習がこの問題に適している本質的な理由は、スパムの特徴が多様で、常に変化し続ける点にあります。手動でルールを定義する方法は、このような動的な問題に対しては脆弱です [4]。一方、機械学習はデータから直接学習するため、複雑なパターンを捉え、新たなスパムの手口にも適応していく能力を持っています。これが、機械学習がスパム分類において固定的なルールよりも優れた性能を発揮する理由です [1]。

1.3. 教師あり学習によるアプローチ

スパムメール分類で用いられる機械学習の手法は、主に**教師あり学習（Supervised Learning）**と呼ばれるものです [3]。教師あり学習では、あらかじめ正解（ラベル）が付けられたデータセット（この場合は、「スパム」または「ハム」というラベルが付いたメールデータ）を使ってモデルを訓練します [3]。

モデルは、入力データ（メールの内容）と正解ラベル（スパムかハムか）の関係性を学習し、新しい未知のメールがどちらのカテゴリに属するかを予測できるようになります。
教師あり学習には、主に「分類」と「回帰」の二つのタスクがあります。「分類」はデータを特定のカテゴリに分けるタスク（例：スパムメール判定、画像認識）であり、「回帰」は連続的な数値を予測するタスク（例：株価予測、住宅価格予測）です [3]。スパムメール分類は、メールを「スパム」と「ハム」の2つのカテゴリのいずれかに分類するため、**二値分類（Binary Classification）**と呼ばれる、分類問題の中でも基本的なタイプに該当します [4]。

1.4. 今回使用するツール：Python, TensorFlow, Keras

このガイドでは、スパムメール分類器を構築するために、以下のツールを使用します。

- **Python:** 機械学習の分野で最も広く使われているプログラミング言語です。豊富なライブラリと活発なコミュニティがあり、データ分析やモデル構築に適しています [5]。
- **TensorFlow:** Googleが開発した、強力なオープンソースの機械学習フレームワークです [3]。特にニューラルネットワークの構築と訓練において、世界中で広く利用されています [12]。
- **Keras:** TensorFlowに統合されている高水準API（Application Programming Interface）です [6]。Kerasを使うことで、ニューラルネットワークの層（レイヤー）を積み重ねるように、直感的かつ簡単にモデルを構築できます。初心者にとって、複雑なモデル構築のハードルを大幅に下げてくれます [16]。
-

Pythonの使いやすさ、TensorFlowの計算能力、そしてKerasのシンプルさが組み合わさることで、初心者でもスパムメール分類のような実践的な機械学習の問題に取り組むための、アクセスしやすく強力な環境が提供されます。Pythonが土台となり、TensorFlowがエンジン、Kerasがユーザーフレンドリーな操作パネルの役割を果たすことで、複雑なタスクへの参入障壁が低くなっています [6]。

2. 環境構築：PythonとTensorFlowの準備

機械学習モデルの開発を始める前に、必要なツールを準備します。

2.1. 必要なツール

- **Python:** バージョン3.x系のインストールが推奨されます。Python公式サイトからダウンロードできます。
- **pip:** Pythonのパッケージ管理システムです。通常、Pythonと一緒にインストールされます。
- **TensorFlow:** 機械学習ライブラリ本体です。

2.2. TensorFlowのインストール

ターミナル（Windowsの場合はコマンドプロンプトまたはPowerShell）を開き、以下のコマンドを実行してTensorFlowをインストールします [14]。

Bash

```
pip install tensorflow
```

これにより、TensorFlow本体と、NumPyなどの関連ライブラリがインストールされます。特定のバージョンが必要な場合もありますが、まずはこの標準的な方法でインストールを進めましょう。
代替手段：Google Colaboratory
ローカル環境にソフトウェアをインストールすることなく、ブラウザ上でPythonコードを実行し、機械学習モデルを構築・訓練できる無料のサービスとしてGoogle Colaboratory (Colab) があります 15。ColabにはTensorFlowがプリインストールされており、GPUなどの計算リソースも無料で利用できるため、特に初心者にとっては環境構築の手間が省け、非常に便利です。

2.3. インストールの確認

インストールが正しく完了したかを確認するために、Pythonのインタラクティブシェルまたはスクリプトで以下のコードを実行します。

Python

```
import tensorflow as tf
print(tf.__version__)
```

エラーが表示されず、TensorFlowのバージョン番号（例：2.15.0）が出力されれば、インストールは成功です [8]。

3. データの収集と準備：生のメールからモデルが読める形式へ

教師あり学習モデルを訓練するには、正解ラベルが付与されたデータが不可欠です [3]。モデルの性能は、使用するデータの質と量に大きく依存します。

3.1. ラベル付きデータの重要性

スパムメール分類の場合、「このメールはスパムである（ラベル=1）」「このメールはハムである（ラベル=0）」のように、各メールに対して正解ラベルが付与されたデータセットが必要です。モデルはこのデータから学習します。

3.2. データセットの入手

練習用のデータセットは、KaggleやUCI Machine Learning Repositoryなどのプラットフォームで公開されています。このチュートリアルでは、広く使われている「**SMS Spam Collection Dataset**」を利用することを想定します [7]。これは、SMSメッセージとそのメッセージがスパムかハムかのラベルで構成されるデータセットです。通常、CSV（Comma Separated Values）形式で提供され、1列目にラベル（例：'spam' or 'ham'）、2列目にメッセージ本文が含まれています [20]。

3.3. データの読み込みと確認

データセット（例：spam.csv）をPythonで扱うには、**pandas**ライブラリが非常に便利です。以下のコードでCSVファイルを読み込み、DataFrameという形式でデータを格納できます [22]。

Python

```
import pandas as pd

# CSVファイルを読み込む（ファイルパスは適宜変更）
# header=None はファイルにヘッダー行がない場合、namesで列名を指定
df = pd.read_csv('spam.csv', encoding='latin-1', header=None, names=['label', 'message'])

# データの一部を表示して確認
print(df.head())

# データの基本情報を表示
print(df.info())

# ラベルの分布を確認
print(df['label'].value_counts())
```

.head()で先頭数行を、.info()で各列のデータ型や欠損値の有無を、.value_counts()で各ラベル（spam/ham）の件数を確認し、データの構造を把握します [20]。

3.4. テキストデータの前処理

生のメールテキストは、そのままでは機械学習モデルが処理できません。HTMLタグ、句読点、大文字・小文字の混在など、分析のノイズとなる要素が含まれており、また、アルゴリズムが理解できる数値形式になっていないためです [24]。そのため、テキスト前処理と呼ばれる、テキストをクレンジングし、数値形式に変換する一連の作業が必要になります。
ステップ1：クリーニング（Cleaning）
メール本文に含まれるHTMLタグ、特殊記号、URL、不要な空白などを除去します。Pythonの正規表現モジュールreがよく用いられます 22。
ステップ2：小文字化（Lowercasing）
テキスト全体を小文字に統一します。これにより、「FREE」と「free」のような、大文字・小文字の違いしかない単語が同じものとして扱われ、モデルが学習すべき単語の種類（語彙数）を減らすことができます 19。
ステップ3：トークン化（Tokenization）
文章を個々の単語（トークン）に分割するプロセスです 19。例えば、「This is a pen.」を「"This", "is", "a", "pen", "."」のように分割します。これにより、単語の出現頻度などを数えることが可能になります。自然言語処理ライブラリNLTKや、TensorFlow自身のTokenizerクラスなどが利用できます 22。
ステップ4：ストップワード除去（Stop Word Removal）（任意）
「the」「a」「is」「in」のような、頻繁に出現するものの、文脈上の意味をあまり持たない単語（ストップワード）を除去する処理です 22。これにより、データ量を削減し、より重要な単語に焦点を当てることができます。ただし、文脈によってはストップワードが重要な場合もあるため、常に除去が最善とは限りません 25。NLTKなどのライブラリには、一般的な英語のストップワードリストが含まれています。
ステップ5：ベクトル化（Vectorization）
前処理されたテキスト（単語のリスト）を、機械学習モデルが扱える数値ベクトルに変換する、最も重要なステップです。主な手法には以下があります。

- **Bag-of-Words (BoW):** 最も基本的な手法の一つ。各文書を、そこに含まれる単語の出現回数を要素とするベクトルで表現します。単語の順序は無視されます。
- **TF-IDF (Term Frequency-Inverse Document Frequency):** BoWの発展形。単語の出現頻度（TF）に、その単語が他の多くの文書にも出現するかどうか（IDF）を加味して重み付けします。多くの文書に出現する一般的な単語の重みは低く、特定の文書に集中して出現する単語の重みは高くなります [23]。
- 単語埋め込み **(Word Embeddings):** 各単語を、意味的に近い単語がベクトル空間上で近くに配置されるような、低次元の密な数値ベクトル（埋め込みベクトル）で表現する手法です。単語間の意味的な関係性を捉えることができます。KerasにはEmbeddingレイヤーがあり、訓練データから埋め込みベクトルを学習できます [22]。また、大規模なテキストコーパスで事前に学習された埋め込みベクトル（Word2Vec, GloVeなど）を利用することもでき、これはTensorFlow Hubなどを通じて利用可能です [9]。

ベクトル化の手法選択は、単純さ（BoW, TF-IDF）と意味表現の豊かさ（単語埋め込み）の間のトレードオフを考慮する必要があります。単語埋め込みは、特にニュアンスが重要なタスクにおいて高い性能を発揮する可能性がありますが、その能力を最大限に引き出すには、通常、ニューラルネットワー

クのようなより複雑なモデルが必要となります [9]。

表1：テキスト前処理の主なステップ

ステップ	目的	一般的なPythonツール/ライブラリ例
クリーニング	ノイズ（HTMLタグ、記号等）の除去	re (正規表現)
小文字化	大文字・小文字の統一	文字列メソッド .lower()
トークン化	文章を単語（トークン）に分割	nltk.word_tokenize, tf.keras.preprocessing.text.Tokenizer
ストップワード除去	頻出するが重要度の低い単語の除去	nltk.corpus.stopwords
ベクトル化	トークンを数値ベクトルに変換	sklearn.feature_extraction.text.TfidfVectorizer, tf.keras.layers.Embedding

表2：主なテキストベクトル化手法

手法	説明	利点	欠点
Bag-of-Words (BoW)	単語の出現頻度に基づきベクトル化	シンプル、実装が容易	単語の順序を無視、語彙数が多いと高次元で疎なベクトルになる
TF-IDF	単語の重要度（希少性）を考慮して重み付け	BoWより単語の重要度を反映できる	単語の順序を無視
単語埋め込み	単語を意味的な関係性を保持した低次元の密なベクトルで表現	単語の意味や文脈を捉えられる	計算コストが高い、十分なデータが必要

3.5. 訓練データとテストデータへの分割

モデルの性能を正しく評価するためには、モデルが訓練中に一度も見たことのないデータ（テストデータ）で評価する必要があります [7]。これにより、モデルが訓練データだけを過剰に学習（過学習）してしまい、未知のデータに対してうまく機能しない状態になっていないかを確認できます。一般的に、データセット全体を訓練用（例：80%）とテスト用（例：20%）に分割します。Pythonの**scikit-learn**ライブラリのtrain_test_split関数を使うと簡単に分割できます [22]。

Python

```
from sklearn.model_selection import train_test_split
```

```
# 特徴量（メッセージ本文のベクトル）とラベルを準備
# X: ベクトル化されたメッセージデータ
# y: ラベルデータ (0 or 1)

# データを訓練用とテスト用に分割（例：テストサイズ20%）
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# random_stateを指定すると、毎回同じように分割される（再現性のため）
```

4. スパム分類器の構築：TensorFlowとKerasを使う

データの前処理が完了したら、いよいよTensorFlowとKerasを使って分類モデル（ニューラルネットワーク）を構築します。

4.1. Kerasによる簡単なモデル構築

Kerasは、ニューラルネットワークの構築を非常にシンプルにします [15]。特に**Sequential API**は、層（レイヤー）を順番に積み重ねていくだけでモデルを定義できるため、初心者にとって非常に分かりやすい方法です [14]。

4.2. シンプルなニューラルネットワークの構築

基本的なテキスト分類のためのニューラルネットワークは、通常、以下の層を組み合わせて構築します。

- **tf.keras.layers.Embedding:** テキスト分類モデルの最初の層として配置されることが多い重要な層です。前処理ステップで整数に変換された各単語IDを受け取り、それを密なベクトル（単語埋め込みベクトル）に変換します。このベクトルは、モデルの訓練中に学習され、単語の意味的な特徴を捉えるようになります [22]。入力として、語彙サイズ（データセット中のユニークな単語数）と、埋め込みベクトルの次元数を指定します。
- **tf.keras.layers.GlobalAveragePooling1D** または **tf.keras.layers.Flatten:** Embedding層の出力は、通常（バッチサイズ, 文の長さ, 埋め込み次元数）という3次元のテンソルになります。これを後続の全結合層（Dense層）に入力するために、1次元のベクトルに変換する必要があります。GlobalAveragePooling1Dは、文の長さの次元に沿って平均を取り、文全体の意味を要約したベクトルを生成します。Flattenは単純に全次元を平坦化します [22]。GlobalAveragePooling1Dは文の長さに依存しない固定長の出力を得られるため、扱いやすいことが多いです。
- **tf.keras.layers.Dense:** 全結合層とも呼ばれます。前の層の出力（ベクトル）を受け取り、内部の重みとバイアスを使って線形変換を行い、活性化関数を適用します。これにより、データからより高次の特徴やパターンを学習します [14]。unitsパラメータで層のニューロン（出力次元

数）を指定し、activationパラメータで活性化関数を指定します。

- 活性化関数 **(Activation Function):** ニューラルネットワークが非線形な関係性を学習できるようにするために不可欠な要素です。中間層（隠れ層）では、**ReLU (Rectified Linear Unit)** ('relu') が広く使われています [14]。出力層では、今回の二値分類（スパムかハムか）のように、出力を0から1の間の確率値に変換したい場合に**Sigmoid** ('sigmoid') 関数が用いられます [14]。

以下は、これらの層をSequentialモデルで組み合わせる簡単な例です。

Python

```
import tensorflow as tf
from tensorflow.keras import layers

# パラメータ例（実際の値はデータセットや前処理に依存）
vocab_size = 10000  # 語彙サイズ
embedding_dim = 16 # 埋め込みベクトルの次元数
max_length = 100   # パディング後のシーケンス長（ベクトル化による）

model = tf.keras.Sequential()

# モデルの構造を確認
model.summary()
```

4.3. モデルのコンパイル：学習プロセスの設定

モデルの構造を定義した後、訓練を開始する前にコンパイルというステップが必要です [14]。
model.compile()メソッドを呼び出し、モデルがどのように学習を進めるかを設定します。

- オプティマイザ **(Optimizer):** モデルが訓練データを見て、自身の内部パラメータ（重み）をどのように更新していくかを決定するアルゴリズムです。**Adam** ('adam') は、多くの場合で良好な性能を発揮し、広く使われている堅牢なオプティマイザです [14]。
- 損失関数 **(Loss Function):** 訓練中に、モデルの予測が実際の正解ラベルとどれだけ乖離しているか（＝損失、誤差）を測定する関数です。モデルの訓練目標は、この損失関数の値を最小化することです。二値分類問題では、**Binary Cross-entropy** ('binary_crossentropy') が標準的な選択肢です [27]。
- メトリクス **(Metrics):** 訓練中および評価時にモデルの性能を監視するための指標です。損失関数とは異なり、モデルの学習プロセス自体には直接影響を与えませんが、人間がモデルの性能を理解するのに役立ちます。最も一般的なメトリクスは**Accuracy**（正解率、['accuracy']）で、全データのうち正しく分類できた割合を示します [14]。

Python

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

このcompileステップは、定義されたモデルのアーキテクチャ（層の構造）と、実際の学習プロセス（どのように誤差を計算し、どのようにパラメータを更新するか）を結びつける重要な橋渡し役となります。ここで選択するオプティマイザや損失関数が、モデルがデータからどのように学習するかの根幹を定義します [14]。

5. モデルの訓練：分類器にスパムを見分ける方法を教える

モデルの構築とコンパイルが完了したら、準備した訓練データを使ってモデルを訓練します。

5.1. 訓練の開始

モデルの訓練は、model.fit()メソッドを呼び出すことで開始します [14]。

Python

```
# 訓練データ (X_train, y_train) を使ってモデルを訓練
# epochs: 訓練データ全体を何回繰り返して学習するか
# batch_size: 1回のパラメータ更新に使うデータ数
# validation_data: 各エポック終了時に性能を評価するための検証データ
history = model.fit(X_train,
                    y_train,
                    epochs=10, # 例として10エポック
                    batch_size=32, # 例としてバッチサイズ32
                    validation_data=(X_test, y_test), # テストデータを検証用に使用
                    verbose=1) # 訓練の進捗状況を表示
```

5.2. 主要な訓練パラメータ

model.fit()にはいくつかの重要なパラメータを渡します。

- X_train, y_train: 訓練用の特徴量データ（ベクトル化されたテキスト）と正解ラベル。

- **epochs**: 訓練データセット全体を何回繰り返して学習に使用するかを指定します [3]。1エポックで、モデルは訓練データ全体を一通り見ることになります。通常、複数エポックの訓練が必要です。
- **batch_size**: 訓練データを一度にすべて処理するのではなく、小さなバッチに分割して処理します。batch_sizeは、1回のパラメータ更新（学習ステップ）で使用するサンプル数を指定します [3]。バッチサイズが小さいと学習のノイズが大きくなる可能性がありますが、汎化性能が向上することもあります。大きいと学習は高速になりますが、局所解に陥りやすくなる可能性もあります。
- **validation_data**: 各エポックの終了時に、モデルの性能を評価するために使用するデータセットを指定します。通常、訓練データとは別に用意した検証データセット、または（この例のように）テストデータセットを指定します。これにより、訓練中のモデルが未知のデータに対してどの程度の性能を発揮しているかを監視し、過学習の兆候を早期に発見できます。

5.3. 訓練の監視

model.fit()を実行すると、各エポックの終了時に損失（loss）と指定したメトリクス（例：accuracy）の値が、訓練データと検証データ（validation_dataで指定したもの）それぞれについて表示されます。これらの値の変化を観察することで、モデルが順調に学習しているか、あるいは過学習や学習不足に陥っていないかなどを判断する手がかりになります。

6. モデル性能の評価：分類器はどれくらい優秀か？

モデルの訓練が完了したら、その性能を客観的に評価する必要があります。訓練データに対する性能だけでなく、モデルが未知のデータに対してどれだけうまく機能するかが重要です [17]。

6.1. なぜ評価が重要か？

訓練データに対する性能（例：訓練時の正解率）が高いだけでは、良いモデルとは限りません。モデルが訓練データに過剰に適合（過学習）してしまい、新しいデータに対してはうまく予測できない可能性があるためです。そのため、訓練には使用しなかったテストデータを使って、モデルの真の汎化性能を評価する必要があります。

6.2. テストデータによる評価

モデルの評価は、model.evaluate()メソッドを使って行います。このメソッドにテストデータ（X_test）と正解ラベル（y_test）を渡すと、コンパイル時に指定した損失とメトリクス（例：損失値と正解率）を計算して返します [14]。

Python

```
# テストデータでモデルの性能を評価
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
```

6.3. 正解率（Accuracy）だけでは不十分な場合

正解率（Accuracy）は直感的に分かりやすい指標ですが、特に**データが不均衡（Imbalanced）**な場合には注意が必要です [29]。例えば、スパムメールが全体の1%しかないデータセットの場合、全てのメールを「ハム（非スパム）」と予測するモデルでも、正解率は99%になってしまいます。しかし、このモデルはスパムメールを全く検出できないため、実用的ではありません。
このような状況でモデルの性能をより正確に評価するために、以下のメトリクスがよく用いられます [27]。

- 適合率 **(Precision):** モデルが「スパム」と予測したメールのうち、実際にスパムであったものの割合。「スパムと予測したものがどれだけ正しいか」を示します。適合率が高いと、誤ってハムをスパムと判定してしまう（False Positive）リスクが低いことを意味します。重要なメールをブロックしたくない場合に重視されます。
- 再現率 **(Recall / Sensitivity):** 実際にスパムであったメールのうち、モデルが正しく「スパム」と予測できたものの割合。「実際のスパムをどれだけ見つけられたか」を示します。再現率が高いと、スパムメールを見逃してしまう（False Negative）リスクが低いことを意味します。できるだけ多くのスパムを検出したい場合に重視されます。
- **F1**スコア **(F1-Score):** 適合率と再現率の調和平均。両者のバランスを取った単一の指標です。適合率と再現率のどちらか一方だけが高くても、F1スコアは高くならないため、バランスの取れた性能評価に適しています。

適合率と再現率はトレードオフの関係にあることが多く、一方を高めようとすると他方が下がることがあります。どちらをより重視するかは、アプリケーションの目的（スパムの見逃しと正常メールの誤ブロックのどちらがより問題か）によって異なります。

表**3**：主な分類評価メトリクス

メトリクス	問い	概要	重要となる場面
正解率 (Accuracy)	全体として、どれくらいの頻度で分類器は正しいか？	全予測に対する正解予測の割合	データクラスのバランスが良い場合、全体的な性能把握

適合率 (Precision)	スパムと予測したとき、どれくらいの頻度で実際にスパムか？	陽性（スパム）と予測されたうち、実際に陽性だった割合	偽陽性（False Positive）のコストが高い場合（例：正常メール誤ブロック）
再現率 (Recall)	実際のスパムのうち、どれだけを検出できたか？	実際の陽性（スパム）のうち、正しく陽性と予測された割合	偽陰性（False Negative）のコストが高い場合（例：スパム見逃し）
F1スコア (F1-Score)	適合率と再現率のバランスは取れているか？	適合率と再現率の調和平均	適合率と再現率の両方をバランス良く考慮したい場合

6.4. 過学習と学習不足について

モデル評価の際には、**過学習（Overfitting）と学習不足（Underfitting）**にも注意が必要です。

- 過学習: モデルが訓練データに過剰に適合し、訓練データに含まれるノイズまで学習してしまった状態。訓練データに対する精度は非常に高いものの、テストデータのような未知のデータに対する精度が低くなります [3]。訓練中の損失が下がり続ける一方で、検証データの損失が上昇し始めた場合、過学習の兆候と考えられます。
- 学習不足: モデルが単純すぎるか、訓練が不十分なために、訓練データの特徴すら十分に捉えられていない状態。訓練データ・テストデータともに精度が低い場合に該当します。

これらは機械学習でよく遭遇する問題であり、対策としては、データ量を増やす、モデルの複雑さを調整する（層を減らす、ユニット数を減らす）、正則化と呼ばれるテクニックを導入するなど、様々な方法があります（これらは本入門ガイドの範囲を超えます）。

7. モデルの実用：新しいメールのスパム判定

訓練と評価が完了したモデルを使って、新しい未知のメールがスパムかどうかを予測（推論）する方法を見ていきましょう。

7.1. 予測の実行

モデルによる予測は、model.predict()メソッドを使って行います [14]。このメソッドに、予測したい新しいデータ（例：新しいメールのテキスト）を入力として渡します。

7.2. 新しいデータの前処理

非常に重要な注意点があります。model.predict()に入力する新しいテキストデータは、訓練データと全く同じ前処理（クリーニング、小文字化、トークン化、ベクトル化など）を施す必要があります。モデルは特定の形式に処理されたデータで学習しているため、異なる形式のデータを与えても正しく機能

しません。
以下は、新しいテキストを前処理して予測する簡単な例です。

Python

```
# 新しいメールテキストの例
new_email_text = ["Congratulations! You've won a free cruise. Click here now!"]

# 訓練時と同じ前処理を適用する関数 (例)
def preprocess_text(texts, tokenizer, max_length):
    # テキストのクリーニングや小文字化など（訓練時と同じ処理）
    #... (省略)...
    # トークン化とシーケンスへの変換
    sequences = tokenizer.texts_to_sequences(texts)
    # パディング
    padded_sequences = tf.keras.preprocessing.sequence.pad_sequences(sequences,
maxlen=max_length, padding='post', truncating='post')
    return padded_sequences

# 訓練時に使用したTokenizerとmax_lengthを使って前処理
# tokenizer は訓練時にfit_on_textsしたもの
new_email_processed = preprocess_text(new_email_text, tokenizer, max_length)

# 予測を実行
prediction_probability = model.predict(new_email_processed)
print(f"Prediction Probability: {prediction_probability}")
```

7.3. 予測結果の解釈

Sigmoid活性化関数を出力層に使用した二値分類モデルの場合、predict()メソッドは通常、0から1の間の確率値を出力します [30]。この値は、入力されたメールが「スパム（クラス1）」である確率を示唆します。
最終的な分類結果（スパムかハムか）を得るためには、**閾値（Threshold）**を設定します。一般的には0.5を閾値とし、確率が0.5より大きければ「スパム（1）」、0.5以下であれば「ハム（0）」と判定します。

Python

```
threshold = 0.5
if prediction_probability > threshold:
    print("Prediction: Spam")
```

```
else:
    print("Prediction: Ham")
```

この閾値は、適合率と再現率のバランスを考慮して調整することも可能です。

8. 実装例：完全なコードウォークスルー

これまでのステップを統合し、データ読み込みから予測までを行う完全なPythonコードの例を示します。ここでは、KerasのTokenizerとpad_sequencesを使った前処理を想定します。

Python

```
import pandas as pd
import numpy as np
import re
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

# --- 1. データ読み込み ---
try:
    # KaggleのSMS Spam Collection Datasetを想定
    df = pd.read_csv('spam.csv', encoding='latin-1')
    # 必要な列だけを選択し、列名を変更
    df = df[['v1', 'v2']]
    df.columns = ['label', 'message']
except FileNotFoundError:
    print("エラー: spam.csvが見つかりません。データセットをダウンロードして同じディレクトリに配置してください。")
    exit()

print("データ読み込み完了:")
print(df.head())

# --- 2. 前処理 ---
```

```
# ラベルを数値に変換 (ham -> 0, spam -> 1)
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['label'])

# テキストクリーニング関数 (例：簡単なクリーニング)
def clean_text(text):
    text = text.lower() # 小文字化
    text = re.sub(r'[^a-z\s]', '', text) # アルファベットと空白以外を除去
    text = re.sub(r'\s+', ' ', text).strip() # 連続する空白を一つに
    return text

df['cleaned_message'] = df['message'].apply(clean_text)
print("\nテキストクリーニング後:")
print(df[['label_encoded', 'cleaned_message']].head())

# Tokenizerの設定
vocab_size = 10000 # 語彙サイズ（頻出上位10000語）
oov_token = "<OOV>" # 語彙に含まれない単語の扱い
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)

# テキストデータでTokenizerを学習
tokenizer.fit_on_texts(df['cleaned_message'])
word_index = tokenizer.word_index # 単語とインデックスのマッピング

# テキストを整数のシーケンスに変換
sequences = tokenizer.texts_to_sequences(df['cleaned_message'])

# シーケンスの長さを揃える (パディング)
max_length = 100 # シーケンスの最大長（適宜調整）
padding_type = 'post' # パディングを後ろに追加
truncation_type = 'post' # 長すぎる場合に後ろを切り捨て
padded_sequences = pad_sequences(sequences, maxlen=max_length,
padding=padding_type, truncating=truncation_type)

print("\nベクトル化（パディング済みシーケンス）の例:")
print(padded_sequences)

# --- 3. データの分割 ---
X = padded_sequences
y = df['label_encoded'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"\n訓練データ数: {len(X_train)}, テストデータ数: {len(X_test)}")
```

```
# --- 4. モデル構築 ---
embedding_dim = 16

model = Sequential([
  Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_length),
  GlobalAveragePooling1D(),
  Dense(24, activation='relu'),
  Dense(1, activation='sigmoid') # 二値分類
])

# --- 5. モデルのコンパイル ---
model.compile(loss='binary_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])

model.summary() # モデル構造の表示

# --- 6. モデル訓練 ---
num_epochs = 10
batch_size = 32

print("\nモデル訓練開始...")
history = model.fit(X_train, y_train,
                epochs=num_epochs,
                batch_size=batch_size,
                validation_data=(X_test, y_test),
                verbose=1)
print("モデル訓練完了。")

# --- 7. モデル評価 ---
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"\nテストデータでの損失 (Loss): {loss:.4f}")
print(f"テストデータでの正解率 (Accuracy): {accuracy:.4f}")

# より詳細な評価 (適合率、再現率、F1スコア)
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten() # 確率を0/1のクラスに変換

print("\nテストデータでの分類レポート:")
print(classification_report(y_test, y_pred, target_names=))
```

```
# --- 8. 新しいメールでの予測 ---
def predict_spam(text, model, tokenizer, max_length):
    # 1. 前処理
    cleaned = clean_text(text)
    sequence = tokenizer.texts_to_sequences([cleaned]) # リストとして渡す
    padded = pad_sequences(sequence, maxlen=max_length, padding=padding_type,
truncating=truncation_type)
    # 2. 予測
    prediction = model.predict(padded)
    # 3. 結果解釈
    probability = prediction
    label = "Spam" if probability > 0.5 else "Ham"
    return label, probability

# 予測例
new_email_1 = "URGENT! You have won a 1 week FREE membership in our £100,000 Prize
Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX
4403LDNW1A7RW18"
new_email_2 = "Hey, are we still on for dinner tonight?"

label1, prob1 = predict_spam(new_email_1, model, tokenizer, max_length)
print(f"\n予測例1: '{new_email_1[:50]}...' -> {label1} (確率: {prob1:.4f})")

label2, prob2 = predict_spam(new_email_2, model, tokenizer, max_length)
print(f"予測例2: '{new_email_2[:50]}...' -> {label2} (確率: {prob2:.4f})")
```

注：このコードを実行するには、*pandas, numpy, tensorflow, scikit-learn*が必要です。また、*spam.csv*データセットが同じディレクトリに存在する必要があります。

9. まとめと次のステップ

9.1. まとめ

このガイドでは、TensorFlowとPythonを使ってスパムメール分類器を構築する基本的なプロセスを一通り解説しました。以下のステップを学びました。

1. 問題理解: スパムメール分類が機械学習（特に教師あり学習の二値分類）の問題であること。
2. 環境構築: PythonとTensorFlowのインストール。
3. データ準備: ラベル付きデータセットの入手、pandasによる読み込み、テキストの前処理（クリーニング、小文字化、トークン化、ベクトル化）、訓練/テストデータへの分割。
4. モデル構築: Keras Sequential APIを使ったシンプルなニューラルネットワーク（Embedding

層、Pooling層、Dense層）の定義とコンパイル（オプティマイザ、損失関数、メトリクスの設定）。
5. モデル訓練: model.fit()による訓練の実行と監視。
6. モデル評価: model.evaluate()によるテストデータでの性能評価、およびAccuracy以外の重要指標（Precision, Recall, F1-Score）の理解。
7. モデル利用: model.predict()による新しいメールに対する予測と結果の解釈。

9.2. 主要な学び

- データ前処理の重要性: 機械学習モデル、特にテキストデータを扱うモデルにとって、適切な前処理は性能を大きく左右する重要なステップです。
- **TensorFlow/Keras**の利便性: TensorFlowとKerasは、複雑なニューラルネットワークの構築と訓練を比較的容易に行える強力なツールです。
- 評価指標の選択: モデルの性能を正しく評価するためには、Accuracyだけでなく、問題の性質（例：不均衡データ）に応じてPrecision, Recall, F1-Scoreなどの適切な指標を用いることが重要です。

9.3. さらなる探求のために

このガイドは入門編ですが、さらに深く学びたい方のために、以下のような発展的なトピックがあります。

- モデルアーキテクチャの実験: より複雑なモデル（例：複数のDense層を追加、畳み込みニューラルネットワーク(CNN) [22]、再帰型ニューラルネットワーク(RNN)やLSTM [3]）を試す。
- ハイパーパラメータ調整: 学習率、バッチサイズ、埋め込み次元数などのハイパーパラメータを調整して性能向上を目指す [3]。Keras Tunerなどのツールも利用できます。
- 異なる前処理・ベクトル化手法: ストップワード除去の有無、異なるトークン化手法、TF-IDFなどの他のベクトル化手法を試す。
- 事前学習済み埋め込みの利用: Word2VecやGloVe、またはTensorFlow Hub [9] で提供される事前学習済みの単語埋め込みを利用することで、少ないデータでも高い性能が得られる可能性があります。
- 不均衡データへの対処: スパムメールのようにクラス間のデータ数が大きく異なる場合、オーバーサンプリング、アンダーサンプリング、クラス重み付けなどのテクニックを適用する [29]。
- モデルのデプロイ: 訓練済みモデルを実際のアプリケーションで利用可能にする（例：TensorFlow Serving [31] を使ったAPI化）。
- 公式ドキュメントの活用: TensorFlowとKerasの公式ウェブサイトには、さらに多くのチュートリアルやガイド、APIリファレンスが豊富に用意されています [15]。

このガイドが、機械学習とTensorFlow/Pythonを使ったテキスト分類の世界への第一歩となることを

願っています。

引用文献

1. 【初心者向け】PythonによるAI（人工知能）のつくり方 - 忍者CODEマガジン, 4月 30, 2025にアクセス、https://ninjacode.work/magazine/basic/how-to-make-python-ai/
2. メール分類の機械学習活用法 | 業務効率化と顧客対応向上の秘訣 - Hakky Handbook, 4月 30, 2025にアクセス、https://book.st-hakky.com/purpose/mail-classification-efficiency/
3. 機械学習・予測モデルに関するTensorFlowとGoogle Colab用語解説 - note, 4月 30, 2025にアクセス、https://note.com/yukikkoaimanabi/n/na3c3790b7256
4. TensorFlow入門 - Machine Learningの用語・概念説明 #Python - Qiita, 4月 30, 2025にアクセス、https://qiita.com/negabaro/items/d79bdc91f521b390e849
5. 【初心者向け】Pythonで行う機械学習モデル「分類」「回帰」の基礎知識 - テックファーム, 4月 30, 2025にアクセス、https://www.techfirm.co.jp/blog/python-classification-regression
6. Pythonで学ぶ教師あり学習の実践例 | データ分析スキルを向上させる方法 - Hakky Handbook, 4月 30, 2025にアクセス、https://book.st-hakky.com/data-science/python-supervised-learning-sample/
7. 機械学習概論/演習 - PukiWiki, 4月 30, 2025にアクセス、https://www2.cmds.kobe-u.ac.jp/wiki/bizreach/?%E6%A9%9F%E6%A2%B0%E5%AD%A6%E7%BF%92%E6%A6%82%E8%AB%96/%E6%BC%94%E7%BF%92
8. 映画レビューのテキスト分類 | TensorFlow Core, 4月 30, 2025にアクセス、https://www.tensorflow.org/tutorials/keras/text_classification?hl=ja
9. TensorFlow Hub によるテキストの分類: 映画レビュー, 4月 30, 2025にアクセス、https://www.tensorflow.org/tutorials/keras/text_classification_with_hub?hl=ja
10. Keras text classification - docs ncloud, 4月 30, 2025にアクセス、https://guide.ncloud-docs.com/docs/ja/tensorflow-keras-text-classification
11. TensorFlow.jsの話 - line engineering, 4月 30, 2025にアクセス、https://engineering.linecorp.com/ja/blog/tensorflow-js
12. 機械学習をやる人におすすめ！ライブラリ「TensorFlow」とその活用方法【入門編】- お多福ラボ, 4月 30, 2025にアクセス、https://otafuku-lab.co/aizine/tensorflow-beginner0521/
13. Pythonによる機械学習入門！できることや勉強方法を紹介 - レバテックキャリア, 4月 30, 2025にアクセス、https://career.levtech.jp/guide/knowhow/article/556/
14. PythonのTensorFlowを使い倒すためのチュートリアル - Qiita, 4月 30, 2025にアクセス、https://qiita.com/automation2025/items/4b58fea9f2abd88bd9d1
15. 【初心者向けTensorFlow入門】ディープラーニングで手書き文字認識する方法を解説！, 4月 30, 2025にアクセス、https://pythonsoba.tech/tensorflow/
16. tensorflow - ニューラルネットワークの実装（分類） - KIKAGAKU, 4月 30, 2025にアクセス、https://free.kikagaku.ai/tutorial/basic_of_deep_learning/learn/tensorflow_classification
17. TensorFlow, Kerasの基本的な使い方（モデル構築・訓練・評価・予測）| note.nkmk.me, 4月 30, 2025にアクセス、

https://note.nkmk.me/python-tensorflow-keras-basics/
18. ナイーブベイズ：Pythonによるスパムフィルタの実装 - SEEDATA, 4月 30, 2025にアクセス、https://seedata.jp/blog-tech-2156/
19. 機械学習による迷惑メールの分類 - File Exchange - MATLAB Central - MathWorks, 4月 30, 2025にアクセス、https://www.mathworks.com/matlabcentral/fileexchange/74531
20. spam detection tensorflow v2.ipynb - GitHub, 4月 30, 2025にアクセス、https://github.com/MGCodesandStats/tensorflow-nlp/blob/master/spam%20detection%20tensorflow%20v2.ipynb
21. 機械学習 〜 迷惑メール分類（ナイーブベイズ分類器） 〜 - Qiita, 4月 30, 2025にアクセス、https://qiita.com/fujin/items/50fe0e0227ef8457a473
22. pythonによるテキストマイニングことはじめ スパムフィルタリングpart.1【プログラムあり】, 4月 30, 2025にアクセス、https://software-data-mining.com/python%E3%81%AB%E3%82%88%E3%82%8B%E3%83%86%E3%82%AD%E3%82%B9%E3%83%88%E3%83%9E%E3%82%A4%E3%83%8B%E3%83%B3%E3%82%B0%E3%81%93%E3%81%A8%E3%81%AF%E3%81%98%E3%82%81%E3%80%90%E3%83%97%E3%83%AD%E3%82%B0/
23. pythonによるテキストマイニングことはじめ スパムフィルタリングpart.2【プログラムあり】, 4月 30, 2025にアクセス、https://software-data-mining.com/python%E3%81%AB%E3%82%88%E3%82%8B%E3%83%86%E3%82%AD%E3%82%B9%E3%83%88%E3%83%9E%E3%82%A4%E3%83%8B%E3%83%B3%E3%82%B0%E3%81%93%E3%81%A8%E3%81%AF%E3%81%98%E3%82%81-part-2%E3%80%90%E3%83%97%E3%83%AD/
24. 【練習問題】スパムメール分類 | SIGNATE - Data Science Competition, 4月 30, 2025にアクセス、https://signate.jp/competitions/104
25. 【練習問題】スパムメール分類 | SIGNATE - Data Science Competition, 4月 30, 2025にアクセス、https://signate.jp/competitions/104/tutorials
26. はじめてのニューラルネットワーク：分類問題の初歩 | TensorFlow Core, 4月 30, 2025にアクセス、https://www.tensorflow.org/tutorials/keras/classification?hl=ja
27. 「ゼロから作るDeep Learning」自習メモ（その15） TensorFlowの初心者向けチュートリアル - Qiita, 4月 30, 2025にアクセス、https://qiita.com/slow_learner/items/f9ba7f84053684da750a
28. 初心者のための TensorFlow 2.0 入門, 4月 30, 2025にアクセス、https://www.tensorflow.org/tutorials/quickstart/beginner?hl=ja
29. 不均衡データの分類 | TensorFlow Core, 4月 30, 2025にアクセス、https://www.tensorflow.org/tutorials/structured_data/imbalanced_data?hl=ja
30. TensorFlow.js: コメントスパム検出システムを構築する - Google for Developers, 4月 30, 2025にアクセス、https://developers.google.com/codelabs/tensorflowjs-comment-spam-detection?hl=ja
31. Flutter アプリを作成してテキストを分類する - Google for Developers, 4月 30, 2025にアクセス、https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving?hl=ja