# Homework 7: Fake News Classification

In [2]:
```python
import tensorflow as tf
import pandas as pd
import numpy as np
from nltk.corpus import stopwords
import nltk
import string
import re
from tensorflow.keras.layers import TextVectorization
from tensorflow.keras import layers, Model, Input
import matplotlib.pyplot as plt

#Worked with Kai Bengston
```

## Acquiring data

Read the csv file from the given link

```
train_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_train.csv?raw=true"
```

In [4]:
```python
# Define the URL for the CSV file
train_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_train.csv?raw=true"

# Read the CSV file from the URL
train_data = pd.read_csv(train_url)
print(train_data.head())
```

```
   Unnamed: 0                                              title  \
0       17366  Merkel: Strong result for Austria's FPO 'big c...
1        5634       Trump says Pence will lead voter fraud panel
2       17487  JUST IN: SUSPECTED LEAKER and “Close Confidant...
3       12217  Thyssenkrupp has offered help to Argentina ove...
4        5535  Trump say appeals court decision on travel ban...

                                                text  fake
0  German Chancellor Angela Merkel said on Monday...     0
1  WEST PALM BEACH, Fla.President Donald Trump sa...     0
2  On December 5, 2017, Circa s Sara Carter warne...     1
3  Germany s Thyssenkrupp, has offered assistance...     0
4  President Donald Trump on Thursday called the ...     0
```

## Make Datasets

Write a function called `make_dataset`. This function should do four things:

1. Change the text to lowercase.
2. Remove stopwords from the article text and title. A stopword is a word that is usually considered to be uninformative, such as “the,” “and,” or “but.” You may find this StackOverFlow thread to be helpful.
3. Construct and return a tf.data.Dataset with two inputs and one output. The input should be of the form (title, text), and the output should consist only of the fake column. You may find it helpful to consult lecture notes or this tutorial for reference on how to construct and use Datasets with multiple inputs.

Helpful resources:

1. Lower case: https://saturncloud.io/blog/how-to-lowercase-a-pandas-dataframe-string-column-if-it-has-missing-values/
2. Remove stopwords: https://stackoverflow.com/questions/29523254/python-remove-stop-words-from-pandas-dataframe. You need to install nltk library. If you do not want to install it, you can use Google colab.
3. Tensorflow dataset with multiple inputs and set batch: https://stackoverflow.com/questions/52582275/tf-data-with-multiple-inputs-outputs-in-keras.

In [6]:
```python
# Ensure that nltk's stopwords are downloaded
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def make_dataset(df):
    """
    Creates separate datasets for different model types.

    Parameters:
    - df: A Pandas DataFrame containing 'title', 'text', and 'fake' columns

    Returns:
    - title_ds: Dataset with title inputs and labels
    - text_ds: Dataset with text inputs and labels
    - combined_ds: Dataset with both title and text inputs and labels
    """
    # Convert Text and Titles to Lowercase
    df['title'] = df['title'].str.lower()
    df['text'] = df['text'].str.lower()

    # Remove stopwords
    def remove_stopwords(text):
        if isinstance(text, str):
            words = text.split()
            filtered = [word for word in words if word not in stop_words]
            return ' '.join(filtered)
        return text

    df['title'] = df['title'].apply(remove_stopwords)
    df['text'] = df['text'].apply(remove_stopwords)

    # Extract inputs and labels
    titles = df['title'].values
    texts = df['text'].values
    labels = df['fake'].values

    # Create separate datasets for each model type
    title_ds = tf.data.Dataset.from_tensor_slices((titles, labels))
    text_ds = tf.data.Dataset.from_tensor_slices((texts, labels))
    combined_ds = tf.data.Dataset.from_tensor_slices(
        ({'title': titles, 'text': texts}, labels)
    )

    return title_ds, text_ds, combined_ds
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kaibe\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

## Train test split:

Write a function `train_test_split` to do train test split for any tensorflow dataset. The passing arguement should be the training size.

Then, you should use your function to do train_test split and the trainning size is 80% of your dataset.

In [8]:
```python
def train_test_split(tf_dataset, train_size=0.8):
    """
    Splits a TensorFlow dataset into training and testing datasets.

    Parameters:
    - tf_dataset: The input TensorFlow dataset to be split.
    - train_size: A float between 0 and 1 representing the proportion of data for training.

    Returns:
    - train_dataset: The training subset of the dataset.
    - test_dataset: The testing subset of the dataset.
    """
    dataset_size = len(list(dataset))
    train_size = int(train_size * dataset_size)
```

```python
    train_dataset = dataset.take(train_size)
    val_dataset = dataset.skip(train_size)

    return train_dataset, val_dataset
```

# Text vectorization

Here is one option:

```python
#preparing a text vectorization layer for tf model
size_vocabulary = 2000

def standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    no_punctuation = tf.strings.regex_replace(lowercase,
                                  '[%s]' % re.escape(string.punctuation),'')
    return no_punctuation

title_vectorize_layer = TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary, # only consider this many words
    output_mode='int',
    output_sequence_length=500)

title_vectorize_layer.adapt(train.map(lambda x, y: x["title"]))
```

You can also use your preferred vectorization, e.g. text vectorization. You are also welcome to change the parameters such as `size_vocabulary` and `output_sequence_length`.

In [10]:
```python
# Constants for vectorization configuration
VOCAB_SIZE = 2000
MAX_LENGTH = 500

def create_text_vectorizer():
    """
    Creates and configures a text vectorization layer for processing input text.

    Returns:
        TextVectorization layer with standardization and integer encoding
    """
    def normalize_text(text):
        """Helper function to standardize input text"""
        # Convert to lowercase
        return tf.strings.regex_replace(
            lowercase,
            f'[{re.escape(string.punctuation)}]',
            ''
        )

        return TextVectorization(
        standardize=normalize_text,
        max_tokens=VOCAB_SIZE,
        output_mode='int',
        output_sequence_length=MAX_LENGTH
    )
```

# Create Models

Please use Keras models to offer a perspective on the following question:

`When detecting fake news, is it most effective to focus on only the title of the article, the full text of the article, or both?` To address this question, create three (3) Keras models.

In the first model, you should use only the article title as an input.

In the second model, you should use only the article text as an input.

In the third model, you should use both the article title and the article text as input.

Train your models on the training data until they appear to be “fully” trained. Assess and compare their performance. Make sure to include a visualization of the training histories.

## Notes:

1. For the first two models, you don’t have to create new Datasets. Instead, just specify the inputs to the keras.Model appropriately, and Keras will automatically ignore the unused inputs in the Dataset.

2. The lecture notes and tutorials linked above are likely to be helpful as you are creating your models as well.

3. You will need to use the Functional API, rather than the Sequential API, for this modeling task.

4. When using the Functional API, it is possible to use the same layer in multiple parts of your model; see this tutorial for examples. I recommended that you share a text vectorization layer and an embedding layer for both the article title and text inputs.

   Note: Do not use the shared embedding layer with separate text vectorization layers. If you do so, you will be embedding two different words on the same coordinate.

5. You may encounter overfitting, in which case Dropout layers can help.

You’re free to be creative when designing your models. If you’re feeling very stuck, start with some of the pipelines for processing text that we’ve seen in lecture, and iterate from there. Please include in your discussion some of the things that you tried and how you determined the models you used.

In [12]:
```
def create_and_adapt_vectorizers(train_data):
    """Creates and adapts text vectorization layers for title and text"""
    # Create vectorization layers
    title_vectorizer = TextVectorization(
        standardize='lower',
        max_tokens=VOCAB_SIZE,
        output_mode='int',
        output_sequence_length=MAX_LENGTH
    )

    text_vectorizer = TextVectorization(
        standardize='lower',
        max_tokens=VOCAB_SIZE,
        output_mode='int',
        output_sequence_length=MAX_LENGTH
    )

    # Adapt the vectorization layers to the training data
    title_vectorizer.adapt(train_data['title'])
    text_vectorizer.adapt(train_data['text'])

    return title_vectorizer, text_vectorizer

# Create and adapt vectorizers
title_vectorize_layer, text_vectorize_layer = create_and_adapt_vectorizers(train_data)

# Now create the models using the vectorization layers
def create_text_processing_model(input_layer, vectorize_layer):
    """Creates a text processing model architecture"""
    # Vectorize text input
    x = vectorize_layer(input_layer)

    # Embedding layer
    x = layers.Embedding(VOCAB_SIZE, 32)(x)

    # Bidirectional LSTM
    x = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(x)
    x = layers.Bidirectional(layers.LSTM(32))(x)

    # Dense layers with dropout
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
```

```python
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dropout(0.2)(x)

    return x

def create_title_model():
    """Creates model that only uses article titles"""
    title_input = Input(shape=(1,), dtype=tf.string, name='title')
    x = create_text_processing_model(title_input, title_vectorize_layer)
    outputs = layers.Dense(1, activation='sigmoid')(x)
    model = Model(inputs=title_input, outputs=outputs)
    model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
    return model

def create_text_model():
    """Creates model that only uses article text"""
    text_input = Input(shape=(1,), dtype=tf.string, name='text')
    x = create_text_processing_model(text_input, text_vectorize_layer)
    outputs = layers.Dense(1, activation='sigmoid')(x)
    model = Model(inputs=text_input, outputs=outputs)
    model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
    return model

def create_combined_model():
    """Creates model that uses both title and text"""
    title_input = Input(shape=(1,), dtype=tf.string, name='title')
    text_input = Input(shape=(1,), dtype=tf.string, name='text')

    title_features = create_text_processing_model(title_input, title_vectorize_layer)
    text_features = create_text_processing_model(text_input, text_vectorize_layer)

    x = layers.concatenate([title_features, text_features])
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dropout(0.2)(x)
    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[title_input, text_input], outputs=outputs)
    model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
    return model
```

In [13]:

```python
# Create datasets using make_dataset function
title_ds, text_ds, combined_ds = make_dataset(train_data)

# Split each dataset into train and validation sets
def train_val_split(dataset, train_size=0.8):
    """
    splits the data into test and train sets
    """
    dataset = dataset.shuffle(10000)
    dataset_size = len(list(dataset))
    train_size = int(train_size * dataset_size)
    train_ds = dataset.take(train_size)
    val_ds = dataset.skip(train_size)
    return train_ds.batch(32), val_ds.batch(32)

# Create train and validation datasets for each model
train_title_ds, val_title_ds = train_val_split(title_ds)
train_text_ds, val_text_ds = train_val_split(text_ds)
train_combined_ds, val_combined_ds = train_val_split(combined_ds)

# Create the models
title_model = create_title_model()
text_model = create_text_model()
combined_model = create_combined_model()
```

```python
# Train the models
history_title = title_model.fit(
    train_title_ds,
    validation_data=val_title_ds,
    epochs=10,
    batch_size=32
)

history_text = text_model.fit(
    train_text_ds,
    validation_data=val_text_ds,
    epochs=10,
    batch_size=32
)

history_combined = combined_model.fit(
    train_combined_ds,
    validation_data=val_combined_ds,
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 170s 294ms/step - accuracy: 0.8160 - loss: 0.3643 - val_accuracy: 0.9588 - val_loss: 0.1075
Epoch 2/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 147s 262ms/step - accuracy: 0.9564 - loss: 0.1156 - val_accuracy: 0.9686 - val_loss: 0.0843
Epoch 3/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 151s 269ms/step - accuracy: 0.9665 - loss: 0.0878 - val_accuracy: 0.9677 - val_loss: 0.0807
Epoch 4/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 147s 262ms/step - accuracy: 0.9595 - loss: 0.1061 - val_accuracy: 0.9771 - val_loss: 0.0635
Epoch 5/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 140s 250ms/step - accuracy: 0.9743 - loss: 0.0692 - val_accuracy: 0.9782 - val_loss: 0.0590
Epoch 6/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 139s 247ms/step - accuracy: 0.9778 - loss: 0.0589 - val_accuracy: 0.9804 - val_loss: 0.0533
Epoch 7/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 143s 254ms/step - accuracy: 0.9816 - loss: 0.0533 - val_accuracy: 0.9851 - val_loss: 0.0408
Epoch 8/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 143s 255ms/step - accuracy: 0.9837 - loss: 0.0438 - val_accuracy: 0.9811 - val_loss: 0.0495
Epoch 9/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 140s 249ms/step - accuracy: 0.9859 - loss: 0.0390 - val_accuracy: 0.9871 - val_loss: 0.0331
Epoch 10/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 145s 259ms/step - accuracy: 0.9856 - loss: 0.0385 - val_accuracy: 0.9875 - val_loss: 0.0329
Epoch 1/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 161s 265ms/step - accuracy: 0.8363 - loss: 0.3078 - val_accuracy: 0.9724 - val_loss: 0.0843
Epoch 2/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 146s 260ms/step - accuracy: 0.9670 - loss: 0.1019 - val_accuracy: 0.9621 - val_loss: 0.1185
Epoch 3/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 145s 258ms/step - accuracy: 0.9668 - loss: 0.0963 - val_accuracy: 0.9808 - val_loss: 0.0545
Epoch 4/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 189s 336ms/step - accuracy: 0.9861 - loss: 0.0414 - val_accuracy: 0.9777 - val_loss: 0.0563
Epoch 5/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 212s 376ms/step - accuracy: 0.9850 - loss: 0.0474 - val_accuracy: 0.9933 - val_loss: 0.0248
Epoch 6/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 176s 312ms/step - accuracy: 0.9926 - loss: 0.0253 - val_accuracy: 0.9947 - val_loss: 0.0194
Epoch 7/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 137s 244ms/step - accuracy: 0.9942 - loss: 0.0212 - val_accuracy: 0.9940 - val_loss: 0.0184
Epoch 8/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 136s 242ms/step - accuracy: 0.9936 - loss: 0.0216 - val_accuracy: 0.9971 - val_loss: 0.0091
Epoch 9/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 136s 242ms/step - accuracy: 0.9958 - loss: 0.0145 - val_accuracy: 0.9962 - val_loss: 0.0116
Epoch 10/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 137s 243ms/step - accuracy: 0.9948 - loss: 0.0187 - val_accuracy: 0.9958 - val_loss: 0.0089
Epoch 1/10
```

```
C:\Users\kaibe\anaconda3\Lib\site-packages\keras\src\models\functional.py:225: UserWarning: The structure of `inputs` does
n't match the expected structure: ['title', 'text']. Received: the structure of inputs={'title': '*', 'text': '*'}
  warnings.warn(
```

```
562/562 ━━━━━━━━━━━━━━━━━━━━ 281s 468ms/step - accuracy: 0.7958 - loss: 0.3632 - val_accuracy: 0.9869 - val_loss: 0.0357
Epoch 2/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 258s 459ms/step - accuracy: 0.9801 - loss: 0.0647 - val_accuracy: 0.9844 - val_loss: 0.0443
Epoch 3/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 265s 471ms/step - accuracy: 0.9841 - loss: 0.0479 - val_accuracy: 0.9742 - val_loss: 0.0766
Epoch 4/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 279s 497ms/step - accuracy: 0.9879 - loss: 0.0397 - val_accuracy: 0.9898 - val_loss: 0.0250
Epoch 5/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 324s 577ms/step - accuracy: 0.9921 - loss: 0.0246 - val_accuracy: 0.9971 - val_loss: 0.0091
Epoch 6/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 401s 714ms/step - accuracy: 0.9938 - loss: 0.0197 - val_accuracy: 0.9980 - val_loss: 0.0069
Epoch 7/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 420s 747ms/step - accuracy: 0.9937 - loss: 0.0209 - val_accuracy: 0.9918 - val_loss: 0.0221
Epoch 8/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 417s 742ms/step - accuracy: 0.9935 - loss: 0.0178 - val_accuracy: 0.9984 - val_loss: 0.0045
Epoch 9/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 352s 626ms/step - accuracy: 0.9983 - loss: 0.0056 - val_accuracy: 0.9915 - val_loss: 0.0281
Epoch 10/10
562/562 ━━━━━━━━━━━━━━━━━━━━ 350s 623ms/step - accuracy: 0.9912 - loss: 0.0247 - val_accuracy: 0.9969 - val_loss: 0.0081
```

In [14]:

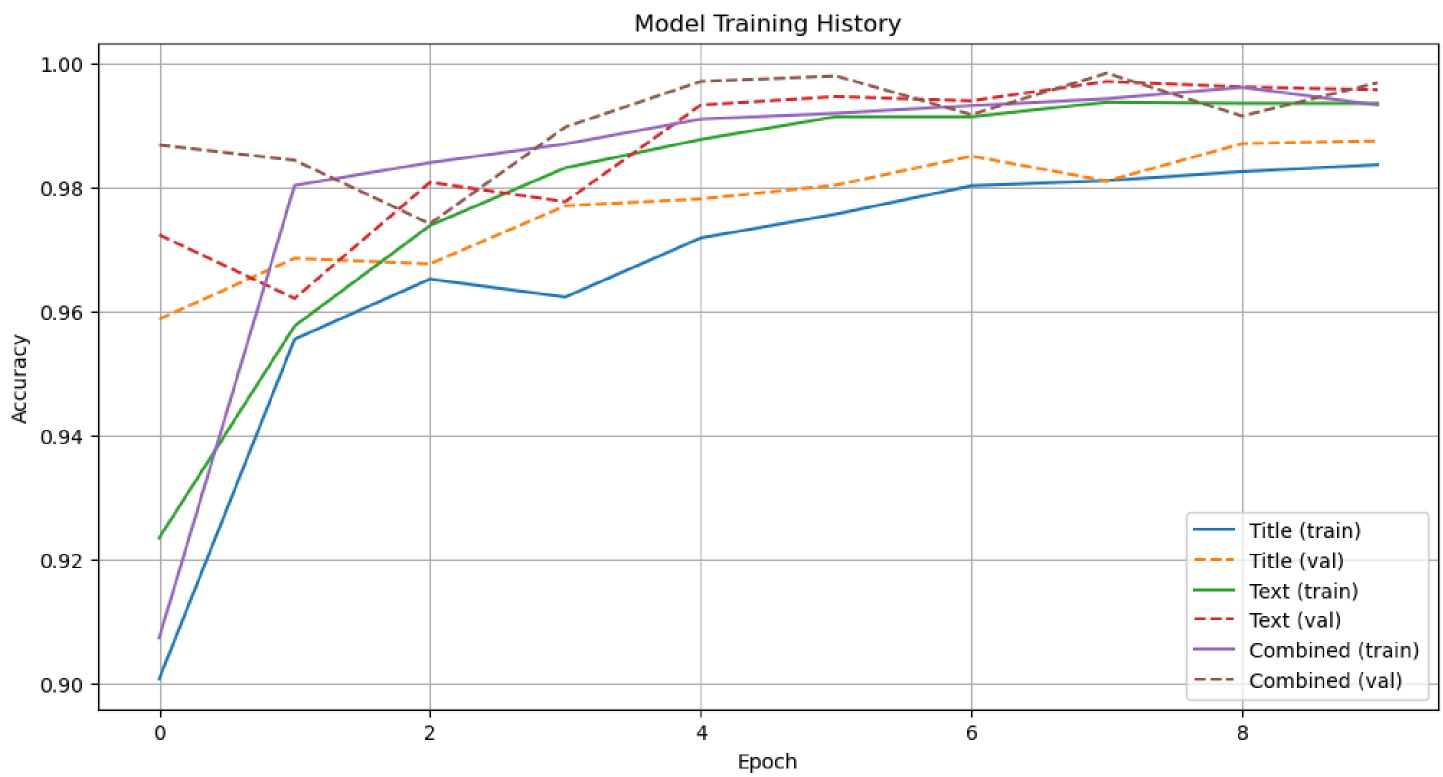
```python
# Plot training histories
def plot_training_history(histories):
    plt.figure(figsize=(12, 6))

    # Plot training accuracies
    for name, history in histories.items():
        plt.plot(history.history['accuracy'],
                label=f'{name} (train)',
                linestyle='-')
        plt.plot(history.history['val_accuracy'],
                label=f'{name} (val)',
                linestyle='--')

    plt.title('Model Training History')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

# Create dictionary of histories
histories = {
    'Title': history_title,
    'Text': history_text,
    'Combined': history_combined
}

# Plot histories
plot_training_history(histories)
```




### What Accuracy Should You Aim For?

Your three different models might have noticeably different performance. Your best model should be able to consistently score at least 97% validation accuracy.

After comparing the performance of each model on validation data, make a recommendation regarding the question at the beginning of this section. Should algorithms use the title, the text, or both when seeking to detect fake news?

Algorithms should use the combined model when seeking to detect fake news

## Model Evaluation

Now we'll test your model performance on unseen test data. For this part, you can focus on your best model, and ignore the other two.

Once you're satisfied with your best model's performance on validation data, download the test data here:

```
test_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_test.csv?raw=true"
```

In [18]:

```
# Download and prepare test data
test_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_test.csv?raw=true"
test_data = pd.read_csv(test_url)

# Create test dataset for combined model (using both title and text)
test_titles = tf.constant(test_data["title"].values)
test_texts = tf.constant(test_data["text"].values)
test_labels = test_data["fake"].values

# Create combined test dataset
test_combined_ds = tf.data.Dataset.from_tensor_slices(
    ({
        'title': test_titles,
        'text': test_texts
    }, test_labels)
).batch(32)

# Evaluate combined model on test set
test_loss, test_accuracy = combined_model.evaluate(test_combined_ds)
print(f"\nTest accuracy: {test_accuracy:.4f}")
```

```python
# Generate predictions using combined model
predictions = combined_model.predict(test_combined_ds)
predicted_labels = (predictions > 0.5).astype(int)

# Print detailed classification metrics
from sklearn.metrics import classification_report
print('\nClassification Report:')
print(classification_report(test_labels, predicted_labels))
```

```
702/702 ━━━━━━━━━━━━━━━━━━━━ 72s 103ms/step - accuracy: 0.9749 - loss: 0.0773

Test accuracy: 0.9759
C:\Users\kaibe\anaconda3\Lib\site-packages\keras\src\models\functional.py:225: UserWarning: The structure of `inputs` does
n't match the expected structure: ['title', 'text']. Received: the structure of inputs={'title': '*', 'text': '*'}
  warnings.warn(
702/702 ━━━━━━━━━━━━━━━━━━━━ 67s 93ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97     10708
           1       0.97      0.99      0.98     11741

    accuracy                           0.98     22449
   macro avg       0.98      0.98      0.98     22449
weighted avg       0.98      0.98      0.98     22449
```

# Grading items:

### Data Prep (15 pts)

1. Stopwords are removed during the construction of the data set. (3 pts)
2. make_dataset is implemented as a function, and used to create both the training/validation and testing data sets. (3 pts)
3. The constructed Dataset has multiple inputs. (3 pts)
4. Write a function to do train test split. (3 pts)
5. 20% of the training data is split off for validation. (3 pts)

### Models (40 pts)

6. Model 1 uses only the article title. (5 pts)
7. Model 2 uses only the article text. (5 pts)
8. Model 3 uses both the article title and text. (5 pts)
9. For model 3, embedding is consistent with the text vectorization method. i.e., if you use shared embedding layer, the preceding text vectorization layer also should be shared. (5 pts)
10. The training history is plotted for each of the three models, including the training and validation performance. (5 pts)
11. The most performant model is evaluated on the test data set. (5 pts)
12. The best model consistently obtains at least 97% accuracy on the validation set. (5 pts)
13. The best model's performance on the test set is shown. (5 pts)

### Style and Documentation (10 pts)

14. Throughout the HW, function docstrings, incline comments, and markdown are required. (10 pts)