```
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from nltk.tokenize import RegexpTokenizer
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

Generic Fucntions

```
labels=pickle.load( open('labels.pkl', 'rb'))
features=pickle.load( open('features.pkl', 'rb'))
features_tfidf= pickle.load( open('features_tfidf.pkl', 'rb'))
word_tfidf_weights=pickle.load( open('word_tfidf_weights.pkl', 'rb'))
non_text_features_np = features.drop(columns=['title', 'text',
'combined_text', 'label']).to_numpy()

def get_metrics(clf,test_ft,test_labels):
    pred_labels=clf.predict(test_ft)

print(classification_report(pred_labels,test_labels,target_names=["fak
e","real"]))

def plot_ROC(test):

    pred_labels=clf.predict(test_ft)
    fpr, tpr, thresholds = roc_curve(test,pred_labels)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label='ROC curve (AUC =
{:.2f})'.format(roc_auc))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (FPR)')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.legend(loc='lower right')
```

```
    plt.show()


def plot_loss_acc(history):

    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

1. TF-IDF vectors + text characteristics

Machine Learning models

```
#normalizing the data

scaler=StandardScaler()
norm_features=scaler.fit_transform(features_tfidf)

train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
print("----------------Train data-----------------------\n")
print(" train data features shape {} \n train data labels shape {}\
n".format(train_ft.shape,train_labels.shape))
print("-----------------Test data-----------------------\n")
print(" test data features shape {} \n test data labels shape
{}".format(test_ft.shape,test_labels.shape))
```

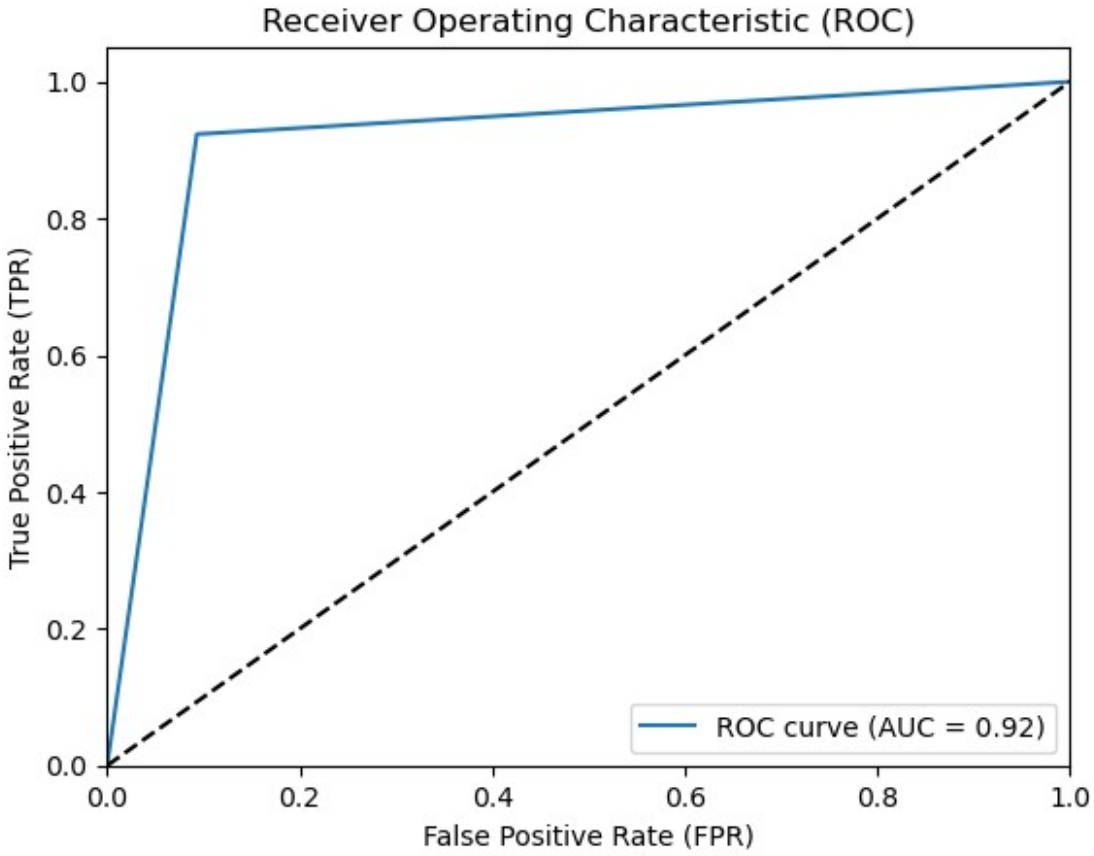
```
----------------Train data-----------------------

 train data features shape (57229, 115)
 train data labels shape (57229,)

-----------------Test data-----------------------

 test data features shape (14308, 115)
 test data labels shape (14308,)
```

Logistic Regression

```
clf = LogisticRegression(random_state=0).fit(train_ft, train_labels)

get_metrics(clf,test_ft,test_labels)
```

```
              precision    recall  f1-score   support

        fake       0.91      0.92      0.92      6895
        real       0.93      0.92      0.92      7413

    accuracy                           0.92     14308
   macro avg       0.92      0.92      0.92     14308
weighted avg       0.92      0.92      0.92     14308
```

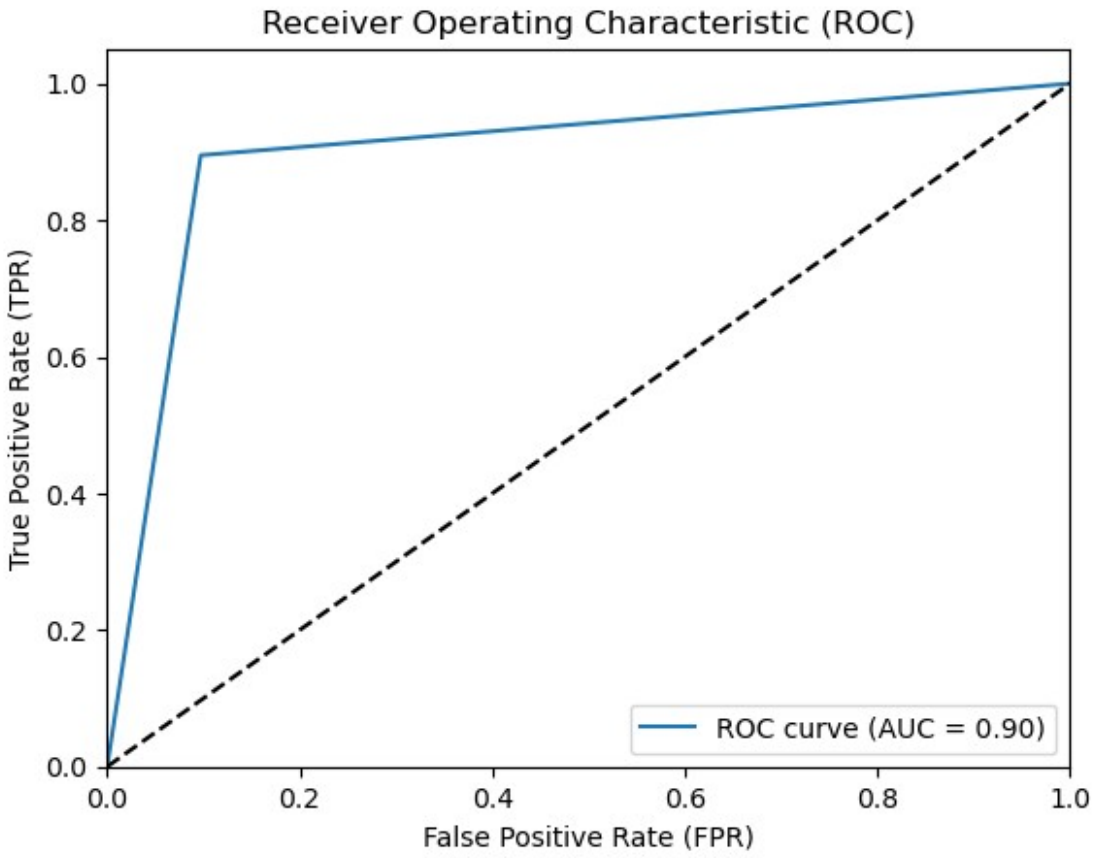
```
plot_ROC(test_labels)
```



Linear Discriminant Analysis

```
clf = LinearDiscriminantAnalysis().fit(train_ft, train_labels)

get_metrics(clf,test_ft,test_labels)
```

```
              precision    recall  f1-score   support

        fake       0.89      0.92      0.91      6762
        real       0.93      0.90      0.92      7546

    accuracy                           0.91     14308
   macro avg       0.91      0.91      0.91     14308
weighted avg       0.91      0.91      0.91     14308


plot_ROC(test_labels)
```



Artificial Neural network

```
class ANN_model(tf.keras.models.Model):
    def __init__(self):
        super(ANN_model,self).__init__()
        self.layer1= Dense(64, activation="relu")
        self.layer2= Dense(32,activation="relu")
        self.label= Dense(1,activation="sigmoid")

    def call(self,inputs):
        x=self.layer1(inputs)
```

```
        x=self.layer2(x)
        x=self.label(x)

        return x
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[2], line 1
----> 1 class ANN_model(tf.keras.models.Model):
      2     def __init__(self):
      3         super(ANN_model,self).__init__()

NameError: name 'tf' is not defined
```

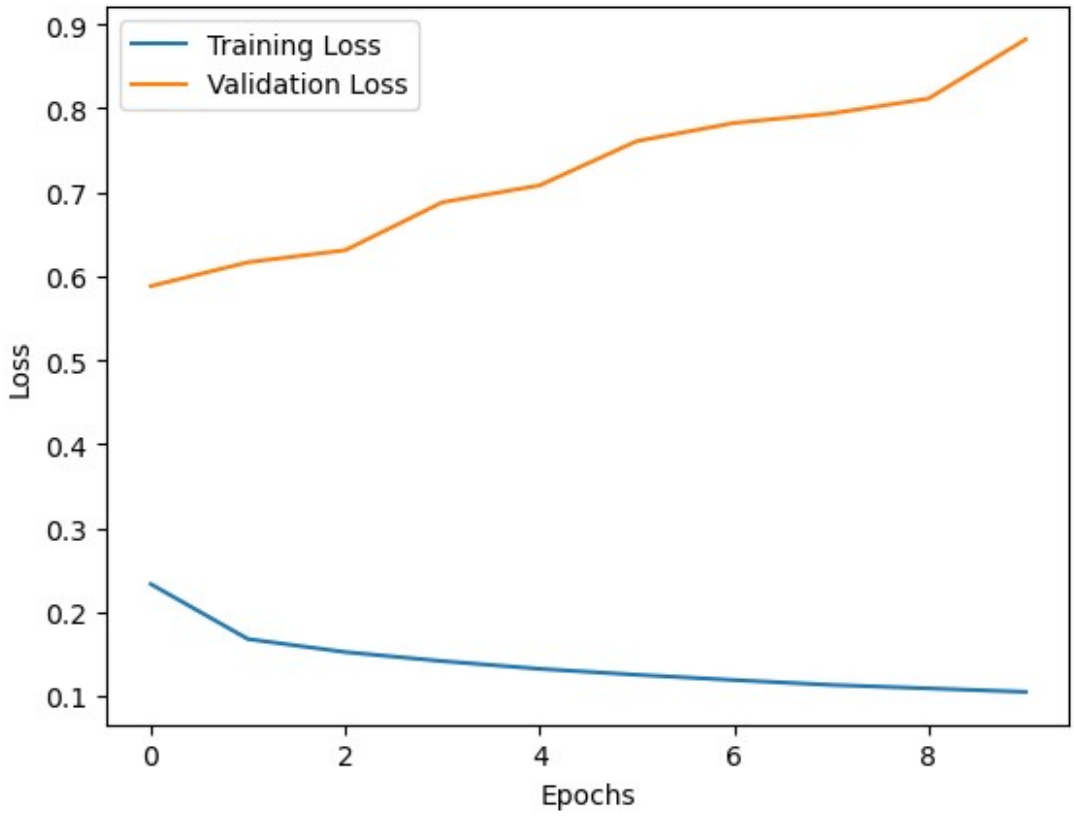
```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(115,)))  # Input
layer
model.add(Dense(64, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer
model.summary()
```

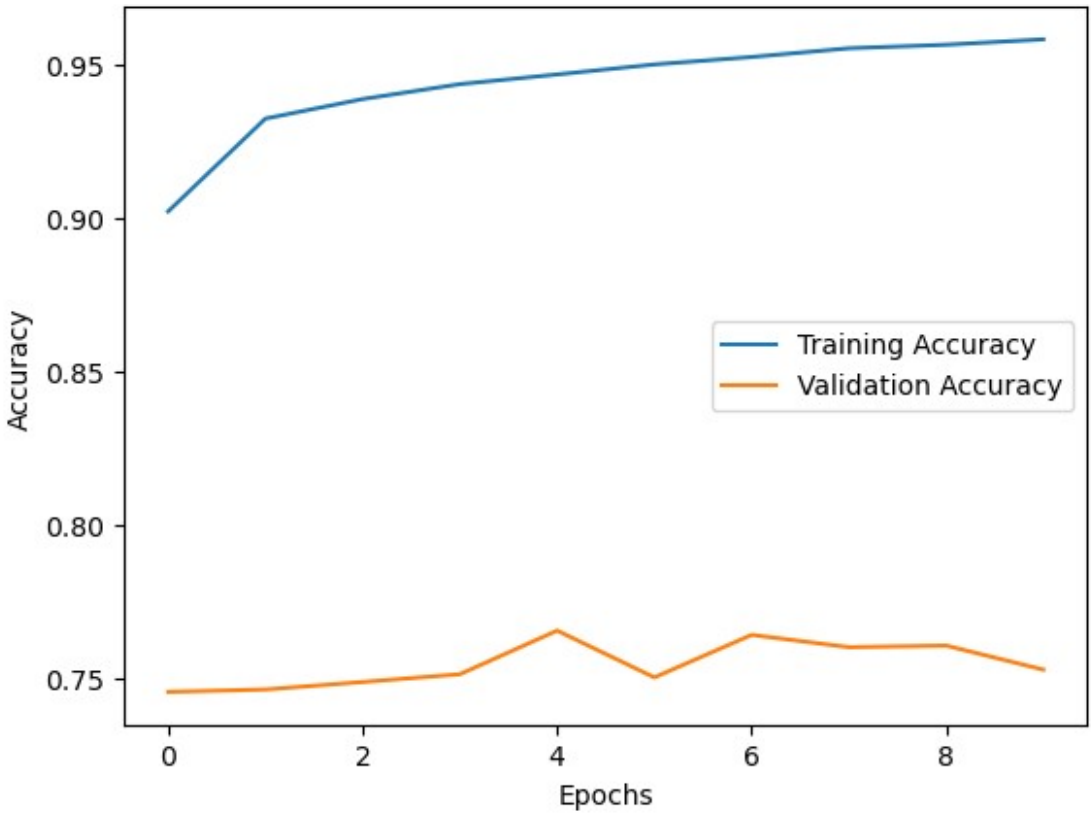
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 32)                3712

 dense_13 (Dense)            (None, 64)                2112

 dense_14 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,889
Trainable params: 5,889
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_ft, train_labels, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
```

```
Epoch 1/10
1789/1789 [==============================] - 8s 2ms/step - loss:
0.2334 - accuracy: 0.9022 - val_loss: 0.5883 - val_accuracy: 0.7453
Epoch 2/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1678 - accuracy: 0.9325 - val_loss: 0.6167 - val_accuracy: 0.7461
```

```
Epoch 3/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1524 - accuracy: 0.9388 - val_loss: 0.6310 - val_accuracy: 0.7486
Epoch 4/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1416 - accuracy: 0.9437 - val_loss: 0.6881 - val_accuracy: 0.7511
Epoch 5/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1324 - accuracy: 0.9469 - val_loss: 0.7083 - val_accuracy: 0.7653
Epoch 6/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1254 - accuracy: 0.9501 - val_loss: 0.7609 - val_accuracy: 0.7500
Epoch 7/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1191 - accuracy: 0.9526 - val_loss: 0.7825 - val_accuracy: 0.7639
Epoch 8/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1133 - accuracy: 0.9554 - val_loss: 0.7938 - val_accuracy: 0.7599
Epoch 9/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1092 - accuracy: 0.9566 - val_loss: 0.8115 - val_accuracy: 0.7604
Epoch 10/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1050 - accuracy: 0.9583 - val_loss: 0.8822 - val_accuracy: 0.7526

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_ft, test_labels)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')

448/448 [==============================] - 1s 2ms/step - loss: 0.1723
- accuracy: 0.9332
Test Loss: 0.17231875658035278, Test Accuracy: 0.9331842064857483

plot_loss_acc(history)
```


Training Loss
Validation Loss
Loss
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8


Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0
2
4
6
8

2. Word2vec vectors + text characteristics

Machine Learning Models

```
doc=list(features["combined_text"])
non_text_features = features.drop(columns=['title', 'text',
'combined_text', 'label'])
non_text_features_np=non_text_features.to_numpy()

def tokenize_words(text):
    tokenized_words,vocab=[],[]
    for i in text:
        tokenizer = RegexpTokenizer(r'\w+')     #using regular
expression tokenizer
        wrd_token=tokenizer.tokenize(i)
        tokenized_words.append(wrd_token)
        vocab.extend(wrd_token)
    return tokenized_words, vocab


doc_tokens_w2v,vocab=tokenize_words(doc)  #tokenizing the doc(ans+que)

model_doc=Word2Vec(doc_tokens_w2v,min_count=2,window=4)

sentence_vectors = []
for sentence in doc:
    tokens = word_tokenize(sentence.lower())
    sentence_vector = np.zeros(model_doc.vector_size)
    if len(tokens)==0:
        sentence_vectors.append([0]*100)
    else:
        for token in tokens:
            if token in model_doc.wv:
                token_vector = model_doc.wv[token]
                sentence_vector += token_vector
        sentence_vector /= len(tokens)
        sentence_vectors.append(sentence_vector)

# Convert the list of sentence vectors to a NumPy array
sentence_vectors = np.array(sentence_vectors)

sentence_vectors.shape

(71537, 100)

non_text_features = features.drop(columns=['title', 'text',
'combined_text', 'label'])
non_text_features_np=non_text_features.to_numpy()
```

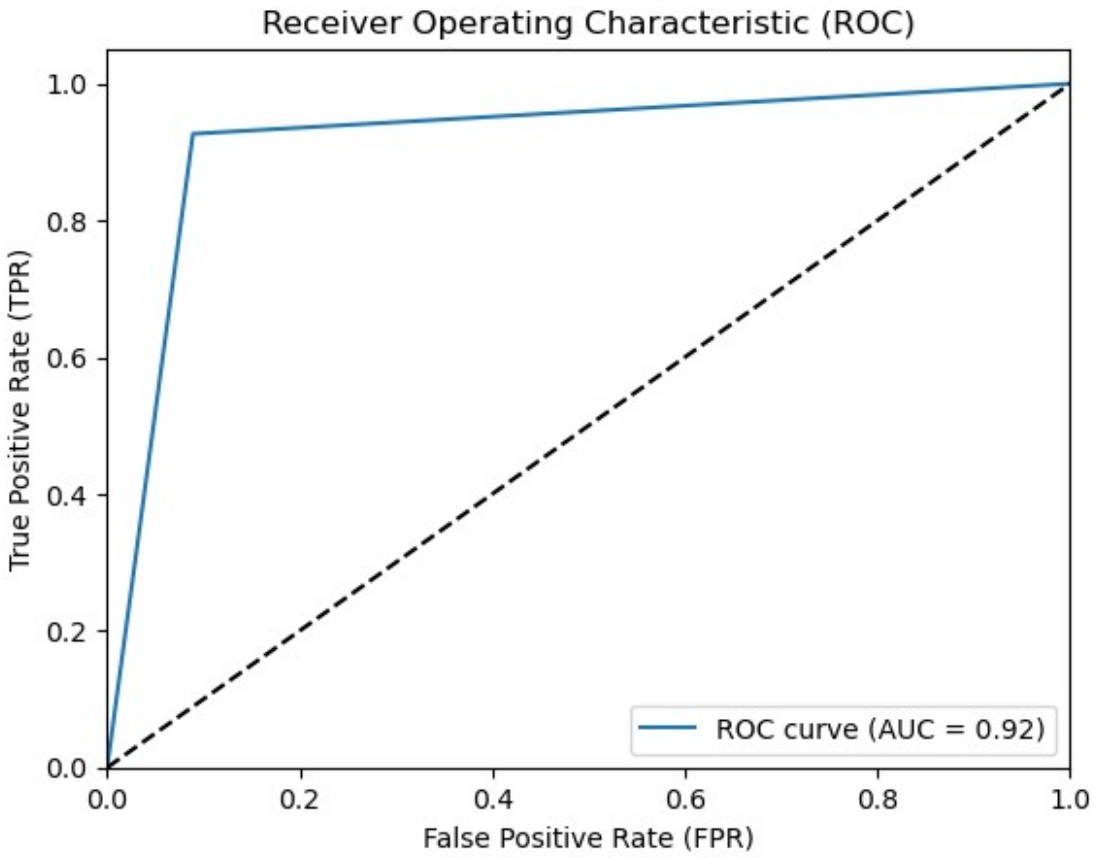
```
features_w2v = np.hstack((sentence_vectors, non_text_features_np))

pickle.dump(features_w2v, open('features_w2v.pkl', 'wb'))

features_w2v = pickle.load( open('features_tfidf.pkl', 'rb'))

scaler=StandardScaler()
norm_features=scaler.fit_transform(features_w2v)

train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
print("----------------Train data-----------------------\n")
print(" train data features shape {} \n train data labels shape {}\
n".format(train_ft.shape,train_labels.shape))
print("-----------------Test data------------------------\n")
print(" test data features shape {} \n test data labels shape
{}".format(test_ft.shape,test_labels.shape))
```

```
----------------Train data-----------------------

 train data features shape (57229, 115)
 train data labels shape (57229,)

-----------------Test data------------------------

 test data features shape (14308, 115)
 test data labels shape (14308,)
```

Logistic Regression

```
clf = LogisticRegression(random_state=0).fit(train_ft, train_labels)
get_metrics(clf,test_ft,test_labels)
plot_ROC(test_labels)
```

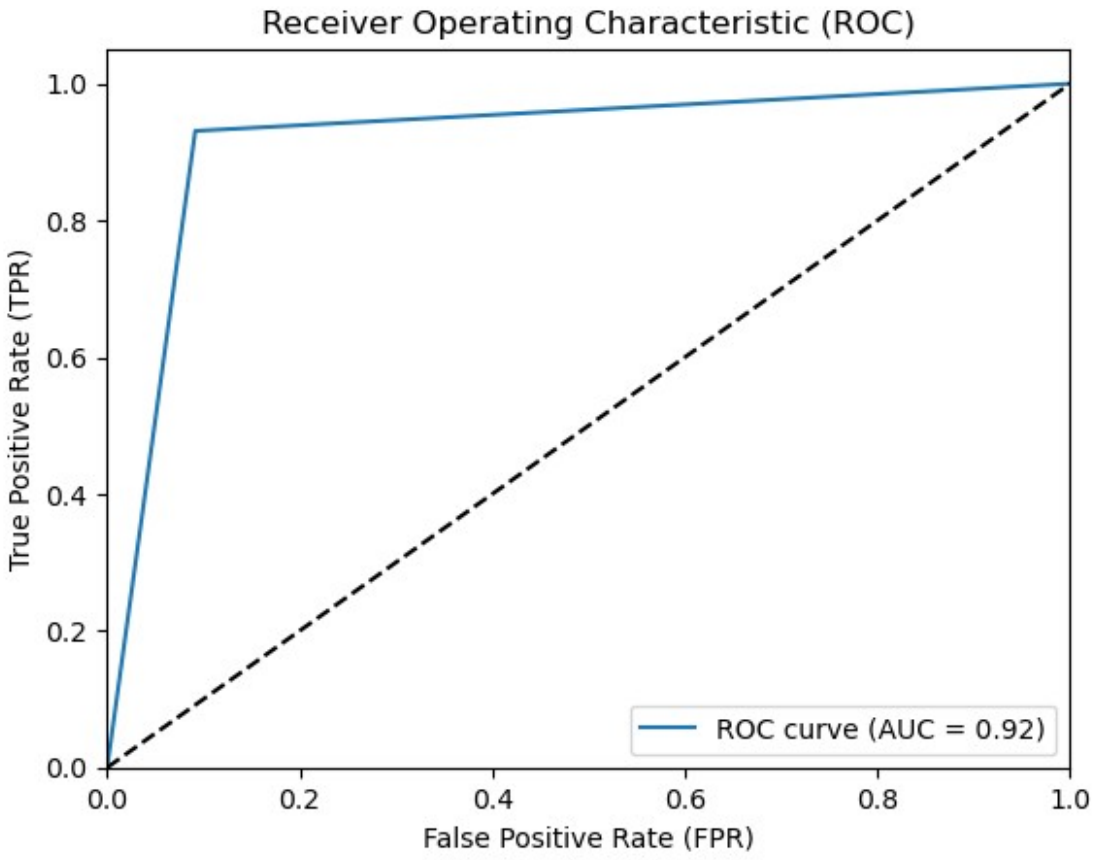
```
              precision    recall  f1-score   support

        fake       0.91      0.92      0.92      6895
        real       0.93      0.92      0.92      7413

    accuracy                           0.92     14308
   macro avg       0.92      0.92      0.92     14308
weighted avg       0.92      0.92      0.92     14308
```




```
get_metrics(clf,test_ft,test_labels)
```

	precision	recall	f1-score	support
fake	0.91	0.92	0.92	6895
real	0.93	0.92	0.92	7413
accuracy			0.92	14308
macro avg	0.92	0.92	0.92	14308
weighted avg	0.92	0.92	0.92	14308

```
plot_ROC(test_labels)
```

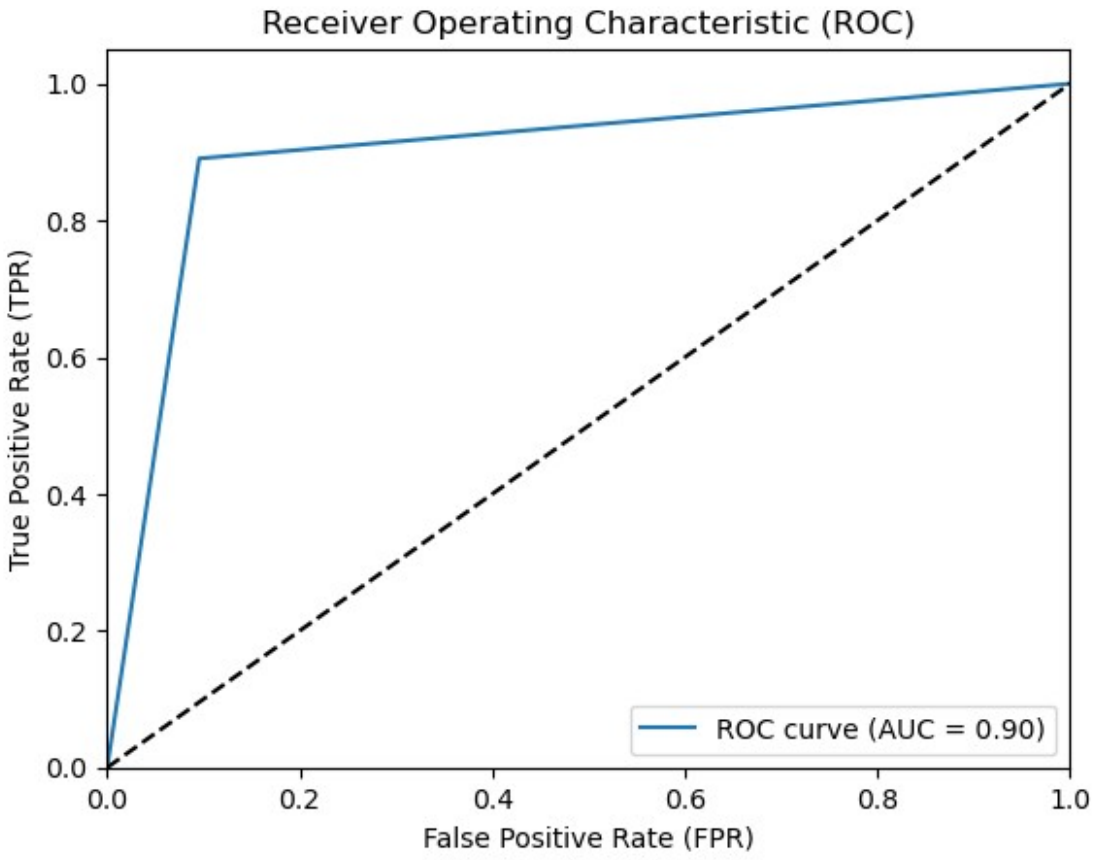


Linear Discriminant Analysis

```
clf = LinearDiscriminantAnalysis().fit(train_ft, train_labels)

get_metrics(clf,test_ft,test_labels)
```

	precision	recall	f1-score	support
fake	0.89	0.92	0.91	6762
real	0.93	0.90	0.92	7546
accuracy			0.91	14308
macro avg	0.91	0.91	0.91	14308
weighted avg	0.91	0.91	0.91	14308

```
plot_ROC(test_labels)
```



Artificial Neural Network

```
train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
X_train, X_val, y_train, y_val = train_test_split(train_ft,
train_labels, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", test_ft.shape)

Training set shape: (45783, 115)
Validation set shape: (11446, 115)
Test set shape: (14308, 115)

import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Assuming you have already preprocessed and split your dataset into
X_train, y_train, X_val, y_val

# Define the model architecture
model = Sequential()
```

```
model.add(Dense(64, activation='relu', input_shape=(115,)))  # Input
layer
model.add(Dense(64, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                7424

 dense_16 (Dense)            (None, 64)                4160

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 11,649
Trainable params: 11,649
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_ft, train_labels, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
```

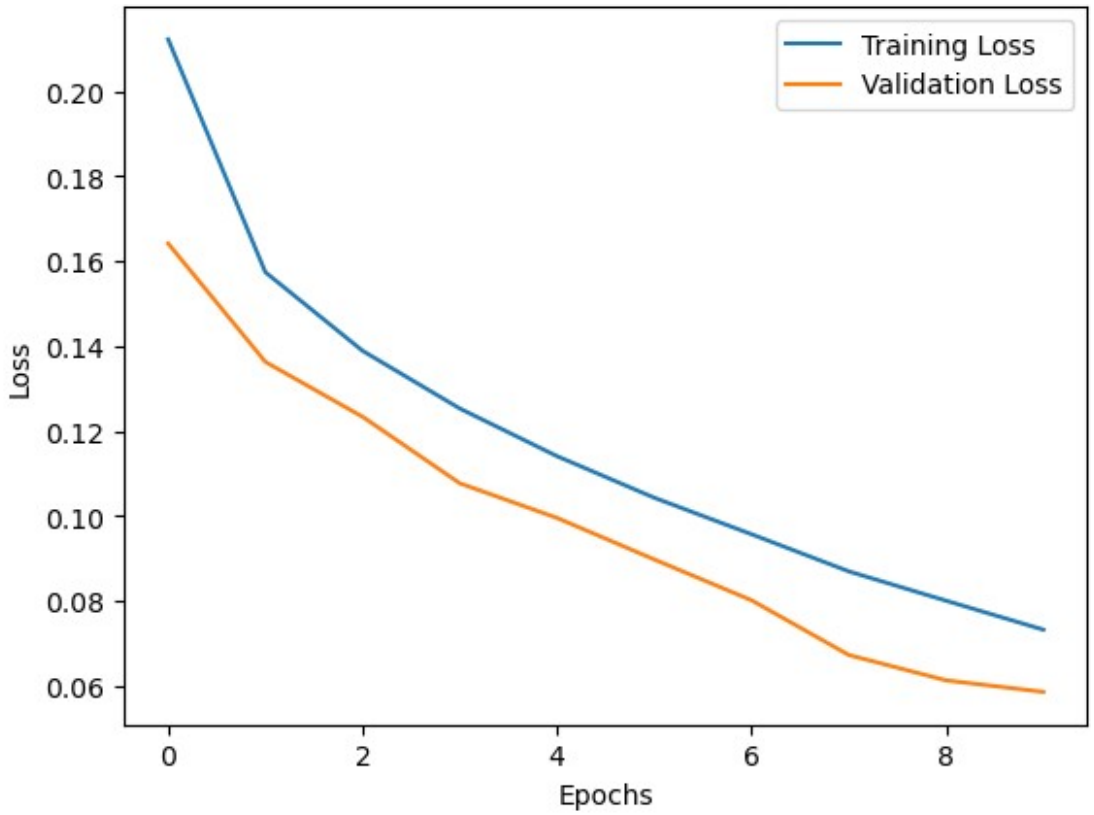
```
Epoch 1/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.2122 - accuracy: 0.9140 - val_loss: 0.1641 - val_accuracy: 0.9363
Epoch 2/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1574 - accuracy: 0.9371 - val_loss: 0.1362 - val_accuracy: 0.9462
Epoch 3/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1388 - accuracy: 0.9442 - val_loss: 0.1233 - val_accuracy: 0.9529
Epoch 4/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1253 - accuracy: 0.9505 - val_loss: 0.1076 - val_accuracy: 0.9582
Epoch 5/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1140 - accuracy: 0.9551 - val_loss: 0.0995 - val_accuracy: 0.9599
Epoch 6/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.1042 - accuracy: 0.9591 - val_loss: 0.0897 - val_accuracy: 0.9661
Epoch 7/10
1789/1789 [==============================] - 5s 3ms/step - loss:
0.0956 - accuracy: 0.9635 - val_loss: 0.0801 - val_accuracy: 0.9710
Epoch 8/10
1789/1789 [==============================] - 4s 2ms/step - loss:
```

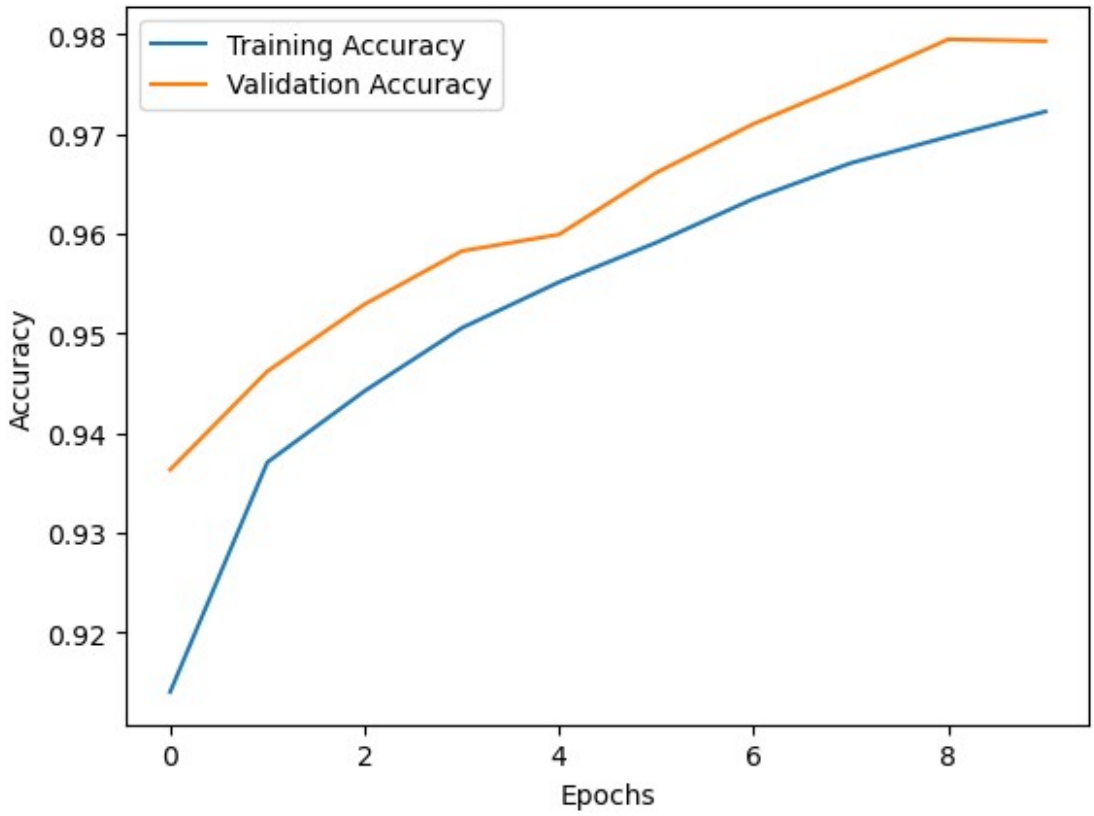
```
0.0869 - accuracy: 0.9671 - val_loss: 0.0672 - val_accuracy: 0.9751
Epoch 9/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.0800 - accuracy: 0.9697 - val_loss: 0.0612 - val_accuracy: 0.9795
Epoch 10/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.0732 - accuracy: 0.9722 - val_loss: 0.0585 - val_accuracy: 0.9793
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_ft, test_labels)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
```

```
448/448 [==============================] - 1s 1ms/step - loss: 0.1954
- accuracy: 0.9354
Test Loss: 0.19544467329978943, Test Accuracy: 0.9353508353233337
```

```
plot_loss_acc(history)
```





3. TF-IDF weighted Word2vec vectors

Machine learning models

```
sentence_vectors_tf = []

# Process each sentence and compute its vector
for i,sentence in enumerate(doc):
    tokens = word_tokenize(sentence.lower())
    sentence_vector = np.zeros(model_doc.vector_size)
    if len(tokens)==0:
         sentence_vectors_tf.append([0]*100)
    else:
        for token in tokens:
            if token in model_doc.wv:
                token_vector = model_doc.wv[token]
                sentence_vector += token_vector

        sentence_vector /= len(tokens)
        sentence_vector *= word_tfidf_weights[i]
        sentence_vectors_tf.append(sentence_vector)

# Convert the list of sentence vectors to a NumPy array
```

```
sentence_vectors_tf = np.array(sentence_vectors)
sentence_vectors_tf = np.nan_to_num(sentence_vectors_tf, nan=0.0)

pickle.dump(sentence_vectors_tf, open('w2v_tf_vectors.pkl', 'wb'))

sentence_vectors_tf=pickle.load( open('w2v_tf_vectors.pkl', 'rb'))

sentence_vectors_tf.shape
```

```
(71537, 100)
```

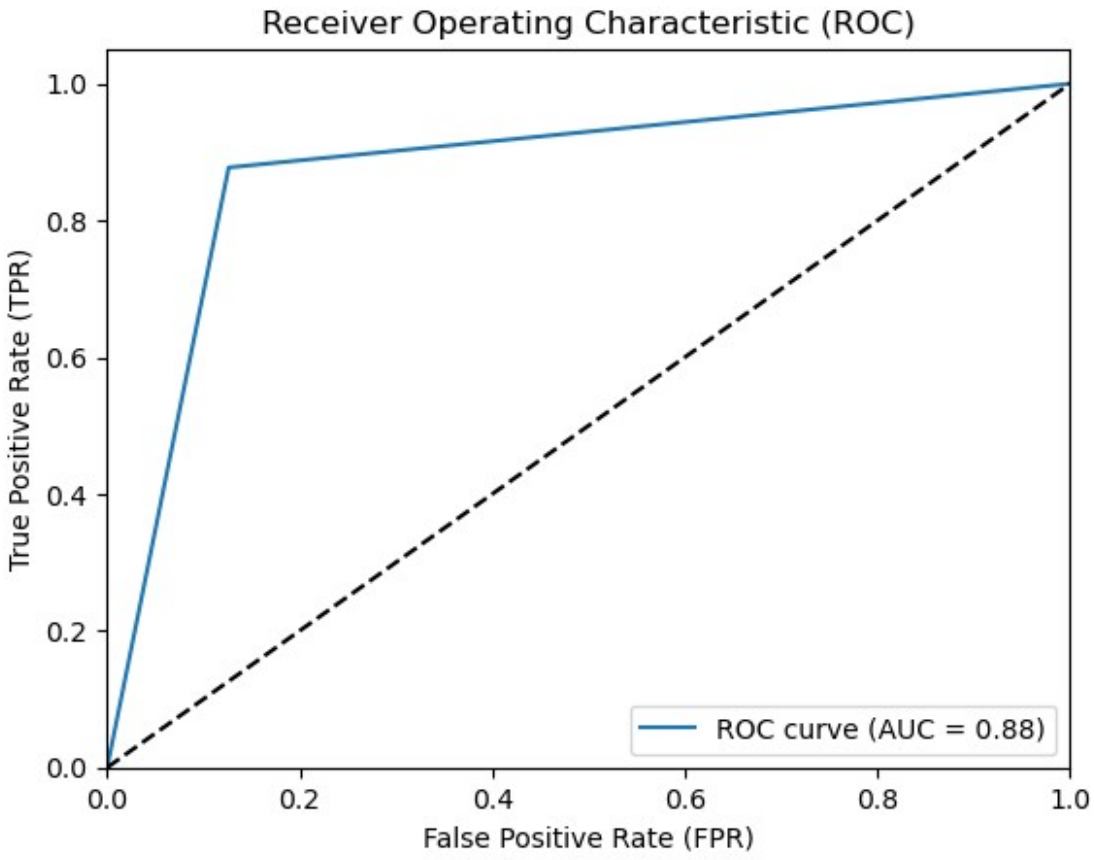
Logistic Regression

```
scaler=StandardScaler()
norm_features=scaler.fit_transform(sentence_vectors_tf)

train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
print("----------------Train data-----------------------\n")
print(" train data features shape {} \n train data labels shape {}\
n".format(train_ft.shape,train_labels.shape))
print("----------------Test data-----------------------\n")
print(" test data features shape {} \n test data labels shape
{}".format(test_ft.shape,test_labels.shape))
```

```
----------------Train data-----------------------

 train data features shape (57229, 100)
 train data labels shape (57229,)

----------------Test data-----------------------

 test data features shape (14308, 100)
 test data labels shape (14308,)
```

```
clf = LogisticRegression(random_state=0, max_iter=300).fit(train_ft,
train_labels)
get_metrics(clf,test_ft,test_labels)
plot_ROC(test_labels)
```

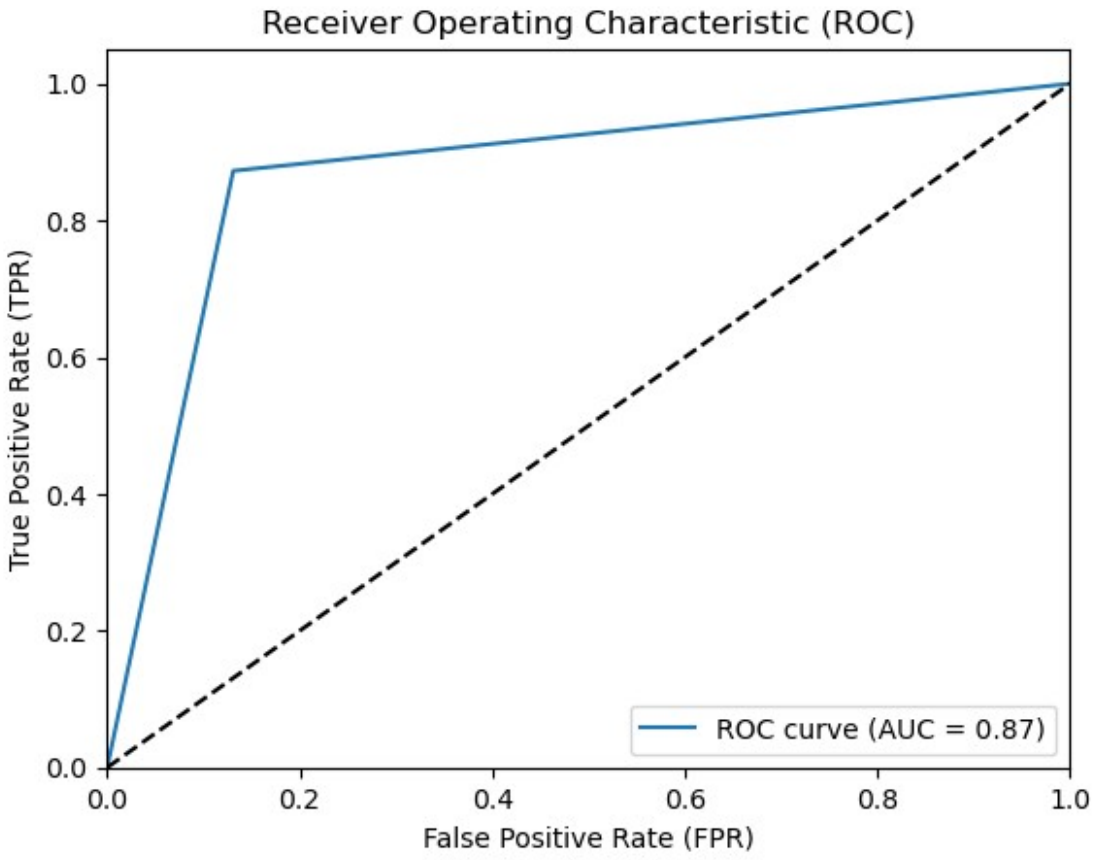
```
              precision    recall  f1-score   support

        fake       0.87      0.87      0.87      7008
        real       0.88      0.88      0.88      7300

    accuracy                           0.88     14308
   macro avg       0.88      0.88      0.88     14308
weighted avg       0.88      0.88      0.88     14308
```



Linear Discriminant Analysis

```
clf = LinearDiscriminantAnalysis().fit(train_ft, train_labels)
get_metrics(clf,test_ft,test_labels)
plot_ROC(test_labels)
```

```
              precision    recall  f1-score   support

        fake       0.87      0.87      0.87      7011
        real       0.87      0.87      0.87      7297

    accuracy                           0.87     14308
   macro avg       0.87      0.87      0.87     14308
weighted avg       0.87      0.87      0.87     14308
```



Artificial Neural Networks

```
train_ft,test_ft,train_labels,test_labels=train_test_split(sentence_ve
ctors_tf,labels,test_size=0.2, train_size=0.8)
X_train, X_val, y_train, y_val = train_test_split(train_ft,
train_labels, test_size=0.2, random_state=42)

# Print the shape of each set
print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", test_ft.shape)
```

```
Training set shape: (45783, 100)
Validation set shape: (11446, 100)
Test set shape: (14308, 100)
```

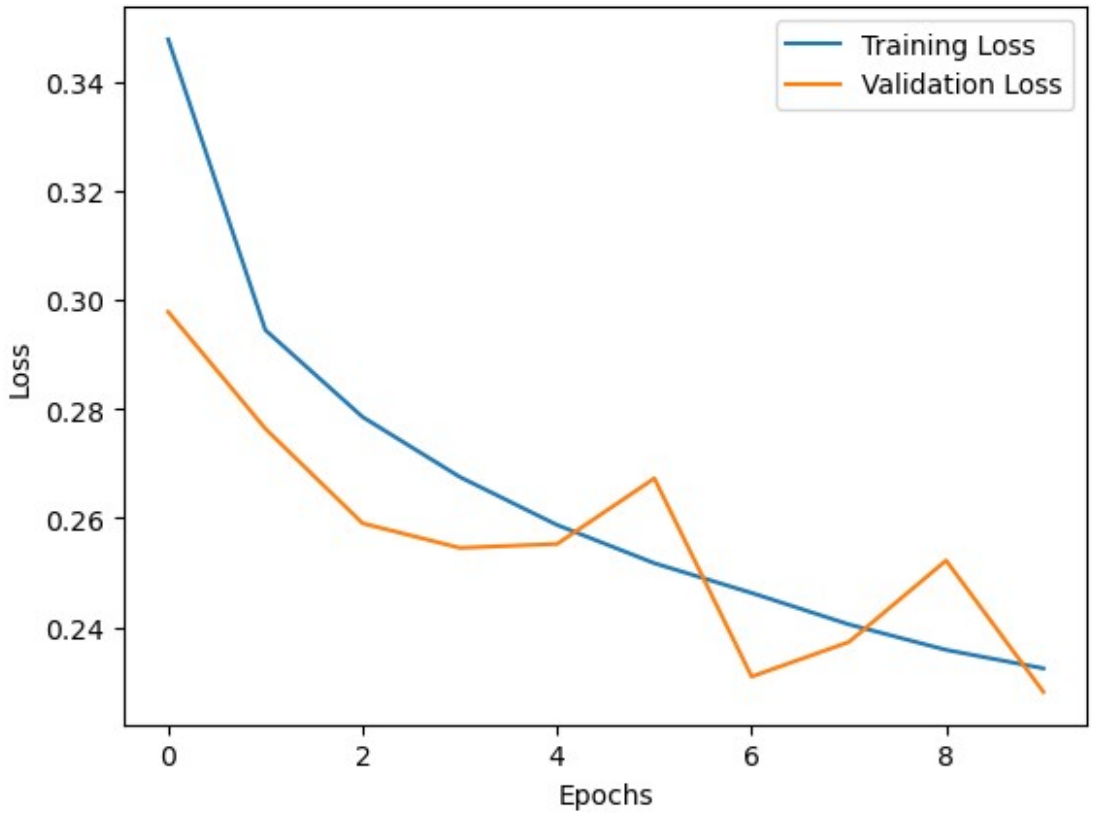
```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(100,)))  # Input
layer
model.add(Dense(64, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer
model.summary()
```

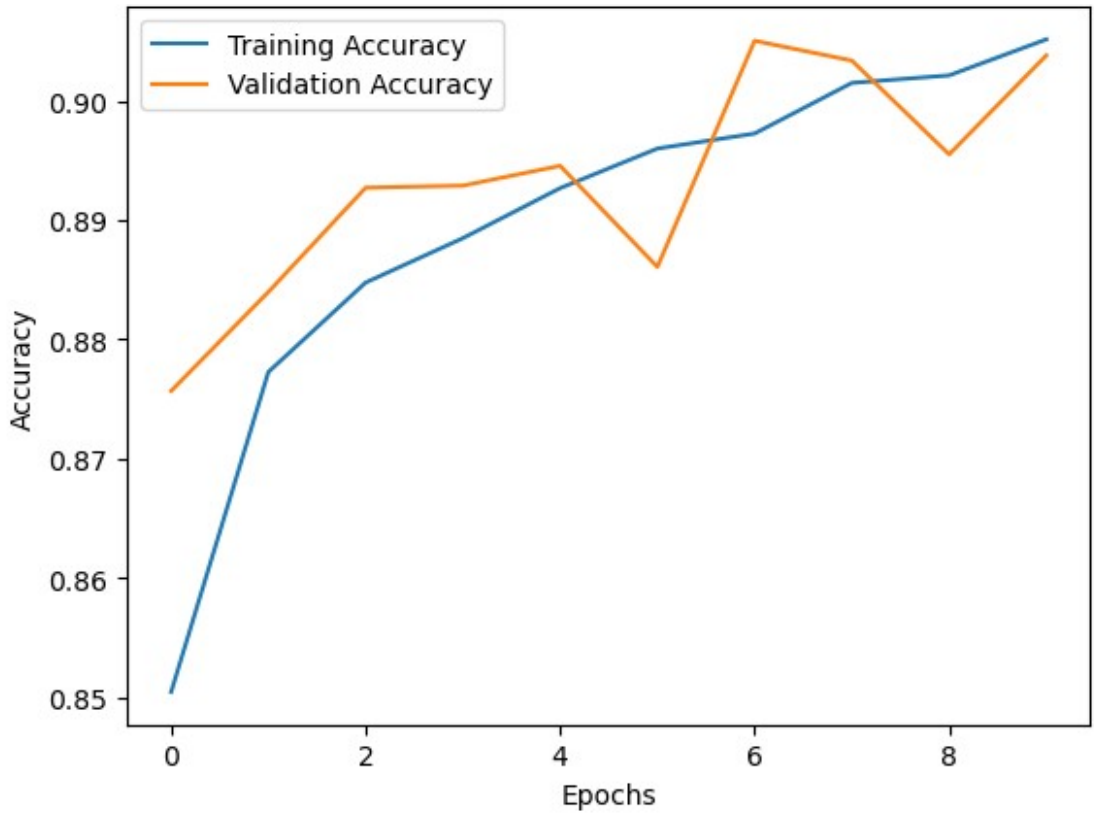
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_3 (Dense)              (None, 32)                3232

 dense_4 (Dense)              (None, 64)                2112

 dense_5 (Dense)              (None, 1)                 65

=================================================================
Total params: 5,409
Trainable params: 5,409
Non-trainable params: 0
_________________________________________________________________
```

```
opt = keras.optimizers.SGD(learning_rate=0.1)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_ft, train_labels, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
```

```
Epoch 1/10
1789/1789 [==============================] - 4s 2ms/step - loss:
0.3477 - accuracy: 0.8504 - val_loss: 0.2978 - val_accuracy: 0.8757
Epoch 2/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2944 - accuracy: 0.8773 - val_loss: 0.2764 - val_accuracy: 0.8840
Epoch 3/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2785 - accuracy: 0.8848 - val_loss: 0.2591 - val_accuracy: 0.8927
Epoch 4/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2675 - accuracy: 0.8885 - val_loss: 0.2546 - val_accuracy: 0.8929
Epoch 5/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2588 - accuracy: 0.8927 - val_loss: 0.2553 - val_accuracy: 0.8945
Epoch 6/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2518 - accuracy: 0.8960 - val_loss: 0.2673 - val_accuracy: 0.8861
Epoch 7/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2463 - accuracy: 0.8973 - val_loss: 0.2310 - val_accuracy: 0.9050
Epoch 8/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2406 - accuracy: 0.9015 - val_loss: 0.2373 - val_accuracy: 0.9034
Epoch 9/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2359 - accuracy: 0.9021 - val_loss: 0.2523 - val_accuracy: 0.8955
Epoch 10/10
```

```
1789/1789 [==============================] - 3s 2ms/step - loss:
0.2325 - accuracy: 0.9052 - val_loss: 0.2282 - val_accuracy: 0.9038

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_ft, test_labels)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')

448/448 [==============================] - 1s 1ms/step - loss: 0.2482
- accuracy: 0.8947
Test Loss: 0.24815793335437775, Test Accuracy: 0.8947442173957825

plot_loss_acc(history)
```





4 TF-IDF weighted Word2vec with non numeric features

Machine Learning Models

Logistic Regression

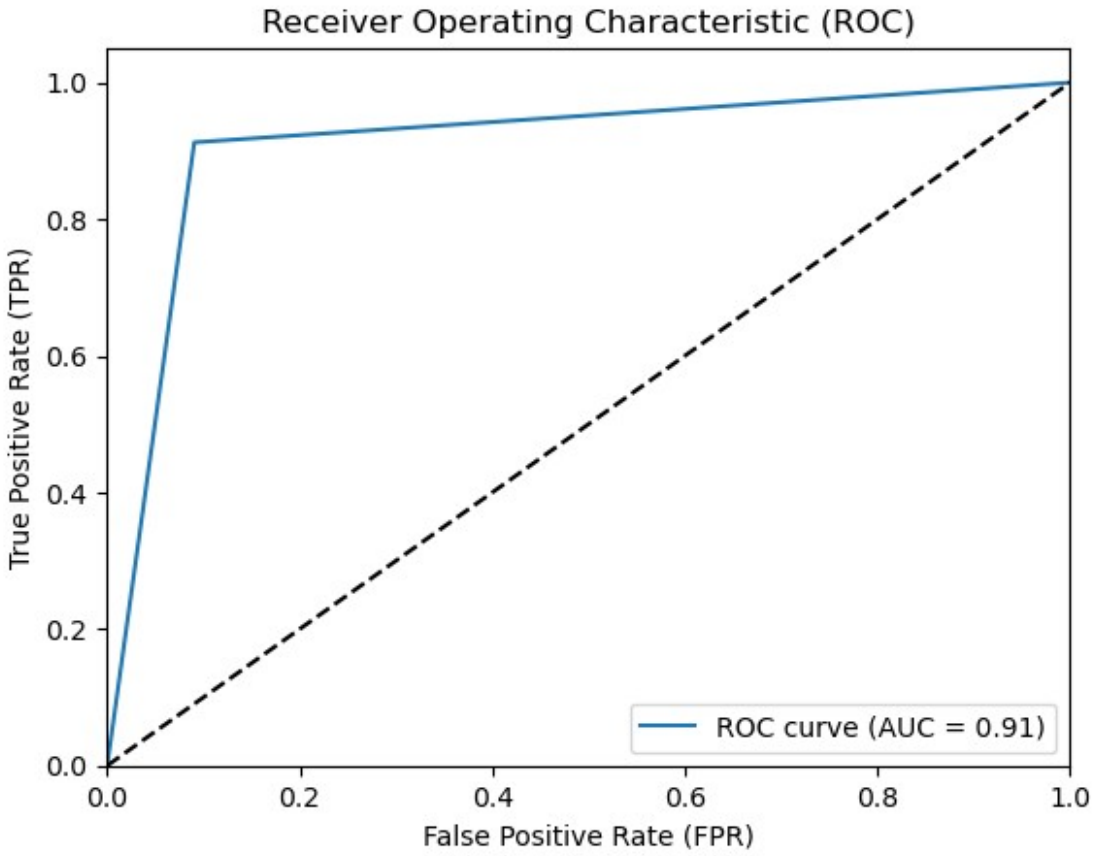
```
features_w2v = np.hstack((sentence_vectors_tf, non_text_features_np))
scaler=StandardScaler()
norm_features=scaler.fit_transform(features_w2v)

train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
print("----------------Train data-----------------------\n")
print(" train data features shape {} \n train data labels shape {}\
n".format(train_ft.shape,train_labels.shape))
print("-----------------Test data------------------------\n")
print(" test data features shape {} \n test data labels shape
{}".format(test_ft.shape,test_labels.shape))
```

```
----------------Train data-----------------------
```

```
 train data features shape (57229, 115)
 train data labels shape (57229,)

-----------------Test data-------------------------

 test data features shape (14308, 115)
 test data labels shape (14308,)
```

```
clf = LogisticRegression(random_state=0, max_iter=300).fit(train_ft,
train_labels)
get_metrics(clf,test_ft,test_labels)
plot_ROC(test_labels)
```

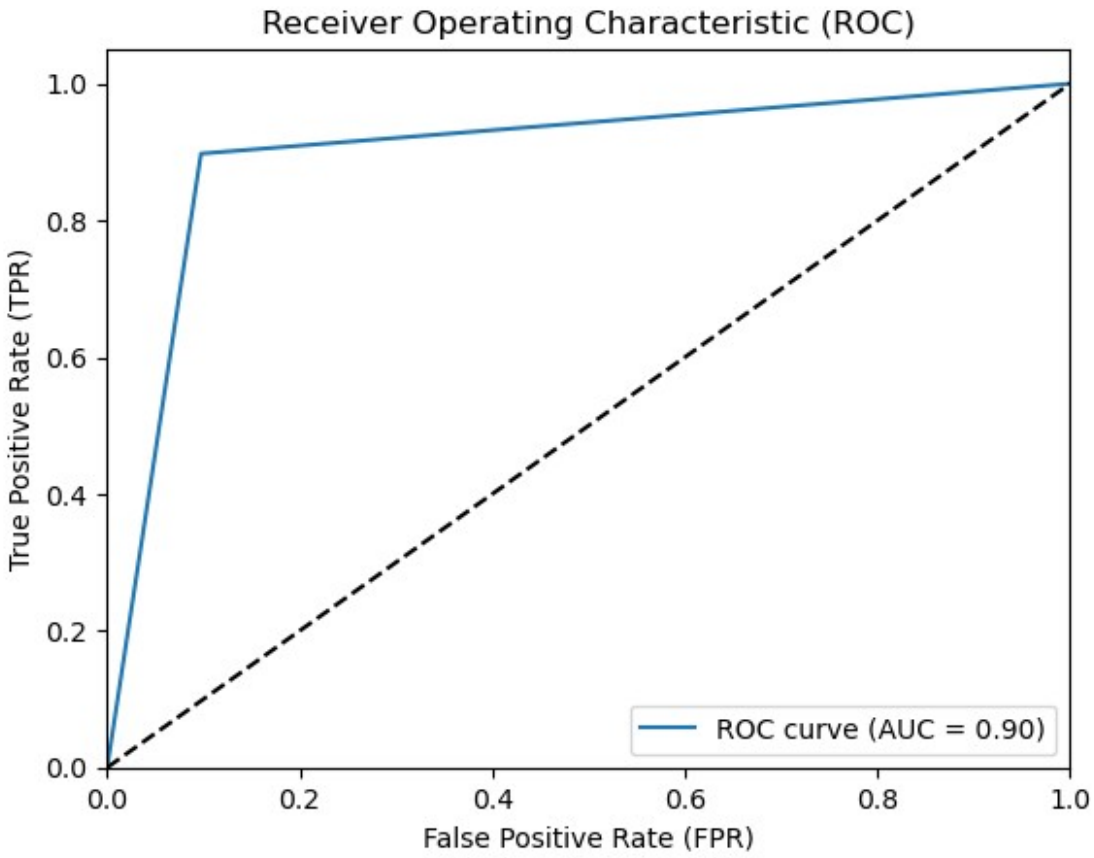
```
              precision    recall  f1-score   support

        fake       0.91      0.91      0.91      7048
        real       0.91      0.91      0.91      7260

    accuracy                           0.91     14308
   macro avg       0.91      0.91      0.91     14308
weighted avg       0.91      0.91      0.91     14308
```



Linear Discriminant Analysis

```
clf = LinearDiscriminantAnalysis().fit(train_ft, train_labels)
get_metrics(clf,test_ft,test_labels)
plot_ROC(test_labels)
```

```
              precision    recall  f1-score   support

        fake       0.90      0.90      0.90      7106
        real       0.90      0.90      0.90      7202

    accuracy                           0.90     14308
   macro avg       0.90      0.90      0.90     14308
weighted avg       0.90      0.90      0.90     14308
```



Artificial Neural Networks

```
features_w2v = np.hstack((sentence_vectors_tf, non_text_features_np))
scaler=StandardScaler()
norm_features=scaler.fit_transform(features_w2v)

train_ft,test_ft,train_labels,test_labels=train_test_split(norm_featur
es,labels,test_size=0.2, train_size=0.8)
```

```
X_train, X_val, y_train, y_val = train_test_split(train_ft,
train_labels, test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", test_ft.shape)
```

```
Training set shape: (45783, 115)
Validation set shape: (11446, 115)
Test set shape: (14308, 115)
```

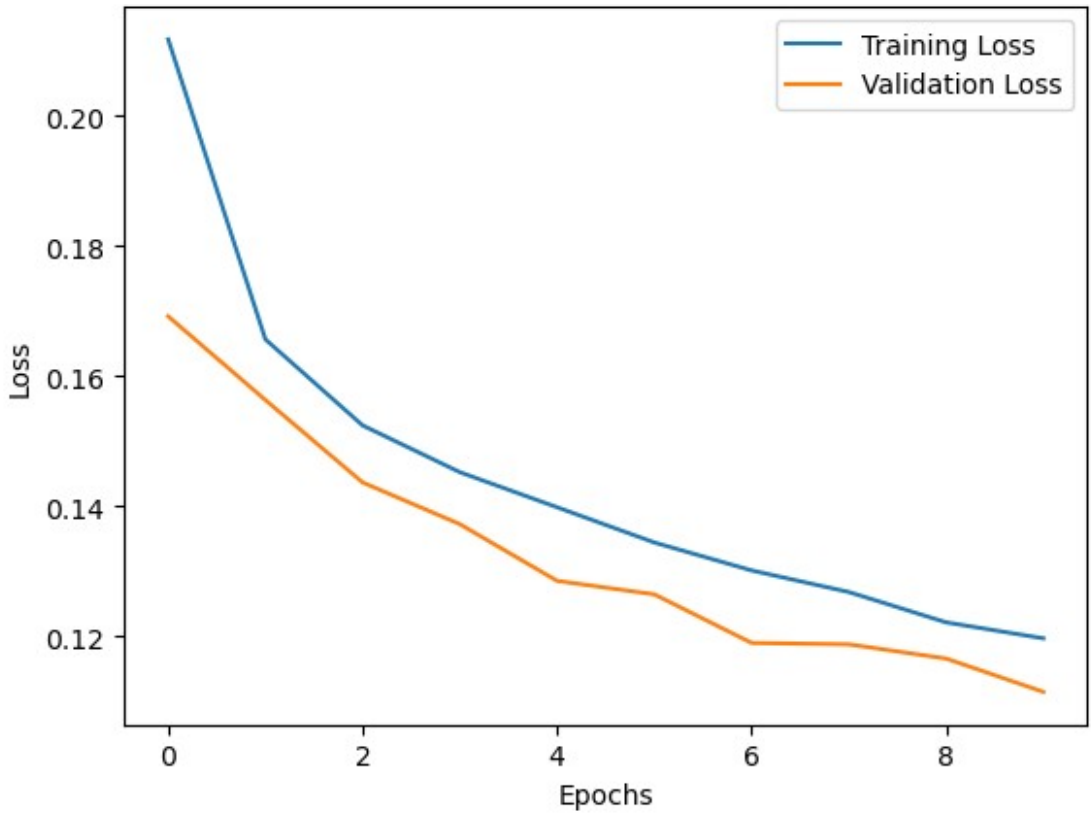
```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(115,)))  # Input
layer
model.add(Dense(64, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer
model.summary()
```

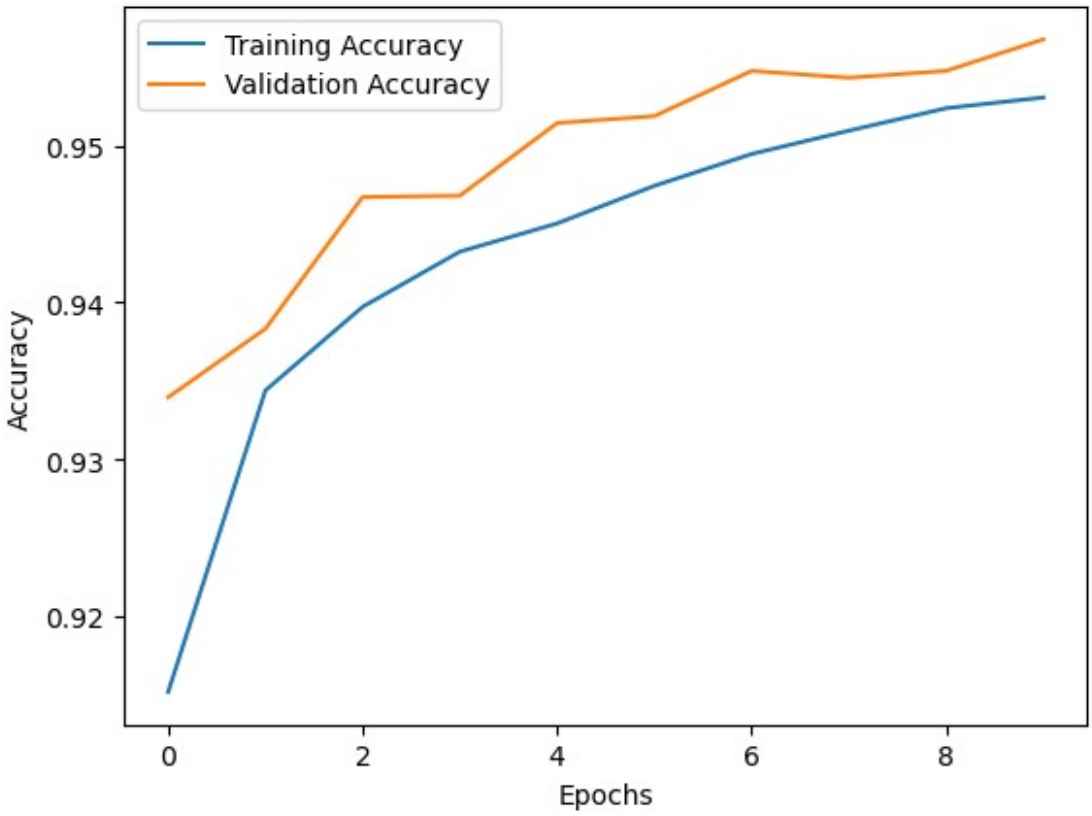
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 32)                3712

 dense_10 (Dense)            (None, 64)                2112

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 5,889
Trainable params: 5,889
Non-trainable params: 0
_________________________________________________________________
```

```
opt = keras.optimizers.SGD(learning_rate=0.1)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_ft, train_labels, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
```

```
Epoch 1/10
1789/1789 [==============================] - 5s 2ms/step - loss:
0.2118 - accuracy: 0.9152 - val_loss: 0.1692 - val_accuracy: 0.9340
Epoch 2/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1656 - accuracy: 0.9344 - val_loss: 0.1563 - val_accuracy: 0.9383
Epoch 3/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1524 - accuracy: 0.9397 - val_loss: 0.1436 - val_accuracy: 0.9467
Epoch 4/10
```

```
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1452 - accuracy: 0.9432 - val_loss: 0.1372 - val_accuracy: 0.9468
Epoch 5/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1398 - accuracy: 0.9450 - val_loss: 0.1285 - val_accuracy: 0.9514
Epoch 6/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1344 - accuracy: 0.9474 - val_loss: 0.1264 - val_accuracy: 0.9519
Epoch 7/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1301 - accuracy: 0.9494 - val_loss: 0.1190 - val_accuracy: 0.9547
Epoch 8/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1268 - accuracy: 0.9509 - val_loss: 0.1188 - val_accuracy: 0.9543
Epoch 9/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1221 - accuracy: 0.9524 - val_loss: 0.1166 - val_accuracy: 0.9547
Epoch 10/10
1789/1789 [==============================] - 3s 2ms/step - loss:
0.1197 - accuracy: 0.9530 - val_loss: 0.1114 - val_accuracy: 0.9568

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_ft, test_labels)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')

448/448 [==============================] - 1s 1ms/step - loss: 0.1572
- accuracy: 0.9417
Test Loss: 0.1571870893239975, Test Accuracy: 0.9417109489440918

plot_loss_acc(history)
```


Training Loss
Validation Loss
0.20
0.18
0.16
0.14
0.12
Loss
0
2
4
6
8
Epochs


Training Accuracy
Validation Accuracy
0.95
0.94
0.93
0.92
Accuracy
0
2
4
6
8
Epochs