```
import pandas as pd
import tensorflow as tf
import joblib
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from app.globals import train_set, test_set, train_labels, test_labels
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler

def under_sampling(df, target_column):
    X = df.drop(target_column, axis=1)
    y = df[target_column]
    rus = RandomUnderSampler(sampling_strategy='not minority',
random_state=101)
    X_res, y_res = rus.fit_resample(X, y)
    df_undersampled = pd.DataFrame(X_res, columns=X.columns)
    df_undersampled[target_column] = y_res
    return df_undersampled

# Load and process the dataset
# Define the list of CSV file paths
file_paths = [
    'data/imdb_cleaned_1.csv',
    'data/imdb_cleaned_2.csv',
    'data/imdb_cleaned_3.csv',
    'data/imdb_cleaned_4_5_6.csv',
    'data/imdb_cleaned_7.csv',
    'data/imdb_cleaned_8.csv',
    'data/imdb_cleaned_9.csv',
]
# Load all CSV files into a single DataFrame
dataframes = [pd.read_csv(file, usecols=['review', 'sentiment']) for
file in file_paths]
# Concatenate all dataframes
imdb_df = pd.concat(dataframes, ignore_index=True)

# Drop any rows with missing values
imdb_df.dropna(inplace=True)

imdb_df = under_sampling(imdb_df, 'sentiment')

review_df = imdb_df['review']  # DataFrame that contains review text
(feature column)
sentiment_df = imdb_df['sentiment']  # DataFrame that contains
sentiment label (target column)

# Split the dataset into training and testing sets
train_set, test_set, train_labels, test_labels =
```

```
train_test_split(review_df, sentiment_df, test_size=0.2,
random_state=101)

def preprocess_text(text):
    return text.lower()

def feature_extraction(X_train, X_test):
    # Replace NaN values with an empty string
    X_train = ["" if pd.isna(text) else text for text in X_train]
    X_test = ["" if pd.isna(text) else text for text in X_test]

    vectorizer = TfidfVectorizer()
    X_train_vectors = vectorizer.fit_transform(X_train)
    X_test_vectors = vectorizer.transform(X_test)
    return X_train_vectors, X_test_vectors

def preprocess_data_for_deep_learning(data):
    train_set, test_set, train_labels, test_labels = data
    train_set = ["" if pd.isna(text) else str(text) for text in
train_set]
    test_set = ["" if pd.isna(text) else str(text) for text in
test_set]
    vocab_size = max(len(set(text.split())) for text in train_set +
test_set)
    all_texts = train_set + test_set
    sequence_lengths = [len(text.split()) for text in all_texts]
    max_length = min(max(sequence_lengths), 500)
    oov_tok = '<OOV>'
    tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
    tokenizer.fit_on_texts(train_set)
    train_sequences = tokenizer.texts_to_sequences(train_set)
    test_sequences = tokenizer.texts_to_sequences(test_set)
    X_train = pad_sequences(train_sequences, maxlen=max_length)
    X_test = pad_sequences(test_sequences, maxlen=max_length)
    return X_train, X_test, train_labels, test_labels, vocab_size,
max_length

def pred_user_sentence(pred_sentences, model, tokenizer=None,
type='ml'):
    labels = ['Negative', 'Neutral', 'Positive']
    processed_sentences = [preprocess_text(sentence) for sentence in
pred_sentences]
    if type == 'pretrained':
        encoded_input = tokenizer(processed_sentences, max_length=512,
padding=True, truncation=True, return_tensors='pt')
        dummy_labels = [0] * len(processed_sentences)
        pred_dataset = IMDbDataset(encoded_input, dummy_labels)
        predictions = model.predict(pred_dataset)
        y_pred = np.argmax(predictions.predictions, axis=-1)
        result_labels = [labels[idx] for idx in y_pred]
```

```
    elif type == 'ml':
        _, transformed_sentences = feature_extraction(train_set,
processed_sentences)
        label_indices = model.predict(transformed_sentences)
        result_labels = [labels[idx] for idx in label_indices]
    elif type == 'dl':
        _, processed_sentences_sequences, _, _, _, _ =
preprocess_data_for_deep_learning(data=(train_set,
processed_sentences, None, None))
        predictions = model.predict(processed_sentences_sequences)
        result_labels = [labels[np.argmax(pred)] for pred in
predictions]
    return result_labels

def load_and_predict(pred_sentence, model_name, tokenizer=None):
    type = 'ml' if model_name in ['lr', 'svm', 'rf'] else 'dl'
    print(f"Loading model: {model_name}")
    if type == 'dl':
        filename = f'model/{model_name.lower()}.keras'
        model = tf.keras.models.load_model(filename)
    else:
        filename = f'model/{model_name.lower()}.sav'
        model = joblib.load(filename)
    return pred_user_sentence([pred_sentence], model=model,
tokenizer=tokenizer, type=type)

text = 'cincai'
model_name = 'svm'

sentiment =load_and_predict(text, model_name)
print(sentiment)
```

```
Loading model: svm

c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\base.py:318:
UserWarning: Trying to unpickle estimator SVC from version 1.5.0 when
using version 1.2.2. This might lead to breaking code or invalid
results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-
maintainability-limitations
  warnings.warn(

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[8], line 4
      1 text = 'cincai'
      2 model_name = 'svm'
----> 4 sentiment =load_and_predict(text, model_name)
```

```
      5 print(sentiment)

Cell In[7], line 60, in load_and_predict(pred_sentence, model_name,
tokenizer)
     58     filename = f'model/{model_name.lower()}.sav'
     59     model = joblib.load(filename)
---> 60 return pred_user_sentence([pred_sentence], model=model,
tokenizer=tokenizer, type=type)

Cell In[7], line 43, in pred_user_sentence(pred_sentences, model,
tokenizer, type)
     41 elif type == 'ml':
     42     _, transformed_sentences = feature_extraction(train_set,
processed_sentences)
---> 43     label_indices = model.predict(transformed_sentences)
     44     result_labels = [labels[idx] for idx in label_indices]
     45 elif type == 'dl':

File c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\svm\
_base.py:820, in BaseSVC.predict(self, X)
    818     y = np.argmax(self.decision_function(X), axis=1)
    819 else:
--> 820     y = super().predict(X)
    821 return self.classes_.take(np.asarray(y, dtype=np.intp))

File c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\svm\
_base.py:433, in BaseLibSVM.predict(self, X)
    417 def predict(self, X):
    418     """Perform regression on samples in X.
    419
    420     For an one-class model, +1 (inlier) or -1 (outlier) is
returned.
   (...)
    431         The predicted values.
    432     """
--> 433     X = self._validate_for_predict(X)
    434     predict = self._sparse_predict if self._sparse else
self._dense_predict
    435     return predict(X)

File c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\svm\
_base.py:613, in BaseLibSVM._validate_for_predict(self, X)
    610 check_is_fitted(self)
    612 if not callable(self.kernel):
--> 613     X = self._validate_data(
    614         X,
    615         accept_sparse="csr",
    616         dtype=np.float64,
    617         order="C",
    618         accept_large_sparse=False,
```

```
    619         reset=False,
    620     )
    622 if self._sparse and not sp.isspmatrix(X):
    623     X = sp.csr_matrix(X)

File c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\base.py:588, in
BaseEstimator._validate_data(self, X, y, reset, validate_separately,
**check_params)
    585     out = X, y
    587 if not no_val_X and check_params.get("ensure_2d", True):
--> 588     self._check_n_features(X, reset=reset)
    590 return out

File c:\Users\Asus\anaconda3\Lib\site-packages\sklearn\base.py:389, in
BaseEstimator._check_n_features(self, X, reset)
    386     return
    388 if n_features != self.n_features_in_:
--> 389     raise ValueError(
    390         f"X has {n_features} features, but
{self.__class__.__name__} "
    391         f"is expecting {self.n_features_in_} features as
input."
    392     )

ValueError: X has 10363 features, but SVC is expecting 10201 features
as input.
```