```
!git clone https://github.com/bellevue-university/dsc650.git
```

```
Cloning into 'dsc650'...
remote: Enumerating objects: 120326, done.
remote: Counting objects: 100% (128/128), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 120326 (delta 54), reused 121 (delta 53), pack-reused 120198
Receiving objects: 100% (120326/120326), 360.60 MiB | 13.99 MiB/s, done.
Resolving deltas: 100% (7337/7337), done.
Updating files: 100% (114699/114699), done.
```

```
pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.9/dist-packages (3.8.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from nltk) (8.1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from nltk) (4.65.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.9/dist-packages (from nltk) (2022.10.31)
```

```
pip install scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-packages (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn) (3.1.0)
```

```
import os
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Define the path to the IMDB dataset in your Google Colab environment
data_folder = '/content/dsc650/data/external/imdb/aclImdb'

# Define a function to load the reviews from a folder
def load_reviews(folder):
    reviews = []
    for file in os.listdir(folder):
        with open(os.path.join(folder, file), 'r', encoding='utf-8') as f:
            review = f.read()
            reviews.append(review)
    return reviews

# Load the positive and negative reviews from the train and test folders
train_pos_folder = os.path.join(data_folder, 'train', 'pos')
train_neg_folder = os.path.join(data_folder, 'train', 'neg')
test_pos_folder = os.path.join(data_folder, 'test', 'pos')
test_neg_folder = os.path.join(data_folder, 'test', 'neg')

train_pos_reviews = load_reviews(train_pos_folder)
train_neg_reviews = load_reviews(train_neg_folder)
test_pos_reviews = load_reviews(test_pos_folder)
test_neg_reviews = load_reviews(test_neg_folder)

# Combine the positive and negative reviews into a single list, and label them
train_reviews = train_pos_reviews + train_neg_reviews
test_reviews = test_pos_reviews + test_neg_reviews
train_labels = [1] * len(train_pos_reviews) + [0] * len(train_neg_reviews)
test_labels = [1] * len(test_pos_reviews) + [0] * len(test_neg_reviews)

# Define a function to preprocess the reviews
def preprocess(review):
    # Convert to lowercase
    review = review.lower()
```

```
    # Tokenize into words
    words = word_tokenize(review)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    # Join the words back into a string
    review = ' '.join(words)
    return review

# Preprocess the train and test reviews
train_reviews = [preprocess(review) for review in train_reviews]
test_reviews = [preprocess(review) for review in test_reviews]

# Define a pipeline for the classifier
classifier = Pipeline([
    ('vectorizer', TfidfVectorizer()),
    ('clf', LinearSVC())
])

# Split the train data into a smaller training set and a validation set
train_data, val_data, train_labels, val_labels = train_test_split(train_reviews, train_labels, test_size=0.2)

# Train the classifier on the smaller training set
classifier.fit(train_data, train_labels)

# Evaluate the classifier on the validation set
val_preds = classifier.predict(val_data)
val_acc = accuracy_score(val_labels, val_preds)
print('Validation accuracy:', val_acc)

# Test the classifier on the test set
test_preds = classifier.predict(test_reviews)
test_acc = accuracy_score(test_labels, test_preds)
print('Test accuracy:', test_acc)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Validation accuracy: 0.8946
Test accuracy: 0.86724
```

```
!pip install keras
!pip install tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.9/dist-packages (2.12.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.9/dist-packages (2.12.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.2.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.1)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.32.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.53.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow) (67.6.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.4.7)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/pythor
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.9/dist-packages (from astunparse>=1.6.0->tensorflow) (0.40.(
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow) (1.10.1)
Requirement already satisfied: ml-dtypes>=0.0.3 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow) (0.0.4)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->t
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->te
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow) (3.4
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard<
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from google-auth-oauthlib<1.1,>=0.5->
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.9/dist-packages (from markdown>=2.6.8->tensorboard<2.13
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.1
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensor
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oau
```

```
import os
import numpy as np
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


# Define the data folder
data_folder = '/content/dsc650/data/external/imdb/aclImdb'

# Load the IMDb movie review dataset
max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
train_dir = os.path.join(data_folder, 'train')
test_dir = os.path.join(data_folder, 'test')

# Load the train reviews and labels
train_reviews = []
train_labels = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname.endswith('.txt'):
            with open(os.path.join(dir_name, fname), encoding='utf-8') as f:
                train_reviews.append(f.read())
            train_labels.append(0 if label_type == 'neg' else 1)

# Load the test reviews and labels
test_reviews = []
test_labels = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname.endswith('.txt'):
            with open(os.path.join(dir_name, fname), encoding='utf-8') as f:
                test_reviews.append(f.read())
            test_labels.append(0 if label_type == 'neg' else 1)

print(len(train_reviews), 'train sequences')
print(len(test_reviews), 'test sequences')

# Tokenize the train and test reviews
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(train_reviews)
train_sequences = tokenizer.texts_to_sequences(train_reviews)
test_sequences = tokenizer.texts_to_sequences(test_reviews)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# Pad the train and test sequences
train_data = pad_sequences(train_sequences, maxlen=maxlen)
test_data = pad_sequences(test_sequences, maxlen=maxlen)
```

```
# Convert the train and test labels to numpy arrays
train_labels = np.asarray(train_labels)
test_labels = np.asarray(test_labels)

# Build the model
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Train the model
history = model.fit(train_data, train_labels,
                    epochs=10,
                    batch_size=batch_size,
                    validation_split=0.2)

# Evaluate the model on the test data
score, acc = model.evaluate(test_data, test_labels, batch_size=batch_size)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
Loading data...
25000 train sequences
25000 test sequences
Found 88582 unique tokens.
Epoch 1/10
625/625 [==============================] - 59s 93ms/step - loss: 0.5120 - acc: 0.7426 - val_loss: 0.5201 - val_acc: 0.8024
Epoch 2/10
625/625 [==============================] - 58s 92ms/step - loss: 0.3322 - acc: 0.8625 - val_loss: 0.3013 - val_acc: 0.9020
Epoch 3/10
625/625 [==============================] - 58s 92ms/step - loss: 0.2595 - acc: 0.8984 - val_loss: 0.4940 - val_acc: 0.8300
Epoch 4/10
625/625 [==============================] - 57s 91ms/step - loss: 0.2195 - acc: 0.9155 - val_loss: 0.5550 - val_acc: 0.7802
Epoch 5/10
625/625 [==============================] - 57s 92ms/step - loss: 0.1879 - acc: 0.9296 - val_loss: 0.5757 - val_acc: 0.7882
Epoch 6/10
625/625 [==============================] - 61s 98ms/step - loss: 0.1711 - acc: 0.9371 - val_loss: 0.5342 - val_acc: 0.8154
Epoch 7/10
625/625 [==============================] - 60s 95ms/step - loss: 0.1388 - acc: 0.9487 - val_loss: 0.5797 - val_acc: 0.8002
Epoch 8/10
625/625 [==============================] - 61s 98ms/step - loss: 0.1358 - acc: 0.9488 - val_loss: 0.7146 - val_acc: 0.7672
Epoch 9/10
625/625 [==============================] - 61s 97ms/step - loss: 0.1074 - acc: 0.9625 - val_loss: 0.8526 - val_acc: 0.7290
Epoch 10/10
625/625 [==============================] - 58s 93ms/step - loss: 0.0823 - acc: 0.9714 - val_loss: 0.8286 - val_acc: 0.7612
782/782 [==============================] - 18s 23ms/step - loss: 0.6358 - acc: 0.8096
Test score: 0.6357991695404053
Test accuracy: 0.8095999956130981
```

```
import numpy as np
from keras.datasets import reuters
from keras import models
from keras import layers
from keras.utils import to_categorical

# Load the news article dataset
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)

# Preprocess the data
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)

# Define the neural network architecture
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
```

```
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=512,
                    validation_data=(x_test, y_test))

# Evaluate the model
results = model.evaluate(x_test, y_test)

# Make predictions
predictions = model.predict(x_test)

# Print the accuracy and loss score
print("Test Accuracy:", results[1])
print("Test Loss:", results[0])
```

```
Epoch 1/10
18/18 [==============================] - 1s 54ms/step - loss: 2.7835 - accuracy: 0.5120 - val_loss: 1.8821 - val_accuracy: 0.6420
Epoch 2/10
18/18 [==============================] - 1s 39ms/step - loss: 1.5267 - accuracy: 0.6837 - val_loss: 1.4025 - val_accuracy: 0.6937
Epoch 3/10
18/18 [==============================] - 1s 37ms/step - loss: 1.1432 - accuracy: 0.7550 - val_loss: 1.2100 - val_accuracy: 0.7315
Epoch 4/10
18/18 [==============================] - 1s 38ms/step - loss: 0.9270 - accuracy: 0.7982 - val_loss: 1.1037 - val_accuracy: 0.7520
Epoch 5/10
18/18 [==============================] - 1s 40ms/step - loss: 0.7671 - accuracy: 0.8350 - val_loss: 1.0422 - val_accuracy: 0.7556
Epoch 6/10
18/18 [==============================] - 1s 59ms/step - loss: 0.6371 - accuracy: 0.8644 - val_loss: 0.9769 - val_accuracy: 0.7832
Epoch 7/10
18/18 [==============================] - 1s 62ms/step - loss: 0.5315 - accuracy: 0.8898 - val_loss: 0.9432 - val_accuracy: 0.7850
Epoch 8/10
18/18 [==============================] - 1s 45ms/step - loss: 0.4479 - accuracy: 0.9060 - val_loss: 0.9235 - val_accuracy: 0.7858
Epoch 9/10
18/18 [==============================] - 1s 39ms/step - loss: 0.3783 - accuracy: 0.9204 - val_loss: 0.9090 - val_accuracy: 0.7965
Epoch 10/10
18/18 [==============================] - 1s 39ms/step - loss: 0.3238 - accuracy: 0.9297 - val_loss: 0.9118 - val_accuracy: 0.7925
71/71 [==============================] - 0s 3ms/step - loss: 0.9118 - accuracy: 0.7925
71/71 [==============================] - 0s 2ms/step
Test Accuracy: 0.7925200462341309
Test Loss: 0.9118285775184631
```

```
import numpy as np
from keras.datasets import boston_housing
from keras import models
from keras import layers

# Load the Boston Housing dataset
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

# Normalize the data
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)
train_data -= mean
train_data /= std
test_data -= mean
test_data /= std

# Define the neural network architecture
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model

# Train the model using k-fold cross-validation
k = 4
```

```
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
    print('Processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=16, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)

# Print the validation scores
print("Validation Scores:", all_scores)
print("Mean Validation Score:", np.mean(all_scores))

# Train the final model
model = build_model()
model.fit(train_data, train_targets, epochs=80, batch_size=16, verbose=0)

# Evaluate the final model on the test data
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)

# Print the test results
print("Test MSE Score:", test_mse_score)
print("Test MAE Score:", test_mae_score)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
Processing fold # 0
Processing fold # 1
Processing fold # 2
Processing fold # 3
Validation Scores: [1.994585394859314, 2.286994457244873, 2.5223820209503174, 2.3804290294647217]
Mean Validation Score: 2.2960977256298065
4/4 [==============================] - 0s 3ms/step - loss: 17.5477 - mae: 2.5969
Test MSE Score: 17.54768943786621
Test MAE Score: 2.596897840499878
```