```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/drive
```

Load training data

The training data for building the model will be from the previous run of extracting keywords from NLTK.

```
import pandas as pd
filepath = '/content/drive/My Drive/final-project/keywords/keyword-context.txt'
df = pd.read_csv(filepath, names=['context', 'keywords'], sep=',')
print(df.iloc[0])
```

```
context     borqs beijing qualified high technology enterp...
keywords                                          borqs beijing
Name: 0, dtype: object
```

Generate Test and Train Data

First, we are going to split the data into a training and testing set which will allow us to evaluate the accuracy and se well.

This means whether the model is able to perform well on data it has not seen before.

```
from sklearn.model_selection import train_test_split

sentences = df['context'].values
y = df['keywords'].values

sentences_train, sentences_test, y_train, y_test = train_test_split(sentences, y, test_size=
```

Building the Keras Model

Keras supports two main types of models. You have the Sequential model API and the functional API which can do e model but it can be also used for advanced models with complex network architectures.

The Sequential model is a linear stack of layers, where we can use the large variety of available layers in Keras. The r Dense layer which is your regular densely connected neural network layer with all the weights and biases that we are

Build a helper function to visualize the loss and accuracy for the training and testing data based on the History callb automatically applied to each Keras model, records the loss and additional metrics that can be added in the .fit() me interested in the accuracy. This helper function employs the matplotlib plotting library:

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

Text preprocessing and sequence preprocessing.

Now we need to tokenize the data into a format that can be used by the word embeddings. Keras offers a couple of preprocessing and sequence preprocessing which you can employ to prepare your text.

Convolutional Neural Networks (CNN)

CNN has hidden layers which are called convolutional layers.

This is the very core of the technique, the mathematical process of convolution. With each convolutional layer the ne complex patterns.

When we are working with sequential data, like text, you work with one dimensional convolutions, but the idea and th You still want to pick up on patterns in the sequence which become more complex with each added convolutional la

```
embedding_dim = 100

model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, input_length=maxlen))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
history = model.fit(X_train, y_train,
                    epochs=10,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

Hyperparameters Optimization

One crucial steps of deep learning and working with neural networks is hyperparameter optimization.

One popular method for hyperparameter optimization is grid search. What this method does is it takes lists of param with each parameter combination that it can find. It is the most thorough way but also the most computationally hea common way, random search, which you'll see in action here, simply takes random combinations of parameters.

In order to apply random search with Keras, we use the KerasClassifier which serves as a wrapper for the scikit-learn able to use the various tools available with scikit-learn like cross-validation. The class is RandomizedSearchCV whic with cross-validation. Cross-validation is a way to validate the model and take the whole data set and separate it into data sets.

There are various types of cross-validation. One type is the k-fold cross-validation . In this type the data set is partitic where one set is used for testing and the rest of the partitions are used for training. This enables you to run k differe once used as a testing set. So, the higher k is the more accurate the model evaluation is, but the smaller each testin

First step for KerasClassifier is to have a function that creates a Keras model. We will use the previous model, but w to be set for the hyperparameter optimization:

```
def create_model(num_filters, kernel_size, vocab_size, embedding_dim, maxlen):
    model = Sequential()
    model.add(layers.Embedding(vocab_size, embedding_dim, input_length=maxlen))
    model.add(layers.Conv1D(num_filters, kernel_size, activation='relu'))
    model.add(layers.GlobalMaxPooling1D())
    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(1, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model


param_grid = dict(num_filters=[32, 64, 128],
                  kernel_size=[3, 5, 7],
                  vocab_size=[5000],
                  embedding_dim=[50],
                  maxlen=[100])
```

The resulting instance and the parameter grid are then used as the estimator in the RandomSearchCV class. Additio folds in the k-folds cross-validation, which is in this case 4. Besides the RandomSearchCV and KerasClassifier, there handling the evaluation:

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import RandomizedSearchCV
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras import layers

# Main settings
epochs = 20
embedding_dim = 50
maxlen = 100
output_file = '/content/output.txt'

# Run grid search for each source (yelp, amazon, imdb)
sentences = df['context'].values
y = df['keywords'].values

# Train-test split
sentences_train, sentences_test, y_train, y_test = train_test_split(
sentences, y, test_size=0.25, random_state=1000)

# Tokenize words
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(sentences_train)
X_train = tokenizer.texts_to_sequences(sentences_train)
X_test = tokenizer.texts_to_sequences(sentences_test)

# Adding 1 because of reserved 0 index
vocab_size = len(tokenizer.word_index) + 1

# Pad sequences with zeros
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
```

```
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

# Parameter grid for grid search
param_grid = dict(num_filters=[32, 64, 128],
                  kernel_size=[3, 5, 7],
                  vocab_size=[vocab_size],
                  embedding_dim=[embedding_dim],
                  maxlen=[maxlen])
model = KerasClassifier(build_fn=create_model,
                        epochs=epochs, batch_size=10,
                        verbose=False)
grid = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
                          cv=4, verbose=1, n_iter=5)
grid_result = grid.fit(X_train, y_train)

# Evaluate testing set
test_accuracy = grid.score(X_test, y_test)

# Save and evaluate results
with open(output_file, 'a') as f:
  s = ('Running {} data set\nBest Accuracy : '
       '{:.4f}\n{}\nTest Accuracy : {:.4f}\n\n')
  output_string = s.format(
      'edgar',
      grid_result.best_score_,
      grid_result.best_params_,
      test_accuracy)
  print(output_string)
  f.write(output_string)
```

Evaluating the performance of different model.

1) Below is a simple sequential model with 1 layer.

```
from keras.models import Sequential
from keras import layers
from sklearn.preprocessing import LabelEncoder

input_dim = X_train.shape[1]  # Number of features

model = Sequential()
model.add(layers.Dense(10, input_dim=input_dim, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
le = LabelEncoder()
y_test = le.fit_transform(y_test)
y_train =le.fit_transform(y_train)
history = model.fit(X_train, y_train,
```

```
                    epochs=25,
                    verbose=True,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy: {}".format(accuracy))
```

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(sentences_train)

X_train = tokenizer.texts_to_sequences(sentences_train)
X_test = tokenizer.texts_to_sequences(sentences_test)

vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0 index

maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

What Is a Word Embedding?

Text is considered a form of sequence data similar to time series data that you would have in weather data or financ
model, you have seen how to represent a whole sequence of words as a single feature vector.

There are various ways to vectorize text, such as:

- Words represented by each word as a vector
- Characters represented by each character as a vector
- N-grams of words/characters represented as a vector (N-grams are overlapping groups of multiple succeedin

Below we will use Keras for word embedding as well the he glove word embedding to see if they make a difference t

```
from keras.models import Sequential
from keras import layers

embedding_dim = 50

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
history = model.fit(X_train, y_train,
                    epochs=20,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
le = LabelEncoder()
y_test = le.fit_transform(y_test)
y_train =le.fit_transform(y_train)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
from keras.models import Sequential
from keras import layers

embedding_dim = 50

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))
```

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

```
le = LabelEncoder()
y_test = le.fit_transform(y_test)
y_train =le.fit_transform(y_train)
history = model.fit(X_train, y_train,
                    epochs=50,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
import numpy as np
```

```
def create_embedding_matrix(filepath, word_index, embedding_dim):
    vocab_size = len(word_index) + 1  # Adding again 1 because of reserved 0 index
    embedding_matrix = np.zeros((vocab_size, embedding_dim))

    with open(filepath) as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix[idx] = np.array(
                    vector, dtype=np.float32)[:embedding_dim]

    return embedding_matrix


embedding_dim = 100
embedding_matrix = create_embedding_matrix(
  '/content/drive/My Drive/final-project/raw/glove.6B.100d.txt',
   tokenizer.word_index, embedding_dim)


model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
                           weights=[embedding_matrix],
                           input_length=maxlen,
                           trainable=False))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
le = LabelEncoder()
y_test = le.fit_transform(y_test)
y_train =le.fit_transform(y_train)
history = model.fit(X_train, y_train,
                    epochs=25,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
                           weights=[embedding_matrix],
                           input_length=maxlen,
                           trainable=True))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
history = model.fit(X_train, y_train,
                    epochs=50,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
```

```
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```