Speech emotion recognition id30

Machine learning specialization-

source-https://www.coursera.org/specializations/machine-learning-introduction

What I learned

Linear regression -

It is a machine learning algorithm which can be used to predict certain continues outcomes from given data .The algorithm figures out how much weight to give to each feature using gradient decent

Feature scaling -

Feature scaling helps our algorithm perform faster. Some of the feature scaling methods are given below.

1. Mean normalization
2. Z-score normalization

Polynomial regression-

It is same as the linear regression but it has low bias but high variance

Classification

Binary classification

It uses a sigmoid function to give the final output as 1 or 0 hence it is called binary classification. It can be used for diagnosis of a disease .

Advance Learning algorithms

Neural networks

It consists of an input layer ,hidden layers and output layer . Neural networks are versatile and can be used in many applications . different machine learning algorithms can be used in different parts of the neural network layer to get final desirable output.

Relu functions can be also used in neural networks for optimization of the algorithm in case of regression and mostly used in hidden layers.

Dense layer-Each neuron output is a function of all the activation outputs of the previous layer.

Convolutional Layer-Each Neuron only looks at part of the previous layer's inputs.

- Faster computation
- Need less training data

Multiclass classification(softmax regression)

It can be used for multiclass classification using the probability principle of one condition given another event. cross entropy function is used to do gradient descent

Bias and variance-

Bias can be explained by the test train error and variance can be explained by cross validation test error.
Different type of regularization techniques can be used to reduce variance which is caused by the algorithm as it overfits with the training data

K-means Clustering

The algorithm assigns data points to the nearest centroid based on shortest distance.After assigning points, the average is computed to update the centroids.Elbow curve demonstrates abrupt changes in WCSS value vs. k value. Validation of model performance using silhouette score after finding optimal k value.

Deep learning -

source-https://www.youtube.com/watch?v=d2kxUVwWWwU

Perceptron-

Perceptrons are often used for binary classification problems where they learn to classify input data into two categories based on training examples.

Forward propagation in deep learning-

Forward propagation involves predicting output based on initialized weights.
The difference between predicted value and truth value needs to be minimized.

Training a neural network involves three key stages.

First, during forward propagation, the network processes input data to make predictions.Next, backward propagation, also known as backpropagation, comes

into play. Here, the network uses the errors from its predictions to fine-tune its internal settings, adjusting how much each piece of information influences its decisions.Finally, there's the weight update stage. This is where the network actually adjusts its internal settings based on the refinements calculated during backpropagation. It uses methods like gradient descent to reduce the errors in its predictions.Together, these steps enable the neural network to learn from its experiences and become better at making accurate predictions over time.

Chain rule of derivative

In deep learning, I've learned that the chain rule of derivatives plays a crucial role. It's similar to using a map to navigate complex routes. Initially, you figure out the impact of the outer function, guiding you through the first step. Then, you delve into the inner function's influence, which assists in the subsequent steps. Finally, applying the chain rule means combining these effects to see how even small input changes can affect the final outcome.In simpler terms, I now understand that the chain rule is like a guidebook for the neural network, showing how tiny adjustments at each stage build up to accurate predictions. It's like learning to tackle intricate puzzles, helping the network interpret data intricacies and enhance its predictive abilities over time.

Convolutional neural networks-

A specific kind of deep learning neural network called a convolutional neural network (CNN) is made to process and analyze visual data, including images. They now play a crucial role in many computer vision applications, offering solutions for problems including object identification, image categorization, and image recognition.Convolutional layers, pooling layers, and fully linked layers are usually the main parts of CNNs.Convolutional layers: These layers employ the convolution method to extract pertinent information from the input data using filters (kernels).
Layers for pooling: These layers help control the complexity of the network and avoid overfitting by reducing the spatial dimensions of the data.
Fully linked layers: These layers provide predictions by interpreting the features that were extracted by the preceding layers.

Natural language processing-

Nlp pipeline-

1. Data acquisition-collecting data for required project
2. Text preparation- preparing the text for further processing
3. Feature engineering-converting text in to numbers so that we can use it in neural networks
4. Modeling - Choosing models according to characteristic of our data
5. Deployment - Finally we can deploy the project and check if our models are performing well then according to the outcomes we can fine tune the models.

Text preprocessing step-

- Lowercasing -Lowercasing involves converting all the text to lowercase.Spelling correction is an important step in text preprocessing.So that our program does not differ between lower and upper case and it is the same word
- HTML tags need to be removed in text preprocessing-HTML tags are not required in machine learning models and can cause confusion Regular expressions can be used to remove HTML tags
- Removing punctuation is important for language formation and text analysis.-Punctuation marks like full stop, question mark, and comma are used in the English language.Removing punctuation helps in feature engineering, sentiment analysis, and tax documentation.
- Removing stopwords-Text preprocessing involves removing stopwords and punctuation marks.Text preprocessing can be done using the 'remove_stopwords' and 'remove_punctuation' functions.
- Text preprocessing for handling emojis-Two options for handling emojis: remove or replaceUsing regular expressions to identify and substitute emojis
- Text preprocessing involves tokenization and handling various problems-
- The processing stage involves proper tokenization of text Common problems -include handling prefixes, suffixes, and special charactersTokenization difficulties can arise due to these problems Algorithms may struggle with understanding certain characters and word structures

- Lexical dictionaries help in finding word meanings and lemmas-Lexical dictionaries store different words of the English language.They are used to find the lemma (root word) of a given word.

Text Representation-

- Text representation techniques and their importance-Explanation of fundamental text representation techniques such as Bag of Words and Tf-Idf.Advantages and disadvantages of these techniques and assignment related to text representation.
- Text representation techniques convert text into vectors.-Two famous techniques covered: Bag of Words and Tf-Idf.Text conversion to numbers is essential for machine learning algorithms.
- Advantages of the Bag of Words technique-The technique is intuitive and easy to implement, allowing for quick coding.It helps in understanding further techniques and overcoming potential disadvantages.
- Text Representation techniques in NLP-Exploration of Bag of Words, Tf-Idf, N-grams, Bi-grams and Uni-grams.Applications in text classification and performance evaluation
- NLP techniques like Bag of Words, Tf-Idf, and N-grams are key in text representation-N-grams help in analyzing sequences of words for better understanding.Tf-Idf helps in measuring the importance of words in a document.N-grams can capture the hidden meaning in sentences.
- Text Representation using Bag of Words and Tf-Idf-Bag of Words and Tf-Idf are used to find the most frequent words and their relevance.These techniques have advantages such as high usage in information retrieval systems, but also have drawbacks relating to vocabulary size and inability to capture complex relationships.

Word2vec- a method in natural language processing, is used for a variety of tasks that involve understanding and processing human language.

1. Text Classification: Word2vec helps in identifying and categorizing text into different classes or categories, such as spam detection, topic labeling, or sentiment analysis.
2. Sentiment Analysis: It assists in determining the sentiment expressed within a piece of text, whether it's positive, negative, or neutral.

3. Machine Translation: Word2vec aids in translating text from one language to another by capturing semantic and syntactic features of words and phrases.
4. Information Retrieval: It supports searching and retrieving relevant information from a large corpus of text based on the semantic similarity between words and documents.
5. Named Entity Recognition: It helps in identifying and classifying entities mentioned in text, such as names of people, organizations, or locations.
6. Recommendation Systems: Word2vec can be used to capture the semantic similarity between words and phrases, which is useful in building recommendation systems based on user preferences and item characteristics.

Text Classification-

Text classification is a type of supervised machine learning power that involves categorizing data into different classes.It is used in various fields like weather prediction, image classification, and sentiment analysis.It involves categorizing text data, such as email content and movie reviews, into different categories.

- Text classification methods including algorithms and existing APIs-Tourist approach is rarely used due to powerful algorithms. Big companies use existing APIs for text classification and analysis
- Key data preprocessing steps for text classification-Handled duplicate reviews and processed text by converting to lowercase. Transformed reviews into numerical labels for machine learning model
- Using pre-trained word2vec model or training a model on your data can improve text classification results.-The type of matrix factorization, like ATM Friday, and the algorithm used can influence the results.Using techniques like word2vec can improve results by creating embeddings based on the relationships within your data.

POS Tagging-

Assigning parts of speech to respective word in given sentence-

- Importance of POS Tagging in NLP-POS Tagging is crucial for understanding and disambiguating sentence meanings.Process involves importing libraries, creating documents, and extracting values for analysis.
- Understanding parts of speech and their classification-Parts of speech can be easily accessed by using index position dot POS and scores. Parts of speech

can be categorized as coarse-grained and fine-grained, with detailed classification and examples.

- POS Tagging and Hidden Markov Models-Explanation about the importance of understanding POS tagging using Hidden Markov Models.Process of applying POS tagging on a sentence using Hidden Markov Models.
- POS tagging using HMM and Viterbi algorithm-The frequency of words and their occurrences in different contexts.The process of converting word counts into probabilities using the Viterbi algorithm is explained.
- Understanding and utilizing translation probabilities and admission probabilities in POS tagging-Creating a Denmark off model with different components like noun, verb, and model along with their associated words.Analyzing the probability transitions between different components like start, end, nail, and walk in the model.
- POS Tagging and Hidden Markov Models in NLP-The model integrates admission and translation probabilities.The process involves calculating the probabilities of different parts of speech for each word.
- Using Viterbi Algorithm to find the highest probability combination of POS tagging-The algorithm calculates the product of probabilities for different combinations of POS tags to find the most probable sequence.By evaluating all possible combinations, the algorithm determines the sequence with the highest probability.
- Viterbi Algorithm optimizes part of speech tagging process-Long sentences lead to complex combinations, making calculation difficult.Viterbi Algorithm enables fast calculation by optimizing part of speech tagging process.
- Hidden Markov Models involve translation probabilities and path elimination.Translation probabilities determine the likelihood of moving between certain states.Path elimination involves removing paths with zero probability to simplify the model.

Actual project
code-https://github.com/himanshukarekar/speechemotionrecognition

Code was taking too much time on google collab and i was having problems running on my local machine(macbook privacy and system access issues)

Approach-

1)In “radvenessemotionalspeech/song” file name is given in such manner

We can extract the third part for our final out put

Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
Vocal channel (01 = speech, 02 = song).
Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
Repetition (01 = 1st repetition, 02 = 2nd repetition).
Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

2)In Tess data set emotion data is given in the file name.Hence we can extract the emotion from file name

Pre - processing of the data-

Objective:

The objective of this code is to normalize an audio signal to a specific decibel level and then visualize the normalized audio waveform.

Code Breakdown:

Normalization of Audio Signal:

python

```
normalizedsound = effects.normalize(rawsound, headroom
= 5.0)
```

1.
    - **`effects.normalize(rawsound, headroom = 5.0)`**: This function normalizes the audio signal contained in `rawsound`.
        - **`rawsound`**: This is the initial audio signal which is presumably loaded or processed before this step.
        - **`headroom = 5.0`**: This parameter specifies the desired headroom in decibels (dBFS). The audio is normalized so that the peak amplitude of the audio signal reaches -5.0 dBFS. This means the maximum amplitude of the audio will be at 5.0 dB below the maximum possible level.
    - The result, `normalizedsound`, is an `AudioSegment` object where the audio signal has been adjusted to the specified headroom level.

Conversion to Numpy Array:
python
```
normal_x =
np.array(normalizedsound.get_array_of_samples(), dtype
= 'float32')
```

2.
    - **`normalizedsound.get_array_of_samples()`**: Extracts the raw audio sample data from the `AudioSegment` object as a sequence of integers.
    - **`np.array(..., dtype = 'float32')`**: Converts this sequence into a NumPy array of type `float32`. This step is important for further processing or visualization using libraries like `librosa` or `matplotlib`.

Visualization of Normalized Audio:
python

```
plt.figure(figsize=(12,2))
librosa.display.waveshow(normal_x, sr=sr)
plt.title('Normalized audio')
```

3.
    - **`plt.figure(figsize=(12,2))`**: Creates a new figure for plotting with a specified size of 12 units wide and 2 units high.
    - **`librosa.display.waveshow(normal_x, sr=sr)`**: Plots the waveform of the normalized audio signal.
        - **`normal_x`**: The audio signal data in NumPy array format.
        - **`sr`**: The sample rate of the audio signal (should be defined earlier in the code).
    - **`plt.title('Normalized audio')`**: Sets the title of the plot to "Normalized audio".

This code snippet demonstrates the normalization and visualization of an audio signal:

1. **Normalization**:
    - The audio signal, initially in `rawsound`, is normalized to a peak level of -5.0 dBFS. This ensures that the signal's amplitude is adjusted such that its peak reaches the desired level, providing a consistent loudness across different audio signals.
2. **Conversion**:
    - After normalization, the audio signal is converted into a NumPy array. This conversion facilitates further numerical processing and visualization.
3. **Visualization**:
    - The waveform of the normalized audio signal is displayed using `librosa`. The plot provides a visual representation of the audio signal’s amplitude over time, allowing for analysis of the signal’s structure and content.

By normalizing the audio and visualizing it, you ensure that the audio signal is at a consistent level and can be easily analyzed or compared to other signals.

Objective:

The goal of this code is to trim silence from the beginning and end of an audio signal and then visualize and play the trimmed audio.

Code Breakdown:

Trimming Silence from Audio:

python

```
xt, index = librosa.effects.trim(normal_x, top_db = 30)
```

1.
    - **`librosa.effects.trim(normal_x, top_db = 30)`**: This function removes silent parts from the beginning and end of the audio signal.
        - **`normal_x`**: The audio signal that has been normalized and converted to a NumPy array.
        - **`top_db = 30`**: This parameter specifies the threshold for what is considered silence. In this case, any part of the audio where the amplitude is more than 30 dB below the maximum amplitude is considered silent and is trimmed. This value is adjustable depending on how much silence you want to remove.
    - **`xt`**: The resulting audio signal after trimming, which no longer contains the silent parts.
    - **`index`**: The indices of the non-silent portions of the original audio, which is returned but not used further in this snippet.

Visualization of Trimmed Audio:

python

```
plt.figure(figsize=(6,2))
librosa.display.waveshow(xt, sr=sr)
plt.title('Trimmed audio')
```

2.
   - **`plt.figure(figsize=(6,2))`**: Creates a new figure for plotting with a specified size of 6 units wide and 2 units high.
   - **`librosa.display.waveshow(xt, sr=sr)`**: Plots the waveform of the trimmed audio signal.
     - **`xt`**: The trimmed audio signal data in NumPy array format.
     - **`sr`**: The sample rate of the audio signal (should be defined earlier in the code).
   - **`plt.title('Trimmed audio')`**: Sets the title of the plot to "Trimmed audio".

Playing the Trimmed Audio:
python
```
ipd.display(ipd.Audio(data = xt, rate=sr))
```

3.
   - **`ipd.display(ipd.Audio(data = xt, rate=sr))`**: Displays an audio player widget that allows you to listen to the trimmed audio.
     - **`data = xt`**: Passes the trimmed audio data.
     - **`rate = sr`**: Passes the sample rate of the audio.

This code snippet performs the following tasks:

1. **Trimming Silence**:
   - **Function**: The `librosa.effects.trim` function is used to remove silent portions from the beginning and end of the audio signal.

    - **Parameters**: The `top_db` parameter is set to 30 dB, meaning that any audio segment where the amplitude is below 30 dB of the maximum is considered silence and is trimmed out.
    - **Output**: The trimmed audio is stored in `xt`, which excludes the non-informative silent parts, making the audio more concise and potentially more relevant for further analysis.
2. **Visualization**:
    - **Plot**: The waveform of the trimmed audio is visualized using `librosa.display.waveshow`. This plot shows the amplitude of the audio signal over time, highlighting the areas of the audio that have been retained after trimming.
3. **Playback**:
    - **Audio Player**: An audio player is displayed using `IPython.display`, allowing you to listen to the trimmed audio directly within a Jupyter notebook or similar environment.

By trimming silence from the audio, visualizing it, and providing playback capabilities, you can ensure that the audio is clean and focused on the relevant content, which is especially useful for analysis and processing tasks.

Objective:

The objective of this code is to pad an audio signal to a fixed length and then visualize .

Code Breakdown:

Padding the Audio Signal:

python

```
padded_x = np.pad(xt, (0, 173056 - len(xt)),
'constant')
```

1.

- **`np.pad(xt, (0, 173056 - len(xt)), 'constant')`**: Pads the audio signal `xt` to ensure it reaches a specified length of 173,056 samples.
  - **`xt`**: The audio signal that has been trimmed previously.
  - **`(0, 173056 - len(xt))`**: Specifies the padding configuration:
    - `0`: No padding is added at the beginning of the signal.
    - `173056 - len(xt)`: Adds padding to the end of the signal to make its total length 173,056 samples.
  - **`'constant'`**: The type of padding used, which pads the signal with zeros. This is useful for ensuring a consistent input size, especially for machine learning models or other processing that requires fixed-length inputs.
- **`padded_x`**: The resulting audio signal after padding, which has a total length of 173,056 samples.

Visualization of Padded Audio:
python

```
plt.figure(figsize=(12,2))

librosa.display.waveshow(padded_x, sr=sr)

plt.title('Padded audio')
```

2.

- **`plt.figure(figsize=(12,2))`**: Creates a new figure for plotting with a specified size of 12 units wide and 2 units high.
- **`librosa.display.waveshow(padded_x, sr=sr)`**: Plots the waveform of the padded audio signal.
  - **`padded_x`**: The padded audio signal data in NumPy array format.
  - **`sr`**: The sample rate of the audio signal (should be defined earlier in the code).

- `plt.title('Padded audio')`: Sets the title of the plot to "Padded audio".

Playing the Padded Audio:
python

```
ipd.display(ipd.Audio(data = padded_x, rate=sr))
```

3.
   - `ipd.display(ipd.Audio(data = padded_x, rate=sr))`: Displays an audio player widget that allows you to listen to the padded audio.
     - `data = padded_x`: Passes the padded audio data.
     - `rate = sr`: Passes the sample rate of the audio.

This code snippet performs the following tasks:

1. **Padding**:
   - **Function**: The `np.pad` function is used to extend the length of the audio signal to a fixed size of 173,056 samples.
   - **Padding Type**: Zero-padding is applied, meaning that the additional samples are filled with zeros. This is commonly done to standardize input lengths for audio processing tasks or neural network models.
   - **Output**: The padded audio signal is stored in `padded_x`, which has a total length of 173,056 samples, with zeros added to the end if the original signal was shorter.
2. **Visualization**:
   - **Plot**: The waveform of the padded audio is visualized using `librosa.display.waveshow`. This plot shows the amplitude of the padded audio signal over time, allowing for an assessment of how padding affects the signal's representation.
3. **Playback**:
   - **Audio Player**: An audio player is displayed using `IPython.display`, allowing you to listen to the padded audio directly within a Jupyter notebook or similar environment.

Padding the audio signal ensures that all audio inputs are of a consistent length, which is important for processing and analysis, particularly in machine learning models that require fixed-size inputs. Visualization and playback provide a way to verify that padding has been applied correctly and to assess the quality of the padded audio.

Noise Reduction-

1. **Noise Reduction**:
    - **Function**: The `nr.reduce_noise` function from the `noisereduce` library is used to reduce noise from the audio signal.
    - **Parameters**:
        - **`y=padded_x`**: The input audio signal that has been padded.
        - **`sr=sr`**: The sample rate of the audio signal, necessary for effective noise reduction.
        - **`stationary=True`**: Assumes that the noise is stationary, which simplifies the noise reduction process.
    - **Output**: The noise-reduced audio is stored in `final_x`, which should have a cleaner signal with reduced background noise.
2. **Visualization**:
    - **Plot**: The waveform of the noise-reduced audio is visualized using `librosa.display.waveshow`. This plot shows the amplitude of the audio signal over time, providing a visual representation of how the noise reduction has affected the signal.

Noise reduction improves the quality of the audio signal by removing unwanted background noise, making it clearer and more suitable for further analysis or processing. Visualization and playback help verify the effectiveness of the noise reduction and assess the quality of the cleaned audio.

This code snippet extracts various audio features from the processed audio signal `final_x` using the `librosa` library. The extracted features provide different aspects of the audio signal's characteristics:

Feature Extraction

1. **RMS Energy**: Measures the energy of the audio signal.
2. **Zero Crossing Rate (ZCR)**: Indicates how frequently the audio signal crosses zero amplitude.
3. **Mel-Frequency Cepstral Coefficients (MFCCs)**: Represents the short-term power spectrum of the audio.
4. **Spectral Centroid**: Represents the "center of mass" of the spectrum, indicating where the bulk of the signal's energy is concentrated.
5. **Spectral Rolloff**: Measures the frequency below which a specified percentage of the total spectral energy is contained.
6. **Chroma Feature (Chromagram)**: Relates to the 12 different pitch classes in the chromatic scale.
7. **Mel Spectrogram**: Represents the frequency content of the audio signal on the Mel scale.
8. **Spectral Bandwidth**: Measures the width of the spectral band.
9. **Spectral Contrast**: Measures the difference in amplitude between peaks and valleys in the sound spectrum.

Each feature is computed with specific parameters (e.g., frame length, hop length) and its shape is printed to provide insight into the dimensionality of the extracted features. These features are crucial for various audio processing and analysis tasks, including classification, recognition, and synthesis.

LSTM -Model

```
model = Sequential()

model.add(layers.LSTM(64, return_sequences = True, input_shape=(X.shape[1:9])))

model.add(layers.LSTM(64))

model.add(layers.Dense(8, activation = 'softmax'))

print(model.summary())
```

```
batch_size = 32




# Compile & train

model.compile(loss='categorical_crossentropy',

              optimizer='RMSProp',

              metrics=['categorical_accuracy'])



history = model.fit(x_train, y_train_class,

                    epochs=340, batch_size = batch_size,

                    validation_data = (x_val, y_val_class))
```

The code snippet provided builds, compiles, and trains an LSTM-based neural network model for a classification task. The model consists of:

Two LSTM Layers:

The first LSTM layer has 64 units and is configured to return the full sequence of outputs for each input sequence.

The second LSTM layer also has 64 units but returns only the output for the last time step.

One Dense Layer:

A dense layer with 8 units and a softmax activation function, which outputs a probability distribution over 8 classes.

The model is compiled using categorical crossentropy as the loss function, RMSProp as the optimizer, and categorical accuracy as the evaluation metric. The

model is trained for 340 epochs with a batch size of 32, and its performance is evaluated using a validation dataset.