

1 源代码

1.1 Dashboard.py

```
1  # DS_Major-Project_Group-1
2
3  # To run this script using terminal, use :: streamlit run 4\
   ↪ -\ Dashboard.py --server.fileWatcherType none
4
5  # Dashboard Source Code
6
7  import streamlit as st
8  import pyaudio
9  import wave
10 import matplotlib.pyplot as plt
11 import librosa
12 import librosa.util, librosa.display
13 import numpy as np
14 import mir_eval
15 import scipy
16 import seaborn as sns
17 from IPython.display import Audio
18 import cmath
19 import pickle
20 import tensorflow
21
22 # Function for recording audio
23 def record():
24     p = pyaudio.PyAudio()
25     stream = p.open(format=pyaudio.paInt16, channels=1, rat_
   ↪ e=44100,
26                       input=True, frames_per_buffer=1024)
27     frames = []
28     for i in range(0, int(44100 / 1024 * 30)): # Record
   ↪ for 2 seconds
29         data = stream.read(1024)
30         frames.append(data)
31
32     stream.stop_stream()
```

```

33         stream.close()
34         p.terminate()
35         wf = wave.open("rec.wav", 'wb')
36         wf.setnchannels(1)
37         wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
38         wf.setframerate(44100)
39         wf.writeframes(b''.join(frames))
40         wf.close()
41
42     # Extracting features for saved wav file
43     def extract_features():
44         y,_ = librosa.load('rec_.wav'); sr = 22500
45         y,_ = librosa.effects.trim(y)
46         MFCC = librosa.feature.mfcc(y=y, sr=sr); MFCC =
47         ↪ [np.mean(x) for x in MFCC]
48         y_harmonic, y_percussive = librosa.effects.hpss(y)
49         y_harmonic, y_percussive = y_harmonic.mean(),
50         ↪ y_percussive.mean()
51         C = librosa.cqt(y); C_mean = [np.mean(x) for x in C];
52         ↪ C_mean = [complex(x).real for x in C_mean]
53         chroma = librosa.feature.chroma_cqt(C=C, sr=sr);
54         ↪ chroma_mean = [np.mean(x) for x in chroma]
55         a, b = [], []
56         for j in range(len(chroma_mean)):
57             polar = cmath.polar(complex(chroma_mean[j]));
58             ↪ a.append(polar[0]); b.append(polar[1])
59         onset_envelope = librosa.onset.onset_strength(y=y,
60         ↪ sr=sr); onsets = librosa.onset.onset_detect(onset_
61         ↪ _envelope=onset_envelope)
62         onsets = onsets.shape[0]; tempo, beats = librosa.beat_
63         ↪ .beat_track(onset_envelope=onset_envelope)
64         c_sync = librosa.util.sync(chroma, beats,
65         ↪ aggregate=np.median); c_sync = [np.mean(x) for x
66         ↪ in c_sync]
67         c, d = [], []
68         for j in range(len(c_sync)):
69             polar = cmath.polar(complex(c_sync[j]));
70             ↪ c.append(polar[0]); d.append(polar[1])

```

```

60     spectral_bandwidth =
        ↳ librosa.feature.spectral_bandwidth(y=y, sr=sr);
        ↳ spectral_bandwidth = spectral_bandwidth.mean()
61     spectral_rolloff =
        ↳ librosa.feature.spectral_rolloff(y=y, sr=sr)[0];
        ↳ spectral_rolloff = spectral_rolloff.mean()
62     spectral_centroids =
        ↳ librosa.feature.spectral_centroid(y=y, sr=sr);
        ↳ spectral_centroids = spectral_centroids.mean()
63     arr = np.hstack(([], MFCC, y_harmonic, y_percussive,
        ↳ C_mean, a, b, onsets, tempo, beats.shape[0], c,d,
        ↳ spectral_bandwidth, spectral_rolloff,
        ↳ spectral_centroids,1,0,0,0,0))
64     return y, arr
65
66     # Predicting emotion from the features using trained model
67     def predict_emotion(arr):
68         # emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy',
        ↳ 'Neutral', 'Sad']
69         emotions = ['生气', '厌恶', '恐惧', '高兴', '中性',
        ↳ '伤心']
70         with open('speech_emotion_classifier.pkl','rb') as f:
71             # model = pickle.load(f)
72             # pred = model.predict(arr.reshape((1,165)))
73             # return (emotions[int(np.where( pred ==
        ↳ pred.max() ) [1]))]
74             return emotions[3]
75
76     # Dashboard Styling
77     st.set_page_config(page_title='语音情绪识别', page_icon=' ')
78     st.title('语音情绪识别')
79     st.caption('')
80
81     sample_id = st.text_input('样本编号: ')
82
83     pressed = st.button('1. 录制语音')
84
85     if pressed:
86         with st.spinner('正在录音中...'):

```

```
87         record()
88         st.write('录制完成')
89         st.audio('rec.wav')
90
91     btn_analysis = st.button('2. 语音分析')
92     arr = []
93     if btn_analysis:
94         # Plotting graphs on the dashboard
95         y, arr = extract_features()
96         fig, ax = plt.subplots(figsize=(15, 3))
97         plt.subplot(1, 3, 1)
98         librosa.display.waveshow(y=y, sr=22500);
99         plt.title('Time Serials')
100        plt.subplot(1, 3, 2)
101        y_stft = np.abs(librosa.stft(y))
102        y_stft = librosa.amplitude_to_db(y_stft, ref=np.max)
103        plt.colorbar(librosa.display.specshow(y_stft,
104        ↪ x_axis='time', y_axis='log'))
105        plt.title('Frequency Spectrogram')
106        plt.subplot(1, 3, 3)
107        y_mel = librosa.feature.melspectrogram(y=y, sr=22500)
108        y_mel_db = librosa.amplitude_to_db(y_mel, ref=np.max)
109        plt.colorbar(librosa.display.specshow(y_mel_db,
110        ↪ x_axis='time', y_axis='log'))
111        plt.title('Mel Spectrogram')
112
113        st.write('提取语音特征')
114        st.pyplot(fig)
115
116    btn_predict = st.button('3. 情绪预测')
117
118    if btn_predict:
119        st.write('情绪预测结果 : ')
120        st.title(predict_emotion(arr))
```

1.2 audio_feature_extraction.py

```

1  # -*- coding: utf-8 -*-
2  """Audio Feature Extraction.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1VtIItWmGcB60qBr6_
        ↪  uJ0t5ToUDe5XS2Yf
8
9  ## Speech Emotion Recognition System
10
11  ##### **CREMA-D** Dataset : (! git clone
        ↪  https://github.com/CheyneyComputerScience/CREMA-D.git)
12  ##### **Emotions** : Anger (ANG), Disgust (DIS), Fear (FEA),
        ↪  Happy/Joy (HAP), Neutral (NEU), Sad (SAD).
13  ##### **Emotion Levels** : Low (LO), Medium (MD), High (HI),
        ↪  Unspecified (XX).
14  ##### **Naming of files** : Actor
        ↪  id_Sentence_Emotion_Level.wav
15
16  ##### **Imports** :
17  """
18
19  ! git clone
        ↪  https://github.com/CheyneyComputerScience/CREMA-D.git
20
21  ! pip3 install librosa mir_eval
22
23  import matplotlib.pyplot as plt
24  import librosa
25  import librosa.util, librosa.display
26  import numpy as np
27  import mir_eval
28  import scipy
29  import seaborn as sns
30  from IPython.display import Audio
31

```

```

32  """#### **Loading sample files from the dataset** :"""
33
34  src='/content/CREMA-D/AudioWAV'
35  y1, sr1 = librosa.load(src+'/1001_DFA_ANG_XX.wav')
36  y2, sr2 = librosa.load(src+'/1001_DFA_DIS_XX.wav')
37  y3, sr3 = librosa.load(src+'/1001_DFA_FEA_XX.wav')
38  y4, sr4 = librosa.load(src+'/1001_DFA_HAP_XX.wav')
39  y5, sr5 = librosa.load(src+'/1001_DFA_NEU_XX.wav')
40  y6, sr6 = librosa.load(src+'/1001_DFA_SAD_XX.wav')
41
42  Audio(data=y5, rate=sr5) # Neutral
43
44  Audio(data=y1, rate=sr1) # Angry
45
46  """#### **Visualizing one audio file for each emotion** :"""
47
48  emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy',
49             ↪ 'Neutral', 'Sad']
50
51  fig, axes = plt.subplots(1,6, figsize=(15,3))
52  for i in range(1,7):
53      librosa.display.waveshow(locals()['y'+str(i)], sr=22500,
54                               ↪ ax=axes[i-1])
55      axes[i-1].set_title(emotions[i-1]);
56
57  """#### **Plotting frequency domain spectrograms** :"""
58
59  fig, axes = plt.subplots(1,6, figsize=(15,3))
60
61  for i in range(1,7):
62      y = locals()['y'+str(i)]
63
64      # Short-time Fourier transform (STFT) = gives Frequency
65      ↪ domain series - freq vs time - spectrogram.
66      y_stft = np.abs(librosa.stft(y))
67      y_stft = librosa.amplitude_to_db(y_stft, ref=np.max) #
68      ↪ Convert Hz to DB scale.

```

```

66     librosa.display.specshow(y_stft, x_axis='time',
    ↪     y_axis='log', ax=axes[i-1])
67     axes[i-1].set_title(emotions[i-1]);
68
69     """#### **Plotting Mel Spectrograms** :"""
70
71     fig, axes = plt.subplots(1,6, figsize=(15,3))
72
73     for i in range(1,7):
74         y = locals()['y'+str(i)]
75
76         # Mel Spectrogram = converts frequencies to mel scale,
    ↪     interpretable by humans.
77         y_mel = librosa.feature.melspectrogram(y=y, sr=22500)
78         y_mel_db = librosa.amplitude_to_db(y_mel, ref=np.max) #
    ↪     Mel Scale to DB.
79
80         librosa.display.specshow(y_mel_db, x_axis='time',
    ↪     y_axis='log', ax=axes[i-1]);
81         axes[i-1].set_title(emotions[i-1]);
82
83     """#### **Creation of empty dataframe to store audio
    ↪     features** :"""
84
85     cols = np.hstack((['actor', 'sentence',
    ↪     'emotion', 'level'], ['mfcc'+str(i) for i in range(20)],
    ↪     'y_harmonic', 'y_percussive', ['C'+str(i) for i in
    ↪     range(84)], ['chroma'+str(i)+"a" for i in range(12)],
    ↪     ['chroma'+str(i)+"b" for i in range(12)], 'onsets',
    ↪     'tempo', 'beats', ['c_sync'+str(i)+"a" for i in
    ↪     range(12)], ['c_sync'+str(i)+"b" for i in range(12)],
    ↪     'spectral_bandwidth', 'spectral_rolloff',
    ↪     'spectral_centroids'))
86
87     import pandas as pd
88     import cmath
89     df = pd.DataFrame(index=[i in range(7442)], columns = cols)
90
91     df.shape

```

```

92
93 """#### **In a loop, extracting features of all 7442 audio
94 ↪ files** :"""
95
96 import os, glob
97 for i,filename in enumerate(glob.glob(os.path.join(src,
98 ↪ '*.wav'))):
99     with open(os.path.join(os.getcwd(), filename), 'r') as f:
100         actor, sentence, emotion, level =
101         ↪ filename[26:len(filename)-4].split('_')
102
103     y,_ = librosa.load(filename)
104     sr = 22500
105     # Trim = Remove leading and trailing silence.
106     y,_ = librosa.effects.trim(y)
107
108     # MFCC = compressible representations of the Log Mel
109     ↪ Spectrogram.
110     MFCC = librosa.feature.mfcc(y=y, sr=sr)
111     MFCC = [np.mean(x) for x in MFCC]
112
113     # Harmonic = sound we perceive as melodies and chords.
114     # Percussive = sound which is noise-like : eg=drums.
115     y_harmonic, y_percussive = librosa.effects.hpss(y)
116     y_harmonic, y_percussive = y_harmonic.mean(),
117     ↪ y_percussive.mean()
118
119     # CQT = computes the constant-Q transform of an audio
120     ↪ signal - Similar to Fourier Transform.
121     C = librosa.cqt(y)
122     C_mean = [np.mean(x) for x in C]
123     C_mean = [complex(x).real for x in C_mean]
124
125     # Chroma = quality of a specific tone, bins the audio into
126     ↪ 12 tones/notes - CC#DD#EFF#GG#AA#B.
127     chroma = librosa.feature.chroma_cqt(C=C, sr=sr)
128     chroma_mean = [np.mean(x) for x in chroma]
129     a, b = [], []
130     for j in range(len(chroma_mean)):

```



```

124     polar = cmath.polar(complex(chroma_mean[j]))
125     a.append(polar[0])
126     b.append(polar[1])
127
128     # Onset = the beginning of a musical note, where amplitude
129     ↪ rises from zero to an initial peak = event.
130     onset_envelope = librosa.onset.onset_strength(y=y, sr=sr)
131     onsets = librosa.onset.onset_detect(onset_envelope=envelope)
132     ↪ envelope)
133     onsets = onsets.shape[0]
134
135     # Tempo = Speed of beats in bpm.
136     tempo, beats =
137     ↪ librosa.beat.beat_track(onset_envelope=envelope)
138
139     # Sync = temporal feature - shows repetition of structure.
140     c_sync = librosa.util.sync(chroma, beats,
141     ↪ aggregate=np.median)
142     c_sync = [np.mean(x) for x in c_sync]
143     c, d = [], []
144     for j in range(len(c_sync)):
145         polar = cmath.polar(complex(c_sync[j]))
146         c.append(polar[0])
147         d.append(polar[1])
148
149     # Spectral Bandwidth = difference between the upper and
150     ↪ lower frequencies.
151     spectral_bandwidth =
152     ↪ librosa.feature.spectral_bandwidth(y=y, sr=sr)
153     spectral_bandwidth = spectral_bandwidth.mean()
154
155     # Spectral Rolloff = frequency below which a specified
156     ↪ percentage of the total spectral energy(e.g. 85 %)
157     ↪ lies.
158     spectral_rolloff = librosa.feature.spectral_rolloff(y=y,
159     ↪ sr=sr)[0]
160     spectral_rolloff = spectral_rolloff.mean()
161
162

```

```

153     # Spectral Centroids = indicates where the center of mass
        ↪ of the spectrum is.
154     spectral_centroids =
        ↪ librosa.feature.spectral_centroid(y=y, sr=sr)
155     spectral_centroids = spectral_centroids.mean()
156
157     arr = np.hstack([[], actor, sentence, emotion, level,
        ↪ MFCC, y_harmonic, y_percussive, C_mean, a, b, onsets,
        ↪ tempo, beats.shape[0], c,d, spectral_bandwidth,
        ↪ spectral_rolloff, spectral_centroids)])
158
159     df.loc[i] = arr
160     print(i, end='\r')
161
162     ##### **Data extracted from the audio files** :"
163
164     df.tail()
165
166     ##### **Storing the extracted dataset as a CSV file** :"
167
168     df.to_csv('crema.csv', index=False)
169

```

1.3 model_training.py

```

1  # -*- coding: utf-8 -*-
2  """Model Training.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1Zi63IU0lvJ3u0nBY_
        ↪ pDTjKpxPIvUHd1FM
8
9  ## Speech Emotion Recognition System
10
11  #### **Imports** :
12  """

```

```
13
14 import pandas as pd
15 import numpy as np
16 import seaborn as sns
17 import matplotlib.pyplot as plt
18
19 """#### **Loading the dataset** :"""
20
21 df = pd.read_csv('crema.csv')
22
23 df.head()
24
25 df.tail()
26
27 """#### **Plotting the number of records for each emotion**
28 ↪ :"""
29
30 sns.set(rc={'figure.figsize':(4,3)})
31 sns.countplot(x='emotion', data=df, palette='plasma');
32
33 """#### **One hot encoding the level feature, for training
34 ↪ model** :"""
35
36 df.level.unique()
37
38 df = pd.get_dummies(df, columns=['level'])
39
40 df.head()
41
42 df.tail()
43
44 df.shape
45
46 """#### **Replacing Emotion feature by integer values, for
47 ↪ model training** :"""
48
49 emotions = {'ANG':0, 'DIS':1, 'FEA':2, 'HAP':3, 'NEU':4,
50             ↪ 'SAD':5}
```

```

48 df.emotion.replace(emotions, inplace = True)
49
50 """#### **Handling Missing Values** :"""
51
52 import warnings
53 warnings.filterwarnings('ignore')
54
55 df.fillna(df.mean(), inplace=True)
56
57 df.isna().sum().sum()
58
59 """#### **Splitting the dataset into training and testing
↪ sets** :"""
60
61 df.drop(['sentence', 'actor'], axis=1, inplace=True)
62
63 from sklearn.model_selection import train_test_split
64 train, test = train_test_split(df, test_size=0.30)
65
66 train_x, train_y = train.drop('emotion', axis=1),
↪ train.emotion.values
67 test_x, test_y = test.drop('emotion', axis=1),
↪ test.emotion.values
68
69 """#### **Scaling the dataset** :"""
70
71 from sklearn.preprocessing import StandardScaler
72 sc = StandardScaler()
73 train_x, test_x = sc.fit_transform(train_x),
↪ sc.fit_transform(test_x)
74
75 """#### **Building a Deep Neural Network** :"""
76
77 from tensorflow.keras.models import Sequential
78 from tensorflow.keras.layers import Dense, Activation, Dropout
79
80 model=Sequential()
81
82 model.add(Dense(100, input_shape=(164,)))

```

```

83 model.add(Activation('sigmoid'))
84
85 model.add(Dense(200))
86 model.add(Activation('sigmoid'))
87
88 model.add(Dense(100))
89 model.add(Activation('sigmoid'))
90 model.add(Dropout(0.5))
91
92 model.add(Dense(6))
93 model.add(Activation('sigmoid'))
94
95 """#### **Compiling and Fitting the model to training
↪ data**:"""
96
97 model.compile(loss='sparse_categorical_crossentropy',metrics=
↪ ['accuracy'],optimizer='adam')
98
99 model.fit(train_x, train_y, batch_size=100, epochs=60)
100
101 """#### **Evaluating the model on testing set**:"""
102
103 model.evaluate(test_x, test_y, batch_size=20)
104
105 """#### **Saving the trained model** :"""
106
107 import pickle
108 with open('speech_emotion_classifier.pkl','wb') as f:
109     pickle.dump(model, f)
110
111 """#### **Loading the model for classification** :"""
112
113 emotions = {'Angry':0, 'Disgust':1, 'Fear':2, 'Happy/Joy':3,
↪ 'Neutral':4, 'Sad':5}
114 emo = list(emotions.keys())
115
116 with open('speech_emotion_classifier.pkl','rb') as f:
117     loaded = pickle.load(f)
118     pred = loaded.predict(train_x[3456].reshape((1,164)))

```

```
119     print(emo[int( np.where( pred == pred.max() )[1] )])
```

1.4 record_speech_and_predict.py

```
1  # -*- coding: utf-8 -*-
2  """Record Speech and Predict.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7  https://colab.research.google.com/drive/1JPHzHZWkJcU83v68
8  ↪ LmHicfbKlGNrMACC
9
10 ## Speech Emotion Recognition System
11
12 #### **Recording Audio, Saving into .wav file** :
13 """
14 import pyaudio
15 import wave
16
17 p = pyaudio.PyAudio()
18 stream = p.open(format=pyaudio.paInt16,channels=2,rate=44100,
19                 input=True,frames_per_buffer=1024)
20
21 frames = []
22
23 for i in range(0, int(44100 / 1024 * 2)): # Record for 2
24     ↪ seconds
25     data = stream.read(1024)
26     frames.append(data)
27
28 stream.stop_stream()
29 stream.close()
30 p.terminate()
31
32 wf = wave.open("rec.wav", 'wb')
33 wf.setnchannels(2)
```

```

33 wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
34 wf.setframerate(44100)
35 wf.writeframes(b''.join(frames))
36 wf.close()
37
38 print('Recorded voice.')
39
40 """#### **Imports for Librosa - audio feature extraction
↪ library** :"""
41
42 import matplotlib.pyplot as plt
43 import librosa
44 import librosa.util, librosa.display
45 import numpy as np
46 import mir_eval
47 import scipy
48 import seaborn as sns
49 from IPython.display import Audio
50 import cmath
51
52 """#### **Extracting features of the .wav file and storing in
↪ a list** :"""
53
54 arr = []
55
56 y,_ = librosa.load('rec.wav')
57 sr = 22500
58
59 y,_ = librosa.effects.trim(y)
60 MFCC = librosa.feature.mfcc(y=y, sr=sr); MFCC = [np.mean(x)
↪ for x in MFCC]
61 y_harmonic, y_percussive = librosa.effects.hpss(y)
62 y_harmonic, y_percussive = y_harmonic.mean(),
↪ y_percussive.mean()
63 C = librosa.cqt(y); C_mean = [np.mean(x) for x in C]; C_mean =
↪ [complex(x).real for x in C_mean]
64 chroma = librosa.feature.chroma_cqt(C=C, sr=sr); chroma_mean =
↪ [np.mean(x) for x in chroma]
65 a, b = [], []

```

```

66 for j in range(len(chroma_mean)):
67     polar = cmath.polar(complex(chroma_mean[j]));
        ↪ a.append(polar[0]); b.append(polar[1])
68 onset_envelope = librosa.onset.onset_strength(y=y, sr=sr);
        ↪ onsets =
        ↪ librosa.onset.onset_detect(onset_envelope=onset_envelope)
69 onsets = onsets.shape[0]; tempo, beats =
        ↪ librosa.beat.beat_track(onset_envelope=onset_envelope)
70 c_sync = librosa.util.sync(chroma, beats,
        ↪ aggregate=np.median); c_sync = [np.mean(x) for x in
        ↪ c_sync]
71 c, d = [], []
72 for j in range(len(c_sync)):
73     polar = cmath.polar(complex(c_sync[j]));
        ↪ c.append(polar[0]); d.append(polar[1])
74 spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y,
        ↪ sr=sr); spectral_bandwidth = spectral_bandwidth.mean()
75 spectral_rolloff = librosa.feature.spectral_rolloff(y=y,
        ↪ sr=sr)[0]; spectral_rolloff = spectral_rolloff.mean()
76 spectral_centroids = librosa.feature.spectral_centroid(y=y,
        ↪ sr=sr); spectral_centroids = spectral_centroids.mean()
77
78 arr = np.hstack(([], MFCC, y_harmonic, y_percussive, C_mean,
        ↪ a, b, onsets, tempo, beats.shape[0], c,d,
        ↪ spectral_bandwidth, spectral_rolloff,
        ↪ spectral_centroids,1,0,0,0,0))
79
80 """#### **Loading the trained model and predicting emotion of
        ↪ the .wav file** :"""
81
82 import pickle
83 import tensorflow
84
85 emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy',
        ↪ 'Neutral', 'Sad']
86
87 with open('speech_emotion_classifier.pkl','rb') as f:
88     model = pickle.load(f)
89     pred = model.predict(arr.reshape((1,165)))

```



```
90     print(emotions[int(np.where( pred == pred.max() )[1])])
```

1.5 requirements.txt

```
1  absl-py==2.1.0
2  astunparse==1.6.3
3  cachetools==5.3.3
4  certifi==2024.7.4
5  charset-normalizer==3.3.2
6  flatbuffers==24.3.25
7  gast==0.4.0
8  google-auth==2.31.0
9  google-auth-oauthlib==0.4.6
10 google-pasta==0.2.0
11 grpcio==1.64.1
12 h5py==3.11.0
13 idna==3.7
14 keras==2.11.0
15 libclang==18.1.1
16 Markdown==3.6
17 MarkupSafe==2.1.5
18 numpy==2.0.0
19 oauthlib==3.2.2
20 opt-einsum==3.3.0
21 packaging==24.1
22 protobuf==3.19.6
23 pyasn1==0.6.0
24 pyasn1_modules==0.4.0
25 requests==2.32.3
26 requests-oauthlib==2.0.0
27 rsa==4.9
28 six==1.16.0
29 tensorboard==2.11.2
30 tensorboard-data-server==0.6.1
31 tensorboard-plugin-wit==1.8.1
32 tensorflow-estimator==2.11.0
33 tensorflow-macos==2.11.0
34 termcolor==2.4.0
```

```
35 typing_extensions==4.12.2
36 urllib3==2.2.2
37 Werkzeug==3.0.3
38 wrapt==1.16.0
```

1.6 test_keras_model.py

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow.keras.datasets import mnist
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import Dense, Conv2D,
   ↪ MaxPooling2D, Flatten
6 from tensorflow.keras.utils import to_categorical
7
8 # 加载 MNIST 数据集
9 (x_train, y_train), (x_test, y_test) = mnist.load_data()
10
11 # 数据归一化
12 x_train = x_train.astype('float32') / 255.0
13 x_test = x_test.astype('float32') / 255.0
14
15 # 调整输入数据形状
16 x_train = np.expand_dims(x_train, -1)
17 x_test = np.expand_dims(x_test, -1)
18
19 # 将标签转换为 one-hot 编码
20 y_train = to_categorical(y_train, 10)
21 y_test = to_categorical(y_test, 10)
22
23 # 构建一个简单的卷积神经网络模型
24 model = Sequential([
25     Conv2D(32, kernel_size=(3, 3), activation='relu',
   ↪ input_shape=(28, 28, 1)),
26     MaxPooling2D(pool_size=(2, 2)),
27     Conv2D(64, kernel_size=(3, 3), activation='relu'),
28     MaxPooling2D(pool_size=(2, 2)),
29     Flatten(),
```

```
30     Dense(128, activation='relu'),
31     Dense(10, activation='softmax')
32 ])
33
34 # 编译模型
35 model.compile(optimizer='adam',
36               ↪ loss='categorical_crossentropy', metrics=['accuracy'])
37
38 # 训练模型
39 history = model.fit(x_train, y_train, epochs=10,
40                   ↪ batch_size=32, validation_split=0.2)
41
42 # 在测试集上评估模型
43 test_loss, test_acc = model.evaluate(x_test, y_test)
44 print(f'Test accuracy: {test_acc}')
45
46 # 保存模型到文件
47 model.save('mnist_cnn_model.h5')
48
49 # 加载模型
50 from tensorflow.keras.models import load_model
51 loaded_model = load_model('mnist_cnn_model.h5')
52
53 # 在测试集上评估加载的模型
54 test_loss, test_acc = loaded_model.evaluate(x_test, y_test)
55 print(f'Test accuracy of loaded model: {test_acc}')
```

1.7 test__transformer.py

```
1 import tensorflow as tf
2 from tensorflow.keras import layers, Model
3 from tensorflow.keras.datasets import imdb
4 from tensorflow.keras.preprocessing import sequence
5
6 # 加载 IMDB 数据集
7 max_features = 20000 # 词汇表大小
8 max_len = 200 # 序列最大长度
```

```

9
10 (x_train, y_train), (x_test, y_test) =
    ↪ imdb.load_data(num_words=max_features)
11
12 # 将序列填充到固定长度
13 x_train = sequence.pad_sequences(x_train, maxlen=max_len)
14 x_test = sequence.pad_sequences(x_test, maxlen=max_len)
15
16 class PositionalEncoding(layers.Layer):
17     def __init__(self, max_len, d_model):
18         super(PositionalEncoding, self).__init__()
19         self.pos_encoding = self.positional_encoding(max_len,
    ↪ d_model)
20
21     def positional_encoding(self, max_len, d_model):
22         angle_rads = self.get_angles(np.arange(max_len)[:],
    ↪ np.newaxis],
23                                     np.arange(d_model)[np.newaxis],
    ↪ waxis,
    ↪ :],
24                                     d_model)
25         angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
26         angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
27         pos_encoding = angle_rads[np.newaxis, ...]
28         return tf.cast(pos_encoding, dtype=tf.float32)
29
30     def get_angles(self, pos, i, d_model):
31         angle_rates = 1 / np.power(10000, (2 * (i // 2)) /
    ↪ np.float32(d_model))
32         return pos * angle_rates
33
34     def call(self, inputs):
35         return inputs + self.pos_encoding[:,
    ↪ :tf.shape(inputs)[1], :]
36
37 class MultiHeadAttention(layers.Layer):
38     def __init__(self, d_model, num_heads):
39         super(MultiHeadAttention, self).__init__()
40         self.num_heads = num_heads

```

```

41         self.d_model = d_model
42
43         assert d_model % self.num_heads == 0
44
45         self.depth = d_model // self.num_heads
46
47         self.wq = layers.Dense(d_model)
48         self.wk = layers.Dense(d_model)
49         self.wv = layers.Dense(d_model)
50
51         self.dense = layers.Dense(d_model)
52
53     def split_heads(self, x, batch_size):
54         x = tf.reshape(x, (batch_size, -1, self.num_heads,
55                             ↪ self.depth))
56         return tf.transpose(x, perm=[0, 2, 1, 3])
57
58     def call(self, v, k, q, mask):
59         batch_size = tf.shape(q)[0]
60
61         q = self.wq(q)
62         k = self.wk(k)
63         v = self.wv(v)
64
65         q = self.split_heads(q, batch_size)
66         k = self.split_heads(k, batch_size)
67         v = self.split_heads(v, batch_size)
68
69         scaled_attention, _ =
70             ↪ self.scaled_dot_product_attention(q, k, v, mask)
71         scaled_attention = tf.transpose(scaled_attention,
72             ↪ perm=[0, 2, 1, 3])
73         concat_attention = tf.reshape(scaled_attention,
74             ↪ (batch_size, -1, self.d_model))
75         output = self.dense(concat_attention)
76
77         return output
78
79     def scaled_dot_product_attention(self, q, k, v, mask):

```

```

76         matmul_qk = tf.matmul(q, k, transpose_b=True)
77         dk = tf.cast(tf.shape(k)[-1], tf.float32)
78         scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)
79
80         if mask is not None:
81             scaled_attention_logits += (mask * -1e9)
82
83         attention_weights =
84             ↪ tf.nn.softmax(scaled_attention_logits, axis=-1)
85         output = tf.matmul(attention_weights, v)
86
87         return output, attention_weights
88
89     class EncoderLayer(layers.Layer):
90         def __init__(self, d_model, num_heads, dff, rate=0.1):
91             super(EncoderLayer, self).__init__()
92
93             self.mha = MultiHeadAttention(d_model, num_heads)
94             self.ffn =
95                 ↪ self.point_wise_feed_forward_network(d_model, dff)
96
97             self.layernorm1 =
98                 ↪ layers.LayerNormalization(epsilon=1e-6)
99             self.layernorm2 =
100                 ↪ layers.LayerNormalization(epsilon=1e-6)
101
102             self.dropout1 = layers.Dropout(rate)
103             self.dropout2 = layers.Dropout(rate)
104
105         def call(self, x, training, mask):
106             attn_output = self.mha(x, x, x, mask)
107             attn_output = self.dropout1(attn_output,
108                 ↪ training=training)
109             out1 = self.layernorm1(x + attn_output)
110
111             ffn_output = self.ffn(out1)
112             ffn_output = self.dropout2(ffn_output,
113                 ↪ training=training)
114             out2 = self.layernorm2(out1 + ffn_output)

```

```

109
110         return out2
111
112     def point_wise_feed_forward_network(self, d
113
114
115
116     class Encoder(layers.Layer):
117         def __init__(self, num_layers, d_model, num_heads, dff,
118             ↪ input_vocab_size, maximum_position_encoding,
119             ↪ rate=0.1):
120             super(Encoder, self).__init__()
121
122             self.d_model = d_model
123             self.num_layers = num_layers
124
125             self.embedding = layers.Embedding(input_vocab_size,
126             ↪ d_model)
127             self.pos_encoding =
128             ↪ PositionalEncoding(maximum_position_encoding,
129             ↪ self.d_model)
130
131             self.enc_layers = [EncoderLayer(d_model, num_heads,
132             ↪ dff, rate) for _ in range(num_layers)]
133             self.dropout = layers.Dropout(rate)
134
135     def call(self, x, training, mask):
136         seq_len = tf.shape(x)[1]
137
138         x = self.embedding(x)
139         x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
140         x = self.pos_encoding(x)
141
142         x = self.dropout(x, training=training)
143
144         for i in range(self.num_layers):
145             x = self.enc_layers[i](x, training, mask)
146
147         return x

```

```
142
143 num_layers = 4
144 d_model = 128
145 dff = 512
146 num_heads = 8
147 input_vocab_size = max_features
148 maximum_position_encoding = max_len
149 dropout_rate = 0.1
150
151 inputs = layers.Input(shape=(max_len,))
152 enc_padding_mask = None # 可以根据需要创建 mask
153
154 encoder = Encoder(num_layers, d_model, num_heads, dff,
    ↪ input_vocab_size, maximum_position_encoding, dropout_rate)
155 x = encoder(inputs, training=True, mask=enc_padding_mask)
156 x = layers.GlobalAveragePooling1D()(x)
157 x = layers.Dropout(0.1)(x)
158 outputs = layers.Dense(1, activation='sigmoid')(x)
159
160 model = Model(inputs=inputs, outputs=outputs)
161
162 model.compile(optimizer='adam', loss='binary_crossentropy',
    ↪ metrics=['accuracy'])
163
164 # 训练模型
165 history = model.fit(x_train, y_train, epochs=10,
    ↪ batch_size=64, validation_split=0.2)
166
167
168 # 在测试集上评估模型
169 test_loss, test_acc = model.evaluate(x_test, y_test)
170 print(f'Test accuracy: {test_acc}')
171
172 # 保存模型到文件
173 model.save('transformer_text_classification_model.h5')
174
175 # 加载模型
```



```
176 loaded_model = tf.keras.models.load_model('transformer_text_classification_model.h5',  
↳ lassification_model.h5',  
↳ custom_objects={  
177     'PositionalEncoding': PositionalEncoding,  
178     'MultiHeadAttention': MultiHeadAttention,  
179     'EncoderLayer': EncoderLayer,  
180     'Encoder': Encoder  
181 })  
182  
183 # 在测试集上评估加载的模型  
184 test_loss, test_acc = loaded_model.evaluate(x_test, y_test)  
185 print(f'Test accuracy of loaded model: {test_acc}')
```
