Курсовая работа с датасетом ESC-50

Автор: Никитин Кирилл

Датасет ESC-50

Описание:

- ESC-50 — это стандартный датасет для классификации звуков, включающий 50 различных классов, таких как природные звуки, шумы, звуки животных и многое другое.
- Каждый из 50 классов представлен 40 аудиозаписями, всего в датасете 2 000 файлов в формате WAV длительностью 5 секунд.
- Датасет широко используется для обучения и оценки моделей машинного обучения в задачах классификации звуков.

Применение:

- Анализ звуковой среды в реальном времени (например, определение звуков в умных домах).
- Разработка алгоритмов для устройств с голосовым управлением и систем мониторинга.
- Исследования в области акустики и машинного обучения.

Подготовка данных (Dataloader и Dataset)

В рамках сборки Dataset были реализованы:

- ➔ Загрузка аудио файлов из набора ESC-50.
- ➔ Преобразование аудио в моноформат с частотой дискретизации 16,000 Гц.
- ➔ Обеспечивает нормализацию длительности аудиозаписей до 1 секунды (16,000 семплов).
- ➔ Для каждого аудиофайла возвращает тензор аудиоданных, метку класса и имя файла.
- ➔ Данные разбиты на обучающую и валидационную выборки (80/20) с использованием `train_test_split`.

```
class ESCDataset(Dataset):
    def   init  (self, dataset dir, labels_path, transform=None):
        self.dataset dir = dataset dir
        self.labels = pd.read csv(labels_path)
        self.transform = transform

    def   len  (self):
        return len(self.labels)

    def   getitem  (self, idx):
        audio filename = self.labels.iloc[idx, 0]
        audio label = self.labels.iloc[idx, 2]  # Второй столбец содержит метку класса
        audio path = os.path.join(self.dataset dir, audio filename)
        audio data, sampling rate = librosa.load(audio path, sr=16000, mono=True)
        audio data = audio data[:16000] if len(audio_data) > 16000 else np.pad(audio_data, (0,
max(0, 16000 - len(audio data))), "constant")
        if self.transform:
            audio data = self.transform(audio data)
        return torch.tensor(audio data, dtype=torch.float32), torch.tensor(audio_label,
dtype=torch.long), audio_filename


#Загрузка набора данных
ESC50 audio dir = r"C:\Users\kirill.nikitin\Downloads\ESC-50-master\ESC-50-master\audio"  # Путь
к папке с аудиофайлами ESC-50
ESC50 labels path =
r"C:\Users\kirill.nikitin\Downloads\ESC-50-master\ESC-50-master\meta\esc50.csv"  # Путь к мета-
файлу

ESC50_dataset = ESCDataset(ESC50_audio_dir, ESC50_labels_path)

# Разделение данных
train indices, validation_indices = train_test_split(range(len(ESC50_dataset)), test_size=0.2,
random state=42)
train data subset = torch.utils.data.Subset(ESC50 dataset, train indices)
validation_data_subset = torch.utils.data.Subset(ESC50_dataset, validation_indices)

train data loader = DataLoader(train data subset, batch size=16, shuffle=True)
validation_data_loader = DataLoader(validation_data_subset, batch_size=16, shuffle=False)

# Отображение нескольких примеров из dataloader
for batch index, (audio batch, label batch, filenames_batch) in enumerate(train_data_loader):
    print(f"Пакет {batch index + 1}")
    print(f"Форма аудиоданных: {audio_batch.shape}")
    print(f"Метки: {label batch}")
    print(f"Имена файлов: {filenames batch}")
    if batch index == 2:  # Отобразить только первые 3 пакета
        break
```

Статистический анализ

- В датасете содержится **2,000 записей**, каждая из которых соответствует отдельному аудио клипу из набора ESC-50.
- Датасет состоит из **5 атрибутов:**
  - `filename` – имя аудиофайла. (строковый тип)
  - `fold` – номер подмножества для кросс-валидации. (целочисленный)
  - `target` – числовая метка класса. (целочисленный)
  - `category` – текстовая категория звукового события. (строковый тип)
  - `esc10` – флаг, показывающий, относится ли событие к упрощённой версии набора ESC-10. (целочисленный)
- Набор данных сбалансирован: каждая из **50 категорий** представлена равномерно
- Отсутствуют пропущенные значения во всех полях.

```
def dataset statistics(dataframe):
    print("Статистика набора данных:\n")
    print("Всего записей:", len(dataframe))
    print("Поля:", dataframe.columns.tolist())
    print("Типы данных:")
    print(dataframe.dtypes)
    print("Распределение классов:")
    print(dataframe['category'].value_counts(normalize=True) * 100)  # Столбец с метками классов
    print("Пропущенные значения:")
    print(dataframe.isnull().sum())
    record size = dataframe.memory_usage(index=True).sum() / len(dataframe) * 8  # Размер одной
записи в битах
    print(f"Объем одной записи (в битах): {record_size:.2f}")


labels dataframe = pd.read csv(ESC50 labels_path)
dataset_statistics(labels_dataframe)

# Создание словаря меток
label to category map = labels dataframe[['target',
'category']].drop_duplicates().set_index('target')['category'].to_dict()
```

Алгоритм машинного обучения для классификации аудио

- ➔ Основная задача для данного пункта – **классификация аудиоклипов** по 50 категориям, представленным в наборе данных ESC-50. Каждая категория соответствует уникальному типу звуковых событий, таких как лай собаки, пение птиц или шум дождя. Эта задача решается с использованием **нейронной сети прямого распространения**, построенный на базе **PyTorch**.

- ➔ **Архитектура модели:**
  - Входной слой принимает одномерный вектор длиной **16,000** (преобразованная форма аудиосигнала).
  - Два скрытых слоя с размерностями 128 и 64, каждый из которых использует функцию активации **ReLU**.
  - Выходной слой с 50 нейронами, каждый из которых соответствует одной категории звукового события.
- ➔ Для обучения модели используется функция потерь **CrossEntropyLoss**
- ➔ Набор данных ESC-50 разделён на обучающую и валидационную выборки в пропорции 80:20.
- ➔ Модель обучается на протяжении **20 эпох** с использованием оптимизатора **Adam**.
- ➔ По окончании обучения была построена диаграмма, показывающая уменьшение величины потерь по эпохам.

```
class SimpleAudioClassifier(nn.Module):
    def   init  (self):
        super(SimpleAudioClassifier, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(16000, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 50)  # 50 классов в ESC-50
        )

    def forward(self, x):
        return self.fc(x)

def train model(classifier model, training_loader, validation_loader, epochs=20):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(classifier_model.parameters(), lr=0.001)
    losses = []

    for epoch in range(epochs):
        classifier model.train()
        total loss = 0
        for audio batch, label batch,   in training loader:
            predictions = classifier model(audio batch)
            loss = criterion(predictions, label_batch)
            optimizer.zero grad()
            loss.backward()
            optimizer.step()
            total loss += loss.item()
        average loss = total loss / len(training_loader)
        losses.append(average loss)
        print(f"Эпоха {epoch + 1}, Потеря: {average_loss:.4f}")

    # Построение графика потерь
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, epochs + 1), losses, marker='o', label='Потери на обучении')
    plt.title("График потерь от эпох")
    plt.xlabel("Эпоха")
    plt.ylabel("Потеря")
    plt.legend()
    plt.grid()
    plt.show()

classifier model instance = SimpleAudioClassifier()
train_model(classifier_model_instance, train_data_loader, validation_data_loader)
```

Инференс

- После обучения модель была протестирована на нескольких аудио клипах из набора данных. Процедура включала:
  - Прогон аудиоклипа через модель.
  - Определение наиболее вероятной категории на основе выходных данных модели.
  - Сопоставление числовой метки класса текстовой категории звука.

```
def inference(classifier model, audio_sample, sample_filename, actual_label):
    classifier model.eval()
    with torch.no grad():
        prediction scores = classifier model(audio sample.unsqueeze(0))
        predicted label = torch.argmax(prediction scores, dim=1).item()
    predicted category = label to category map.get(predicted label, "Неизвестно")
    actual_category = label_to_category_map.get(actual_label, "Неизвестно")

    print(f"Название аудиоклипа: {sample filename}")
    print(f"Предсказанная категория: {predicted_category}")

    return predicted_category

example audio, example label, example filename = ESC50 dataset[0]
predicted category_output = inference(classifier_model_instance, example_audio, example_filename,
example label)
example audio, example label, example filename = ESC50_dataset[1]
predicted_category_output = inference(class
```

Кластеризация и понижение размерности

Для работы с многомерными данными важно уменьшить их размерность для визуализации и анализа.

- ➔ Из каждого аудиофайла вычислялись **13 MFCC (Mel-frequency Cepstral Coefficients)**. Эти коэффициенты представляют собой компактное представление аудиоклипа.
- ➔ Они усреднялись по временной оси, чтобы получить вектор фиксированной длины для каждого файла.
- ➔ Для улучшения работы алгоритмов применялась стандартизация с помощью **StandardScaler**, чтобы привести данные к единому масштабу.
- ➔ Использовался метод **PCA (Principal Component Analysis)**, чтобы снизить размерность данных до двух главных компонент.
- ➔ Применялся алгоритм **K-Means** с заданным числом кластеров (`n_clusters=5`), который группирует данные в кластеры на основе их сходства.

```
def clustering and dim reduction(audio_dir, n_clusters=5):
    audio features = []
    audio filenames = []
    for audio filename in os.listdir(audio dir):
        if audio filename.endswith(".wav"):
            audio data, sampling rate = librosa.load(os.path.join(audio_dir,
audio filename), sr=16000, mono=True)
            mfcc features = librosa.feature.mfcc(y=audio data, sr=16000, n_mfcc=13)
            audio features.append(np.mean(mfcc features, axis=1))
            audio_filenames.append(audio_filename)

    audio features = np.array(audio features)
    feature scaler = StandardScaler()
    scaled_features = feature_scaler.fit_transform(audio_features)

    # Уменьшение размерности
    pca model = PCA(n components=2)
    reduced_features = pca_model.fit_transform(scaled_features)

    # Кластеризация
    kmeans model = KMeans(n clusters=n clusters, random state=42)
    cluster_assignments = kmeans_model.fit_predict(reduced_features)

    # Создание словаря для отображения кластеров в легенду
    cluster labels = [f"Кластер {i}" for i in range(n clusters)]
    color palette = plt.cm.viridis(np.linspace(0, 1, n clusters))
    cluster_color_mapping = {i: color for i, color in enumerate(color_palette)}

    # Визуализация
    plt.figure(figsize=(10, 7))
    for cluster index in range(n clusters):
        cluster points = reduced features[cluster assignments == cluster_index]
        plt.scatter(cluster points[:, 0], cluster points[:, 1],
color=cluster_color_mapping[cluster_index], label=f"Кластер {cluster_index}")

    plt.title("Кластеризация аудиоклипов")
    plt.xlabel("Главный компонент 1")
    plt.ylabel("Главный компонент 2")
    plt.legend()
    plt.show()

clustering_and_dim_reduction(ESC50_audio_dir)
```

Обнаружение выбросов

Методы обнаружения выбросов:

- ➔ Алгоритм **Isolation Forest** основывается на принципе изоляции аномальных точек. Он случайным образом строит деревья, чтобы определить, насколько сложно изолировать каждую точку. Чем проще точка изолируется, тем выше вероятность, что это выброс.
- ➔ **Local Outlier Factor** использовался для подтверждения обнаруженных аномалий.

```
def detect audio outliers(audio_dir):
    audio features = []
    for audio filename in os.listdir(audio dir):
        if audio filename.endswith(".wav"):
            audio data, sampling rate = librosa.load(os.path.join(audio_dir,
audio filename), sr=16000, mono=True)
            mfcc features = librosa.feature.mfcc(y=audio data, sr=16000, n_mfcc=13)
            audio_features.append(np.mean(mfcc_features, axis=1))

    audio features = np.array(audio features)
    feature scaler = StandardScaler()
    scaled_features = feature_scaler.fit_transform(audio_features)

    # Использование IsolationForest для поиска выбросов
    isolation forest model = IsolationForest(contamination=0.05, random state=42)
    isolation_outlier_predictions = isolation_forest_model.fit_predict(scaled_features)

    # Использование LocalOutlierFactor для подтверждения выбросов
    local outlier factor model = LocalOutlierFactor(n neighbors=20, contamination=0.05)
    lof_outlier_predictions = local_outlier_factor_model.fit_predict(scaled_features)

    isolation outliers = np.where(isolation outlier predictions == -1)[0]
    confirmed outliers = np.where((isolation_outlier_predictions == -1) &
(lof_outlier_predictions == -1))[0]

    print(f"Обнаружено выбросов (Isolation Forest): {len(isolation outliers)}")
    print(f"Подтверждено выбросов (LOF): {len(confirmed_outliers)}")

    for outlier index in confirmed outliers:
        print(f"Выброс: {os.listdir(audio_dir)[outlier_index]}")

detect_audio_outliers(ESC50_audio_dir)
```

Примеры использования

- Системы умного дома или безопасности:
  - распознавание звуков, таких как плач ребёнка, сирена, звон стекла или шаги.

    Преимущества:

    - Помогает автоматически реагировать на важные события.
    - Упрощает взаимодействие пользователя с устройством.

    Недостатки:

    - Может не справляться с новыми или редкими звуками, которых не было в обучающих данных.
    - Шумы или помехи в окружающей среде снижают точность.

Примеры использования

- Мониторинг оборудования в промышленности:
  - анализ звуков работы машин, чтобы вовремя находить поломки.

Преимущества:

- Позволяет предотвратить поломки, что экономит деньги и время.
- Автоматически сигнализирует о проблемах.

Недостатки:

- Требуется адаптация к звукам конкретного оборудования.

Примеры использования

➔ Диагностика заболеваний по звукам в медицинских приложениях.

Преимущества:

- Помогает быстро обнаружить потенциальные проблемы со здоровьем.

Недостатки:

- Высокие требования к качеству и объёму данных для обучения.
- Не заменяет полноценную диагностику врачом.

Примечание

➔ Импортируемые библиотеки

```
import os
import numpy as np
import pandas as pd
import librosa
import librosa.display
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
```

➔ Результаты выполнения кода

```
Пакет 1
Форма аудиоданных: torch.Size([16, 16000])
Метки: tensor([28, 43, 12, 49, 13, 30, 49, 17, 43, 26, 39, 42, 37, 37, 24, 30])
Имена файлов: ('1-39937-A-28.wav', '4-175846-A-43.wav', '5-215658-A-12.wav', '3-102583-A-49.wav',
'2-85139-A-13.wav', '4-186518-A-30.wav', '4-251645-B-49.wav', '2-126433-A-17.wav', '1-19026-A-43.wav',
'3-126113-A-26.wav', '4-204123-A-39.wav', '2-70052-B-42.wav', '4-209536-A-37.wav', '1-34853-A-37.wav',
'3-151213-A-24.wav', '5-218980-A-30.wav')
...
Всего записей: 2000
Поля: ['filename', 'fold', 'target', 'category', 'esc10', 'src_file', 'take']
Типы данных:
filename    object
...
take        object
dtype: object
Распределение классов:
category
dog                 2.0
...
crickets            2.0
Name: proportion, dtype: float64
Пропущенные значения:
filename    0
...
take        0
dtype: int64
Объем одной записи (в битах): 392.53
Эпоха 1, Потеря: 3.9042
...
Эпоха 20, Потеря: 0.0859
Название аудиоклипа: 1-100032-A-0.wav
Предсказанная категория: mouse_click
Название аудиоклипа: 1-100038-A-14.wav
Предсказанная категория: chirping_birds
Обнаружено выбросов (Isolation Forest): 100
Подтверждено выбросов (LOF): 45
Выброс: 1-260640-C-2.wav
...
Выброс: 5-243449-A-14.wav
```