Week_4_Data_Preprocessing

September 29, 2024

[1]:
```
import numpy as np
import pandas as pd
import h5py

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.utils.class_weight import compute_class_weight

import torch
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import io
```

0.1 Load data

[2]:
```
import pandas as pd
train = pd.read_csv('../data/processed/train-metadata.csv')
validation = pd.read_csv('../data/processed/validation-metadata.csv')
test = pd.read_csv('../data/processed/test-metadata.csv')
train.head()
```

[2]:
```
        isic_id  age_approx   sex anatom_site_general  clin_size_long_diam_mm  \
0  ISIC_5622764        65.0  male     posterior torso                    7.79
1  ISIC_8296521        50.0  male     posterior torso                    6.90
2  ISIC_1262887        65.0  male     posterior torso                    6.98
3  ISIC_9555133        55.0  male     lower extremity                    9.50
4  ISIC_2851454        60.0  male     posterior torso                    3.97

   target
0       0
1       0
2       0
3       0
```

```
4       0
```

0.2 Handle data imbalance in training set

[3]:
```python
import pandas as pd

# Assuming 'train' is your DataFrame with the target column 'target'

try:
    # Print class distribution before sampling
    print("Class Distribution Before Sampling (%):")
    display(train.target.value_counts(normalize=True) * 100)

    # Check if the 'target' column exists in the DataFrame
    if 'target' not in train.columns:
        raise KeyError("The 'target' column is not found in the DataFrame.")

    # Sampling process
    try:
        # Sample the majority class (0) with a fraction of 0.01
        majority_df = train.query("target == 0").sample(frac=0.01,␣
 ↪random_state=42)  # Fixed random seed for reproducibility

        # Sample the minority class (1) with a factor of 5.0, allowing␣
 ↪replacement
        minority_df = train.query("target == 1").sample(frac=5.0, replace=True,␣
 ↪random_state=42)

        # Combine the sampled data into a new balanced DataFrame
        train_balanced = pd.concat([majority_df, minority_df], axis=0).
 ↪sample(frac=1.0, random_state=42)  # Shuffle the combined DataFrame
    except ValueError as e:
        raise ValueError(f"Error during sampling: {e}")

    # Print class distribution after sampling
    print("\nClass Distribution After Sampling (%):")
    display(train_balanced.target.value_counts(normalize=True) * 100)

except Exception as e:
    print(f"An error occurred: {e}")
```

```
Class Distribution Before Sampling (%):

target
0    99.902045
1     0.097955
Name: proportion, dtype: float64
```

```
Class Distribution After Sampling (%):

target
0    67.105263
1    32.894737
Name: proportion, dtype: float64
```

[4]:
```
# Set class weight for training purpose
class_weights = compute_class_weight('balanced', classes=np.
 ↪unique(train['target']), y=train['target'])
class_weights = torch.tensor(class_weights,dtype=torch.float)
print("Class Weights:", class_weights)
```

```
Class Weights: tensor([5.0049e-01, 5.1044e+02])
```

0.3 Process Data

[5]:
```
# Custom transformer for handling missing values
class MissingValueHandler(BaseEstimator, TransformerMixin):
    # Fit method, not modifying any parameters, just returning self
    def fit(self, X, y=None):
        return self

    # Transform method to handle missing values
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Identify categorical columns
        cat_cols = X.select_dtypes(include=['object', 'category']).columns

        # Create imputer for numerical data using median
        num_imputer = SimpleImputer(strategy="median")
        # Apply imputer to numerical columns
        X[num_cols] = num_imputer.fit_transform(X[num_cols])

        # Create imputer for categorical data using the most frequent value
        cat_imputer = SimpleImputer(strategy="most_frequent")
        # Apply imputer to categorical columns
        X[cat_cols] = cat_imputer.fit_transform(X[cat_cols])

        return X  # Return the transformed DataFrame
```

```python
# Custom transformer for one-hot encoding
class OneHotEncoderTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Initialize the OneHotEncoder with specified parameters
        self.encoder = OneHotEncoder(sparse_output=False,␣
 ↪handle_unknown="ignore")

    # Fit method to learn the categories for encoding
    def fit(self, X, y=None):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Fit the encoder to categorical columns
        self.encoder.fit(X.select_dtypes(include=['object', 'category']))
        return self

    # Transform method to apply one-hot encoding
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Transform categorical columns to one-hot encoding
        encoded_cols = self.encoder.transform(X.
 ↪select_dtypes(include=['object', 'category']))
        # Get the new column names after encoding
        new_columns = self.encoder.get_feature_names_out(X.
 ↪select_dtypes(include=['object', 'category']).columns)

        # Create a DataFrame for the encoded columns
        encode_df = pd.DataFrame(encoded_cols, columns=new_columns, index=X.
 ↪index)
        # Concatenate the original DataFrame (excluding categorical columns)␣
 ↪with the encoded DataFrame
        return pd.concat([X.select_dtypes(exclude=['object', 'category']),␣
 ↪encode_df], axis=1)

# Custom transformer for scaling numerical features
class NumericalScaler(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Initialize the StandardScaler for scaling numerical features
        self.scaler = StandardScaler()

    # Fit method to learn the scaling parameters
    def fit(self, X, y=None):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Fit the scaler to the numerical columns
        self.scaler.fit(X[num_cols])
        return self

    # Transform method to apply scaling
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Apply scaling to the numerical columns
        X[num_cols] = self.scaler.transform(X[num_cols])
        return X  # Return the scaled DataFrame

# Custom transformer for handling age approximation
class AgeApproxTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self  # No fitting required for this transformer

    # Transform method to round age approximations
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Check if 'age_approx' is in the DataFrame
        if 'age_approx' in X.columns:
            # Round the age and convert to integer type
            X['age_approx'] = X['age_approx'].round().astype('Int64')
        return X  # Return the transformed DataFrame

# Create the complete pipeline for preprocessing
def process_pipeline() -> Pipeline:
    # Define a pipeline with the specified transformers
    pipeline = Pipeline(steps=[
        ('age_transformer', AgeApproxTransformer()),  # Age approximation
        ('missing_value_handler', MissingValueHandler()),  # Handling missing␣
 ↪values
        ('num_scaler', NumericalScaler()),  # Scaling numerical features
        ('cat_encoder', OneHotEncoderTransformer())  # One-hot encoding␣
 ↪categorical features
    ])
    return pipeline  # Return the constructed pipeline
```

0.4 Transform train, validation and test data

[7]:
```
#seperate case id and target variable from dependable variables
X_train = train_balanced.drop(columns=['isic_id','target'])
temp_train = train_balanced[['target','isic_id']]
pipeline = process_pipeline()
train_processed_df = pd.concat([pipeline.
 ↪fit_transform(X_train),temp_train],axis=1)

# Process validation data
X_validation = validation.drop(columns=['isic_id', 'target'])
temp_validation = validation[['target', 'isic_id']]
validation_processed_df = pd.concat([pipeline.transform(X_validation),␣
 ↪temp_validation], axis=1)

# Process test data
X_test = test.drop(columns=['isic_id', 'target'])
temp_test = test[['target', 'isic_id']]
test_processed_df = pd.concat([pipeline.transform(X_test), temp_test], axis=1)

# Save the processed dataframes
train_processed_df.to_csv('../data/processed/processed-train-metadata.csv',␣
 ↪index=False)
validation_processed_df.to_csv('../data/processed/processed-validation-metadata.
 ↪csv', index=False)
test_processed_df.to_csv('../data/processed/processed-test-metadata.csv',␣
 ↪index=False)
```

[8]:
```
# Custom Dataset class for loading images from HDF5 files
class HDF5ImageDataset(Dataset):
    def __init__(self, hdf5_file, csv_file, transform=None):
        # Open the HDF5 file with error handling
        try:
            self.hdf5_file = h5py.File(hdf5_file, 'r')  # Read-only mode
        except Exception as e:
            raise IOError(f"Could not open HDF5 file: {hdf5_file}. Error: {e}")

        # Read the CSV file containing image labels and IDs
        try:
            self.labels_df = pd.read_csv(csv_file)
        except Exception as e:
            raise IOError(f"Could not read CSV file: {csv_file}. Error: {e}")
        # Ensure that all image IDs from the CSV are present in the HDF5 file
        self.image_ids = self.labels_df['isic_id'].values
        for image_id in self.image_ids:
            if str(image_id) not in self.hdf5_file.keys():
                raise ValueError(f"Image id {image_id} not found in HDF5 file.")

        # Store any transformations to be applied to the images
        self.transform = transform

    def __len__(self):
        # Return the total number of samples in the dataset
        return len(self.labels_df)

    def __getitem__(self, idx):
        # Get the image ID from the CSV file based on index
        image_id = str(self.labels_df.iloc[idx]['isic_id'])

        # Load the image data from the HDF5 file
        try:
            image_bytes = self.hdf5_file[image_id][()]  # Read image bytes
        except KeyError:
            raise KeyError(f"Image id {image_id} not found in HDF5 file during␣
↪__getitem__.")
        except Exception as e:
            raise RuntimeError(f"Error loading image id {image_id} from HDF5␣
↪file. Error: {e}")

        # Convert the image bytes to a PIL Image
        try:
            image = Image.open(io.BytesIO(image_bytes))  # Use io.BytesIO to␣
↪handle bytes
        except Exception as e:
            raise RuntimeError(f"Could not convert image bytes to PIL Image.␣
↪Error: {e}")

        # Apply any specified transformations to the image
        if self.transform:
            image = self.transform(image)

        # Retrieve the corresponding label for the image
        label = self.labels_df.iloc[idx]['target']

        return image, label  # Return the image and its label
```

[9]:
```
# Usage
if __name__ == "__main__":
    # Define paths to the HDF5 and CSV files
    hdf5_file = '../data/raw/train-image.hdf5'
    csv_file = '../data/processed/processed-train-metadata.csv'

    # Optionally define any transformations (e.g., resize, normalize, etc.)
    from torchvision import transforms
    transform = transforms.Compose([
        transforms.Resize((128, 128)),  # Example resize
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x / 255.0),
    ])

    # Create Dataset instance
    dataset = HDF5ImageDataset(hdf5_file=hdf5_file, csv_file=csv_file,␣
 ↪transform=transform)

    # Create DataLoader
    dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[11]:
```
#check if the class works
for images, labels in dataloader:
    print(f"Image batch shape: {images.shape}, Labels batch shape: {labels.
 ↪shape}")  # Print shapes
    break  # Remove this line to iterate through the entire DataLoader
```

```
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
```

```
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
Image batch shape: torch.Size([32, 3, 128, 128]), Labels batch shape:
torch.Size([32])
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[11], line 2
      1 #check if the class works
----> 2 for images, labels in dataloader:
      3     print(f"Image batch shape: {images.shape}, Labels batch shape:␣
 ↪{labels.shape}")  # Print shapes
```

```
File ~/.local/lib/python3.10/site-packages/torch/utils/data/dataloader.py:630,␣
↪in _BaseDataLoaderIter.__next__(self)
    627 if self._sampler_iter is None:
    628     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    629     self._reset()  # type: ignore[call-arg]
--> 630 data = self._next_data()
    631 self._num_yielded += 1
    632 if self._dataset_kind == _DatasetKind.Iterable and \
    633         self._IterableDataset_len_called is not None and \
    634         self._num_yielded > self._IterableDataset_len_called:

File ~/.local/lib/python3.10/site-packages/torch/utils/data/dataloader.py:673,␣
↪in _SingleProcessDataLoaderIter._next_data(self)
    671 def _next_data(self):
    672     index = self._next_index()  # may raise StopIteration
--> 673     data = self._dataset_fetcher.fetch(index)  # may raise StopIteratio
    674     if self._pin_memory:
    675         data = _utils.pin_memory.pin_memory(data, self.
↪_pin_memory_device)

File ~/.local/lib/python3.10/site-packages/torch/utils/data/_utils/fetch.py:52,
↪in _MapDatasetFetcher.fetch(self, possibly_batched_index)
     50         data = self.dataset.__getitems__(possibly_batched_index)
     51     else:
---> 52         data = [self.dataset[idx] for idx in possibly_batched_index]
     53 else:
     54     data = self.dataset[possibly_batched_index]

File ~/.local/lib/python3.10/site-packages/torch/utils/data/_utils/fetch.py:52,
↪in <listcomp>(.0)
     50         data = self.dataset.__getitems__(possibly_batched_index)
     51     else:
---> 52         data = [self.dataset[idx] for idx in possibly_batched_index]
     53 else:
     54     data = self.dataset[possibly_batched_index]

Cell In[8], line 35, in HDF5ImageDataset.__getitem__(self, idx)
     33 # Load the image data from the HDF5 file
     34 try:
---> 35     image_bytes = self.hdf5_file[image_id][()]  # Read image bytes
     36 except KeyError:
     37     raise KeyError(f"Image id {image_id} not found in HDF5 file during␣
↪__getitem__.")

File h5py/_objects.pyx:54, in h5py._objects.with_phil.wrapper()

File h5py/_objects.pyx:55, in h5py._objects.with_phil.wrapper()
File /opt/tljh/user/lib/python3.10/site-packages/h5py/_hl/dataset.py:823, in Dataset.__getitem__(self, args, new_dtype)
    821     arr = numpy.zeros(selection.mshape, dtype=new_dtype)
    822 for mspace, fspace in selection:
--> 823     self.id.read(mspace, fspace, arr, mtype)
    824 if selection.mshape is None:
    825     return arr[()]

KeyboardInterrupt:
```