01-data-preprocessing

June 10, 2024

Import libraries

[ ]:
```
import os
import time
import pickle
import pandas as pd
import pyarrow.parquet as pa
from sklearn.feature_extraction import DictVectorizer
```

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

Start time

[ ]:
```
start_time = time.time()
```

Setting path to the data directory

[ ]:
```
CURRENT_DIRECTORY = os.getcwd()
PARENT_DIRECTORY = os.path.dirname(CURRENT_DIRECTORY)
DATA_PATH = os.path.join(PARENT_DIRECTORY, '_data')
PICKLE_PATH = os.path.join(CURRENT_DIRECTORY, '_pickle')
DEST_PATH = os.path.join(CURRENT_DIRECTORY, 'DEST_PATH')  # Create a specific␣
↪directory
```

Ensure the DEST_PATH directory exists

[ ]:
```
if not os.path.exists(DEST_PATH):
    print(f"Creating directory: {DEST_PATH}")
    os.makedirs(DEST_PATH)
else:
    print(f"Directory already exists: {DEST_PATH}")
```

[ ]:
```
vectorise = DictVectorizer()
```

Notes: 1. We shall use the code of Data Pre-processing written for Week-01. 2. Here we are using Yellow taxi data of January, February, and March months. 3. train => January, validation => February, test => March.

[ ]:
```
def path_join(train, val, test):
    """
    Join the paths for the train, validation, and test datasets.
    Args:
    train (str): Filename for the train dataset.
    val (str): Filename for the validation dataset.
    test (str): Filename for the test dataset.
    Returns:
    list: List containing the full paths for the train, validation, and test␣
 ↪datasets.
    """
    train_data_path = os.path.join(DATA_PATH, train)
    val_data_path = os.path.join(DATA_PATH, val)
    test_data_path = os.path.join(DATA_PATH, test)
    return [train_data_path, val_data_path, test_data_path]
```

[ ]:
```
def read_data(data):
    """
    Read the data from a file and return it as a pandas DataFrame.
    Args:
    data (str): Path to the data file.
    Returns:
    pd.DataFrame: DataFrame containing the data.
    """
    if data.endswith('.parquet'):
        data = pa.read_table(data)
        df = data.to_pandas()  # Converting to pandas DataFrame
        df.columns = df.columns.str.lower()
        return df
    elif data.endswith('.csv'):
        df = pd.read_csv(data)
        df.columns = df.columns.str.lower()
        return df
    else:
        return 'Not valid format'
```

[ ]:
```
def save_pickle(obj, filename: str):
    """
    Save an object to a pickle file.
    Args:
    obj: Object to be saved.
    filename (str): Name of the file where the object will be saved.
    """
    with open(filename, "wb") as f_out:
        return pickle.dump(obj, f_out)
```

[ ]:
```python
def calculate_duration(data):
    """
    Calculate the duration of each trip in minutes.
    Args:
    data (pd.DataFrame): DataFrame containing the trip data.
    Returns:
    pd.DataFrame: DataFrame with an added 'duration' column.
    """
    data['duration'] = pd.to_datetime(data['lpep_dropoff_datetime']) - pd.
 ↪to_datetime(data['lpep_pickup_datetime'])
    data['duration'] = data['duration'].dt.total_seconds() / 60  # Convert␣
 ↪seconds to minutes
    return data
```

[ ]:
```python
def outliers(data):
    """
    Filter out trips with durations outside the range [1, 60] minutes.
    Args:
    data (pd.DataFrame): DataFrame containing the trip data.
    Returns:
    pd.DataFrame: DataFrame with outliers removed.
    """
    data_outliers = data[(data['duration'] >= 1) & (data['duration'] <= 60)]
    data_outliers['pulocationid'] = data_outliers['pulocationid'].astype(str)
    data_outliers['dolocationid'] = data_outliers['dolocationid'].astype(str)
    return data_outliers
```

[ ]:
```python
def convert_to_dict(data_outliers):
    """
    Convert the DataFrame to a list of dictionaries for vectorization.
    Args:
    data_outliers (pd.DataFrame): DataFrame containing the filtered data.
    Returns:
    list: List of dictionaries representing the data.
    """
    return data_outliers[['pulocationid', 'dolocationid', 'trip_distance']].
 ↪to_dict(orient='records')
```

[ ]:
```python
def fit_transform_(df_dict):
    """
    Fit and transform the data using DictVectorizer.
    Args:
    df_dict (list): List of dictionaries representing the data.
    Returns:
    scipy.sparse.csr_matrix: Transformed data.
    """
    return vectorise.fit_transform(df_dict)
```

[ ]:
```
def fit_(df_dict):
    """
    Transform the data using an already fitted DictVectorizer.
    Args:
    df_dict (list): List of dictionaries representing the data.
    Returns:
    scipy.sparse.csr_matrix: Transformed data.
    """
    return vectorise.transform(df_dict)
```

[ ]:
```
def pre_processing(data, choice):
    """
    Pre-process the data by calculating duration, removing outliers, and␣
 ↪vectorizing the data.
    Args:
    data (pd.DataFrame): DataFrame containing the trip data.
    choice (int): Choice for vectorization (0 for training data, 1 for␣
 ↪validation/test data).
    Returns:
    tuple: Tuple containing the vectorized data and the DataFrame with outliers␣
 ↪removed.
    """
    data = calculate_duration(data)
    data_outliers = outliers(data)
    df_dict = convert_to_dict(data_outliers)
    if choice == 0:
        X_train = fit_transform_(df_dict)
        return X_train, data_outliers
    elif choice == 1:
        X_val = fit_(df_dict)
        return X_val, data_outliers
    else:
        return 'Enter Choice 0 or 1'
```

[ ]:
```
def main(train, val, test):
    """
    Main function to execute the data pre-processing pipeline.
    Args:
    train (str): Filename for the train dataset.
    val (str): Filename for the validation dataset.
    test (str): Filename for the test dataset.
    """
    data_path_files = path_join(train, val, test)
    df_train = read_data(data_path_files[0])  # Read January data
    df_val = read_data(data_path_files[1])  # Read February data
    df_test = read_data(data_path_files[2])  # Read March data
    X_train, df_train = pre_processing(df_train, choice=0)
```

```
    X_val, df_val = pre_processing(df_val, choice=1)
    X_test, df_test = pre_processing(df_test, choice=1)
    y_train = df_train['duration']
    y_val = df_val['duration']
    y_test = df_test['duration']

    # Save DictVectorizer and datasets
    save_pickle(vectorise, os.path.join(DEST_PATH, "vectorise.pkl"))
    save_pickle((X_train, y_train), os.path.join(DEST_PATH, "train.pkl"))
    save_pickle((X_val, y_val), os.path.join(DEST_PATH, "val.pkl"))
    save_pickle((X_test, y_test), os.path.join(DEST_PATH, "test.pkl"))
```

[ ]:
```
if __name__ == '__main__':
    # File Names
    january_file_name = 'green_tripdata_2023-01.parquet'
    february_file_name = 'green_tripdata_2023-02.parquet'
    march_file_name = 'green_tripdata_2023-03.parquet'
    main(january_file_name, february_file_name, march_file_name)

    # End time
    end_time = time.time()
    print(f"Total time taken to run the script: {end_time - start_time}␣
 ↪seconds")
```