03-hypo

June 10, 2024

[ ]:
```
import os, pickle, mlflow, logging
import numpy as np
```

[ ]:
```
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
from hyperopt.pyll import scope
```

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Configure logging

[ ]:
```
logging.basicConfig(filename='logs/hypo.log', level=logging.INFO)
```

[ ]:
```
mlflow.set_tracking_uri("http://127.0.0.1:5000")
mlflow.set_experiment("random-forest-hyperopt")
```

Define functions

[ ]:
```
def load_pickle(fileName: str):
    """
    Load data from a pickle file.
    Args:
    fileName (str): Path to the pickle file.
    Returns:
    object: Data loaded from the pickle file.
    """
    try:
        with open(fileName, 'rb') as f:
            return pickle.load(f)
    except FileNotFoundError:
        logging.error(f"Error: File '{fileName}' not found.")
        return None
```

[ ]:
```
def optimisation_(Data_path: str = 'DEST_PATH', num_trails = int):

    # Load training and validation data
    X_train, y_train = load_pickle(os.path.join(Data_path, 'train.pkl'))
    X_val, y_val = load_pickle(os.path.join(Data_path, 'val.pkl'))
```

```
    # Convert target variables to numpy arrays
    y_train = y_train.to_numpy()
    y_val = y_val.to_numpy()
    def objective(params):
        # Start MLflow run
        with mlflow.start_run():
            logging.info("Training random forest regressor model...")
            mlflow.log_params(params)
            # Initialize and train random forest regressor model
            rf = RandomForestRegressor(**params)
            rf.fit(X_train, y_train)
            y_pred = rf.predict(X_val)

            # Calculate root mean square error
            rmse = mean_squared_error(y_val, y_pred, squared=False)
            mlflow.log_metric("rmse", rmse)
            logging.info(f'Root Mean Square Error = {rmse}')
        return {'loss':rmse, 'status':STATUS_OK}
    search_space = {
        'max_depth' : scope.int(hp.quniform('max_dept', 1,20,1)),
        'n_estimators': scope.int(hp.quniform('n_estimator', 10,50,1)),
        'min_samples_split': scope.int(hp.quniform('min_samples_split',␣
↪2,10,1)),
        'random_state':42
    }
    rstate = np.random.default_rng(42)  # For Reproducable Results
    fmin(
        fn=objective,
        space=search_space,
        algo=tpe.suggest,
        max_evals=num_trails,
        trials=Trials(),
        rstate=rstate
    )
```

Entry point of the script

[ ]:
```
if __name__ == '__main__':
    # Set the path to the data directory
    CURRENT_DIRECTORY = os.getcwd()
    DEST_PATH = os.path.join(CURRENT_DIRECTORY, 'DEST_PATH')

    # Train the model using the data in DEST_PATH
    optimisation_(DEST_PATH, num_trails=30)
```