Automated Hyperparameter Optimization Project Report

Introduction

Hyperparameter optimization is crucial in machine learning as it significantly influences model performance. This project focuses on automating the hyperparameter optimization process using Bayesian optimization, Randomized Search, and a custom Hyperopt-like algorithm. Our architecture integrates these optimization techniques seamlessly with various machine learning models to handle different data types efficiently.

Specifications:

- **Programming Language**: Python
- **Libraries**: scikit-learn, pandas, numpy, matplotlib
- **Optimization Techniques**: Bayesian optimization, Randomized Search, Hyperopt-like optimization
- **Models**: Multiple machine learning models integrated through the `models.py` script
- **Dataset**: Dataset was too large to upload, a link is provided instead

Project Details

Data Preprocessing

Data preprocessing is crucial for model performance. We start by handling missing values, scaling numerical features, and encoding categorical variables.

Snippet:

```
def load_and_preprocess_data():
    data_path = 'data/dataset.csv'
    data = pd.read_csv(data_path)

    num_feats = ['CNT_CHILDREN', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START']
    cat_feats = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE',
                 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
'NAME_HOUSING_TYPE', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_6',
                 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY', 'REGION_RATING_CLIENT',
                 'REGION_RATING_CLIENT_W_CITY']

    num_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())
    ])

    cat_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant',
fill_value='Missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
```

```
            ('num', num_transformer, num_feats),
            ('cat', cat_transformer, cat_feats)
        ]
    )

    X = data[num_feats + cat_feats]
    y = data['TARGET']
    X_preprocessed = preprocessor.fit_transform(X)

    return train_test_split(X_preprocessed, y, test_size=0.2,
random_state=42)
```

Purpose: This code ensures that the dataset is cleaned and properly formatted for model training.

Hyperparameter Optimization

The core of this project lies in implementing and comparing different hyperparameter optimization techniques.

1. **Randomized Search**:
    - **Implementation**: Randomly samples hyperparameter combinations.
    - **Purpose**: Provides a baseline for hyperparameter optimization.
2. **Bayesian Optimization**:
    - **Implementation**: Uses probabilistic models to predict the performance of hyperparameter combinations.
    - **Purpose**: Efficiently explores the hyperparameter space to find optimal configurations.
3. **Hyperopt-like Optimization**:
    - **Implementation**: Custom algorithm inspired by Hyperopt.
    - **Purpose**: Further explore and validate alternative optimization methods.

Evaluation: After training the models with optimized hyperparameters, we evaluate their performance using ROC AUC scores and visualize the results using ROC curves.

Snippet:

```
def evaluate_models(models, X_test, y_test):
    for name, model in models.items():
        y_pred = model.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, y_pred)
        print(f"{name} AUC: {auc}")

        plot_roc_curve(model, X_test, y_test)
        plt.title(f"{name} ROC Curve")
        plt.show()

models = {
    "Randomized Search": random_model,
    "Bayesian Optimization": bayesian_model,
    "Hyperopt-like Optimization": hyperopt_model
}

evaluate_models(models, X_test, y_test)
```

Purpose: This code evaluates and visualizes the performance of each model, providing insights into their effectiveness.

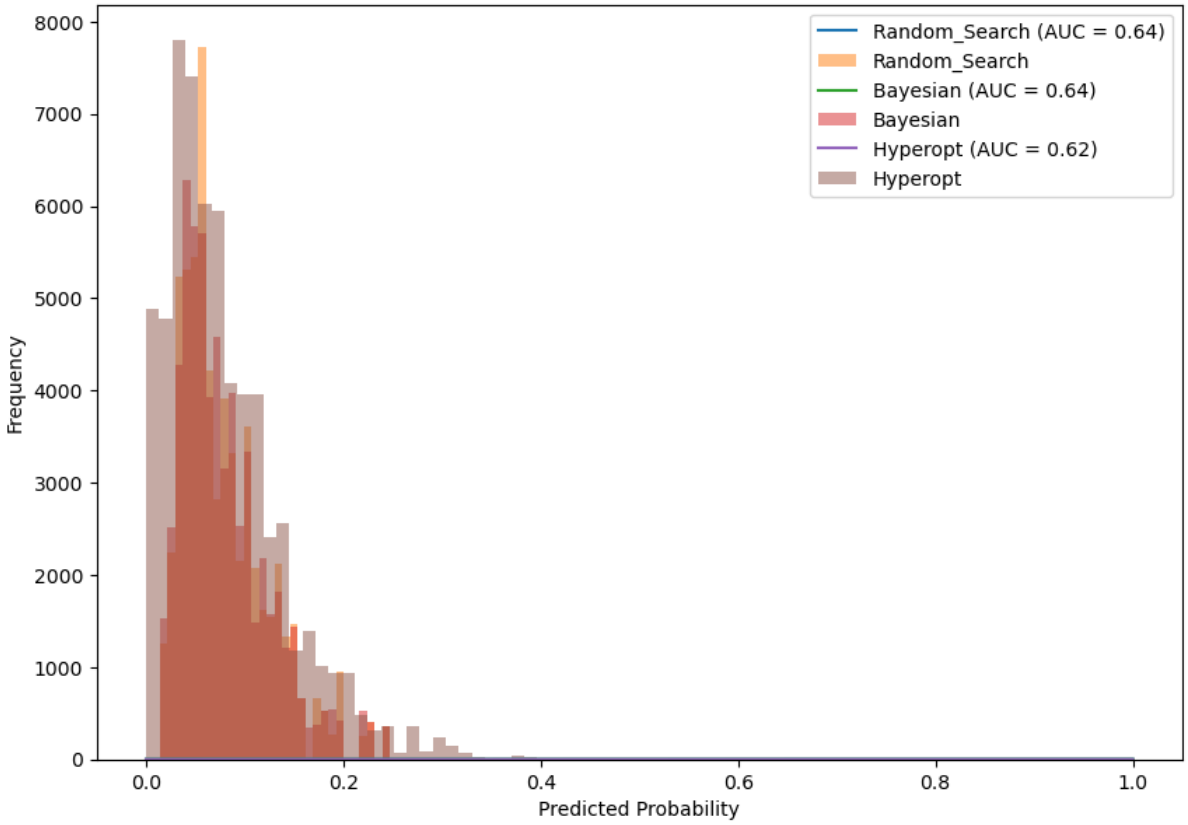
Results

- **Randomized Search ROC AUC**: 0.637
- **Bayesian Optimization ROC AUC**: 0.638
- **Hyperopt-like Optimization ROC AUC**: 0.618

Diagrams and Plots

ROC Curves:

Purpose: ROC curves provide a visual representation of model performance, highlighting the trade-off between true positive rate and false positive rate.



Summary

This project successfully demonstrates the implementation of automated hyperparameter optimization techniques. Bayesian optimization proved to be the most effective, yielding the highest ROC AUC score. The comprehensive comparison of different techniques offers valuable insights into their strengths and limitations.

Findings:

- Bayesian optimization is efficient in finding optimal hyperparameters.
- Randomized Search, while simpler, provides a useful baseline.
- Custom Hyperopt-like optimization offers flexibility and robust performance.

Future Improvements:

- **Expand the Dataset**: Include more diverse datasets to enhance model robustness.
- **Parallel Processing**: Implement parallel processing to speed up hyperparameter search.
- **Advanced Models**: Integrate more complex models to explore their hyperparameter spaces.
- **Ensemble Methods**: Combine multiple models to improve overall performance.

In conclusion, this project provides a solid foundation for automated hyperparameter optimization, demonstrating the effectiveness of various techniques and offering directions for future enhancements.