ExhBMA

Koki Obinata

Mar 20, 2023

CONTENTS:

Exhaustive search with Bayesian model averaging is an estimation method with confidence evaluation. Feature selection can be performed with confidence evaluation.

Installation

You can install this package via pip command.

```
pip install exhbma
```

CHAPTER

ONE

EXHBMA WITH LINEAR REGRESSION

Apply ExhBMA with linear regression model. The number of features is restricted to under around twenty to perform an exhaustive search.

1.1 Generate sample dataset

We consider a linear model $y = x_1 + x_2 - 0.8x_3 + 0.5x_4 + \epsilon$ for a toy generative model. Noise vaiable ϵ follows the normal distribution with a mean of 0 and a variance of 0.1.

We add dummy features and prepare ten features in total. The number of observed data is 50.

```
import numpy as np

n_data, n_features = 50, 10
sigma_noise = 0.1

nonzero_w = [1, 1, -0.8, 0.5]
w = nonzero_w + [0] * (n_features - len(nonzero_w))

np.random.seed(0)
X = np.random.randn(n_data, n_features)
y = np.dot(X, w) + sigma_noise * np.random.randn(n_data)
```

1.2 Train a model

First, training data should be preprocessed with centering and normalizing operation. Normalization for target variable is optional and is not performed in a below example (`scaling=False` is passed for `y_scaler`).

```
from exhbma import ExhaustiveLinearRegression, inverse, StandardScaler

x_scaler = StandardScaler(n_dim=2)
y_scaler = StandardScaler(n_dim=1, scaling=False)
x_scaler.fit(X)
y_scaler.fit(y)
X = x_scaler.transform(X)
y = y_scaler.transform(y)
```

For constructing an estimator model, prior distribution parameters for σ_{noise} and σ_{coef} need to be specified. You need to specify discrete points for the x-inverse distribution and these points are used for numerical integration (marginalization over σ_{noise} and σ_{coef}). The range of prior distributions should be wide enough to cover the peak of the posterior distribution.

Fit process will finish within a minute.

```
# Settings of prior distributions for sigma_noise and sigma_coef
n_sigma_points = 20
sigma_noise_log_range = [-2.5, 0.5]
sigma_coef_log_range = [-2, 1]

# Model fitting
reg = ExhaustiveLinearRegression(
    sigma_noise_points=inverse(
        np.logspace(sigma_noise_log_range[0], sigma_noise_log_range[1], n_sigma_points),
    ),
    sigma_coef_points=inverse(
        np.logspace(sigma_coef_log_range[0], sigma_coef_log_range[1], n_sigma_points),
    ),
)
reg.fit(X, y)
```

1.3 Results

First, we import plot utility functions and prepare names for features.

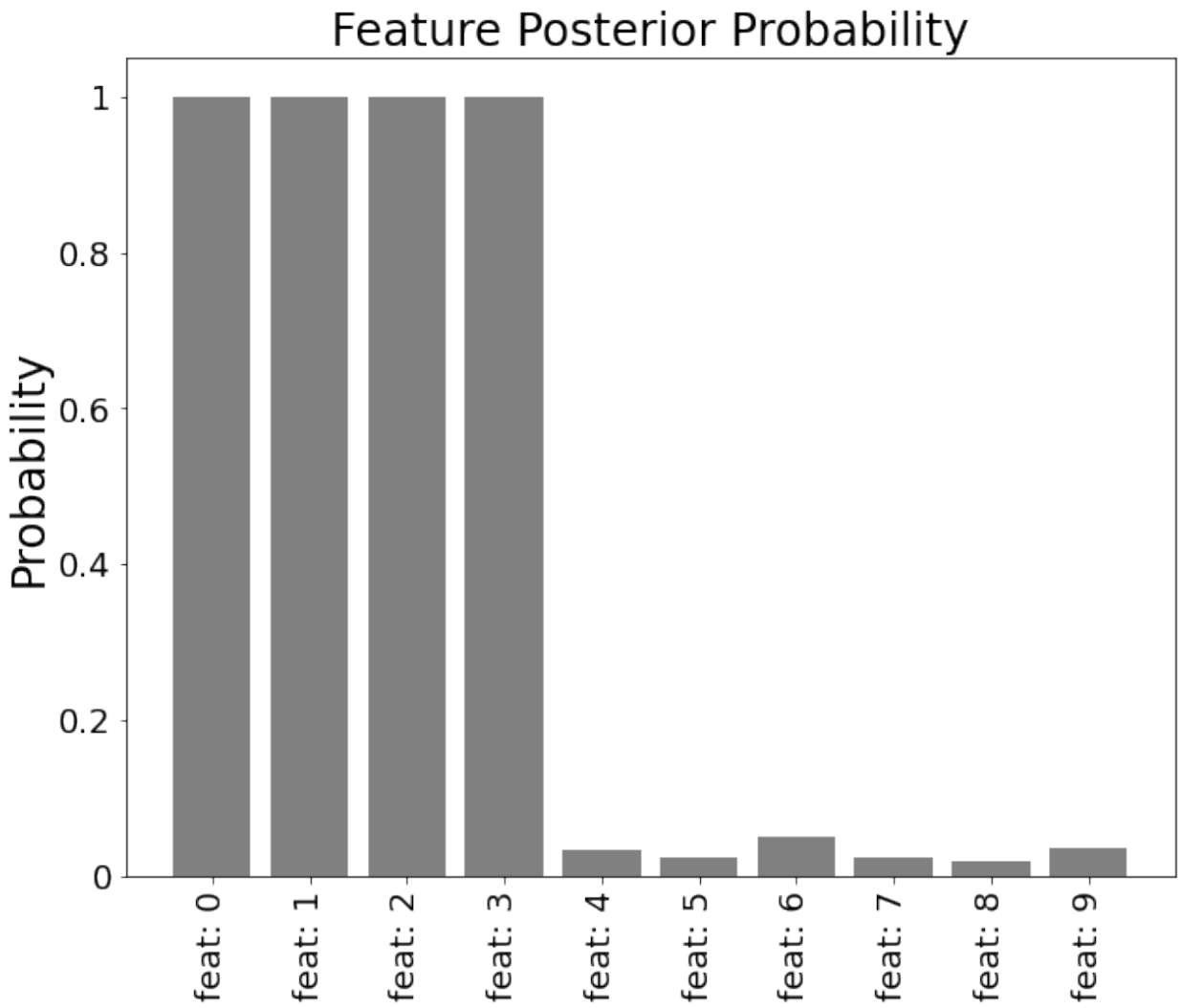
```
from exhbma import feature_posterior, sigma_posterior

columns = [f"feat: {i}" for i in range(n_features)]
```

1.3.1 Feature posterior

Posterior probability that each feature is included in the model is estimated.

You can access the values of posterior probability by `reg.feature_posteriors_`.

```
fig, ax = feature_posterior(
    model=reg,
    title="Feature Posterior Probability",
    ylabel="Probability",
    xticklabels=columns,
)
```

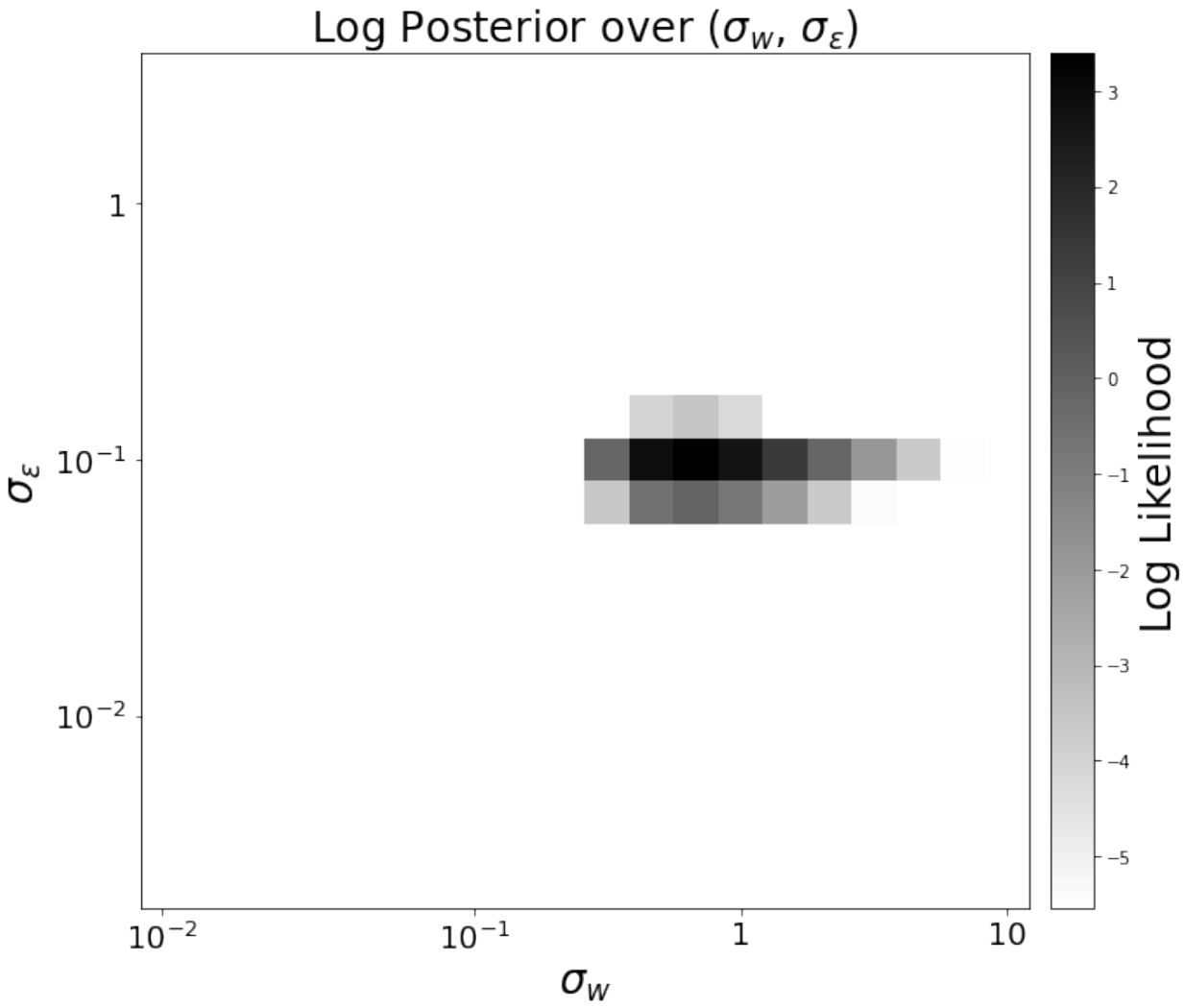


Probabilities for features with nonzero coefficient are close to 1, which indicate that we can select first four features with high confidence. On the other hand, probabilities for dummy features are close to 0.

1.3.2 Sigma posterior distribution

To confirm that defined range of prior distributions properly contains a peak of posterior distribution, plot the posterior distribution of hyperparameters σ_{noise} and σ_{coef}.

```
fig, ax = sigma_posterior(
    model=reg,
    title="Log Posterior over ($\sigma_{w}$, $\sigma_{\epsilon}$)",
    xlabel="$\sigma_{w}$",
    ylabel="$\sigma_{\epsilon}$",
    cbarlabel="Log Likelihood",
)
```



Peak of the distribution is near the $(\sigma_{\text{noise}}, \sigma_{\text{coef}}) = (0.1, 0.8)$, which is close to the predefined values.

1.3.3 Coefficients

Coefficient values estimated by BMA is stored in *ExhaustiveLinearRegression.coef_* attribute.

```
for i, c in enumerate(reg.coef_):
    print(f"Coefficient of feature {i}: {c:.4f}")
```

Output

```
Coefficient of feature 0: 1.0072
Coefficient of feature 1: 0.9766
Coefficient of feature 2: -0.7500
Coefficient of feature 3: 0.5116
Coefficient of feature 4: -0.0006
Coefficient of feature 5: -0.0002
Coefficient of feature 6: 0.0012
```

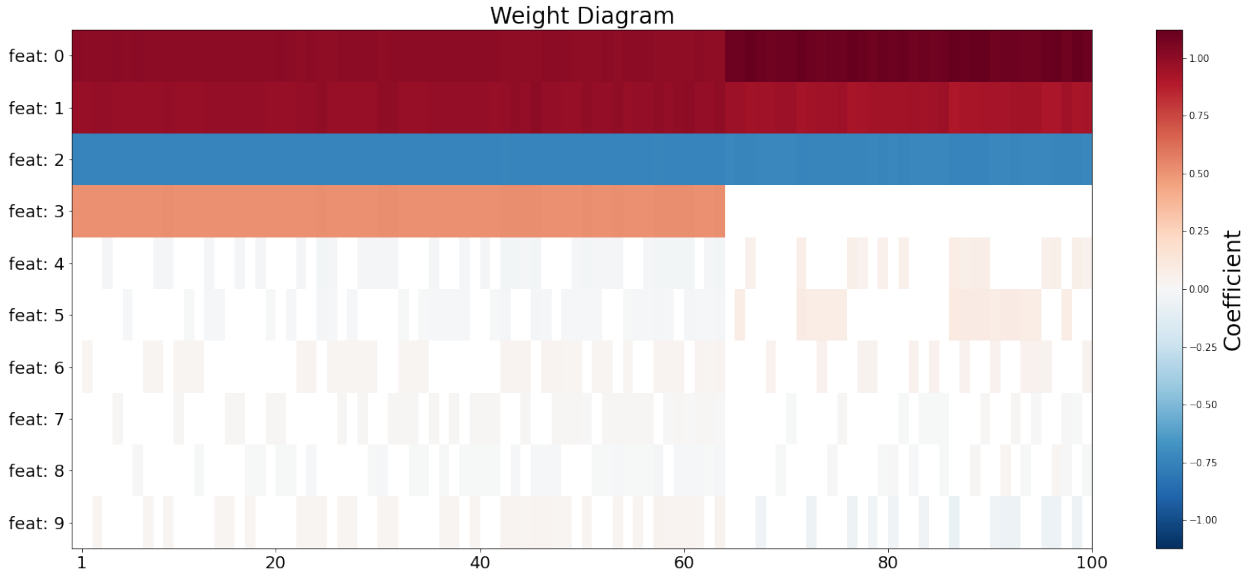
(continues on next page)

(continued from previous page)

```
Coefficient of feature 7: 0.0003
Coefficient of feature 8: -0.0000
Coefficient of feature 9: 0.0007
```

1.3.4 Weight Diagram

For more insights into the model, weight diagram[1] is a useful visualization method.



1.4 Prediction for new data

For new data, prediction method is prepared. First, we prepare test data to evaluate the prediction performance.

```
n_test = 10 ** 3
np.random.seed(10)
test_X = np.random.randn(n_test, n_features)
test_y = np.dot(test_X, w) + sigma_noise * np.random.randn(n_test)
```

For prediction, we use *predict* method. Note that data transformation is necessary for feature data and predicted data.

```
pred_y = y_scaler.restore(
    reg.predict(x_scaler.transform(test_X), mode="full")
)
```

For performance evaluation, we calculate root mean squared error (RMSE).

```
rmse = np.power(test_y - pred_y, 2).mean() ** 0.5
print(f"RMSE for test data: {rmse:.4f}")
```

Output

[1] Y. Igarashi et al., ES-DoS: Exhaustive search and density-of-states estimation as a general framework for sparse variable selection, 2018

```
RMSE for test data: 0.1029
```

RMSE value is close to the predefined noise magnitude, so the estimation is successfully performed.

References

CHAPTER

TWO

API REFERENCE

API documents for exhaustive search with Bayesian model averaging.

2.1 Estimators

class exhbma.exhaustive_search.**ExhaustiveLinearRegression**(*sigma_noise_points: List[RandomVariable]*, *sigma_coef_points: List[RandomVariable]*, *alpha: float = 0.5*, *exclude_null: bool = False*)

ExhaustiveSearchModel with linear_model.LinearRegression

- Intercept of the linear model is assumed to be zero.
  - Assume that target variable y is centralized.
  - Assume that all features x are centralized and normalized.

Parameters

- **sigma_noise_points** (*List[RandomVariable]*) – Data points to explore sigma_noise parameter in exhaustive search.
- **sigma_coef_points** (*List[RandomVariable]*) – Data points to explore sigma_coef parameter in exhaustive search.
- **alpha** (*float (default: 0.5)*) – Alpha parameter is fixed to this value.
- **exclude_null** (*bool (default: False)*) – Whether or not exclude a null model.

n_features_in_

Number of features seen during fit.

Type
int

coef_

Coefficients of the regression model (mean of distribution).

Type
List[float]

`log_likelihood_`

Log-likelihood of the model. Marginalization is performed over sigma_noise, sigma_coef, indicators.

Type
float

`log_likelihood_over_sigma_`

Log-likelihood over σ_{noise} and σ_{coef}, $p(y|\sigma_{noise}, \sigma_{coef}, X)$, which is marginalized over indicators. Prior distributions for both sigma are not included.

Type
List[List[float]]

`feature_posteriors_`

Posterior probabilities for each feature.

Type
List[float]

`indicators_`

List of indicator vectors. Null model *[0, 0, ..., 0]* are excluded, so length is $2^{n\_features\_in\_} - 1$.

Type
List[List[int]]

`log_priors_`

List of log-prior probabilities for each model specified by indicator.

Type
List[float]

`log_likelihoods_`

List of log-likelihood of each model specified by indicator.

Type
List[float]

`models_`

Information for all models specified by indicator vector. Length of this attribute is equal to that of **indicators_** and models correspond to each other.

Type
List[ModelInfo]

`fit`(*X: ndarray*, *y: ndarray*, *verbose: bool = True*)

Train a model

1. Create indicator vectors
2. Fit sub-models for each indicator vector with marginalizing sigma_noise and sigma_coef for each indicator
3. Calculate final model averaged over sub-models

Parameters

- **X** (*np.ndarray with shape (n_data, n_features)*) – Feature matrix. Each row corresponds to single data.
- **y** (*np.ndarray with shape (n_data,)*) – Target value vector.
- **verbose** (*bool (default: True)*) – If this is set to *True*, progress bar is displayed.

predict(*X: ndarray*, *mode: str*, *threshold: float = 0.5*) → ndarray

Predict using the model

Predict about new data.

Parameters

- **X** (*np.ndarray with shape (n_data, n_features)*) – Feature matrix for prediction.
- **mode** (*str*) – Mode of prediction.
- **threshold** (*float (default: 0.5)*) – Feature selection threshold used in mode 'select'.

Returns

y – Prediction values.

Return type

np.ndarray

select_variables(*threshold: float = 0.5*) → List[int]

Return indicator with posterior probability greater than or equal to threshold.

class exhbma.linear_regression.**LinearRegression**(*sigma_noise: float*, *sigma_coef: float*)

Model description:

- Base model: Linear regression model
- Observation noise: Gaussian distribution
- Prior distribution for coefficient: Gaussian distribution
- Intercept of the linear model is assumed to be zero.
  - Assume that target variable y is centralized.
  - Assume that all features x are centralized and normalized.
  - Marginalization over intercept is performed.

Parameters

- **sigma_noise** (*float*) – Standard deviation of gaussian noise. Model assumes that observation noise obeys Gaussian distribution with mean = 0, variance = sigma_noise^2.
- **sigma_coef** (*float*) – Standard deviation of gaussian noise. Model assumes that each coefficient value obeys Gaussian distribution with mean = 0, variance = sigma_coef^2 independently.

n_features_in_

Number of features seen during fit.

Type

int

coef_

Coefficients of the regression model (mean of distribution).

Type

List[float]

`log_likelihood_`

Log-likelihood of the model. Marginalization is performed over sigma_noise, sigma_coef, indicators.

Type
float

`fit`(*X: ndarray*, *y: ndarray*, *skip_preprocessing_validation: bool = False*)

Calculate coefficient used in prediction and log likelihood for this data.

Parameters

- **X** (*np.ndarray with shape (n_data, n_features)*) – Feature matrix. Each row corresponds to single data.
- **y** (*np.ndarray with shape (n_data,)*) – Target value vector.

`predict`(*X*)

Prediction using trained model.

X
[np.ndarray with shape (n_data, n_features)] Feature matrix for prediction.

2.2 Utilities

Utility classes and functions for constructing estimators.

`class exhbma.scaler.StandardScaler`(*n_dim*, *scaling: bool = True*)

`exhbma.probabilities.gamma`(*x: ndarray*, *low: float | None = None*, *high: float | None = None*, *shape: float = 0.001*, *scale: float = 1000.0*) → List[RandomVariable]

Gamma distribution: x ~ (const.) * x^(shape - 1) * exp(- x / scale) Distribution is limited on range of [low, high]. If low(high) is None, low(high) is set as the minimum(maximum) value of x.

`exhbma.probabilities.inverse`(*x: ndarray*, *low: float | None = None*, *high: float | None = None*) → List[RandomVariable]

x-inverse distribution: p(x) = 1 / x This distribution becomes proper when finite interval is considered.

`exhbma.probabilities.uniform`(*x: ndarray*, *low: float = 0.0*, *high: float = 1.0*) → List[RandomVariable]

Uniform distribution: p(x) = 1 / (high - low)

`exhbma.integrate.integrate_log_values_in_line`(*log_values: List[float]*, *x1: List[float]*, *weights: List[float] | None = None*, *expect_positive: bool = True*)

Integrate along line.

log_values: 1 dimension array with length len(x1)
Log values to be integrated along x1

x1: List[float]
1st axis points.

weights: Optional[List[float]], default: None
Weights to log_values.

expect_positive: bool, default: True
If set True, ValueError is raised when result is not positive.

exhbma.integrate.**integrate_log_values_in_square**(*log_values: List[List[float]]*, *x1: List[float]*, *x2: List[float]*, *weights: List[List[float]] | None = None*, *expect_positive: bool = True*)

Integrate box region. int weight * exp(log_value) dx1 dx2

log_values: 2 dimension array with shape (len(x1), len(x2))
Log values to be integrated over x1 and x2 axes.

x1: List[float]
1st axis points.

x2: List[float]
2nd axis points.

weights: Optional[List[List[float]]], default: None
Weights to log_values.

expect_positive: bool, default: True
If set True, ValueError is raised when result is not positive.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX