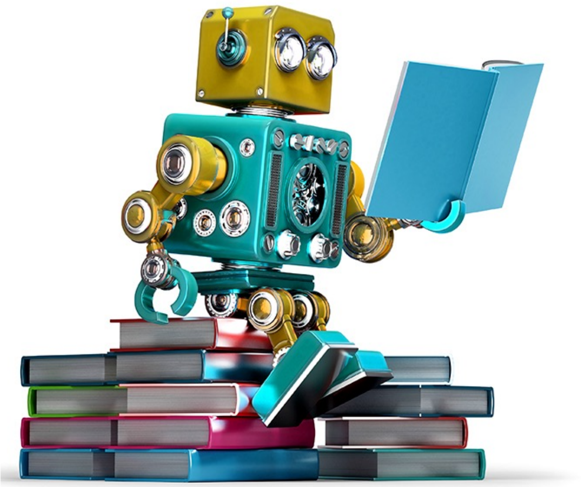
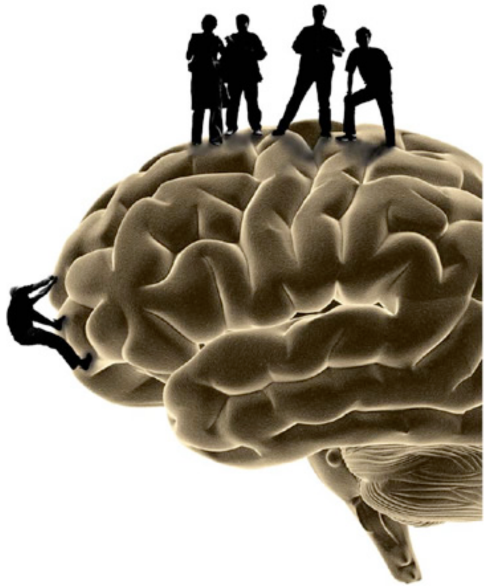
Hyperparameter optimization

Alexandre Gramfort
http://alexandre.gramfort.net



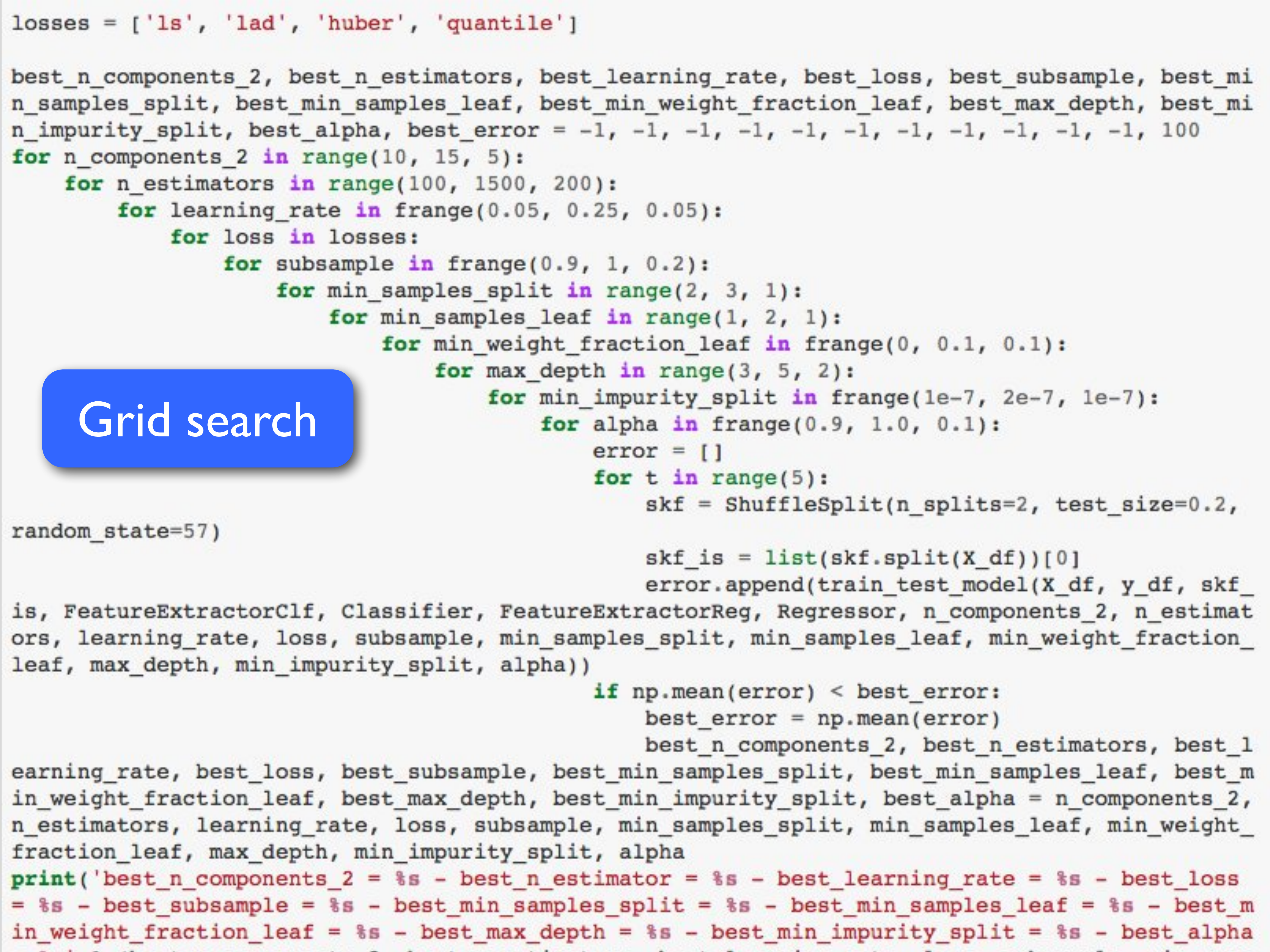


Related concepts

- Meta learning
- “Learning to learn”
- Automatic Machine Learning (AutoML)

Approaches

- Grid search
- Random search
- Bayesian Optimization
- Gradient based optimization
- Evolutionary optimization



Grid search

```
losses = ['ls', 'lad', 'huber', 'quantile']

best_n_components_2, best_n_estimators, best_learning_rate, best_loss, best_subsample, best_mi
n_samples_split, best_min_samples_leaf, best_min_weight_fraction_leaf, best_max_depth, best_mi
n_impurity_split, best_alpha, best_error = -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 100
for n_components_2 in range(10, 15, 5):
    for n_estimators in range(100, 1500, 200):
        for learning_rate in frange(0.05, 0.25, 0.05):
            for loss in losses:
                for subsample in frange(0.9, 1, 0.2):
                    for min_samples_split in range(2, 3, 1):
                        for min_samples_leaf in range(1, 2, 1):
                            for min_weight_fraction_leaf in frange(0, 0.1, 0.1):
                                for max_depth in range(3, 5, 2):
                                    for min_impurity_split in frange(1e-7, 2e-7, 1e-7):
                                        for alpha in frange(0.9, 1.0, 0.1):
                                            error = []
                                            for t in range(5):
                                                skf = ShuffleSplit(n_splits=2, test_size=0.2,
random_state=57)

                                                skf_is = list(skf.split(X_df))[0]
                                                error.append(train_test_model(X_df, y_df, skf_
is, FeatureExtractorClf, Classifier, FeatureExtractorReg, Regressor, n_components_2, n_estimat
ors, learning_rate, loss, subsample, min_samples_split, min_samples_leaf, min_weight_fraction_
leaf, max_depth, min_impurity_split, alpha))
                                            if np.mean(error) < best_error:
                                                best_error = np.mean(error)
                                                best_n_components_2, best_n_estimators, best_l
earning_rate, best_loss, best_subsample, best_min_samples_split, best_min_samples_leaf, best_m
in_weight_fraction_leaf, best_max_depth, best_min_impurity_split, best_alpha = n_components_2,
n_estimators, learning_rate, loss, subsample, min_samples_split, min_samples_leaf, min_weight_
fraction_leaf, max_depth, min_impurity_split, alpha
print('best_n_components_2 = %s - best_n_estimator = %s - best_learning_rate = %s - best_loss
= %s - best_subsample = %s - best_min_samples_split = %s - best_min_samples_leaf = %s - best_m
in_weight_fraction_leaf = %s - best_max_depth = %s - best_min_impurity_split = %s - best_alpha
```

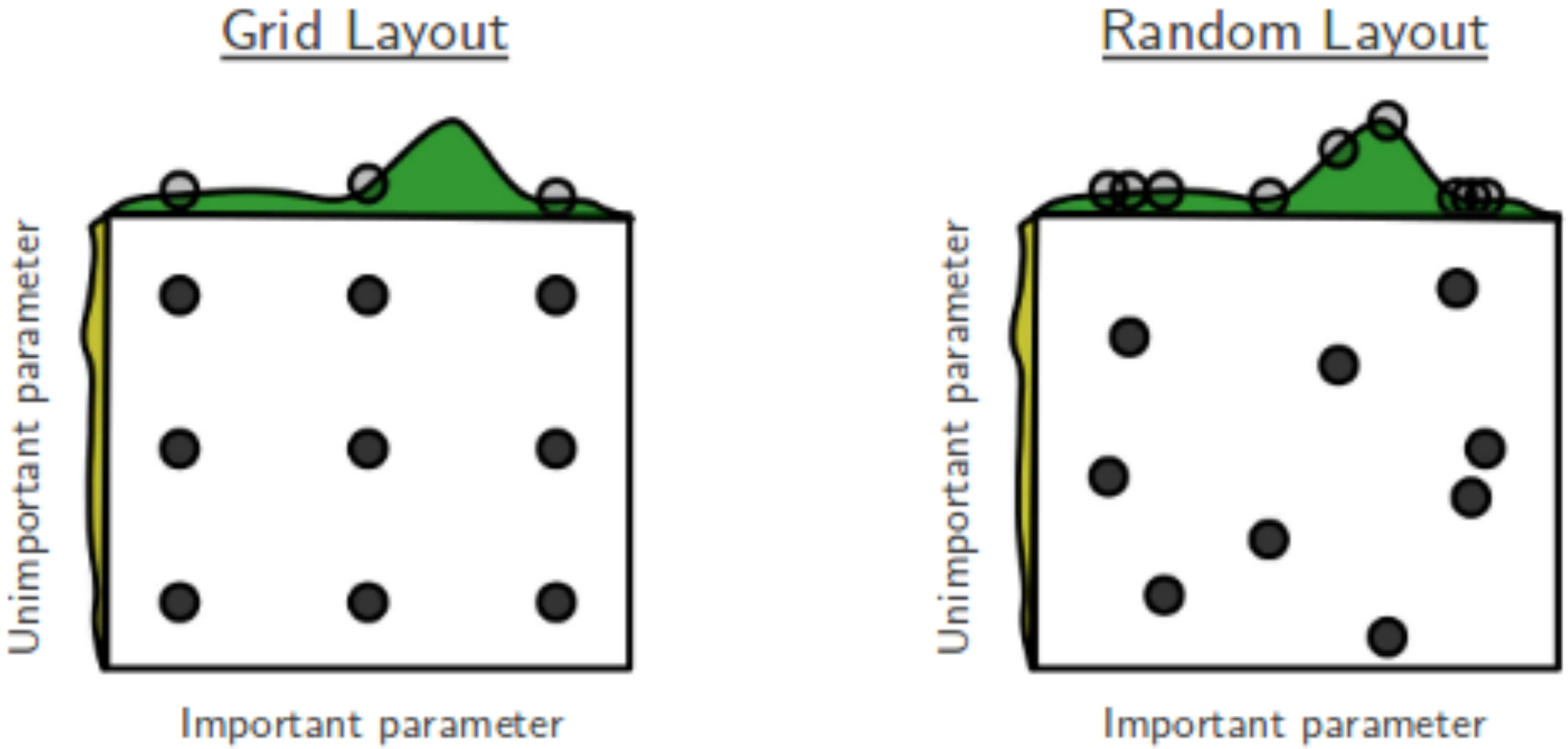
Grid search

```
>>> param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005,
                       0.01, 0.1], }
>>> clf = GridSearchCV(SVC(kernel=‘rbf',
                       class_weight='balanced'),
                       param_grid, cv=5)
>>> clf = clf.fit(X_train_pca, y_train)
>>> print("Best estimator found by grid search:")
>>> print(clf.best_estimator_)
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

Random Search



http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

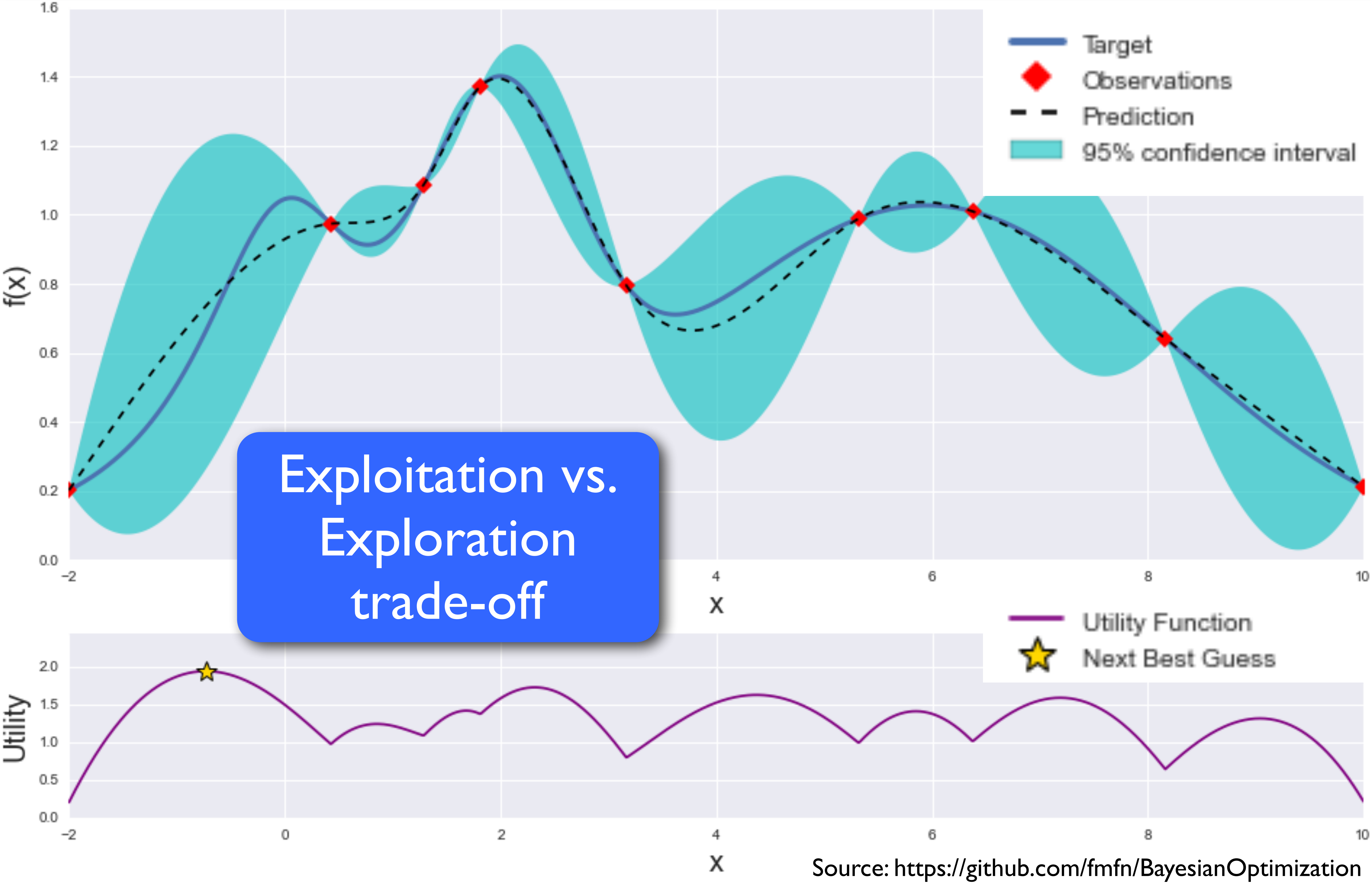
Random Search

```
# specify parameters and distributions to sample from
>>> param_dist = {"max_depth": [3, None],
                  "max_features": randint(1, 11),
                  "min_samples_split": randint(2, 11),
                  "bootstrap": [True, False],
                  "criterion": ["gini", "entropy"]}
>>> clf = RandomForestClassifier(n_estimators=20)
>>> n_iter_search = 20
>>> random_search = \
        RandomizedSearchCV(clf,
                          param_distributions=param_dist,
                           n_iter=n_iter_search, cv=5)
>>> random_search.fit(X, y)
```

https://scikit-learn.org/stable/modules/generated/
sklearn.model_selection.RandomizedSearchCV.html

https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html

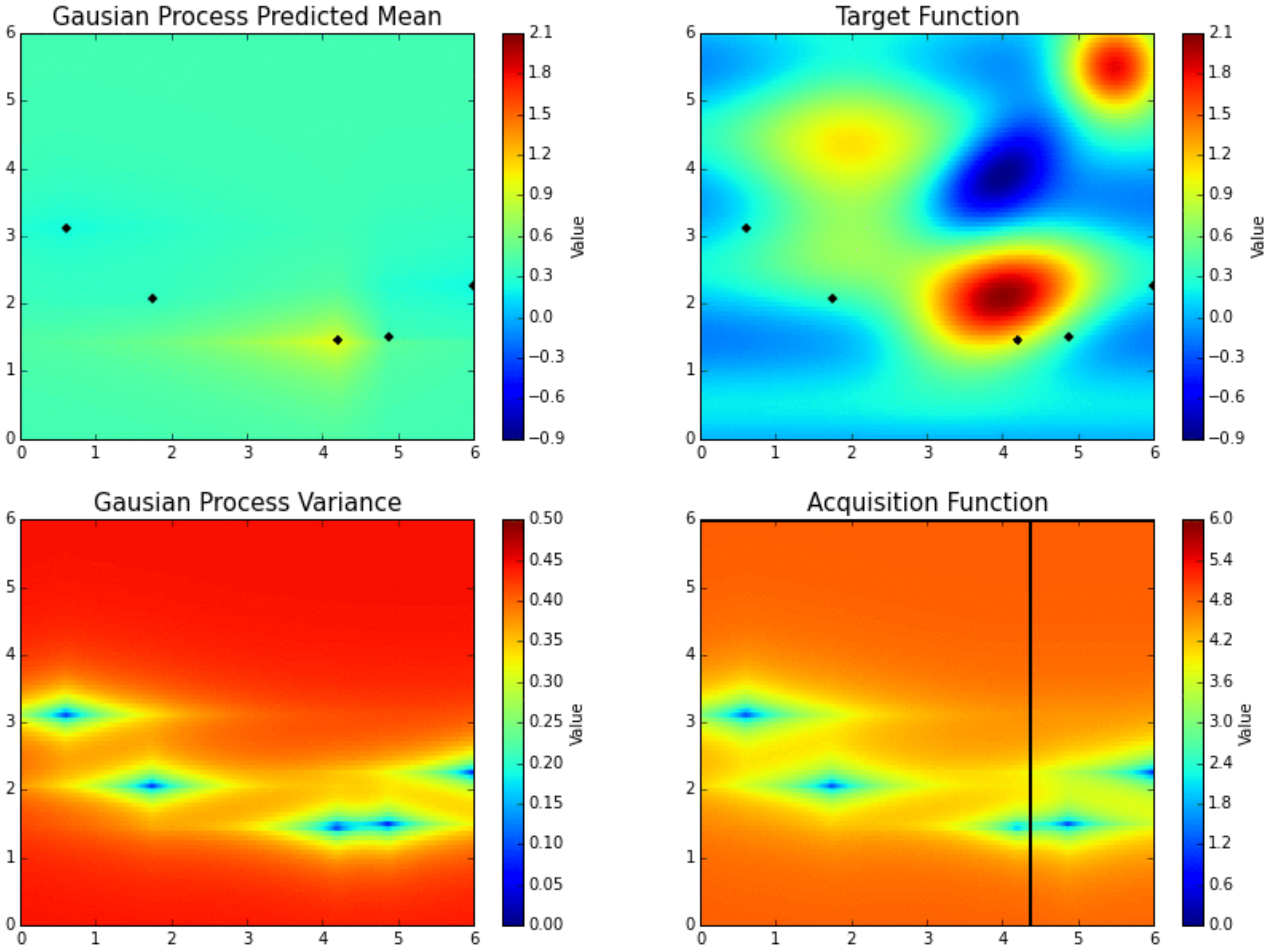
Bayesian Optimization



Exploitation vs. Exploration trade-off

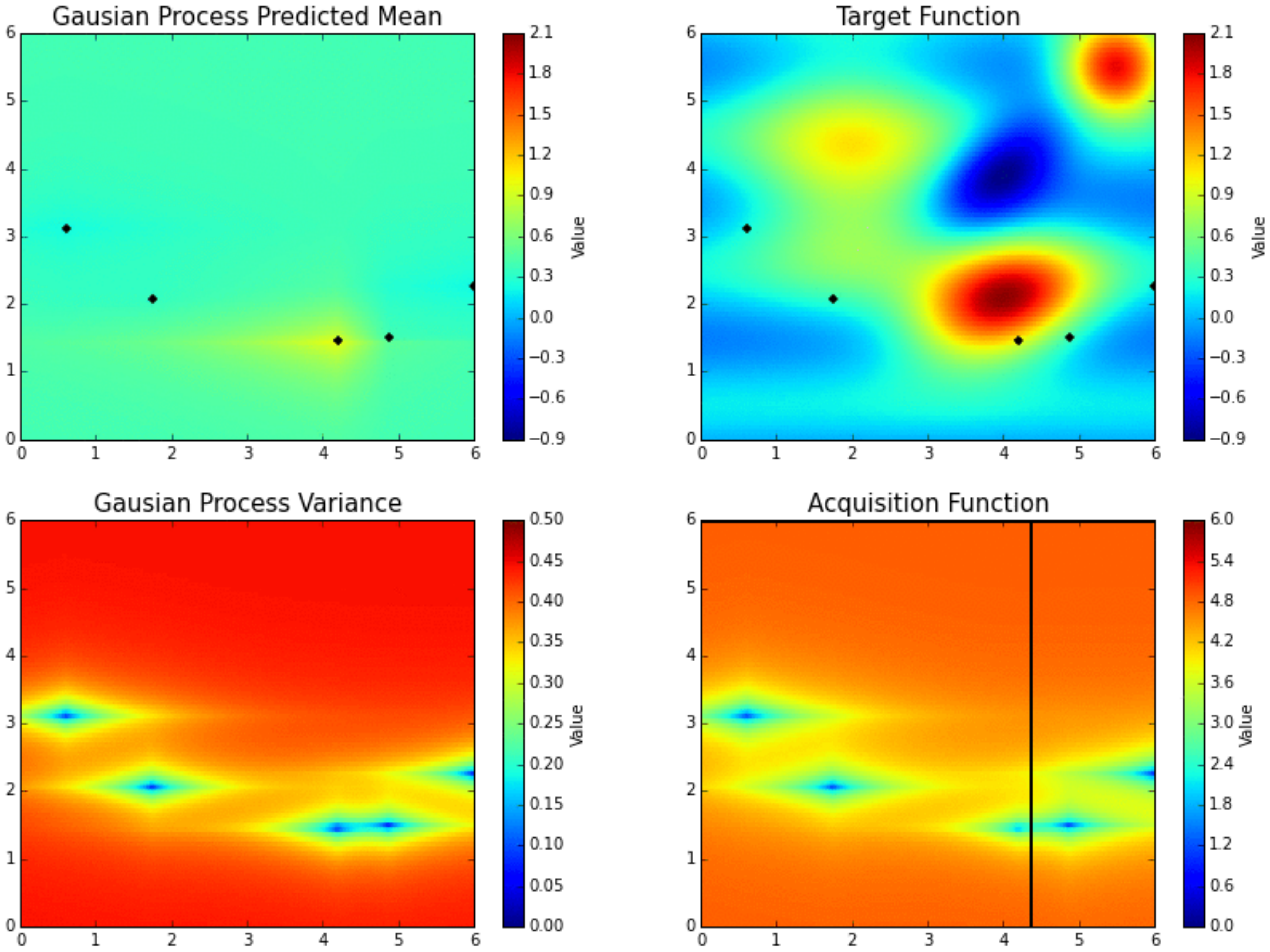
Source: https://github.com/fmfn/BayesianOptimization

Bayesian Optimization in Action



Source: https://github.com/fmfn/BayesianOptimization

Bayesian Optimization in Action



Source: https://github.com/fmfn/BayesianOptimization

Ex. scikit-optimize

```
from skopt import BayesSearchCV

# log-uniform: search over p = exp(x) by varying x
opt = BayesSearchCV(
    SVC(),
    {
        'C': (1e-6, 1e+6, 'log-uniform'),
        'gamma': (1e-6, 1e+1, 'log-uniform'),
        'degree': (1, 8),  # integer valued parameter
        'kernel': ['linear', 'poly', 'rbf'],  # categorical
    },
    n_iter=32
)

opt.fit(X_train, y_train)
```

https://scikit-optimize.github.io/#skopt.BayesSearchCV

Gradient based

- **Idea:** Compute gradient of cross-validation score w.r.t. hyper parameters

- Eg. use automatic differentiation with a smooth loss and an iterative algorithm like gradient descent

https://arxiv.org/abs/1502.03492
https://arxiv.org/abs/1602.02355
https://github.com/HIPS/hypergrad

Software

• hyperopt	https://github.com/hyperopt/hyperopt
• hyperband	https://github.com/zygmuntz/hyperband
• scikit-optimize	https://scikit-optimize.github.io/
• smac	https://github.com/automl/SMAC3
• spearmint	https://github.com/HIPS/Spearmint
• optuna	https://optuna.org/

Hands on

scikit
learn