A Practical Guide to Bayesian Optimization

Jason King, PhD

Data Scientist, XSOLIS

What led me to Bayesian Optimization?

- Like many of you, I compete in Kaggle competitions
- For whatever reason, GBMs tend to be the most popular (single) model types
  - Significantly more hyperparameters to consider compared to GLMs
- Most competitive teams use ensembling and/or stacking
  - Zillow entry was a stacked model: XGBoost, Light GBM, Random Forest, Extra Trees, AdaBoost (tree-based), two Neural Networks, and a K-Nearest-Neighbor.
  - Linear Regression meta-learner

Hyperparameter comparison

- Linear Regression:
  - fit_intercept, normalize
- Ridge:
  - alpha, fit_intercept, normalize, solver, max_iter
- Decision Tree:
  - criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_split, min_impurity_descrease, presort
- Random Forest:
  - above, n_estimators, bootstrap

Hyperparameter comparison

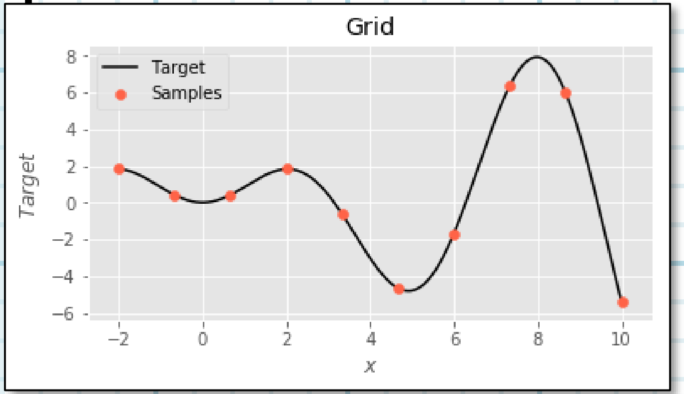
- XGBoost:
  - booster, eta, gamma, max_depth, min_child_weight, max_delta_step, subsample, colsample_bytree, colsample_bylevel, lambda, alpha, tree_method, sketch_eps, scale_pos_weight, updater, refresh_leaf, process_type, grow_policy, max_leaves, max_bin, objective, base_score
- Neural Networks:
  - ∞ (get a grad student!)

Zillow Prize

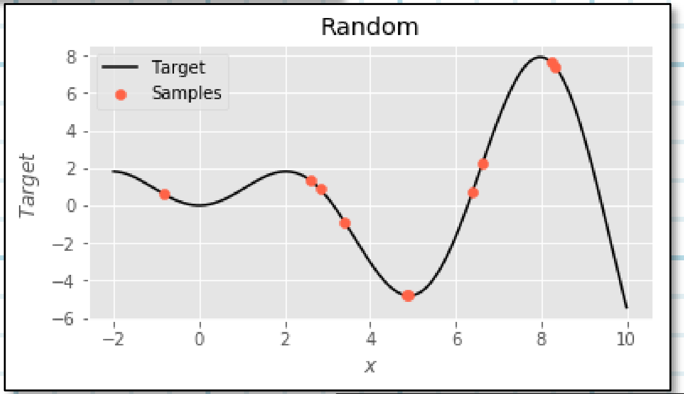
- High-cost training
- Started seeing a lot of chatter about a smarter automated tuning method
  - Learning the hyperparameter space
- A number of Python implementations
  - Metric Optimization Engine (MOE)
  - Bayesian Optimization
  - Hyperopt
- Relatively straightforward to implement in my code
  - Still required a bounded range to search over
  - Took less iterations, and new models performed better

Hyperparameter tuning strategies

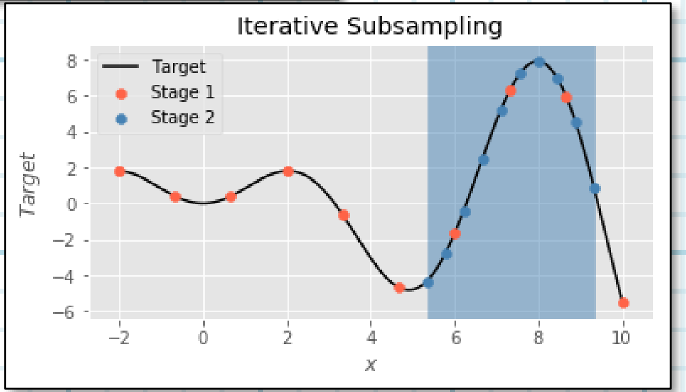
- Grid



- Random



- Iterative subsampling



Objectives

1. Learn how to implement Bayesian Optimization in your code
2. Understand the underlying mechanism
3. Discuss the limitations of the technique
4. Explore some extended use cases

Installation

```
pip install bayesian-optimization
```

Example 1

simple_regression.ipynb

Example 2

simple_classification.ipynb

Example 3

mercari_prep.ipynb

mercari_train.ipynb

Gaussian Process

- A gaussian process uses lazy learning and a measure of similarity between points (kernel function) to predict the value for unseen points
- Given a black box function/process, we can obtain a few samples and generate a predicted function with uncertainty
- This enables derivative-free optimization
- More importantly, if sampling the target function is expensive, we can reduce overall cost

Gaussian Process

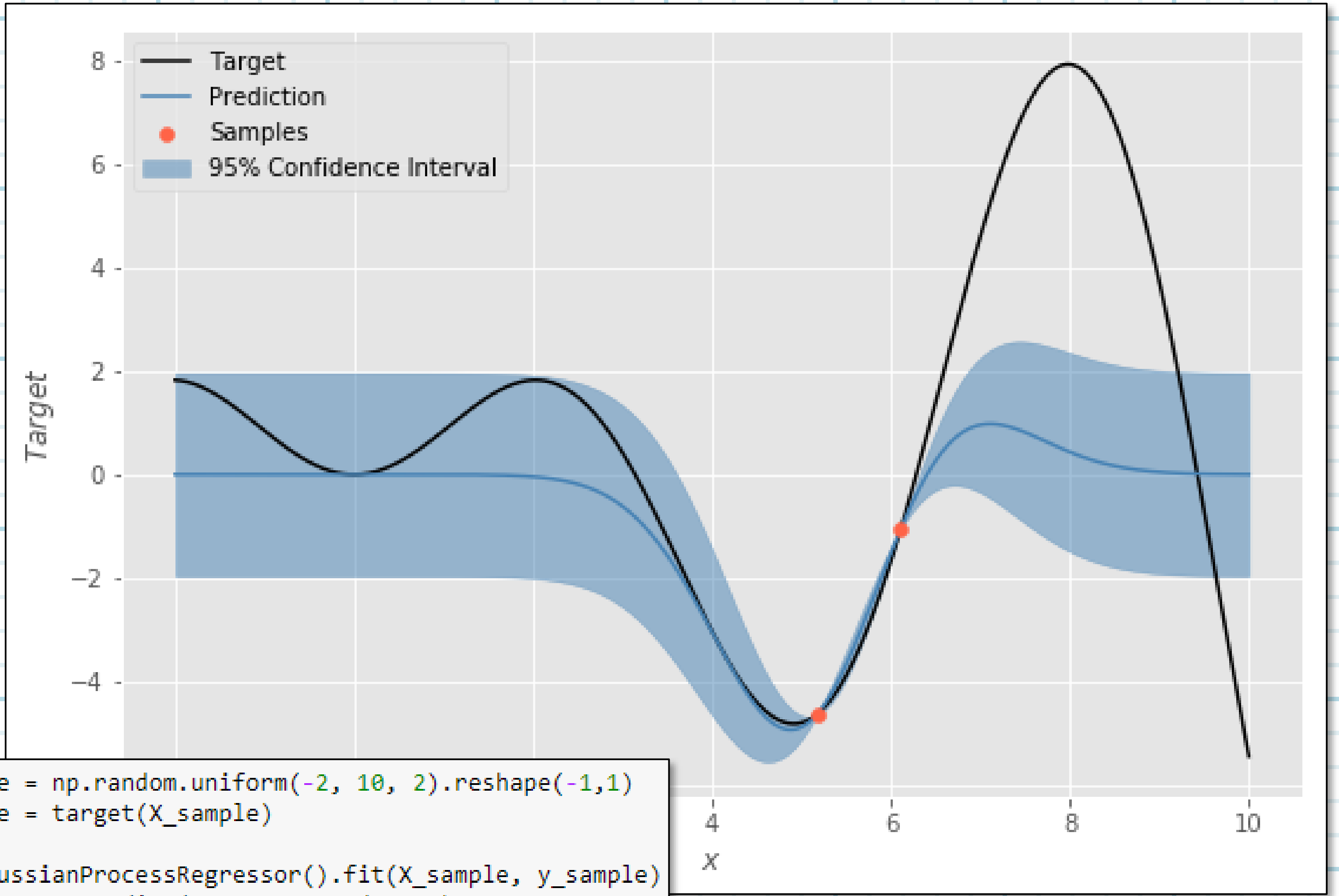
- Noiseless GP Regression

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim N\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix} \cdot \begin{pmatrix} \mathrm{K} & K_* \\ K_*^T & K_{**} \end{pmatrix}\right)$$

$$p(f_*|X_*, X, f) = N(f_*|\mu_*, \Sigma_*)$$
$$\mu_* = \mu(X_*) + K_*^T K^{-1}\big(f - \mu(X)\big)$$
$$\Sigma_* = K_{**} - K_*^T K^{-1} K_*$$

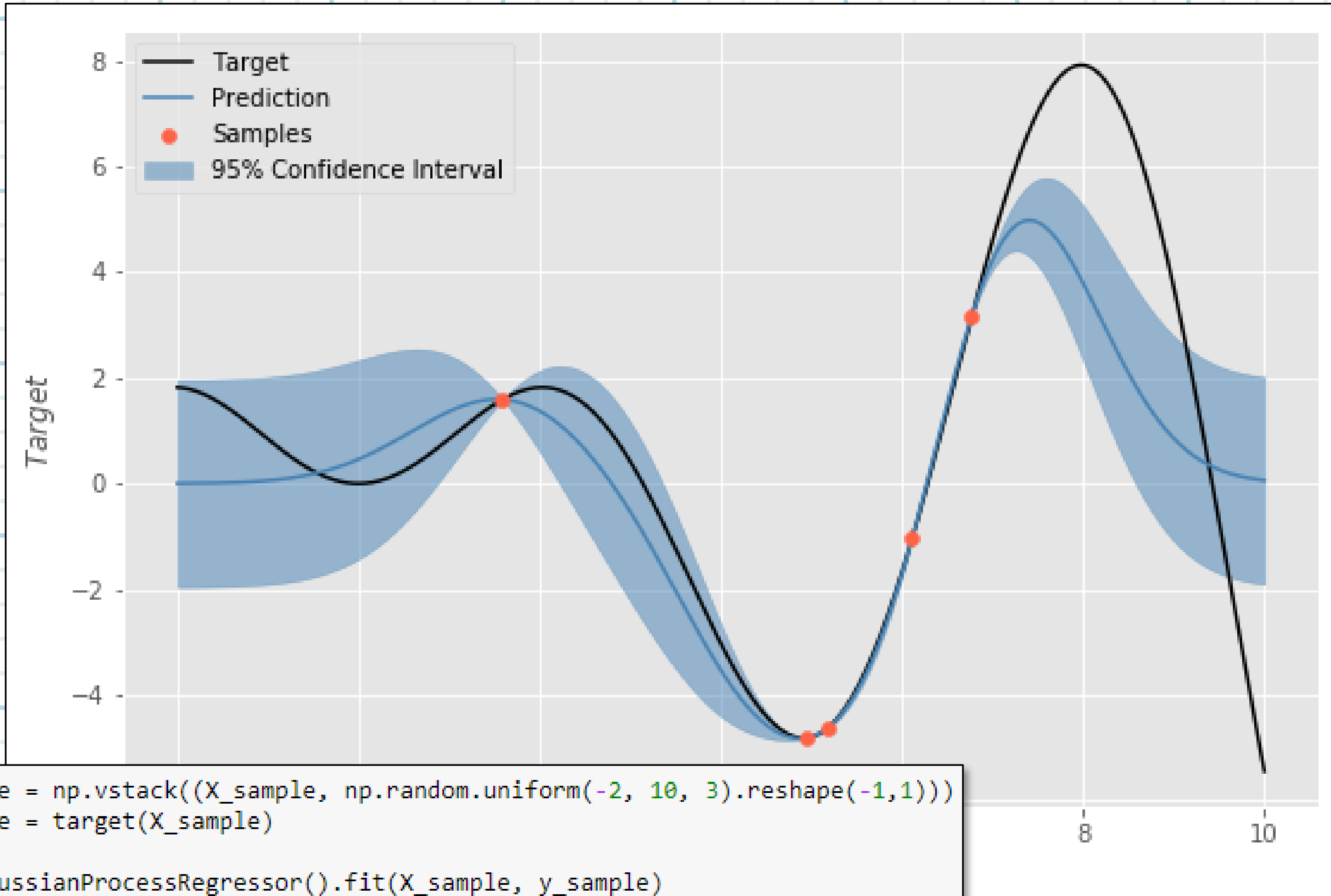
$$K(x, x_*) = e^{\left(-\frac{1}{2}(x - x_*)^2\right)}$$

Gaussian Process Regression




```
X_sample = np.random.uniform(-2, 10, 2).reshape(-1,1)
y_sample = target(X_sample)

gp = GaussianProcessRegressor().fit(X_sample, y_sample)
mu, std = gp.predict(X, return_std=True)
```

Gaussian Process Regression

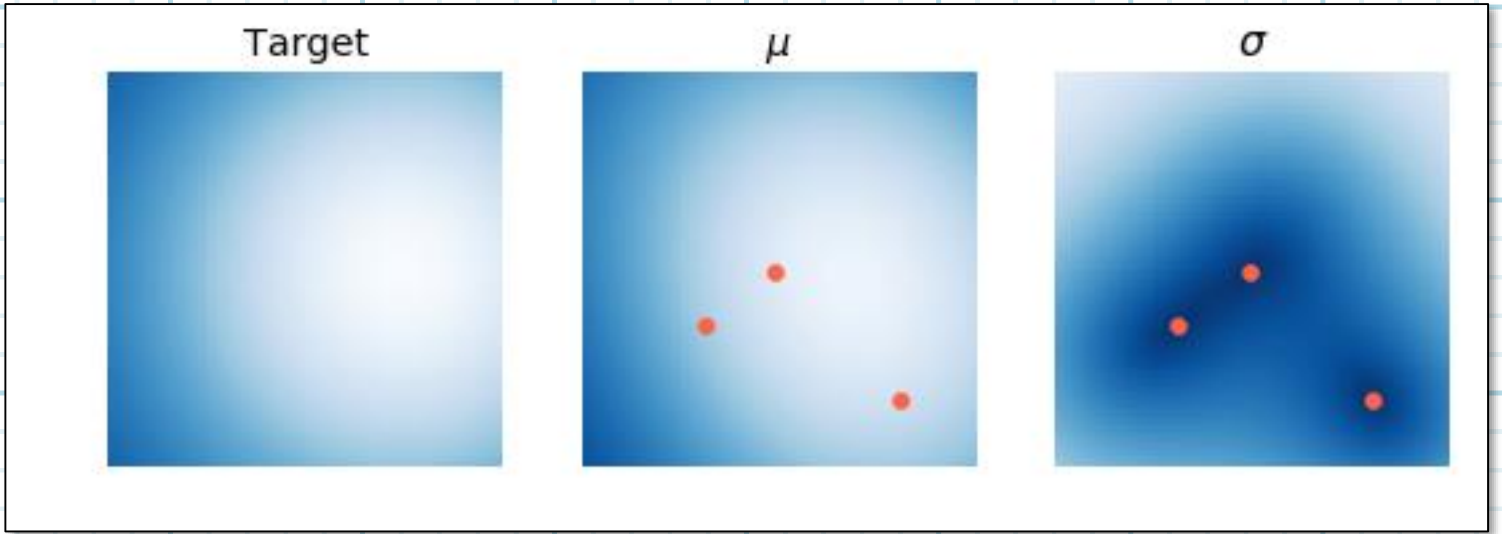



```
X_sample = np.vstack((X_sample, np.random.uniform(-2, 10, 3).reshape(-1,1)))
y_sample = target(X_sample)

gp = GaussianProcessRegressor().fit(X_sample, y_sample)
mu, std = gp.predict(X, return_std=True)
```

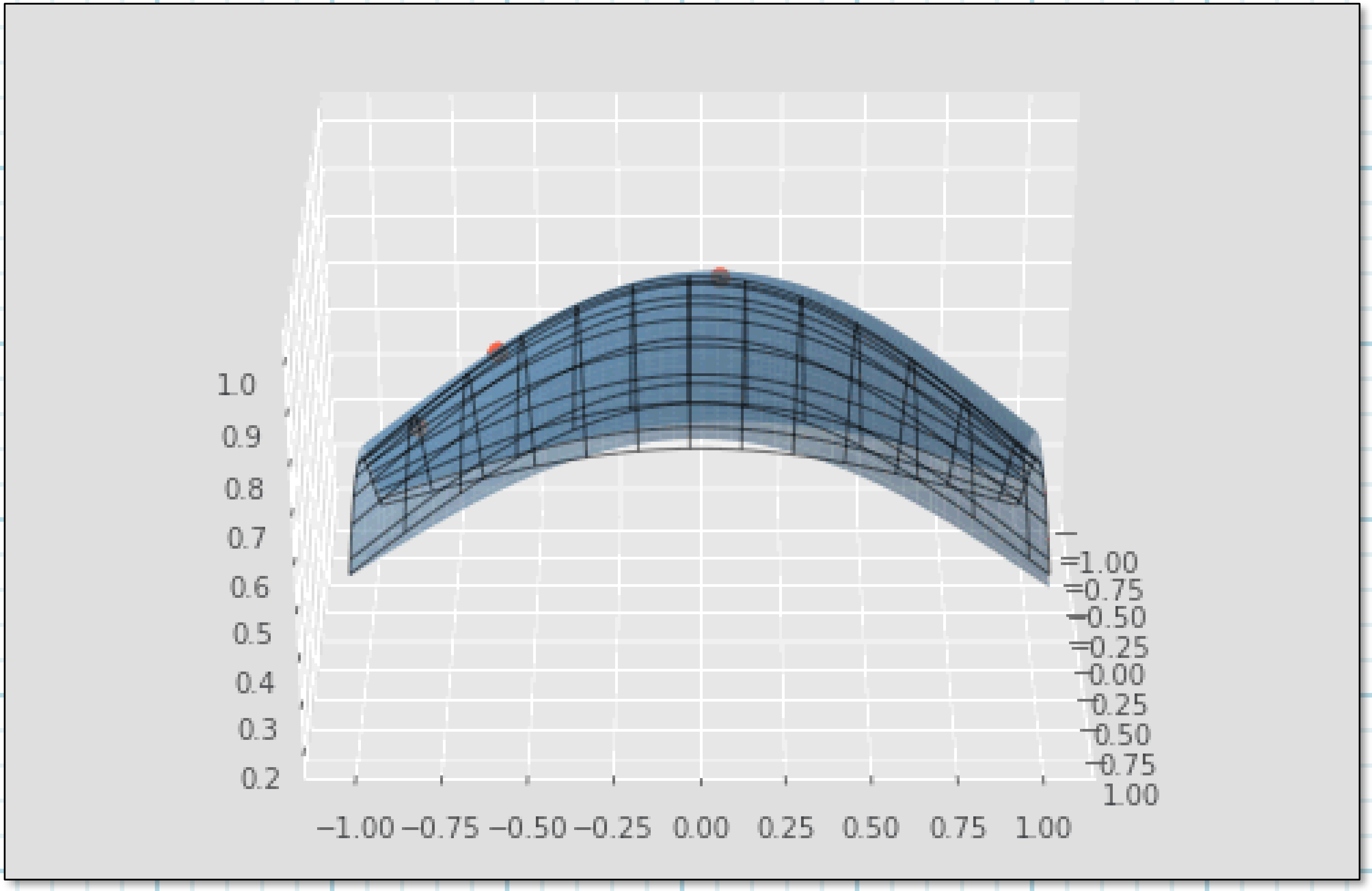
Gaussian Process Regression

```
def gaussian(x, y, x0=0.5, y0=0, sigma=1):
    d = np.sqrt((x-x0)**2 + (y-y0)**2)
    return np.exp(-(d**2/(2.0*sigma**2)))

x, y = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))
z = gaussian(x, y)
```

```
X_sample = np.random.uniform(-1, 1, size=(3,2))
z_sample = gaussian(X_sample[:,0], X_sample[:,1])

gp = GaussianProcessRegressor().fit(X_sample, z_sample)
mu, std = gp.predict(np.hstack((x.reshape(-1,1), y.reshape(-1,1))), return_std=True)

mu, std = mu.reshape(100,100), std.reshape(100,100)
```



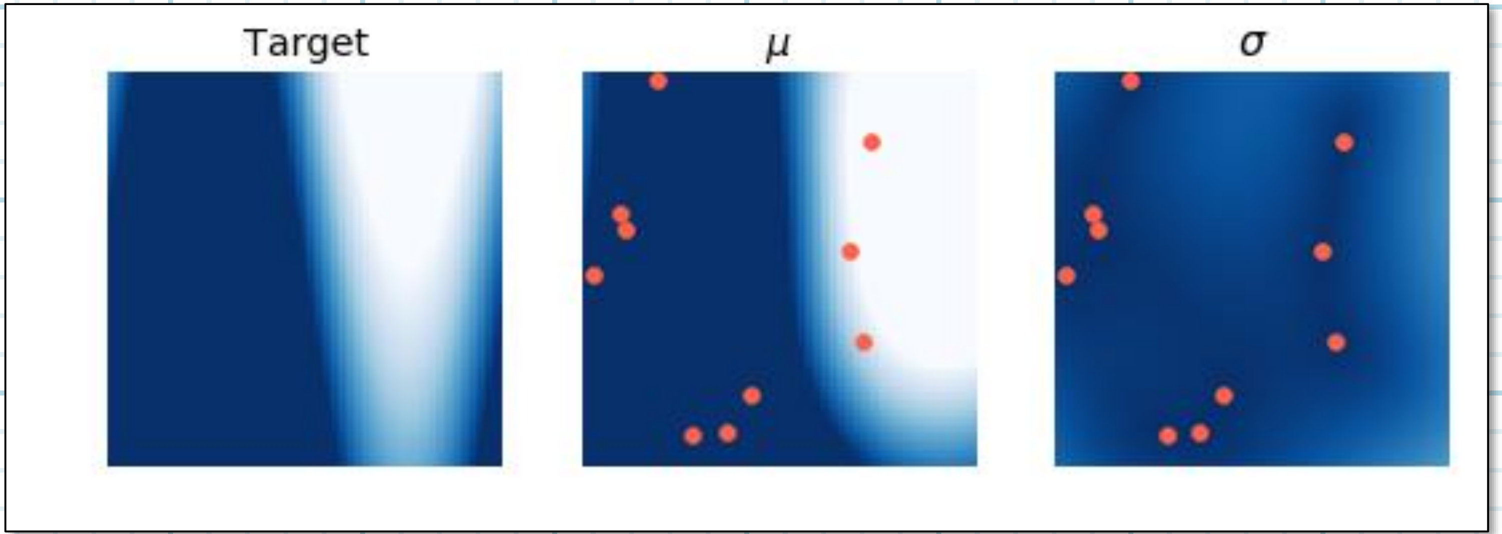
Gaussian Process Regression



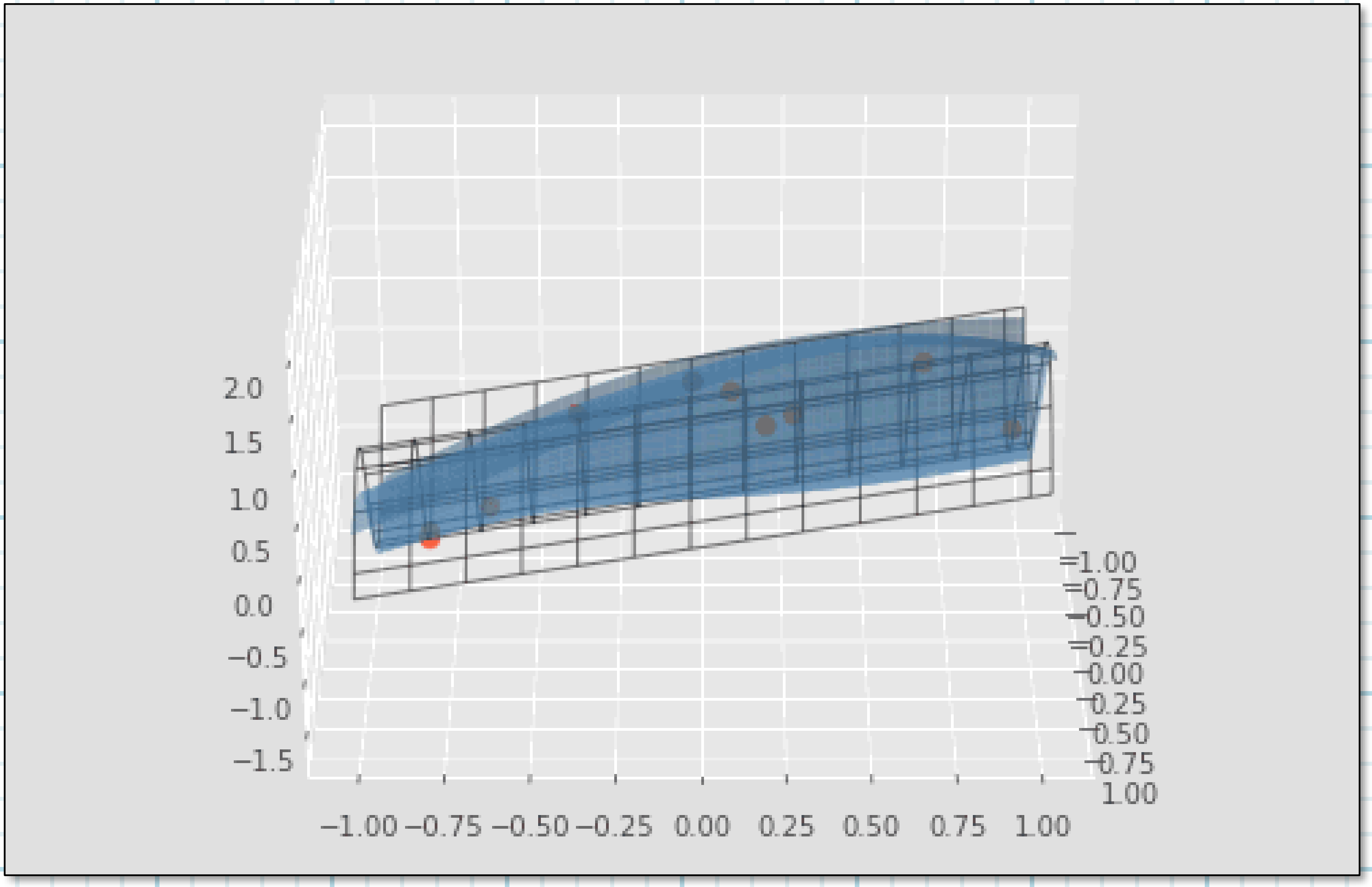
Gaussian Process Regression

```
def sine_plane(x, y, x0=0.5, y0=0):
    return np.sin(3*x) + 0.5*y

x, y = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))
z = sine_plane(x, y)
```

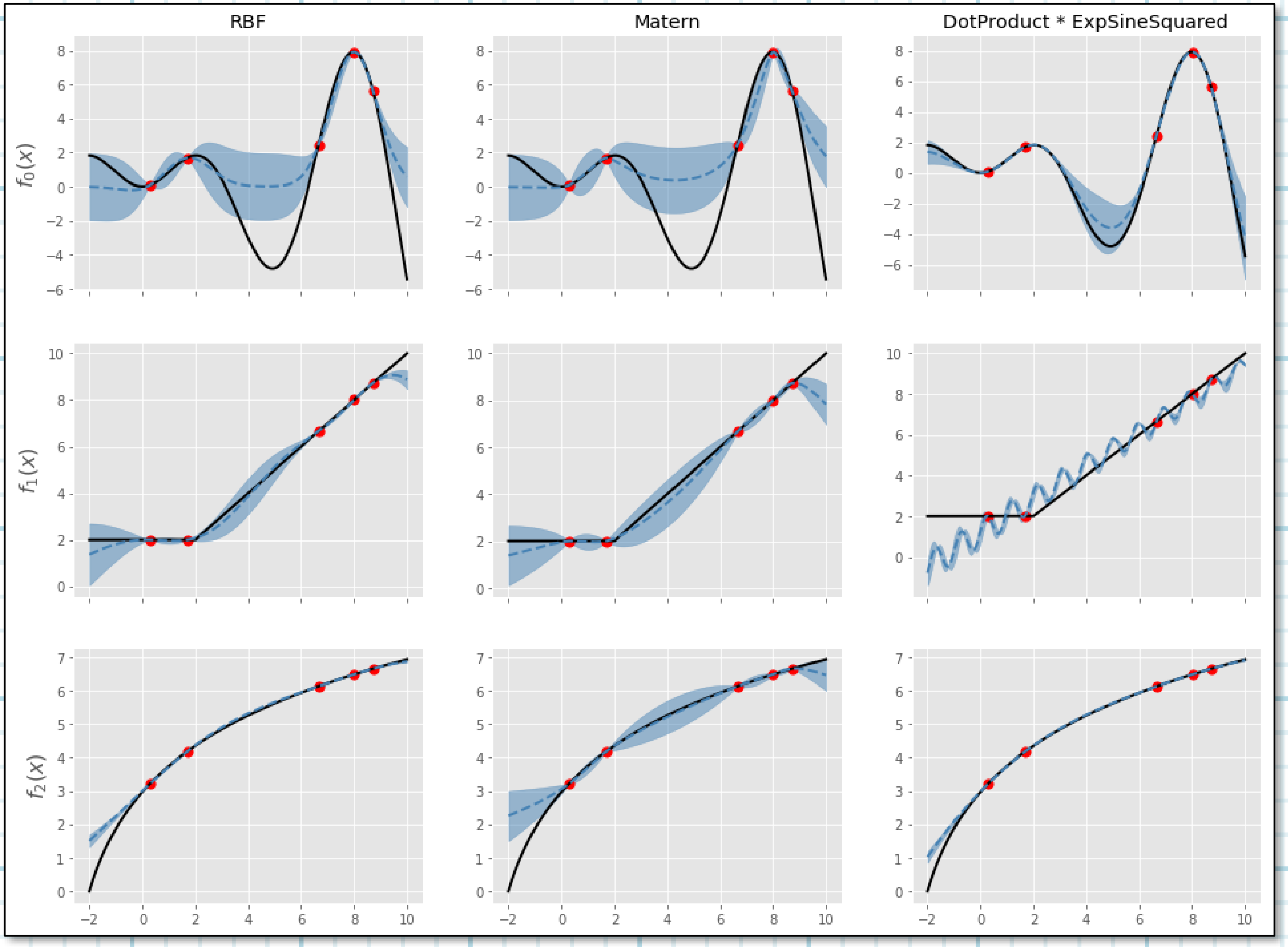
```
X_sample = np.random.uniform(-1, 1, size=(10,2))
z_sample = sine_plane(X_sample[:,0], X_sample[:,1])

gp = GaussianProcessRegressor(alpha=1e-8).fit(X_sample, z_sample)
mu, std = gp.predict(np.hstack((x.reshape(-1,1), y.reshape(-1,1))), return_std=True)

mu, std = mu.reshape(100,100), std.reshape(100,100)
```



Gaussian Process Regression



Gaussian Process - Kernels



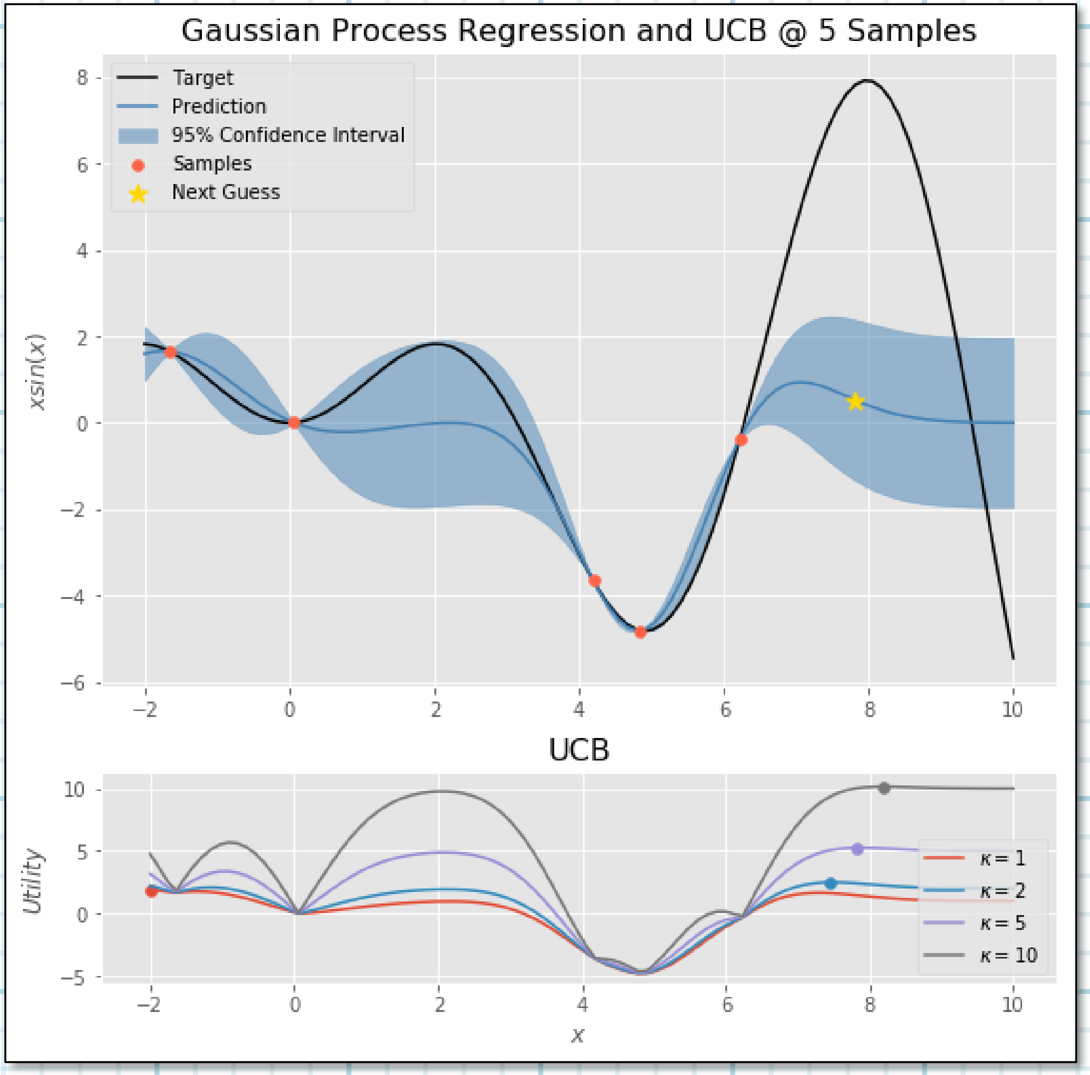
Bayesian Optimization

bayes_opt_example.ipynb

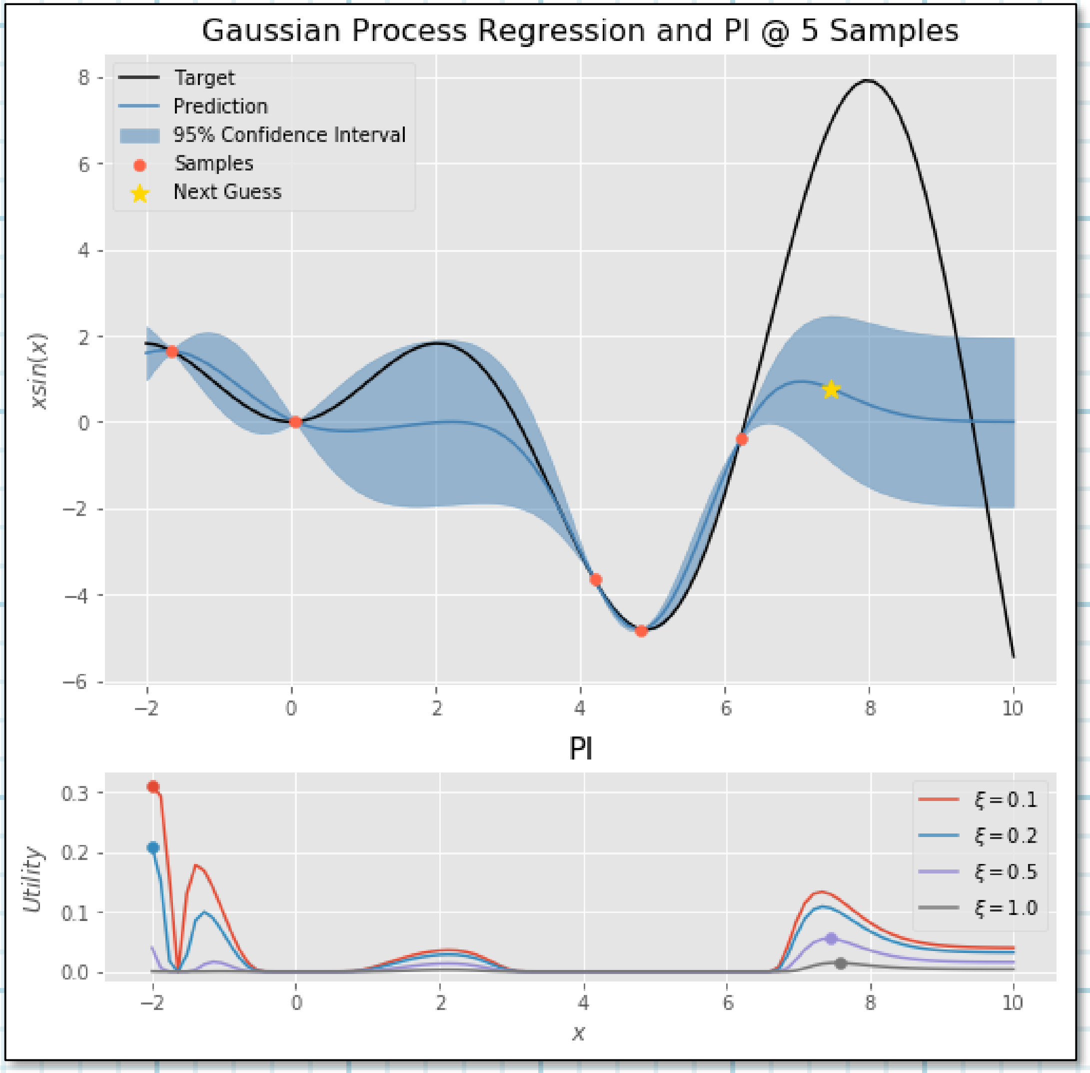
Utility Functions

- Upper Confidence Bound (UCB)
    - $UCB(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x})$
- Probability of Improvement (PI)
    - $PI(\boldsymbol{x}) = P(f(\boldsymbol{x}) > f(\boldsymbol{x}^+) + \xi | D_{1:t})$
    - $f(\boldsymbol{x}^+)$ is the current max
- Expected Improvement (EI)
    - $EI(\boldsymbol{x}) = \boldsymbol{E}[max\{0, f(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi | D_{1:t}\}]$

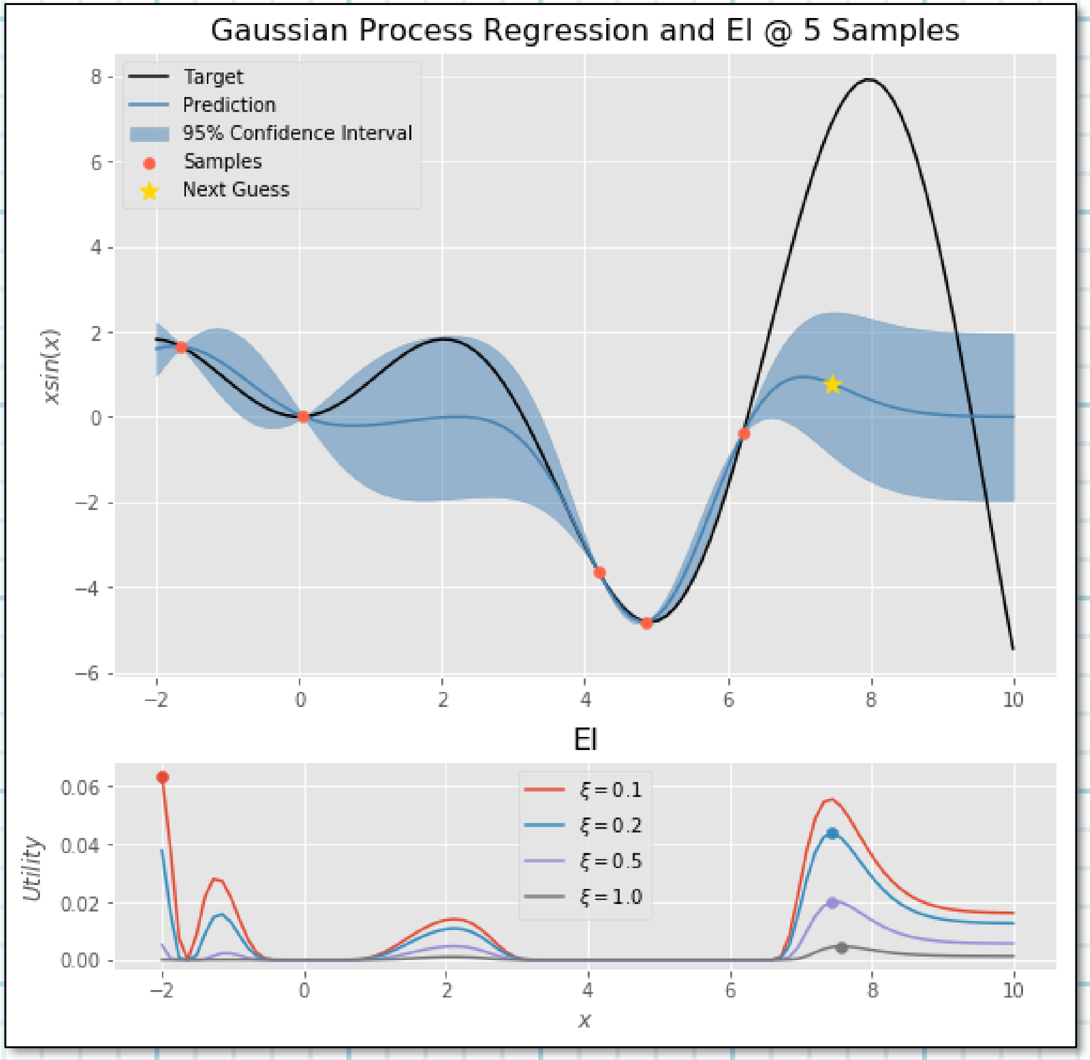
Utility Functions



Utility Functions



Utility Functions



Limitations

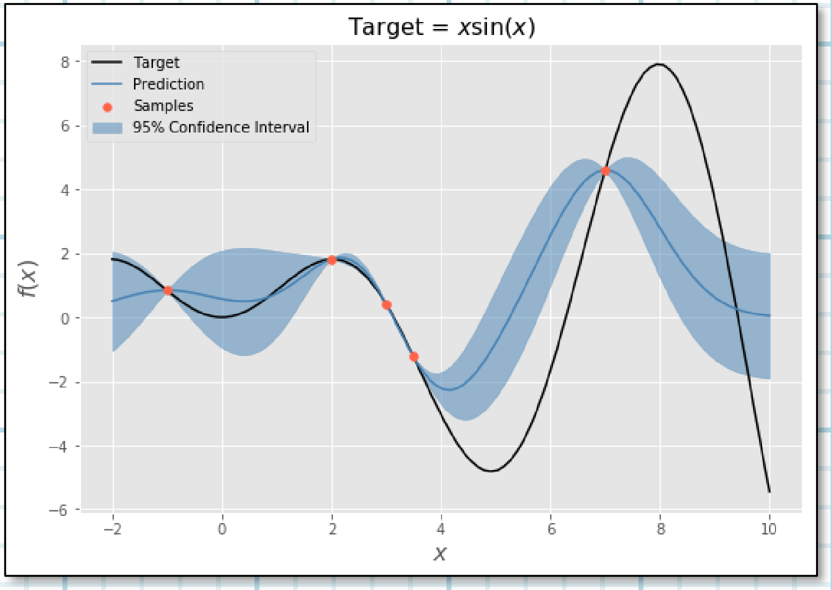
- Overhead
  - Retrain Gaussian process regressor every iteration
- (Hyper)-Hyperparameters
  - Kernel choice
  - Scaling
  - Noise level
- Sequential (traditionally)
  - Smarter initialization
  - Efficient sampling
- Bounds
  - Not unique to Bayesian Optimization

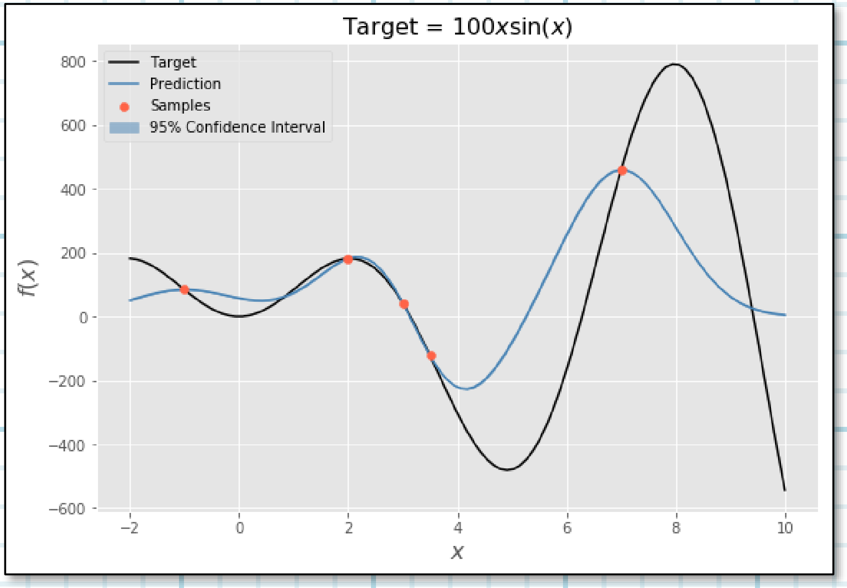
Efficient Initialization

```
rs_params = {'n_estimators':[10, 20, 50, 100],
             'min_samples_split':[2, 5, 10, 20],
             'max_features':[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]}
rs = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=seed),
                        param_distributions=rs_params,
                        n_iter=5,
                        scoring=neg_rmsle,
                        n_jobs=6,
                        cv=3,
                        verbose=3,
                        refit=False,
                        return_train_score=True)

start = time.time()
rs.fit(X_train, y_train)
dump_dill('rs.dill', rs)

params_init = {}
params_init.update({'target': rs.cv_results_['mean_test_score']})
for key in rs_params.keys():
    params_init[key] = [val[key] for val in rs.cv_results_['params']]

params = {'n_estimators':(10,100), 'min_samples_split':(2,20), 'max_features':(0.5,1.0)}
bo = BayesianOptimization(score_model, pbounds=params, verbose=1)
bo.initialize(params_init)
bo.maximize(init_points=0, n_iter=10, acq='ucb')
end = time.time()
print('Time to perform Bayesian optimization: %0.2fs' % (end - start))
bo_time = end - start
```

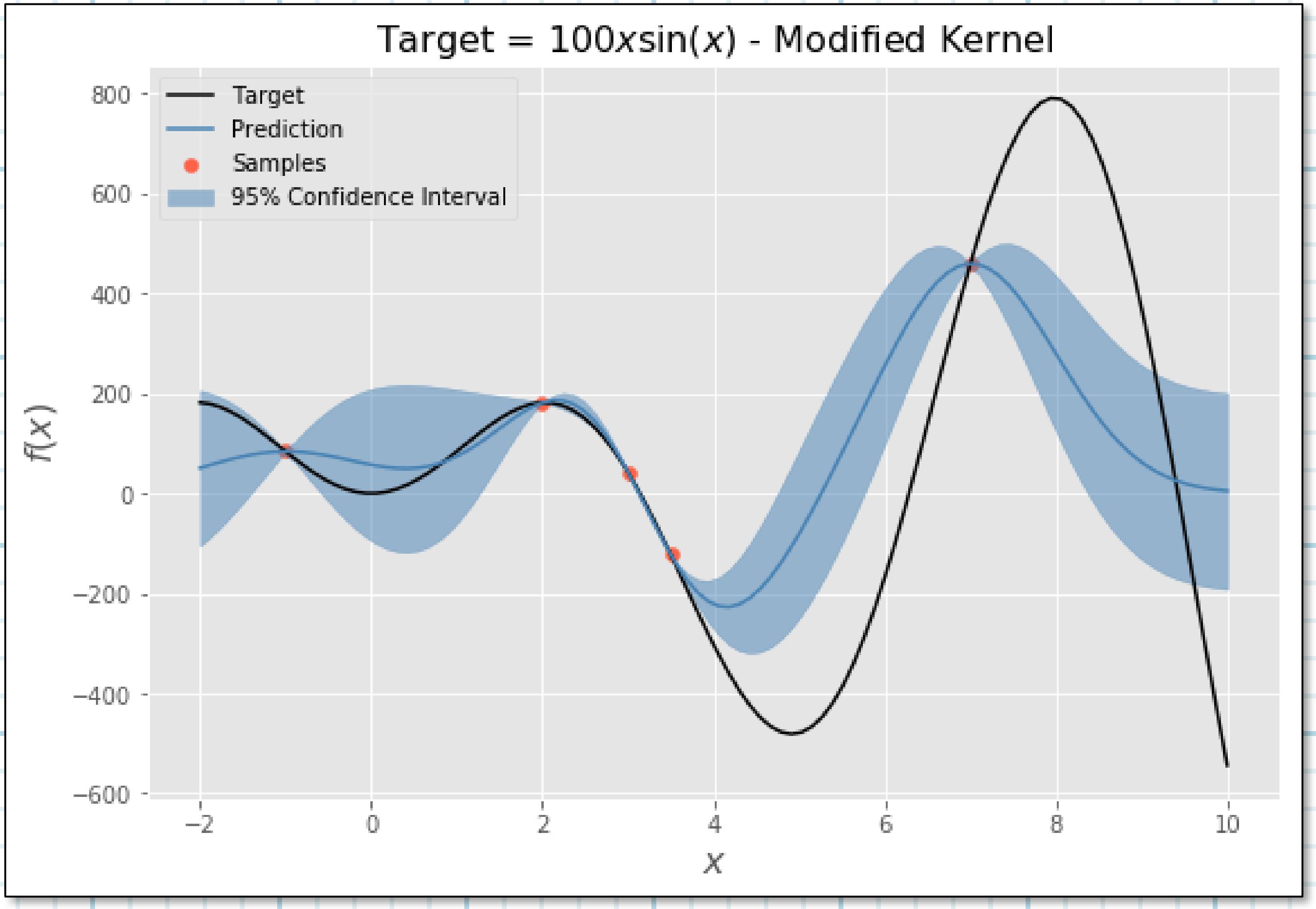
Scaling issues

- Large variances in target values greatly affect standard deviation estimates



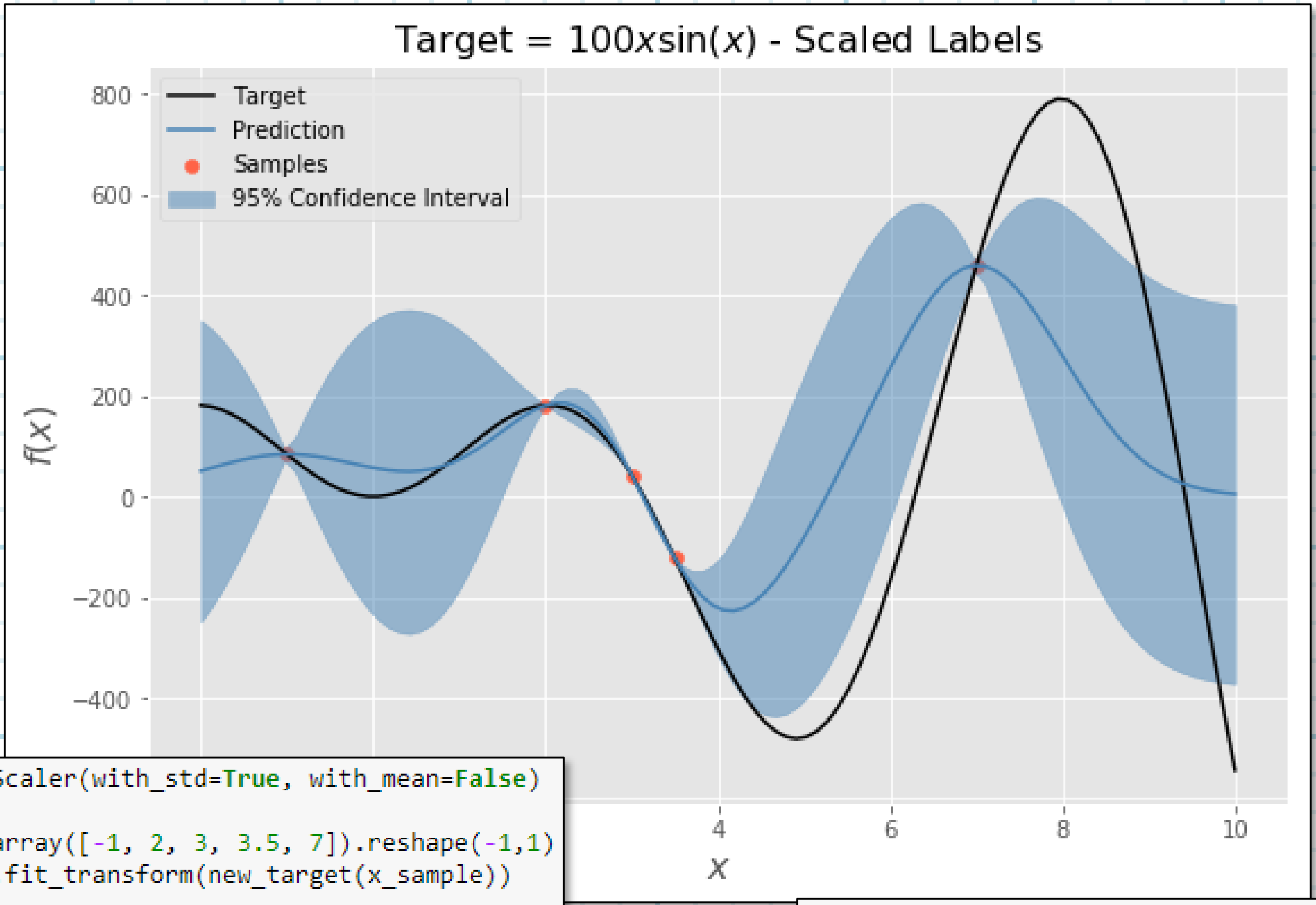


Scaling issues




```
new_kernel = kernels.ConstantKernel(10000.0, constant_value_bounds='fixed')*kernels.RBF(1, length_scale_bounds='fixed')
gp = GaussianProcessRegressor(kernel=new_kernel)
gp.fit(x_sample, y_sample)
```

Scaling issues




```
scl = StandardScaler(with_std=True, with_mean=False)

x_sample = np.array([-1, 2, 3, 3.5, 7]).reshape(-1,1)
y_sample = scl.fit_transform(new_target(x_sample))

gp = GaussianProcessRegressor()
gp.fit(x_sample, y_sample)
```

```
mu, std = gp.predict(x_all, return_std=True)
mu = scl.inverse_transform(mu)
std = scl.inverse_transform(std.reshape(-1,1)).ravel()
```

Modified Target Functions

- Add additional constraints to the scoring method (target)
- Mercari challenge limitation
  - All training must be completed within a Kaggle kernel

```python
def score_model(**params):
    model = RandomForestRegressor(n_estimators=int(params['n_estimators']),
                                  min_samples_split=int(params['min_samples_split']),
                                  max_features=params['max_features'],
                                  n_jobs=1)
    start = time.time()
    scores = cross_val_score(model, X_train, y_train, scoring=neg_rmsle, cv=3)
    end = time.time()
    dt = end - start
    # Remember, model was trained three times on 2/3 of the training data
    if 0.5*dt > 3300: # Add some headroom for computation time variance
        result = -100
    else:
        result = scores.mean()
    return result
```

NASHVILLE TECHNOLOGY COUNCIL

VACO

Questions?

Jason King

jkkphys@gmail.com

Resources

- Machine Learning: Nando de Freitas @ UBC
  - Introduction to Gaussian processes
  - Regression with Gaussian processes
  - Bayesian optimization and multi-armed bandits
- Gaussian Processes for Machine Learning
  - Carl Edward Rasmussen and Christopher K. I. Williams
- A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning
  - Eric Brochu, Vlad M. Cora, and Nando de Freitas
- Scikit-learn: Gaussian Processes