Contents


```
% ECE414 - Bayesian Machine Learning
% Authors   : Junbum Kim, Andy Jeong
% Project 4 : Gaussian Process Regression
% Date      : November 6, 2019
% Reference : Pattern Recognition and Machine Learning by C. M. Bishop (2006)
close all; clear all; clc;   % clear workspace variables
rng('default');              % for reproducibility
```

Equations

1 Gaussian Process Regression

Minimize regularized sum-of-squares error function

$$J(w) = \frac{1}{2}\Sigma_{n=1}^{N} w^T \phi(x_n) - t_n{}^2 + \frac{\lambda}{2} w^T w, \text{ where } \lambda \geq 0 \text{ (Eq. 6.2)}$$

Substitute $\mathrm{w} = \Phi^T a$ into $J(w)$ and Define Gram Matrix $K = \Phi\Phi^T$

Minimize the error function

$$J(a) = \frac{1}{2} a^T K K a - a^T K t + \frac{1}{2} t^T t + \frac{\lambda}{2} a^T K a \text{ (Eq. 6.7)}$$

Parameter a that minimizes J(a):
(t: target, λ: regularization parameter)

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I_N})^{-1} \mathbf{t} \text{ (Eq. 6.8)}$$

Prediction for a new input x:

$$y(x) = w^T \phi(x) = a^T \Phi \phi(x) = k(x_n, x)^T (K + \lambda I_N)^{-1} t \text{ (Eq 6.9)}$$

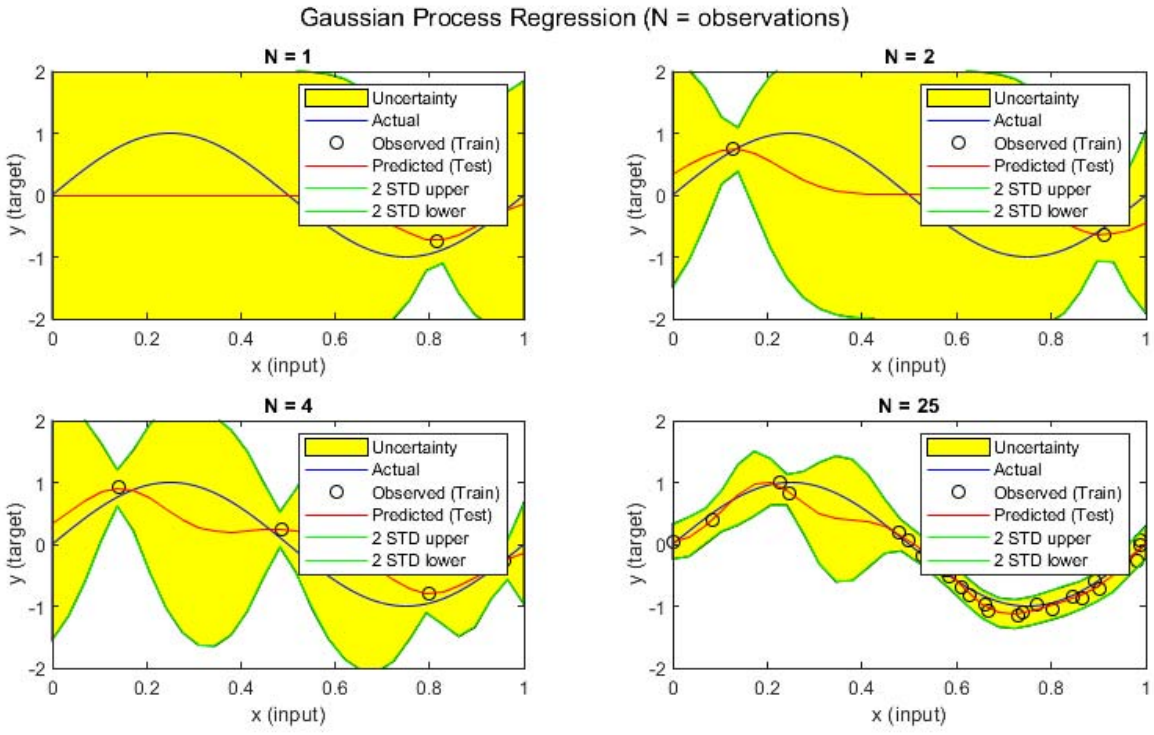
Gram Matrix K (where $\phi(x) \sim \mathrm{N}(\mathrm{x} \mid 0, \sigma)$)

$$K_{nm} = k(x_n, x_m) = \phi(x_n)^T \phi(x_m) \text{ (Eq 6.6)}$$

Bayesian Linear Regression with Gaussian Basis

```
sigma_kernel = 1e-1;          % kernel variance
sigma_noise = 1e-1;           % noise variance
lambda = 1e-2;                % regularization parameter
test_N = 30;                  % number of test examples
sampling_rate = 1000;

% mapping function
f = @(x) sin(2 * pi * x);

% Gaussian basis kernel function
kernel_fcn = @(a, b) exp(-(a - b).^2/(2*sigma_kernel^2));

% actual sinusoidal
X = linspace(0,1,sampling_rate);
y = f(X);

i = 1; % iteration for subplots
% make figure larger for clearer view
figure('Renderer', 'painters', 'Position', [100 100 900 500]);
figure(1);
% iterate over various number of training examples
for train_N = 1:25
    % split into train and test sets
```

```
    % target has Gaussian noise ~ N(zero-mean, sigma_noise)
    train_x = rand(train_N,1);
    test_x = linspace(0,1,test_N)'; % for continuous line plotting
    train_t = f(train_x) + normrnd(0, sigma_noise, [train_N,1]);

    % kernels (with noise)
    K11 = kernel_fcn(train_x, train_x') + lambda * eye(train_N);
    K12 = kernel_fcn(train_x, test_x');
    K22 = kernel_fcn(test_x, test_x') + lambda * eye(test_N);

    % hyperparameter learning to minimize the  error function
    a = pinv(K11) * train_t;

    % make prediction
    mu = K12' * a;

    % covariance and standard deviation
    covariance = K22 - K12' * pinv(K11) * K12;
    std_dev = sqrt(diag(covariance));

    % fill in space for uncertainty
    upper = mu + 2*std_dev;
    lower = mu - 2*std_dev;
    x = [test_x', fliplr(test_x')];
    space = [upper', fliplr(lower')];

    % plot predictions for different observations
    if train_N == 1 || train_N == 2 || train_N == 4 || train_N == 25
        subplot(2,2,i);
        fill(x, space, 'y'); hold on;
        plot(X, y,            'b-', ...     % actual function
            train_x, train_t, 'ko', ...     % training points
            test_x, mu,       'r-', ...     % predicted (test) points
            test_x, upper,    'g', ...      % upper boundary
            test_x, lower,    'g' ...       % lower boundary
        );
        xlabel('x (input)'); ylabel('y (target)');
        legend('Uncertainty','Actual','Observed (Train)','Predicted (Test)', '2 STD upper','2 STD lower');
        title(sprintf('N = %d',train_N)); xlim([0 1]); ylim([-2 2]);
        i = i + 1;
    end
end
% set main title for the figures
t = suptitle('Gaussian Process Regression (N = observations)');
set(t, 'FontSize', 12, 'Position', get(t,'Position')-[0 0.01 0], ...
    'FontWeight', 'normal');
```



```
ECE414 - Bayesian Machine Learning
Authors   : Junbum Kim, Andy Jeong
Project 4 : Support Vector Machine
Date      : November 6, 2019

# -*- coding: utf-8 -*-
"""
proj4_svm.ipynb
"""
import time
import numpy as np
import pandas as pd
```

```
from sklearn.svm import SVC
# define types of classifiers (linear, rbf kernel SVM) in a dictionary
# the hyperparameter ranges is set to logspace(-3,-3,7) for both C and gamma
dict_classifiers = {
    "Linear Kernel SVM":
            {'classifier': SVC(),
                 'params': [ {'C': np.logspace(-3,3,7),
                              'gamma': np.logspace(-3,3,7),
                              'kernel': ['linear']
                             }
                           ]
            },
    "RBF Kernel SVM":
             {'classifier': SVC(),
                'params': [ {'C': np.logspace(-3,3,7),
                                          'gamma': np.logspace(-3,3,7),
                                                 'kernel': ['rbf']
                             }
                           ]
             }
}

# specify the file for Circles dataset
from scipy.io import loadmat
file1 = 'mlData.mat'
```

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV, train_test_split

# load circles x, y data from the provided file
# -- 2 attributes, 400 samples, 2 classes
class Circles():
        def __init__(self, filename = file1):
                data = loadmat(filename)
```

```
                circles = data["circles"]
                # random 80-20 train-test split
                self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(circles["x"][0,0], np.ravel(circles["y"][0,0]), test_size = 0.2,
random_state = 42)

# Wine dataset from UCI Machne Learning Repository
# -- 13 attributes, 178 samples, 3 classes
# https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data
class Wine():
    def __init__(self, raw_wine_data):
        # format the data
        df = pd.DataFrame(data=raw_data['data'],columns=raw_data['feature_names'])
        df['target'] = raw_data['target']
        df['class'] = df['target'].map(lambda index: raw_data['target_names']
[index])
        # random 80-20 train-test split
        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(raw_data['data'],raw_data['target'], test_size=0.2,
random_state=42)

# initialize a support vector machine model
class SVM():
        def __init__(self):
                self.model = SVC()

        def default_model(self, X_train, X_test, Y_train, Y_test):
                # fit into default model
                self.model.fit(X_train, Y_train)

                # predict using default model
                pred = self.model.predict(X_test)
                print("-----------[default model test]-----------")
                print(classification_report(Y_test, pred))

        def tuned_model(self, X_train, X_test, Y_train, Y_test):
                count = 0
                num_classifiers = len(dict_classifiers.keys())

        # iterate through all defined classifiers with the specified search space
                for key, classifier in dict_classifiers.items():
                        df_results = pd.DataFrame(data=np.zeros(shape=(1,4)),
columns = ['classifier', 'train_score', 'test_score', 'training_time'])

                        # perform grid search (and record time taken)
                        t_start = time.clock()
                        self.tunedModel = GridSearchCV(classifier['classifier'],
classifier['params'], refit=True, cv = 10, scoring = 'accuracy', n_jobs = -1)
                        estimator = self.tunedModel.fit(X_train, Y_train)
                        t_end = time.clock()
                        t_diff = t_end - t_start

                        train_score = estimator.score(X_train, Y_train)
                        test_score = estimator.score(X_test, Y_test)
```

```
                        df_results.loc[count,'classifier'] = key
                        df_results.loc[count,'train_score'] = train_score
                        df_results.loc[count,'test_score'] = test_score
                        df_results.loc[count,'training_time'] = t_diff
                        # display trained results
                        print("-----------[tuned model train] Classifier:",key,"--
---------")
                        print("Training Time ({c}): {f:.2f} sec".format(c=key,
f=t_diff))
                        print("Best Parameters:", self.tunedModel.best_params_)
                        print(df_results.iloc[count,1:])

                        # predict and display test results
                        pred = self.tunedModel.predict(X_test)
                        print("-----------[tuned model test] Classifier:",key,"---
--------")
                        print(classification_report(Y_test, pred))
                        print("accuracy: ", round(accuracy_score(Y_test, pred),4))
                return pred
```

```
# run the models and see the performances
print("====================Dataset: Circles====================")
# part 2.a - classify circles data
data1 = Circles()
model = SVM()

# fit data to the model with the default parameters
model.default_model(data1.X_train, data1.X_test, data1.y_train, data1.y_test)

# apply grid search
circles_pred = model.tuned_model(data1.X_train, data1.X_test, data1.y_train,
data1.y_test)
```

```
====================Dataset: Circles====================
-----------[default model test]-----------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        44
           1       1.00      1.00      1.00        36

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80

-----------[tuned model train] Classifier: Linear Kernel SVM -----------
Training Time (Linear Kernel SVM): 3.54 sec
Best Parameters: {'C': 1.0, 'gamma': 0.001, 'kernel': 'linear'}
train_score       0.5875
```

```
test_score           0.6
training_time    7.38116
Name: 0, dtype: object
-----------[tuned model test] Classifier: Linear Kernel SVM -----------
              precision    recall  f1-score   support

           0       0.61      0.77      0.68        44
           1       0.58      0.39      0.47        36

    accuracy                           0.60        80
   macro avg       0.60      0.58      0.57        80
weighted avg       0.60      0.60      0.58        80

accuracy:  0.6
-----------[tuned model train] Classifier: RBF Kernel SVM -----------
Training Time (RBF Kernel SVM): 0.71 sec
Best Parameters: {'C': 0.1, 'gamma': 1.0, 'kernel': 'rbf'}
train_score             1
test_score              1
training_time    0.677268
Name: 0, dtype: object
-----------[tuned model test] Classifier: RBF Kernel SVM -----------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        44
           1       1.00      1.00      1.00        36

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80

accuracy:  1.0
```

```
from sklearn import datasets

print("\n")
print("=====================Dataset: Wine=====================")
# part 2.b - classify wine data
raw_data = datasets.load_wine()
data2 = Wine(raw_data)
model = SVM()

# fit data to the model with the default parameters
model.default_model(data2.X_train, data2.X_test, data2.y_train, data2.y_test)

# apply grid search
wine_pred = model.tuned_model(data2.X_train, data2.X_test, data2.y_train,
data2.y_test)
```

```
====================Dataset: Wine====================
-----------[default model test]-----------
              precision    recall  f1-score   support

           0       1.00      0.07      0.13        14
           1       0.41      1.00      0.58        14
           2       1.00      0.12      0.22         8

    accuracy                           0.44        36
   macro avg       0.80      0.40      0.31        36
weighted avg       0.77      0.44      0.33        36


-----------[tuned model train] Classifier: Linear Kernel SVM -----------
Training Time (Linear Kernel SVM): 18.14 sec
Best Parameters: {'C': 0.1, 'gamma': 0.001, 'kernel': 'linear'}
train_score      0.964789
test_score              1
training_time     19.6669
Name: 0, dtype: object
-----------[tuned model test] Classifier: Linear Kernel SVM -----------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36

accuracy:  1.0
-----------[tuned model train] Classifier: RBF Kernel SVM -----------
Training Time (RBF Kernel SVM): 0.64 sec
Best Parameters: {'C': 100.0, 'gamma': 0.001, 'kernel': 'rbf'}
train_score             1
test_score       0.833333
training_time    0.597421
Name: 0, dtype: object
-----------[tuned model test] Classifier: RBF Kernel SVM -----------
              precision    recall  f1-score   support

           0       0.80      0.86      0.83        14
           1       0.92      0.86      0.89        14
           2       0.75      0.75      0.75         8

    accuracy                           0.83        36
   macro avg       0.82      0.82      0.82        36
weighted avg       0.84      0.83      0.83        36
```

```
accuracy:  0.8333
-------------------------------------------
Summary (accuracy)          Circles    Wine
Linear Kernel SVM:            0.6      1.0
RBF Kernel SVM:               1.0      0.83

Comparison to Project 3 - Linear Classification:
                  Circles
Generative:         1.0
IRLS :              1.0
-------------------------------------------
```

```
# Receiver Operating Characteristic Curves
%matplotlib inline
from sklearn.metrics import roc_curve, roc_auc_score

lw = 2
# ROC for Circles data: 2 classes
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, _ = roc_curve(data1.y_test, circles_pred)
auc = roc_auc_score(data1.y_test, circles_pred)
plt.plot(fpr,tpr,label="AUC=" + str(auc))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, label='reference line', linestyle='-
-')
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Plot for Circles Data')
plt.legend(loc="lower right")
plt.show()


from sklearn.preprocessing import label_binarize

# ROC for Wine data: 3 clases
y = label_binarize(data2.y_test, classes=[0, 1, 2])
pred = label_binarize(wine_pred, classes=[0, 1, 2])
n_classes = y.shape[1]

# Compute ROC curve and ROC area for each class
fpr, tpr, auc, roc_auc = dict(), dict(), dict(), dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y[:, i], pred[:, i])
    auc[i] = roc_auc_score(y[:, i], pred[:, i])
# plot ROC curves for each class
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], linestyle='-', lw=lw, label='Class = %d, AUC= %0.2f'
% (i, auc[i]))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, label='reference line', linestyle='-
```

```
-')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Plot for Wine Data')
plt.legend(loc="lower right")
plt.show()
```