Statistical and Machine Learning (01.113)
Homework 3

Due on 18 MAR, 3 PM

Problem 1

Perform singular value decomposition (SVD) on the following matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = U\Sigma V^T.$$

Solution. We compute AA^T and get

$$AA^T = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

The eigenvalues, in order of decreasing magnitude, are $\lambda_1 = 3$ and $\lambda_2 = 1$, and the corresponding singular values are $\sigma_1 = \sqrt{3}, \sigma_2 = 1$. This means that

$$\Sigma = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The corresponding orthonormal eigenvector of λ_1 is $u_1 = \frac{1}{\sqrt{2}}[1,1]^T$, and that of λ_2 is $u_2 = \frac{1}{\sqrt{2}}[1,-1]^T$. This gives

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

To obtain V, we compute

$$A^T A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

which we know will have eigenvalues 3, 1 and 0. The corresponding orthonormal eigenvectors are $v_1 = \frac{1}{\sqrt{6}}[1,2,1]^T$, $v_2 = \frac{1}{\sqrt{2}}[1,0,-1]^T$ and $v_3 = \frac{1}{\sqrt{3}}[1,-1,1]^T$ respectively. Hence

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix}^T.$$

■

Problem 2

Assume that $f_x \sim \mathcal{N}(\mu_x, \sigma_x)$, and f^* is some constant. Show that

$$\mathbb{E}\left[\max\left\{f_x - f^*, 0\right\}\right] = \sigma_x\left[\gamma_x \mathbf{\Phi}(\gamma_x) + \phi(\gamma_x)\right],$$

where

$$\gamma_x = \frac{\mu_x - f^*}{\sigma_x}$$

and $\mathbf{\Phi}$, ϕ respectively denote the cdf and pdf of the standard normal distribution.

Hint: Use the symmetries of $\mathbf{\Phi}$ and ϕ.

Solution.

$$\begin{aligned}
\mathbb{E}\left[\max\left\{f(x) - f^*, 0\right\}\right] &= \frac{1}{\sigma_x\sqrt{2\pi}} \int_{f^*}^{\infty} (f_x - f^*)\, e^{-\frac{(f_x-\mu_x)^2}{2\sigma_x^2}}\, \mathrm{d}f_x \\
&= \frac{1}{\sqrt{2\pi}} \int_{\frac{f^*-\mu_x}{\sigma_x}}^{\infty} (\sigma_x y_x + \mu_x - f^*)\, e^{-\frac{y_x^2}{2}}\, \mathrm{d}y_x \\
&= \frac{\sigma_x}{\sqrt{2\pi}} \left[-e^{-\frac{y_x^2}{2}}\right]_{\frac{f^*-\mu_x}{\sigma_x}}^{\infty} + (\mu_x - f^*)\left(1 - \mathbf{\Phi}\left(-\gamma_x\right)\right) \\
&= \sigma_x \phi(-\gamma_x) + (\mu_x - f^*)\left(1 - \mathbf{\Phi}\left(-\gamma_x\right)\right) \\
&= \sigma_x \gamma_x \mathbf{\Phi}\left(\gamma_x\right) + \sigma_x \phi(\gamma_x).
\end{aligned}$$

■

Problem 3

Let $X = (X_1, X_2, X_3, X_4)$ be the random variable with $0 \le X_i \le 4$ for $i = 1, 2, 3, 4$, and density function

$$f_X(x) = \frac{1}{Z_1}\exp\{x_1x_2x_3x_4\}, \quad Z_1 = \int_0^4\int_0^4\int_0^4\int_0^4 \exp\{x_1x_2x_3x_4\}.$$

Let $Y = (Y_1, Y_2, Y_3, Y_4)$ be the random variable with $0 \le Y_i \le 4$ for $i = 1, 2, 3, 4$, and density function

$$f_Y(y) = \frac{1}{Z_2}\exp\{y_1y_2y_3 + y_1y_2 + y_4\}, \quad Z_2 = \int_0^4\int_0^4\int_0^4\int_0^4 \exp\{y_1y_2y_3 + y_1y_2 + y_4\}.$$

Let G be the graph in Figure 1.

(a) Show that G is a Markov random field for Y but not X.

(b) Write down a conditional independence relationship for Y that can be read off the graph G due to the global Markov property. (Note that Y_i is the random variable associated with node i.)

(c) Write down the conditional independence relationship for Y that follows from the local Markov property for node 1.

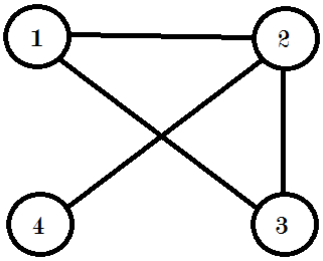

Figure 1: Graph G

(d) Draw the appropriate arrows on G for the Bayesian network associated with the following set of structural equations:

$$\begin{aligned} Y_1 &= -2Y_3 + \epsilon_1, \\ Y_2 &= 4Y_1 + Y_3 + \epsilon_2, \\ Y_3 &= \epsilon_3, \\ Y_4 &= -3Y_2 + \epsilon_4, \\ \epsilon_i &\sim \mathcal{N}(0,1). \end{aligned}$$

Solution. (a) Y factorizes according to the cliques of G, which are $\{1,2,3\}$ and $\{2,4\}$, as

$$f_Y(y) = \frac{1}{Z_2}\exp\{y_1y_2y_3 + y_1y_2\} \cdot \exp\{y_4\}.$$

X does not factorize according to the cliques of G, and $X_4 \not\perp X_1, X_3 \,|\, X_2$, so it fails the global Markov property.

(b) $Y_4 \perp Y_1, Y_3 \mid Y_2$ (there are other conditional independence relationships as well).

(c) $Y_1 \perp Y_4 \mid Y_2, Y_3$.

(d) Refer to Figure 2.

■

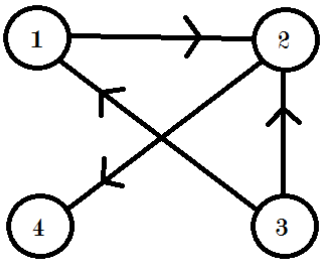

Figure 2: Directed Graph G

Problem 4

In this problem, we will implement Support Vector Machines (SVMs) for classifying two datasets. We start by importing the required packages and modules.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.svm import SVC
from sklearn.datasets.samples_generator import make_blobs, make_circles
```

The "make_blobs" and "make_circles" functions from "sklearn.datasets" can be invoked to generate data for the first and second example, respectively.
The following function will be used later to plot the decision boundary, margins and the support vectors.

```
def plot_svc_decision(model, ax=None):
        if ax is None:
                ax = plt.gca()
        xlim = ax.get_xlim()
        ylim = ax.get_ylim()

        # create grid to evaluate model
        x = np.linspace(xlim[0], xlim[1], 30)
        y = np.linspace(ylim[0], ylim[1], 30)
        Y, X = np.meshgrid(y, x)
        xy = np.vstack([X.ravel(), Y.ravel()]).T
        P = model.decision_function(xy).reshape(X.shape)

        # plot decision boundary and margins
        ax.contour(X, Y, P, colors='k', levels=[-1, 0, 1],
                alpha=0.5, linestyles=['--', '-', '--'])

        # plot support vectors
        ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
                s=300, linewidth=1, edgecolors='black', facecolors='none')

        ax.set_xlim(xlim)
        ax.set_ylim(ylim)
```

(a) Use the following lines of code to plot the first dataset.

```
X, y = make_circles(100, factor=.1, noise=.1)
fig1 = plt.figure()
ax1 = fig1.add_subplot(111)
ax1.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')
```

Use SVC to construct a support vector machine (you will have to specify a kernel and the regularization parameter C) to classify this dataset, then use $fit(X, y)$ to feed in the data and labels. Show your results using the *plot_svc_decision* function.

(b) Now generate and plot the second dataset.

```
X, y = make_blobs(n_samples=100, centers=2, random_state=0, cluster_std=1.0)
fig2 = plt.figure(figsize=(16, 6))
fig2.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

ax2 = fig2.add_subplot(121)
ax2.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')

ax3 = fig2.add_subplot(122)
ax3.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')
```

Your task here is to classify the dataset using different values of the regularization parameter C to understand soft margins in SVM. Indicate clearly what values of C you are using, and plot your results with *plot_svc_decision* using $ax2$ for one model and $ax3$ for the other.

Upload the final script in your Dropbox folder and name it as "**HW3_4.py**'.

Solution.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.svm import SVC
from sklearn.datasets.samples_generator import make_blobs, make_circles

def plot_svc_decision(model, ax=None):
        if ax is None:
                ax = plt.gca()
        xlim = ax.get_xlim()
        ylim = ax.get_ylim()

        # create grid to evaluate model
        x = np.linspace(xlim[0], xlim[1], 30)
        y = np.linspace(ylim[0], ylim[1], 30)
        Y, X = np.meshgrid(y, x)
        xy = np.vstack([X.ravel(), Y.ravel()]).T
        P = model.decision_function(xy).reshape(X.shape)

        # plot decision boundary and margins
        ax.contour(X, Y, P, colors='k', levels=[-1, 0, 1],
                alpha=0.5, linestyles=['--', '-', '--'])

        # plot support vectors
        ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
                s=300, linewidth=1, edgecolors='black', facecolors='none')

        ax.set_xlim(xlim)
        ax.set_ylim(ylim)

# (a)
X, y = make_circles(100, factor=.1, noise=.1)
fig1 = plt.figure()
```

```
ax1 = fig1.add_subplot(111)
ax1.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')

model1 = SVC(kernel='rbf', C=1E6)
model1.fit(X, y)
plot_svc_decision(model1, ax1)

# (b)

X, y = make_blobs(n_samples=100, centers=2, random_state=0, cluster_std=1.0)
fig2 = plt.figure(figsize=(16, 6))
fig2.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

ax2 = fig2.add_subplot(121)
ax2.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')

ax3 = fig2.add_subplot(122)
ax3.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='seismic')

ax2.set_title('C={0:.1f}'.format(10), size=14)
ax3.set_title('C={0:.1f}'.format(0.1), size=14)

model2 = SVC(kernel='linear', C=10)
model2.fit(X, y)

model3 = SVC(kernel='linear', C=0.1)
model3.fit(X, y)

plot_svc_decision(model2, ax2)
plot_svc_decision(model3, ax3)

plt.show()
```

■

Problem 5

We will use Gaussian process regression to approximate the following function and find its global maximum (i.e., $y^* \approx 1$ at $(x_1^*, x_2^*) = (2.25, 2.65)$).

$$y = 0.5 \exp\left\{-0.5\left[(x_1 + 1.25)^2 + (x_2 + 1.75)^2\right]\right\} + \exp\left\{-0.5\left[(x_1 - 2.25)^2 + (x_2 - 2.65)^2\right]\right\}$$

First, we need to import the required packages from NumPy, Matplotlib, SciPy and Scikit-Learn.

```
import numpy as np
from mpl_toolkits import mplot3d
from matplotlib import cm
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.stats import norm
```

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, Matern, RationalQuadratic,
        ExpSineSquared, DotProduct
```

Next, we define our "true" and "noisy" functions as follows:

```
def func(x1, x2):
        return 0.5 * np.exp(-0.5 * ((x1 + 1.25)**2 + (x2 + 1.75)**2)) +
                np.exp(-0.5 * ((x1 - 2.25)**2 + (x2 - 2.65)**2))

def noisy_func(x1, x2):
        output = func(x1, x2)
        noise = np.random.normal(0, 0.1, np.shape(output))
        return output + noise
```

(a) Write code for the following acquisition functions: *probability of improvement, expected improvement* and *upper confidence bound*.

(b) Define a query function using "scipy.optimize.minimize" and the acquisition function of your choice. Your function should look like this:

```
def query(opt_val, gp):
        def obj(x):
                # do Gaussian process prediction
                return -my_acquisition_function(.)

        res = minimize(obj, # initial point)
        return res.x
```

Use the next few lines of code to visualize the true function.

```
res = 50
lin = np.linspace(-5, 5, res)
meshX, meshY = np.meshgrid(lin, lin)
meshpts = np.vstack((meshX.flatten(), meshY.flatten())).T

def add_subplot(gp, subplt):
        mu = gp.predict(meshpts, return_std=False)
        ax = fig.add_subplot(2, 5, subplt, projection='3d')
        ax.plot_surface(meshX, meshY, np.reshape(mu, (50, 50)),
        rstride=1, cstride=1, cmap=cm.jet, linewidth=0, antialiased=False)

if __name__ == '__main__':
        true_y = func(meshX, meshY)

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_surface(meshX, meshY, true_y, rstride=1, cstride=1, cmap=cm.jet,
         linewidth=0, antialiased=False)
        plt.title('True_function')
        plt.show()
```

```
fig = plt.figure(figsize=plt.figaspect(0.5))
```

(c) Initialize 4 random points and evaluate the noisy function at these points:

```
xi = #Complete here
yi = #Complete here
```

(d) Initialize the Gaussian process regressor with a kernel of your choice:

```
gp = #Complete here
```

Finally, use the following code to perform Bayesian optimization and plot the evolution of the mean of the Gaussian process over 10 iterations:

```
for i in range(10):
        gp.fit(xi, yi)

        # find the current optimal value and its location
        opt_val = ...
        opt_x = ...

        print('Best value: ', opt_val)
        print('at ', opt_x)

        next_x = query(opt_val, gp)

        # add next_x to the list of data points
        xi = ...

        next_y = noisy_func(xi[-1][0], xi[-1][1]).reshape(1)

        # add next_y to the list of observations
        yi = ...

        add_subplot(gp, i + 1)

plt.show()
```

How close are you to the optimal value after 10 iterations? Upload the final script in your Dropbox folder and name it as "**HW3_5.py**'.

Solution.

```
import numpy as np
from mpl_toolkits import mplot3d
from matplotlib import cm
import matplotlib.pyplot as plt

from scipy.optimize import minimize
```

```
from scipy.stats import norm

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, Matern, RationalQuadratic,
        ExpSineSquared, DotProduct

def func(x1, x2):
        return 0.5 * np.exp(-0.5 * ((x1 + 1.25)**2 + (x2 + 1.75)**2)) +
                np.exp(-0.5 * ((x1 - 2.25)**2 + (x2 - 2.65)**2))

def noisy_func(x1, x2):
        output = func(x1, x2)
        noise = np.random.normal(0, 0.1, np.shape(output))
        return output + noise

def probability_of_improvement(mu_x, sigma_x, opt_value, kappa=0):
        gamma_x = (mu_x - opt_value - kappa) / sigma_x
        return norm.cdf(gamma_x)

def upper_confidence_bound(mu_x, sigma_x, opt_value, kappa=1.0):
        return mu_x + kappa * sigma_x

def expected_improvement(mu_x, sigma_x, opt_value, kappa=0):
        gamma_x = (mu_x - opt_value - kappa) / sigma_x
        return sigma_x * (gamma_x * norm.cdf(gamma_x) + norm.pdf(gamma_x))

acq = expected_improvement

def query(opt_val, gp):
        def obj(x):
                x = np.array(x).reshape(1, 2)
                mu_x, sigma_x = gp.predict(x, return_std=True)
                return -acq(mu_x, sigma_x, opt_val)

        res = minimize(obj, (0, 0))
        return res.x


res = 50
lin = np.linspace(-5, 5, res)
meshX, meshY = np.meshgrid(lin, lin)
meshpts = np.vstack((meshX.flatten(), meshY.flatten())).T

def add_subplot(gp, subplt):
        mu = gp.predict(meshpts, return_std=False)
        ax = fig.add_subplot(2, 5, subplt, projection='3d')
        ax.plot_surface(meshX, meshY, np.reshape(mu, (50, 50)),
                rstride=1, cstride=1, cmap=cm.jet, linewidth=0, antialiased=False)

if __name__ == '__main__':
        true_y = func(meshX, meshY)
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(meshX, meshY, true_y, rstride=1, cstride=1, cmap=cm.jet,
        linewidth=0, antialiased=False)
plt.title('True function')
plt.show()

fig = plt.figure(figsize=plt.figaspect(0.5))

xi = 10 * np.random.rand(4, 2) - 5
yi = noisy_func(xi[:,0], xi[:,1])

gp = GaussianProcessRegressor(kernel=Matern(length_scale_bounds="fixed"))

for i in range(10):
        gp.fit(xi, yi)

        opt_val = np.max(yi)
        opt_x = xi[np.argmax(yi)]

        print('Best value: ', opt_val)
        print('at ', opt_x)

        next_x = query(opt_val, gp)

        xi = np.vstack((xi, next_x))

        next_y = noisy_func(xi[-1][0], xi[-1][1]).reshape(1)
        yi = np.concatenate((yi, next_y))

        add_subplot(gp, i + 1)

plt.show()
```

■