Homework 3: Bayesian ML and Multiclass

Bayesian logistic regression with Gaussian Prior

Bayesian logistic Regression with Gaussian prior.

→ Binary Classification setting

⇒ input space: $\mathcal{X} \in \mathbb{R}^d$

outcome space: $\mathcal{Y}_{\pm} = \{-1, 1\}$

dataset: $\mathcal{D} = ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots, (x^{(n)}, y^{(n)}))$

$$p(y=1 \mid x; w) = 1/(1 + \exp(-x^T w))$$

for logistic reg. in Bayesian setting.
introduce prior $p(w)$ on $w \in \mathbb{R}^d$

Ans. 1

1. posterior density:

$$p(w \mid D) \propto p(D \mid w) \cdot p(w)$$

$$\Rightarrow p(w \mid D) = k \cdot \underbrace{p(D \mid w)}_{\text{likelihood}} \cdot \underbrace{p(w)}_{\text{prior}}$$

$$\Rightarrow \boxed{p(w \mid D) = k \left(\exp\left(-NLL_{D}(w)\right)\right) \cdot p(w)}$$

(where NLL is the negative log likelihood)

`please turn over`

Ans 2.

(2) let $w \sim N(0, \Sigma)$

MAP estimate for w after observing data D

$$\arg\max_w p(w|D) = \arg\max_w p(D|w) \cdot p(w)$$

$$= \arg\max_w \Big[\underbrace{\log\big(p(D|w)\big)}_{\text{log likelihood function for log.reg.}} + \log\big(p(w)\big)\Big]$$

[taking log as it is strictly inc. f^n & makes compute easy]

$$\arg\max_w \Big[\sum_{i=1}^{n} -\frac{1}{2}\Big(y^{(i)}\log\big(1+\exp(-x^{(i)T}w)\big) + (1-y^{(i)})\big(1+\exp(x^{(i)T}w)\big)\Big) + \log(p(w))\Big]$$

$$= \arg\max_w \Big(-NLL_D(w) + \underbrace{\log|2\pi\Sigma|^{-1/2}}_{} - \frac{1}{2}w^T\Sigma^{-1}w\Big)$$

→ can be removed as it is not dep. on w

$$= \arg\min_w \Big(NLL_D(w) + \frac{1}{2}w^T\Sigma^{-1}w\Big) \equiv \arg\min_w \Big(\frac{1}{n}NLL_D(w) + \frac{1}{2n}w^T\Sigma^{-1}w\Big)$$

→ (1)

L_2 regularized logistic regression

$$= \arg\min_w \Big(\underline{\frac{1}{n}\sum_{i=1}^{n}\log\big(1+\exp(-y^{(i)}x^{(i)T}w)\big)} + \lambda\|w\|_2^2\Big)$$

→ can be rewritten as

$$\frac{1}{2n}\Big[\sum_{i=1}^{n} y^{(i)}\log\big(1+\exp(-x^{(i)T}w)\big) + (1-y^{(i)})\log\big(1+\exp(+x^{(i)T}w)\big)\Big]$$

$$= \frac{1}{n}NLL_D(w)$$

$$\Rightarrow \arg\min_w \Big(\frac{1}{n}NLL_D(w) + \lambda\|w\|_2^2\Big)$$

→ (2)

Equating eq'ns ② and ① we see that

$$\frac{1}{2n}\left(w^T\Sigma^{-1}w\right) = \lambda \|w\|_2^2$$

$$= \lambda w^T w$$

$$\Rightarrow \boxed{\Sigma = \frac{1}{2n\lambda} I}$$

please turn over

Ans 3.

3 for ERM to be the mode of the posterior dist.

such that $\Sigma = I$.

$$\Rightarrow \quad I = \frac{1}{2n\lambda} I$$

$$\Rightarrow \boxed{\lambda = 1/2n}$$

please turn over

Coin Flipping with partial observability

Coin flipping with partial observability.

$p(z = H \mid \theta_1) = \theta_1$ // biased coin.

someone reports z to us as x
there is a chance x is incorrect if z = H.
This is denoted by

$$p(x = H \mid z = H, \theta_2) = \theta_2$$

and

$$p(x = T \mid z = T) = 1.$$

please turn over

Ans 4.

4 To prove

$$p(x=H \mid \theta_1, \theta_2) = \theta_1 \theta_2 .$$

Proof :

$$p(x=H \mid \theta_1, \theta_2) = \Big[\overbrace{\big(p(x=H \mid z=H, \theta_1, \theta_2) \cdot p(z=H \mid \theta_1)\big)}^{①} + \underbrace{p(x=H \mid z=T, \theta_1, \theta_2) \cdot p(z=T \mid \theta_1)}_{②} \Big]$$

Evaluating ①

$$\Rightarrow \quad \underline{p(x=H \mid z=H,} \ \theta_1, \theta_2) \cdot p(z=H \mid \theta_1)$$

↳ since it is known that $z=H$ for this case

$\therefore$ $p(x=H \mid z=H, \theta_1, \theta_2) = p(x=H \mid z=H, \theta_2) = \theta_2$

and $p(z=H \mid \theta_1) = \theta_1$

$\therefore$ ① $= \theta_1 \theta_2$

Evaluating ②

$$p(x=H \mid z=T, \theta_1, \theta_2) \cdot p(z=T \mid \theta_1)$$

↳ since x always correctly reports z if $z=T$

$$\left.\begin{array}{l} \therefore \ p(x=H \mid z=T, \theta_1, \theta_2) = 0 \\ \text{and} \ p(z=T \mid \theta_1) = (1-\theta_1) \end{array}\right\} \therefore ② = 0$$

$$\therefore \boxed{p(x=H \mid \theta_1, \theta_2) = ① + ② = \theta_1 \theta_2 .} \quad // \text{Hence proved}$$

please turn over

Ans 5.

5 assuming all events are independent, we get:

$$\mathcal{L}_{D_r}(\theta_1, \theta_2) = \prod_{i=1}^{N_r} p(x \mid \theta_1, \theta_2)$$

resp.

incorrectly reported head + all tails correctly reported.

$$\boxed{\mathcal{L}_{D_r}(\theta_1, \theta_2) = (\theta_1 \theta_2)^{n_h} (1 - \theta_1 \theta_2)^{n_t}}$$

correctly reported H

please turn over

Ans 6.

6. Estimating θ_1 and θ_2 through MLE is not usable as say for a given distribution, we get

$$\mathcal{L}_{D_n}(\theta_1, \theta_2) = (\theta_1 \theta_2)^{n_h} (1 - \theta_1 \theta_2)^{n_t}$$

if we try MLE, we get

$$\arg \max_{\theta_1, \theta_2} \log \mathcal{L}_{D_n}(\theta_1, \theta_2)$$

$$= \arg \max_{\theta_1, \theta_2} n_h \log \theta_1 \theta_2 + n_t \log (1 - \theta_1 \theta_2)$$

$$\frac{\partial \log \mathcal{L}(\theta_1, \theta_2)}{\partial \theta_1} = 0 = \frac{n_h}{\theta_1 \theta_2} \cdot \theta_2 + \frac{n_t}{1 - \theta_1 \theta_2} \cdot (-\theta_2)$$

$\theta_1, \theta_2 \in (0, 1]$
[for this to be differentiable]

$$\Rightarrow \theta_2 (n_h - n_h \theta_1 \theta_2) - n_t \theta_1 \theta_2^2 = 0$$

$$\Rightarrow n_h \theta_2 - n_h \theta_1 \theta_2^2 - n_t \theta_1 \theta_2^2 = 0$$

$$\Rightarrow (\theta_2)(n_h - \theta_1 \theta_2 (n_h + n_t)) = 0$$

$$\theta_1 \theta_2 = \frac{n_h}{n_h + n_t}$$

$$\frac{\partial \log l(\theta_1, \theta_2)}{\partial \theta_2} = 0 = \frac{n_h}{\theta_1 \theta_2} \theta_1 + \frac{n_t}{1 - \theta_1 \theta_2} \cdot (-\theta_1)$$

$$\Rightarrow (\theta_1)(n_h - (n_h + n_t) \theta_1 \theta_2) = 0$$

$$\Rightarrow \theta_1 \theta_2 = \frac{n_h}{n_h + n_t}$$

identical
$\therefore$ we have
2 variables
1 eq^n
non trivial sol^n only possible.

Hence since we get non trivial sol^{n}s for θ_1 and θ_2 we cannot estimate θ_1 and θ_2 using MLE.

please turn over

Ans 7.

7 Assuming all events are independent, we can write the likelihood as

$$L(\theta_1,\theta_2) = p(\mathcal{D}_r, \mathcal{D}_c \mid \theta_1,\theta_2) = p(\mathcal{D}_r \mid \theta_1,\theta_2) \cdot p(\mathcal{D}_c \mid \theta_1,\theta_2)$$

$$= (\theta_1\theta_2)^{n_h} (1-\theta_1\theta_2)^{n_t} (\theta_1)^{c_h} (1-\theta_1)^{c_t}$$

$$\arg\max_{\theta_1,\theta_2} \left(\log L(\theta_1,\theta_2)\right) \Rightarrow$$

$$l(\theta_1,\theta_2) = n_h \log(\theta_1\theta_2) + n_t \log(1-\theta_1\theta_2) + c_h \log\theta_1 + c_t \log(1-\theta_1)$$

Differentiating wrt θ_1 we get:-

$$\frac{\partial l}{\partial \theta_1} = \frac{n_h}{\theta_1\theta_2}\theta_2 + \frac{n_t(-\theta_2)}{1-\theta_1\theta_2} + \frac{c_h}{\theta_1} + \frac{c_t}{1-\theta_1}(-1) = 0$$

$$= \frac{\theta_2}{(\theta_1\theta_2)(1-\theta_1\theta_2)}\left(n_h - \theta_1\theta_2(n_h+n_t)\right) + \frac{1}{\theta_1(1-\theta_1)}\left(c_h - \theta_1(c_h+c_t)\right) = 0$$

$$\Rightarrow 0 = \frac{(n_h - (\theta_1\theta_2)(n_h+n_t))}{\theta_1(1-\theta_1\theta_2)} + \frac{(c_h - \theta_1(c_h+c_t))}{\theta_1(1-\theta_1)} \quad \text{—(1)}$$

Differentiating wrt θ_2 :-

$$\frac{\partial l}{\partial \theta_2} = 0 = \frac{(n_h - (\theta_1\theta_2)(n_h+n_t))}{\theta_2(1-\theta_1\theta_2)} \quad \text{—(2)}$$

solving these eq^{n}s (1) & (2) we get

$$\boxed{\theta_1 = \frac{c_h}{c_h+c_t}}$$

and

$$\boxed{\theta_2 = \frac{n_h}{n_h+n_t} \cdot \frac{c_h+c_t}{c_h}}$$

non trivial solution.

$\therefore$ we can estimate θ_1, θ_2 using MLE.

`please turn over`

Ans 8.

8

$$\theta_1 \sim \text{Beta}(h, t)$$

$$\Rightarrow p(\theta_1) = \theta_1^{h-1}(1-\theta_1)^{t-1}$$

prior is provided to mitigate the overfitting in D_c

$\Rightarrow$ posterior of θ_1 based on clean results D_c

$$p(\theta_1 | D_c) \propto p(D_c | \theta_1) \cdot p(\theta_1)$$

$$\Rightarrow p(\theta_1 | D_c) = K \cdot \theta_1^{c_h}(1-\theta_1)^{c_t} \cdot \theta_1^{h-1}(1-\theta_1)^{t-1}$$

$$= K\, \theta_1^{c_h+h-1}(1-\theta_1)^{c_t+t-1}$$

$$p(\theta_1 | D_c) = \text{Beta}(c_h+h, c_t+t)$$

$$\therefore \arg\max_{\theta_1} p(\theta_1 | D_c) = \boxed{\frac{c_h+h-1}{c_h+h+c_t+t-2} = \theta_{1\,MAP}}$$

$\to$ formula in slides.

$$\therefore \boxed{\theta_2 = \frac{n_h}{n_h+n_t} \cdot \frac{c_h+h+c_t+t-2}{c_h+h-1}}$$

$\to$ [from eq^n 2 of problem 7]

please turn over

ℓ2-regularized empirical risk function for multiclass hinge loss

Ans 9.

9 To prove: $J(w)$ is convex function of w.

Proof:

Let $f_i(w) = \Delta(y_i, y) + \left(\langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle\right)$

$$\left.\langle w, \psi(x_i, y)\rangle = \langle w, \psi(x_i, y)\rangle + 0\right\} \text{affine function}$$

$\Delta(y_i, y) \in \mathbb{R}$

$\langle w, \psi(x_i, y_i)\rangle \in \mathbb{R}$.

$\therefore f_i(w)$ is an affine function of y on w and thus convex.

$$\Rightarrow \max_{y \in \mathcal{Y}} f_y(w) = \max\{f_1(w), f_2(w) \ldots f_k(w)\}$$

is also convex with domain
$\operatorname{dom} f_1 \cap \operatorname{dom} f_2 \cap \ldots \cap \operatorname{dom} f_k$

→ (Rosenberg's notes 3.2.4)

$$\Rightarrow \quad \frac{1}{n}\sum_{i=1}^{n} \max_{y\in Y} f_y(w) \quad \text{is convex} \Big] \rightarrow \text{(Rosenberg's notes 3.2.1)}$$

Now $\lambda \|w\|_2^2$ is convex as $\lambda \in \mathbb{R}$ and $\lambda > 0$ and $\|w\|_2^2$ is convex
→ (Rosenberg's notes 3.1.3)

$$\therefore \quad \lambda \|w\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{y\in Y} f_y(w) \quad \text{is convex}$$ [sum of 2 convex f^ns is convex]

$$\Rightarrow \quad \lambda \|w\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{y\in Y} \left[\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle \right]$$

is a convex function.

Hence Proved.

please turn over

Ans 10.

10

$$\boxed{J(w) = \lambda \|w\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{y\in\mathcal{Y}} \left[\Delta(y_i, y) + \langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle\right]}$$

Let $\boxed{R(w) = \lambda \|w\|_2^2}$ has subgradient

$$\boxed{g_R(w) = 2\lambda w \in \partial R(w)}$$

$$\boxed{\text{Let } H(w) = \frac{1}{n}\sum_{i=1}^{n} \max_{y\in\mathcal{Y}} \left[\Delta(y_i, y) + \langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle\right]}$$

for each i, $\boxed{\hat{y}_i = \arg\max_{y\in\mathcal{Y}} \left[\Delta(y_i, y) + \langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle\right]}$

$\hookrightarrow$ $\hat{y}_i$ is the y that maximizes above exp.

from Assignment 2 we know that

given $f(x) = \max f_k(x)$ $\quad (k = \{1, 2, \dots, n\})$

if k is any index for which

$f_k(x) = f(x)$

then if $g(x) \in \partial f_k(x)$

$\Rightarrow g(x) \in \partial f(x)$

Let $\boxed{f_y(w) = \Delta(y_i, y) + \langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle}$

$\therefore$ subgradient of $f_y(w)$ at w can be given by subgradient of $f_{\hat{y}_i}(w)$

where $\hat{y}_i = \arg\max_{y\in\mathcal{Y}} f_y(w)$

one subgradient of $f_{\hat{y}_i}(w)$ wrt w is $\Rightarrow$

$$\boxed{g_{f_{\hat{y}_i}}(w) = \psi(x_i, \hat{y}_i) - \psi(x_i, y_i)} \in \partial(f(w))$$

$\therefore$ combining all components of $H(w)$ we get

$$\boxed{g_H(w) = \frac{1}{n}\sum_{i=1}^{n} \left(\psi(x_i, \hat{y}_i) - \psi(x_i, y_i)\right)}$$

we can write

$$g_J(w) = g_R(w) + g_H(w)$$

$$\Rightarrow \boxed{g_J(w) = 2\lambda w + \frac{1}{n}\sum_{i=1}^{n}\left(\psi(x_i, \hat{y}_i) - \psi(x_i, y_i)\right)}$$

$$\left\{\text{where } \hat{y}_i = \arg\max_{y \in Y}\left[\Delta(y_i, y) + \langle w, \psi(x_i, y) - \psi(x_i, y_i)\rangle\right]\right\}$$

$$\boxed{g_J(w) \in \partial J(w)}$$

please turn over

Ans 11.

11 stochastic gradient descent will be based on a single Training eg. (x_i, y_i)

$$\Rightarrow \boxed{\nabla\left(J(w, x_i, y_i)\right) = \left(2\lambda w + \left(\psi(x_i, \hat{y}_i) - \psi(x_i, y_i)\right)\right)}$$

please turn over

Ans 12.

12 minibatch gradient will be based on avg of samples in the mini-batch.

$$\partial\left(J\left(w, (x_i, y_i), (x_{i+1}, y_{i+1}) \ldots\ldots (x_{i+m-1}, y_{i+m-1})\right)\right)$$

$$\|$$

$$\boxed{\left(2\lambda w + \frac{1}{m}\sum_{t=i}^{i+m-1}\left(\psi(x_t, \hat{y}_t) - \psi(x_t, y_t)\right)\right)}$$

please turn over

Hinge Loss is a specialized case of Generalized Hinge Loss

Let $\mathcal{Y} = \{-1, 1\}$. Let $\Delta(y, \hat{y}) = \mathbb{1}y \neq \hat{y}$. If $g(x)$ is the score function in our binary classification setting, then define our compatibility function as

$$\begin{aligned} h(x,1) &= g(x)/2 \\ h(x,-1) &= -g(x)/2. \end{aligned}$$

Show that for this choice of h, the multiclass hinge loss reduces to hinge loss:

$$\ell\left(h,(x,y)\right) = \max_{y' \in \mathcal{Y}} \left[\Delta\left(y,y'\right)\right) + h(x,y') - h(x,y)] = \max\left\{0, 1 - yg(x)\right\}$$

In this problem we will work on a simple three-class classification example. The data is generated and plotted for you in the skeleton code.

Hinge loss is a Specialized Case of Generalized Hinge loss

$$\Delta(y,\hat{y}) = \mathbb{1}_{y\neq \hat{y}} = \begin{cases} 0, & y=\hat{y} \\ 1, & y\neq\hat{y} \end{cases}$$

Compatibility function

$$h(x,1) = g(x)/2 \quad ; \quad h(x,-1) = -g(x)/2 .$$

generalized multiclass hinge loss

$$\ell(h,(x,y)) = \max_{y'\in Y} \left[\Delta(y,\hat{y}) + h(x,y') - h(x,y)\right]$$

Case 1 : correct classification $(y'=y)$

$$\Delta(y,y') = 0$$

$$h(x,y) = h(x,y')$$

$$\Rightarrow \Delta(y,y') + h(x,y') - h(x,y) = 0$$

Case 2 : incorrect classification. $(\hat{y}\neq y)$

$$\Rightarrow \Delta(y,\hat{y}) = 1$$

Case 2a : $y=1, \hat{y}=-1$

$$\Rightarrow \Delta(y,\hat{y}) + h(x,y') - h(x,y)$$

$$= 1 + \frac{-g(x)}{2} - \frac{g(x)}{2}$$

$$= 1 - g(x)$$

Case 2b : $y=-1, \hat{y}=1$

$$\Rightarrow \Delta(y,\hat{y}) + h(x,y') - h(x,y)$$

$$= 1 + g(x).$$

We can combine cases 2a and 2b to get a gen. expr. for case 2 as follows:

Case 2:

$= 1 - y \cdot g(x)$

$\therefore$

$$l(h, (x, y)) = \max_{y \in Y} [\Delta(y, y') + h(x, y') - h(x, y)]$$

$$= \max(0, 1 - y \cdot g(x)) \quad \Big\} \text{ taking the max. of case 1 and case 2}$$

Hence Proved.

please turn over

One-vs-All (also known as One-vs-Rest)

Ans 13.

- ```
  def fit(self, X, y=None):
      #Your code goes here
      for i in range(self.n_classes):
          class_i_labels = np.where(y == i, 1, 0)
          self.estimators[i].fit(X, class_i_labels)
      self.fitted = True
      return self
  ```

- ```
  def decision_function(self, X):
      if not self.fitted:
          raise RuntimeError("You must train classifer \
              before predicting data.")

      if not hasattr(self.estimators[0], "decision_function"):
          raise AttributeError(
              "Base estimator doesn't have a\
                   decision_function attribute.")
      scores = np.empty((X.shape[0], self.n_classes))
      for i in range(self.n_classes):
          class_i_predicted_labels =
  self.estimators[i].decision_function(X)
          scores[:, i] = class_i_predicted_labels
      return scores
  ```

- ```
  def predict(self, X):
      """
      Predict the class with the highest score.
      @param X: array-like, shape = [n_samples,n_features] input data
      @returns array-like, shape = [n_samples,] the predicted classes
  for each input
      """
      scores = self.decision_function(X)
      return np.argmax(scores, axis = 1)
  ```

`please turn over`
  ```

Ans 14.

- ```
  #Here we test the OneVsAllClassifier
  from sklearn import svm
  svm_estimator = svm.LinearSVC(loss='hinge', fit_intercept=False, C=200)
  clf_onevsall = OneVsAllClassifier(svm_estimator, n_classes=3)
  clf_onevsall.fit(X,y)

  for i in range(3) :
      print("Coeffs %d"%i)
      print(clf_onevsall.estimators[i].coef_) #Will fail if you haven't implemented fit yet

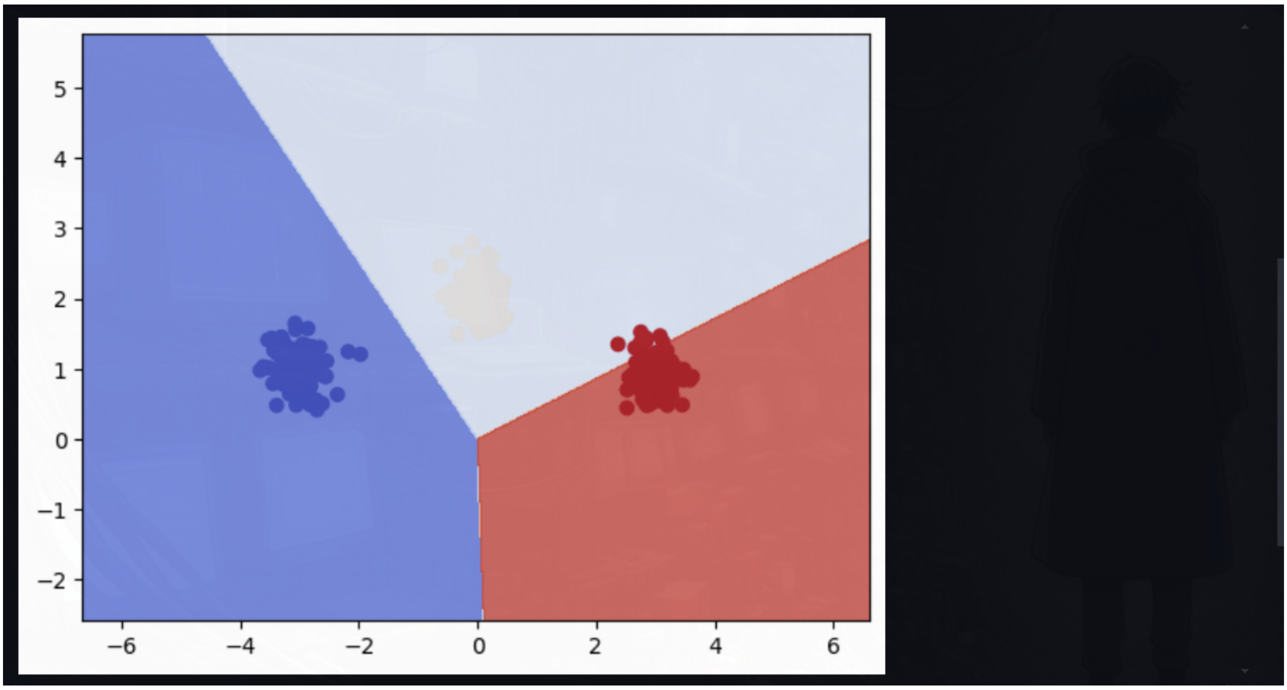
  # create a mesh to plot in
  h = .02  # step size in the mesh
  x_min, x_max = min(X[:,0])-3,max(X[:,0])+3
  y_min, y_max = min(X[:,1])-3,max(X[:,1])+3
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                       np.arange(y_min, y_max, h))
  mesh_input = np.c_[xx.ravel(), yy.ravel()]

  Z = clf_onevsall.predict(mesh_input)
  Z = Z.reshape(xx.shape)
  plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
  # Plot also the training points
  plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)


  from sklearn import metrics
  metrics.confusion_matrix(y, clf_onevsall.predict(X))
  ```

- ```
  Coeffs 0
  [[-1.05853334 -0.90294603]]
  Coeffs 1
  [[0.42121645 0.27171776]]
  Coeffs 2
  [[ 0.89164752 -0.82601734]]
  /home/arjun-prasad/anaconda3/envs/deep_learning/lib/python3.12/site-packages/sklearn/svm/_base
    warnings.warn(
  ```

- ```
  array([[100,   0,   0],
         [  0, 100,   0],
         [  0,  11,  89]])
  ```
  ```

- 

`please turn over`

Multiclass SVM

Ans 15.

- ```
  def featureMap(X,y,num_classes) :
      #The following line handles X being a 1d-array or a 2d-array
      num_samples, num_inFeatures = (1,X.shape[0]) if \
          len(X.shape) == 1 else (X.shape[0],X.shape[1])
      #your code goes here, and replaces following return
      if X.ndim == 1:
          X.reshape(1, -1)

      featureMap = np.zeros((num_samples, num_classes*num_inFeatures))
      loc = num_inFeatures*y
      for i in range(num_samples):
          featureMap[i, loc : loc + num_inFeatures] = X[i]

      return featureMap
  ```

please turn over
  ```

Ans 16.

- ```
  def sgd(X, y, num_outFeatures, subgd, eta = 0.1, T = 10000):
      num_samples = X.shape[0]
      #your code goes here and replaces following return statement
      w = np.zeros(num_outFeatures)
      for iter in range(T):
          idx = np.random.randint(0, num_samples)
          grad = subgd(X[idx], y[idx], w)
          w = w - (eta*grad)
      return w
  ```

`please turn over`
  ```

Ans 17.

- ```
  def subgradient(self,x,y,w):
      '''
      Computes the subgradient at a given data point x,y
      @param x: sample input
      @param y: sample class
      @param w: parameter vector
      @return returns subgradient vector at given x,y,w
      '''
      #Your code goes here and replaces the following return statement
      subgrad_R = 2*self.lam*w
      y_hat = -1

      f_y_w_max = -np.inf
      for yi in range(self.num_classes):
          if yi == y:
              continue # no point in evaluating this
          f_yi_w = self.Delta(yi, y) + (w @ (self.Psi(x, yi).flatten()
  - \
              self.Psi(x, y).flatten()))
          if f_yi_w > f_y_w_max:
              f_y_w_max = f_yi_w
              y_hat = yi
      return subgrad_R + (self.Psi(x, y_hat).flatten() - self.Psi(x,
  y).flatten())
  ```

- ```
  def decision_function(self, X):
      if not self.fitted:
          raise RuntimeError("You must train classifer before
  predicting data.")

      #Your code goes here and replaces following return statement
      num_samples = X.shape[0]
      class_scores = np.zeros((num_samples, self.num_classes))
      for yi in range(self.num_classes):
          class_scores[:, yi] = self.Psi(X, yi) @ self.coef_
      return class_scores
  ```

- ```
  def predict(self, X):
      #Your code goes here and replaces following return statement
      class_scores = self.decision_function(X)
      return np.argmax(class_scores, axis = 1)
  ```
  ```

`please turn over`

Ans 18.

- ```python
  from skeleton_code import zeroOne, featureMap, sgd, MulticlassSVM
  #the following code tests the MulticlassSVM and sgd
  #will fail if MulticlassSVM is not implemented yet
  est = MulticlassSVM(6,lam=1)
  est.fit(X,y)
  print("w:")
  print(est.coef_)
  Z = est.predict(mesh_input)
  Z = Z.reshape(xx.shape)
  plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
  # Plot also the training points
  plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)


  from sklearn import metrics
  metrics.confusion_matrix(y, est.predict(X))
  ```
- ```
  w:
  [-0.41923524 -0.41923524 -0.06249277 -0.06249277  0.48172801  0.48172801]

  array([[100,   0,   0],
         [  0,   0, 100],
         [  0,   0, 100]])
  ```
-