The Australian National University
School of Computing
Convenor: Lexing Xie & Yuan-sen Ting
Other Contributors: Alexander Soen & Josh Nguyen

Semester 1, 2023
Assignment 1
Weight: 18%

COMP4670/8600: Statistical Machine Learning

Release Date. 27th Feburary 2023.
Due Date. 27th March 2023 at 2359 AEDT.
Maximum credit. 100 Marks for COMP4670 and 120 Marks for COMP8600.

For submission, we are using Ed. Check under the submission tab and make sure to follow instructions on what to submit. Although using Ed for submission, we recommend that you work on the assignment locally on your computer as we will not be providing `display.ipynb` on Ed.

If you want to submit in LaTeX, we have a general template which you can use: https://quicklink.anu.edu.au/a4me.

In addition to the files which you need to submit, we have provided an `display.ipynb` Jupter Notebook which you can use to debug the functions that you implement. A correct implementation should run the notebook from start to finish without any errors. **DO NOT** change any files in `framework/.`.

Do not submit:

- The `data/.` folder;
- The `framework/.` folder;
- The Jupyter Notebook `display.ipynb`.

Grading is different for COMP4670 and COMP8600 students. Grades will be calculated as a percentage out of 100 or 120. Questions which are required for COMP8600 students are marked in their titles. These are optional for COMP4670 students. For COMP4670, their final grade will be taken as the maximum percentage of only COMP4670 questions vs all questions.

Installation can be done by using `requirements.txt` provided. For a fresh install, create a new Python environment (*e.g.*, via Conda) and then run `pip install -r requirements.txt` in said environment:

```
$ conda create -n sml
$ conda activate sml
$ pip install -r requirements.txt
```

Collaboration, you are free to discuss the material in the assignment with your classmates. However, every proof and code submitted must be written by yourself without assistant. You should be able to explain your answers if requested.

Regrade requests will be processed via a Microsoft Form. This will be released closer to the due date.

Other notes:

- When writing proofs, use the equation numbers when referring the equations in the assignment, *i.e.*, "Through Equation (3.1) we can show ...";
- You are not required to complete the assignment linearly, you may want to solve which-ever questions you can regardless of order of appearance;
- If you have trouble with proving a statement, try reducing dimensionality of the problem first;
- Code is only graded on correctness of functions implemented, **not** the output of `display.ipynb`.

Section 1: Bayesian Thinking (5 Total Points)

A historic grouping of machine learning methods comes in the form of non-Bayesian vs. Bayesian methods. You will notice that the first half of the course (and much of the Bishop's textbook) follows a pattern of introducing a non-Bayesian model / algorithm and subsequently extendeding it to a Bayesian counter-part, *e.g.*, Logistic Regression to Bayesian Logistic Regression; or Kernel Regression to Gaussian Processes. Although much of the course focuses on mathematical detail, for this first question we will consider some of the "whys" of using Bayesian methods in the first place.

Figure 1: From `http://yann.lecun.com/ex/fun/index.html#allyourbayes`.

Let's first refresh on the application of *Bayes' Rule* to models and data in ML. Given a training dataset $\mathcal{D}$ and a model θ, the posterior distribution is defined as

$$\underbrace{\Pr(\theta \mid \mathcal{D})}_{\text{“posterior”}} = \frac{\overbrace{\Pr(\mathcal{D} \mid \theta)}^{\text{“likelihood”}} \cdot \overbrace{\Pr(\theta)}^{\text{“prior”}}}{\underbrace{\Pr(\mathcal{D})}_{\text{“data”}}} \propto \Pr(\mathcal{D} \mid \theta) \cdot \Pr(\theta). \tag{1.1}$$

On the right-hand side, $\Pr(\mathcal{D})$ is ignored and interpreted as a normalization constant for the posterior. Bayes' Rule incorporates some amount of *prior knowledge* into a model and we can consider using maximizing the *posterior* rather than the *likelihood* in parameter fitting. With this interpretation in mind, we should think about when a non-Bayesian method and a Bayesian method are "equivalent."

Question 1.1: Am I Bayes? (1 Points)

Consider using two methods—maximum likelihood (MLE, non-Bayesian) and maximum a posteriori (MAP, Bayesian)—to estimate the parameters of some probability distribution from the same dataset and parameter space. When would the resulting estimates be equal?

Finding the MAP estimate consists in solving the optimization problem

$$\max_{\theta \in \Theta} \Pr(\theta \mid \mathcal{D}), \tag{1.2}$$

where Θ is the set of all possible values for θ. Sometimes, finding or approximating the solution to Problem (1.2) is straightforward. Unfortunately, most of the time it is not. For the rest of this question, we will make the following assumption.

Assumption: For the problem $\max_{\theta \in \Theta} \Pr(\theta \mid \mathcal{D})$, we assume there exists no method for finding or numerically approximating (with high enough precision) its optimal solution. However, we have access to $N > 0$ i.i.d. samples $\{\theta_i \sim \Pr(\cdot \mid \mathcal{D})\}_{i=1}^{N}$ and their corresponding probabilities $\Pr(\theta_i \mid \mathcal{D}), \forall i = 1, \ldots, N$.

You might notice from the Bayesian Regression lecture that, by the properties of Gaussian distributions, we can find the MAP estimator for θ if sampling from the posterior $\Pr(\theta \mid \mathcal{D})$ is possible. That is, assuming

that (i) the posterior is Gaussian $(*)$ and (ii) we have N i.i.d. samples from $\Pr(\theta \mid \mathcal{D})$, we have

$$\arg\max_{\theta} \Pr(\theta \mid \mathcal{D}) \overset{(*)}{=} \mathop{\mathbb{E}}_{\theta|\mathcal{D}}[\theta] \approx \frac{1}{N} \sum_{i=1}^{N} \theta_i. \tag{1.3}$$

The sample mean on the right-hand side provides a convenient method of approximating the "best" model.

Now consider $\Pr(\theta \mid \mathcal{D})$ is not Gaussian. A tantalizing question emerges: When is it suitable to approximate the MAP estimator using a sample mean?

Question 1.2: Approximating the MAP (2 Points)

Consider a posterior distribution $\Pr(\theta \mid \mathcal{D})$ supported on Θ and N i.i.d. samples $\{\theta_i \in \Theta\}_{i=1}^{N}$ from it. Define the sample mean as $\frac{1}{N}\sum_{i=1}^{N} \theta_i$.

1. For what posterior distributions would it be appropriate to use $\frac{1}{N}\sum_{i=1}^{N} \theta_i$ to approximate $\arg\max_{\theta\in\Theta} \Pr(\theta \mid \mathcal{D})$?
2. Give an example of a posterior distribution for which $\frac{1}{N}\sum_{i=1}^{N} \theta_i$ would be a very poor approximate of $\arg\max_{\theta\in\Theta} \Pr(\theta \mid \mathcal{D})$?

Even in the general case where $(*)$ does not hold, with access to both the samples θ_i and probabilities $\Pr(\theta_i \mid \mathcal{D})$, we can approximate the optimal solution simply by finding the sample maximum: $\arg\max_{i=1,\dots,N} \Pr(\theta_i \mid \mathcal{D})$.

... But should we?

One topic in machine learning which has recently garnered attention is *conformal prediction*.[1] The rise in the method's popularity is thanks to its ability to characterize the *uncertainty* of predictions through a confidence interval. Of course, a Bayesian model should naturally allow us to quantify the uncertainty of a "best" model as we are working with distributions.

Question 1.3: Being Uncertain (2 Points)

For the sample mean $\frac{1}{N}\sum_{i=1}^{N} \theta_i$ and sample maximum $\arg\max_{i=1,\dots,N} \Pr(\theta_i \mid \mathcal{D})$, discuss how we might (or might not be able to) quantify the uncertainty of the parameter estimate θ given a fixed set of N samples. In other words, is it possible to establish "error bars" for these estimates?

Remark. On a more philosophical note, given a posterior $\Pr(\theta \mid \mathcal{D})$, we may want think about what constitutes a "best" model $\theta^\star$. Do we want one that maximizes the posterior, one equal to the average model, or maybe something else like the median? In different scenarios, this definition of "best" can be radically different.

[1] See, for instance: `https://neurips.cc/virtual/2022/invited-talk/55872`

Section 2: Exponential Families (45/65 Total Points)

For this question, we will explore a unified family of distributions: the *exponential family* (not to be confused with an exponential distribution). The exponential family provides a generalization many commonly used distribution (Gaussian distribution, Beta distributions, Binomial distributions, *etc.*) and has many interesting properties which are used throughout ML.

First, let us define the exponential family distributions we will be considering in this question.

Definition 1 (Exponential Family[2])**.** Given a function $\boldsymbol{u} : \mathbb{R}^n \to \mathbb{R}^m$, we denote an n-dimensional exponential family distribution as $\textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta})$, where $\boldsymbol{\eta} \in \mathcal{P} \subset \mathbb{R}^m$ designates the m-dimensional parameters of the distribution within an exponential family[3]. The corresponding densities of the distributions are given by

$$q(\boldsymbol{z} \mid \boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{z}) - \psi(\boldsymbol{\eta})), \tag{2.1}$$

where

$$\psi(\boldsymbol{\eta}) = \log \int_{\mathbb{R}^n} \exp(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{z}))\ \mathrm{d}\boldsymbol{z}. \tag{2.2}$$

The function $\boldsymbol{u}$ is called the sufficient statistics of the exponential family and the function ψ is called the log-partition function of the exponential family. Sometimes, it will be convenient to define $Z(\boldsymbol{\eta}) = \int_{\mathbb{R}^n} \exp(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{z}))\ \mathrm{d}\boldsymbol{z} = \exp(\psi(\boldsymbol{\eta}))$ as the normalizer of the distribution.

Remark. We note that the definite integral in Eq. (2.2) is over the domain of $\boldsymbol{z}$. In this assignment, the domain of integration will simply be over $\mathbb{R}^n$; which can be simplified to integrating over $(-\infty, \infty)$ for different coordinates.

Remark. Notice that Definition 1 represent a broad set of special cases of the definition given in the Bishop textbook Eq. (2.194), with $h(\boldsymbol{x}) = 1$ and $g(\boldsymbol{\eta}) = \exp(-\psi(\boldsymbol{\eta}))$. For notational convenience in the tasks specified below, we use this definition.

At this point, it is useful to verify that the Exponential family indeed generalizes distributions we are familiar with.

Question 2.1: Gaussian R.V. as an Exponential Family (10 Points)

Let $\boldsymbol{u}(z) = (z, z^2)$ and $\boldsymbol{\eta} = (\mu/\sigma^2, -1/2\sigma^2)$ define an $(n = 1)$-dimensional exponential family distribution $\textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta})$ over $\mathbb{R}$.

Verify that $\textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta})$ is equivalent to the 1-dimensional Gaussian distribution with mean μ and variance σ^2.

What is the distribution's normalizer $Z(\boldsymbol{\eta})$?

One interesting aspect of Exponential families is its connections to loss functions in ML one typically finds. The simplest connection comes from 'equating' a loss function minimization procedure to the MLE of an exponential family, *i.e.*,

$$\arg\min \text{“A loss function } \ell\text{”} = \arg\max \log q(\boldsymbol{z} \mid \boldsymbol{\eta}). \tag{2.3}$$

Of course, to do this, one needs to find the correct matching between losses and exponential family. To do so, let us introduce some data.

Assumption: Let us assume that we have data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with features $\boldsymbol{x}_i \in \mathbb{R}^n$ and corresponding labels / targets $y_i \in \mathbb{R}$.

[2]This is actually a simplified version of an exponential family. A more general version can be defined, *i.e.*, non-unit base measure.

[3]The exponential family for a fixed $\boldsymbol{u}$ is the set of distributions given by $\mathcal{M}_{\boldsymbol{u}} = \{\textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta}) \mid \boldsymbol{\eta} \in \mathcal{P}\}$. Two exponential family distributions share the same exponential family if and only if their $\boldsymbol{u}$'s are the same.

Question 2.2: Exponential Families and Losses (5 Points)

Let $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}$ with weights $\boldsymbol{w} \in \mathbb{R}^n$. Let us define a conditional random variable using a 1-dimensional exponential family, such that

$$y \mid \boldsymbol{x} \sim \textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta}), \tag{2.4}$$

where $\boldsymbol{u}(y) = (y, y^2)$ and $\boldsymbol{\eta} = \boldsymbol{\eta}(\boldsymbol{x}) = (f(\boldsymbol{x}; \boldsymbol{w}), -1/2)$.
That is, as per Definition 1, we are defining the following conditional p.d.f. such that:

$$p(y \mid \boldsymbol{x}; \boldsymbol{w}) = q(y \mid \boldsymbol{\eta}(\boldsymbol{x})) = \exp(\boldsymbol{\eta}^\top(\boldsymbol{x})\boldsymbol{u}(y) - \psi(\boldsymbol{\eta}(\boldsymbol{x}))). \tag{2.5}$$

Show that

$$\arg\max_{\boldsymbol{w} \in \mathbb{R}^n} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{w}) = \arg\min_{\boldsymbol{w} \in \mathbb{R}^n} \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i; \boldsymbol{w}))^2. \tag{2.6}$$

Remark. One might notice that in Question 2.2, the function f and its corresponding weights $\boldsymbol{w}$ does not effect the maximization / minimization. Thus such an observation can be identically made when consider f as a neural network whilst optimizing over weights $\boldsymbol{W}_1, \ldots$.

In the general case, optimizing either Eq. (2.6) analytically ('by hand') can be difficult, *e.g.*, when f is a neural network. As such, it is useful to consider the gradients of the log-likelihood function.

Question 2.3: Parameterizing an Exponential Families (15 Points)

Let $f(\boldsymbol{x}; \boldsymbol{\theta})$ be some mapping from feature $\boldsymbol{x}$ parameterized by $\boldsymbol{\theta}$. Let us define a conditional random variable using an exponential family, such that

$$\boldsymbol{y} \mid \boldsymbol{x} \sim \textsc{Exp}(\boldsymbol{u}, \boldsymbol{\eta}), \tag{2.7}$$

where $\boldsymbol{\eta} = f(\boldsymbol{x}; \boldsymbol{\theta})$ for any function $\boldsymbol{u} : \mathbb{R}^n \to \mathbb{R}^m$ (we hide the dependence of $\boldsymbol{x}$ in $\boldsymbol{\eta}$ to reduce notation). We also define $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ similar to Eq. (2.5).
Show that

1. Show that
$$\left.\frac{\partial \psi(\boldsymbol{\eta})}{\partial \boldsymbol{\eta}}\right|_{\boldsymbol{\eta} = f(\boldsymbol{x}; \boldsymbol{\theta})} = \mathop{\mathbb{E}}_{p(\boldsymbol{y} \mid \boldsymbol{x})}[\boldsymbol{u}(y)]. \tag{2.8}$$

2. Show that
$$\frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left(\frac{\partial f(\boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)\left(\boldsymbol{u}(\boldsymbol{y}) - \mathop{\mathbb{E}}_{p(\boldsymbol{y}' \mid \boldsymbol{x})}[\boldsymbol{u}(\boldsymbol{y}')]\right). \tag{2.9}$$

Eq. (2.9) allows us to use *gradient ascent* (confer, gradient descent) to iteratively update the parameters of our model. Of course, we can practically use Eq. (2.9) by using auto-differentiation packages in Python. However, one will need to do some tricks to make the computation efficient.

Question 2.4: Implementing Gradient Ascent (15 Points)

In '`exponential_family.py`' complete the following functions:

- `grad_psi_wrt_eta` (Eq. (2.8));
- `log_likelihood_grad` (Eq. (2.9));
- `update`;

and thus train a neural network in `display.ipynb` for the MNIST dataset.
You will need to use `from torch.autograd.functional import jacobian, vjp`.

Hint: You may want take a look at `framework/exp_fam_model.py` (or the copied comment in `exponential_family.py`).

You will not be graded on optimality of the neural network's performance, only the correctness of implemented functions.

In the next few questions, we will be considering basic fairness properties of the exponential family distributions. There are *many* different criteria for fairness in ML, and rightfully so as fairness needs to take into account the surrounding social context. We will be considering a simple form of *data fairness*, where we will measure the *representation* of different sensitive groups (*e.g.*, gender, age, race, *etc.*).

In the sequel, we will consider n-dimensional distribution, where the first $(n-1)$-dimensions consists of non-sensitive features $\boldsymbol{x} \in \mathcal{X}$ (those which do not affects fairness) and the final dimension consisting of a sensitive feature $s \in \mathcal{S}$.

Assumption: We will assume in the following questions that $\mathcal{S}$ is a finite domain.

Definition 2. Given a p.d.f. $p(\boldsymbol{x}, s)$, its *representation rate* is given by:

$$\mathrm{RR}(p) = \min_{s,s' \in \mathcal{S}} \mathrm{RR}(p, s, s') \in [0, 1], \tag{2.10}$$

where $\mathrm{RR}(p, s, s') = p(s)/p(s')$ (with $p(s)$ as the marginal distribution of $p(\boldsymbol{x}, s)$).

Representation rate simply measure how well the balance of sensitive features are being represented by $p(\boldsymbol{x}, s)$.

Question 2.5 [COMP8600]: Verifying Representation Rate (5 Points)

We have maximal fairness when $\mathrm{RR}(p) = 1$ and minimal fairness when $\mathrm{RR}(p) = 0$.

Argue why this definition of fairness make sense in terms of marginal probabilities of demographic information $p(s)$.

Thus to study the representation rate of (separable) exponential families, we should consider what the sensitive marginal distributions look like.

Question 2.6 [COMP8600]: Sensitive Marginal of Exponential Families (10 Points)

Suppose that we have a separable Exponential Family n-distribution such that the natural parameters $\boldsymbol{\eta} \in \mathbb{R}^m$ and sufficient statistics $\boldsymbol{u} : \mathcal{X} \times \mathcal{S} \to \mathbb{R}^m$ are separable as:

$$\begin{aligned} \boldsymbol{\eta} &= [\boldsymbol{\eta}_0, \boldsymbol{\eta}_1], & \boldsymbol{\eta}_0 \in \mathbb{R}^{m_0} \text{ and } \boldsymbol{\eta}_1 \in \mathbb{R}^{m_1}; \\ \boldsymbol{u} &= [\boldsymbol{u}_0, \boldsymbol{u}_1], & \boldsymbol{u}_0 : \mathcal{X} \times \mathcal{S} \to \mathbb{R}^{m_0} \text{ and } \boldsymbol{u}_1 : \mathcal{X} \times \mathcal{S} \to \mathbb{R}^{m_1}; \end{aligned}$$

such that its p.d.f. $q(\boldsymbol{x}, s)$ is given by

$$q(\boldsymbol{x}, s) = \frac{\exp(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{x}, s))}{Z(\boldsymbol{\eta})}. \tag{2.11}$$

Prove that

$$q(s) = q_0(s) \cdot \frac{Z_0(\boldsymbol{\eta}_0)}{Z(\boldsymbol{\eta})} \cdot \mathop{\mathbb{E}}_{q_0(\boldsymbol{x}|s)}[\exp(\boldsymbol{\eta}_1^\top \boldsymbol{u}_1(\boldsymbol{x}, s))], \tag{2.12}$$

where q_0 and $Z_0(\boldsymbol{\eta}_0)$ are defined via $\textsc{Exp}(\boldsymbol{u}_0, \boldsymbol{\eta}_0)$.

Now we can use this result to prove a bound on the fairness of a (separable) exponential family.

Question 2.7 [COMP8600]: Representation Rate of Exponential Families (5 Points)

Given the same setup as Question 2.6, prove that

$$\mathrm{RR}(q) \geq \mathrm{RR}(q_0) \cdot \frac{\min_{s \in \mathcal{S}} \mathbb{E}_{q_0(\boldsymbol{x}|s)}[\exp(\boldsymbol{\eta}_1^\top \boldsymbol{u}_1(\boldsymbol{x}, s))]}{\max_{s \in \mathcal{S}} \mathbb{E}_{q_0(\boldsymbol{x}|s)}[\exp(\boldsymbol{\eta}_1^\top \boldsymbol{u}_1(\boldsymbol{x}, s))]}. \tag{2.13}$$

Section 3: Classification with Rejection (50 Total Points)

In the majority of the course, we will be learning about how do we teach an algorithm to make a certain prediction. However, should we always make an algorithm make prediction? The *Classification with Rejection (CwR)* framework allows for an algorithm to 'abstain' from making a prediction.

But before that, we will first review some basic in *loss function theory*. In classification, in essence our goal is to simple assign the correct labels to a set of input features. This 'correctness' can be simply quantified by the *0-1-loss function*: given an input $\boldsymbol{x} \in \mathcal{X}$ and a label $y \in \mathcal{Y} = \{1 \dots K\}$, the correctness of a hypothesis $f : \mathcal{X} \to \mathcal{Y}$ can be evaluated by:

$$\ell_{01}(f(\boldsymbol{x}), y) = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) \neq y \\ 0 & \text{otherwise} \end{cases}. \tag{3.1}$$

One can verify, that this loss function is simply assigning incorrect prediction a penalty of 1.

As per the lectures, we will be consider the *Risk Minimization* of these losses. In particular, given a distribution of data points $p(\boldsymbol{x}, y)$ we will denote:

$$R_{01}[f] = \mathbb{E}[\ell_{01}(f(\boldsymbol{x}), y)]. \tag{3.2}$$

Here we note that we use the subscript to designate what type of loss we are calculating the risk for.

To modify the typical notion of classification to add in the option of rejection, we consider a modified version of the 0-1-loss function. First, we modify the class of functions we are interested in to those that can predict an additional *rejection class*, denoted as ®. That is, we consider functions $f : \mathcal{X} \to \mathcal{Y} \cup \{®\}$. This leads us to the *0-1-c-loss function*, a modified 0-1-loss function which accounts for rejection:

$$\ell_{01c}(f(\boldsymbol{x}), y) = \begin{cases} c & \text{if } f(\boldsymbol{x}) = ® \\ \ell_{01}(f(\boldsymbol{x}), y) & \text{otherwise} \end{cases}, \tag{3.3}$$

where $c \in [0, 1/2)$. We define the corresponding risk with "01c" subscripts, *i.e.*, $R_{01c}[f]$.

We will refer to this approach of learning a classifier with rejection as the *general approach*, *i.e.*, learning functions of the type $f : \mathcal{X} \to \mathcal{Y} \cup \{®\}$.

We note that although the classifier f now has the option to reject, the data we are evaluating the loss function is still the same.

Question 3.1: Verify Preferable Rejection (5 Points)

Verify that setting $c \in [0, 1/2)$ in ℓ_{01c} ensures that choosing a rejection is preferable if there is less than $1/2$ chance of f being correct.

Hint: Fix an input $\boldsymbol{x}$ and consider two classifiers, one which reject and on which does not. How do the expected losses compare?

Assumption: In sequel, we will assume $c \in [0, 1/2)$.

One question we can ask, is what is the optimal classifier in the CwR setting, given perfect information.

Question 3.2: Optimal General Rejection Classifier (10 Points)

Given the true distribution of data $p(\boldsymbol{x}, y)$, with class posterior function

$$\eta_y(\boldsymbol{x}) \stackrel{\text{def}}{=} p(y \mid \boldsymbol{x}) \qquad \text{for } y \in \mathcal{Y}. \tag{3.4}$$

Prove that

$$f^\star = \arg\min R_{01c}[f], \tag{3.5}$$

where

$$f^\star(\boldsymbol{x}) = \begin{cases} ® & \text{if } \max_{y\in\mathcal{Y}} \eta_y(\boldsymbol{x}) \leq 1 - c \\ \arg\max_{y\in\mathcal{Y}} \eta_y(\boldsymbol{x}) & \text{otherwise} \end{cases}. \tag{3.6}$$

Hint: Consider the best response to a fixed $\boldsymbol{x}$. When is a ® the best response? Otherwise, what is best response?

More practically, obtaining the optimal CwR classifier difficult. As a result, a number of different methods of altering the learning task has been consider. One of these is called the *classifier-rejector* approach. The intuition of this approach is simple: we learn two separate classifiers where one determines rejection; while the other is a regular classifier,

$$f_{cr}(\boldsymbol{x}; h, r) = \begin{cases} ® & \text{if } r(\boldsymbol{x}) \leq 0 \\ h(\boldsymbol{x}) & \text{otherwise} \end{cases}, \tag{3.7}$$

where $r : \mathcal{X} \to \mathbb{R}$ and $h : \mathcal{X} \to \mathcal{Y}$. The learned function $r(\boldsymbol{x})$ acts as a proxy for the 'confidence' of $h(\boldsymbol{x})$'s prediction. One can verify that

$$\ell_{01c}(f_{cr}(\boldsymbol{x}; h, r), y) = \ell_{01cr}(h(\boldsymbol{x}), r(\boldsymbol{x}), y), \tag{3.8}$$

where

$$\ell_{01cr}(h(\boldsymbol{x}), r(\boldsymbol{x}), y) = \begin{cases} c & \text{if } r(\boldsymbol{x}) \leq 0 \\ \ell_{01}(h(\boldsymbol{x}), y) & \text{otherwise.} \end{cases} \tag{3.9}$$

Question 3.3: Optimal Classifier-Rejection Approach (5 Points)

Given

$$h^\star(\boldsymbol{x}) = \arg\max_{y\in\mathcal{Y}} \eta_y(\boldsymbol{x}); \tag{3.10}$$

$$r^\star(\boldsymbol{x}) = \max_{y\in\mathcal{Y}} \eta_y(\boldsymbol{x}) - (1 - c). \tag{3.11}$$

For the corresponding risk $R_{01cr}[h^\star, r^\star] \stackrel{\text{def}}{=} \mathbb{E}[\ell_{01cr}(h^\star(\boldsymbol{x}), r^\star(\boldsymbol{x}), y)]$, prove that

$$R_{01cr}[h^\star, r^\star] = R_{01c}[f^\star], \tag{3.12}$$

thus proving that $h^\star, r^\star$ are optimal.

~~When would Eq. (3.12) hold for arbitrary h, r, f?~~

From Question 3.2 and 3.3, we have shown that there are two different ways of minimizing ℓ_{01c}: a general method, and one which requires two learned classifiers. There are many pros and cons for why one would pick between these two. We will explore the sample complexity of calculating the risks. First, let us define the *empirical risk* of each of the losses:

$$\hat{R}_{01c}[f] = \frac{1}{N}\sum_{i=1}^{N} \ell_{01c}(f(\boldsymbol{x}_i), y_i)$$

$$\hat{R}_{01cr}[h, r] = \frac{1}{N}\sum_{i=1}^{N} \ell_{01cr}(h(\boldsymbol{x}_i), r(\boldsymbol{x}_i), y_i).$$

Remark. Notice that the notation is slightly different when signifying the empirical risk.

Question 3.4: Sample Complexity of Classifier-Rejection Approach (10 Points)

Suppose that $\mathcal{H}$ and $\mathcal{R}$ are two finite hypothesis classes which are used in the classifier-rejector approach.
Prove that given N samples, for any $\delta > 0$:

$$\Pr\left\{\forall h \in \mathcal{H}, r \in \mathcal{R} : |R_{01cr}[h,r] - \hat{R}_{01cr}[h,r]| \leq \sqrt{\frac{\ln|\mathcal{H}| + \ln|\mathcal{R}| + \ln(2/\delta)}{2N}}\right\} \geq 1 - \delta. \quad (3.13)$$

Question 3.5: Comparing Sample Complexity (10 Points)

Let $\mathcal{H}$, $\mathcal{R}$ be finite hypothesis classes corresponding to the classifier-rejection approach with binary rejector, *i.e.*, classes of functions $h : \mathcal{X} \to \mathcal{Y}$ and $r : \mathcal{X} \to \{-1, +1\}$. Further let $\mathcal{F}$ be a finite hypothesis class for learning the general approach, *i.e.*, class of functions $f : \mathcal{X} \to \mathcal{Y} \cup \{®\}$.

Suppose that hypothesis classes $\mathcal{H}$, $\mathcal{R}$, and $\mathcal{F}$ have size $\mathcal{O}(|\mathcal{Y}|^{|\mathcal{X}|})$, $\mathcal{O}(2^{|\mathcal{X}|})$, and $\mathcal{O}((|\mathcal{Y}|+1)^{|\mathcal{X}|})$, respectively (assuming finite $\mathcal{X}$).

How does the general approach's sample complexity (*i.e.*, equivalent bound to Eq. (3.13)) compare to the classifier-reject approach?

How does this change with the number of classes, k, being predicted over? How does it change as the input space $\mathcal{X}$ increases?

We will end on a practical note. The optimal predictor found in Question 3.3 allows us to utilize a plug-in estimator to make a CwR model. In particular, we can simply replace $\eta_k(\boldsymbol{x})$ with a model which is approximating the probability, *i.e.*, through what we have looked at in Question 2. That is, we will use the classifier you have trained in the Question 2.4 — we are letting $\eta_k(\boldsymbol{x}) = q(y \mid \boldsymbol{x})$, with q defined in Question 2.3.

Question 3.6: Implementing Classifier with Rejection (10 Points)

In `classification_with_rejection.py` complete the following functions:

- `h_classifier_pred` (Eq. (3.10));
- `r_rejector_pred` (Eq. (3.11));
- `cwr_pred` (Eq. (3.7));

One can also check `display.ipynb` to see if the rejected examples make sense.