COMP9417 - Machine Learning
Homework 2

Introduction In this h[illegible] a closer look at feature maps induced by kernels. We then explore a creative use of [illegible] method introduced in homework 1. We will show that gradient descent techniques can be used to construct combinations of models from a base set of models such that the combination can outperform any single base model.

Points Allocation There are a total of 28 marks.

- Question 1 a): 2 marks
- Question 1 b): 2 marks
- Question 1 c): 4 marks
- Question 1 d): 2 marks
- Question 1 e): 2 marks
- Question 1 f): 1 mark
- Question 1 g): 1 mark
- Question 2 a): 2 marks
- Question 2 b): 3 marks
- Question 2 c): 3 marks
- Question 3 a): 2 marks
- Question 3 b): 2 marks
- Question 3 c): 2 marks

What to Submit

- A **single PDF** file which contains solutions to each question. For each question, provide your solution in the form of text and requested plots. For some questions you will be requested to provide screen shots of code used to generate your answer — only include these when they are explicitly asked for.
- **.py file(s) containing all code you used for the project, which should be provided in a separate .zip file.** This code must match the code provided in the report.

- You may be deducted points for not following these instructions.

- You may be deducted points for poorly presented/formatted work. Please be neat and make your solutions clear. Start each question on a new page if necessary.
- You **cannot** submit a Jupyter notebook; this will receive a mark of zero. This does not stop you from developing you[illegible]d then copying it into a .py file though, or using a tool such as **nbconvert** or si[illegible]
- We will set up a [illegible]tions about this homework. Please read the existing questions before posting r[illegible]o some basic research online before posting questions. Please only post clarifi[illegible] questions deemed to be *fishing* for answers will be ignored and/or deleted.
- Please check M[illegible]for updates to this spec. It is your responsibility to check for announcements about the spec.
- Please complete your homework on your own, do not discuss your solution with other people in the course. General discussion of the problems is fine, but you must write out your own solution and acknowledge if you discussed any of the problems in your submission (including their name(s) and zID).

- As usual, we monitor all online forums such as Chegg, StackExchange, etc. Posting homework questions on these site is equivalent to plagiarism and will result in a case of academic misconduct.

- You may **not** use SymPy or any other symbolic programming toolkits to answer the derivation questions. This will result in an automatic grade of zero for the relevant question. You must do the derivations manually.

When and Where to Submit

- Due date: Week 7, Monday **July 10th**, 2023 by **5pm**. Please note that the forum will not be actively monitored on weekends.

- Late submissions will incur a penalty of 5% per day **from the maximum achievable grade**. For example, if you achieve a grade of 80/100 but you submitted 3 days late, then your final grade will be $80 - 3 \times 5 = 65$. Submissions that are more than 5 days late will receive a mark of zero.
- Submission must be made on Moodle, **no exceptions**.

Question 1. Gradient Descent for Learning Combinations of Models.

In this question, we discuss and implement a gradient descent based algorithm for learning combinations of models, which are generally termed 'ensemble models'. The gradient descent idea is a very powerful one that has been used in a large number of creative ways in machine learning beyond direct minimization of loss functions.

The Gradient-Co[illegible]m can be described as follows: Let $\mathcal{F}$ be a set of base learning algorithms[1]. The [illegible] base learners in $\mathcal{F}$ in an optimal way to end up with a good learning algorith[illegible] function, where y is the target, and $\hat{y}$ is the predicted value.[2] Suppose we have [illegible] $\ldots, n$, which we collect into a single data set D_0. We then set the number of de[illegible] and proceed as follows:

(I) Initialize $f_0($[illegible] function.)

(II) For $t = 1, 2, \ldots$ [illegible]

(GC1) Compute:

$$r_{t,i} = -\frac{\partial}{\partial f(x_i)} \sum_{j=1}^{n} \ell(y_j, f(x_j)) \Bigg|_{f(x_j)=f_{t-1}(x_j),\ j=1,\ldots,n}$$

for $i = 1, \ldots, n$. We refer to $r_{t,i}$ as the i-th pseudo-residual at iteration t.

(GC2) Construct a new *pseudo* data set, D_t, consisting of pairs: $(x_i, r_{t,i})$ for $i = 1, \ldots, n$.

(GC3) Fit a model to D_t using our base class $\mathcal{F}$. That is, we solve

$$h_t = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(r_{t,i}, f(x_i))$$

(GC4) Choose a step-size. This can be done by **either** of the following methods:

(SS1) Pick a fixed step-size $\alpha_t = \alpha$

(SS2) Pick a step-size adaptively according to

$$\alpha_t = \arg\min_{\alpha} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)).$$

(GC5) Take the step

$$f_t(x) = f_{t-1}(x) + \alpha_t h_t(x).$$

(III) return f_T.

We can view this algorithm as performing (functional) gradient descent on the base class $\mathcal{F}$. Note that in (GC1), the notation means that after taking the derivative with respect to $f(x_i)$, set all occurences of $f(x_j)$ in the resulting expression with the prediction of the current model $f_{t-1}(x_j)$, for all j. For example:

$$\frac{\partial}{\partial x} \log(x+1) \Bigg|_{x=23} = \frac{1}{x+1} \Bigg|_{x=23} = \frac{1}{24}.$$

[1]For example, you could take $\mathcal{F}$ to be the set of all regression models with a single feature, or alternatively the set of all regression models with 4 features, or the set of neural networks with 2 layers etc.

[2]Note that this set-up is general enough to include both regression and classification algorithms.

(a) Consider the regression setting where we allow the y-values in our data set to be real numbers. Suppose that we use squared error loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$. For round t of the algorithm, show that $r_{t,i} = y_i - f_{t-1}(x_i)$. Then, write down an expression for the optimization problem in step (GC3) that is specific to this setting (you don't need to actually solve it).

What to submit: your working out, either typed or handwritten.

(b) Using the sa[illegible]evious part, derive the step-size expression according to the adaptive app[illegible]

What to subm[illegible]er typed or handwritten.

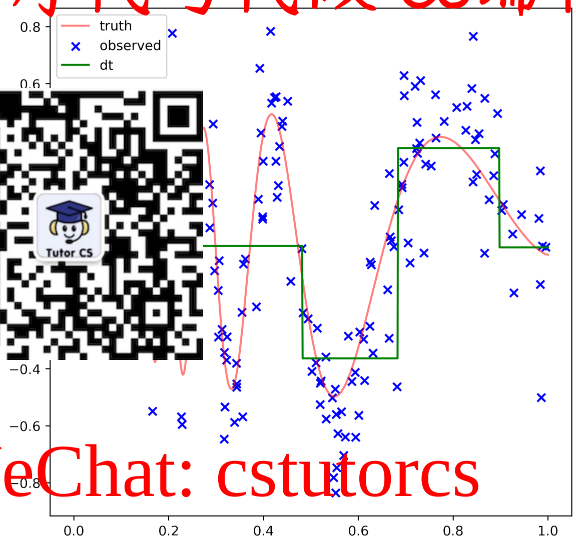
(c) We will now [illegible]t-combination algorithm on a toy dataset from scratch, and we will use the c[illegible](depth 1 decision trees) as our base class ($\mathcal{F}$), and squared error loss as in the[illegible]following code generates the data and demonstrates plotting the predictio[illegible]ee (more details in `q1.py`):

```
np.random.seed(123)
X, y = f_sampler(f, 160, sigma=0.2)
X = X.reshape(-1,1)

fig = plt.figure(figsize=(7,7))
dt = DecisionTreeRegressor(max_depth=2).fit(X,y)          # example model
xx = np.linspace(0,1,1000)
plt.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
plt.scatter(X,y, marker='x', color='blue', label='observed')
plt.plot(xx, dt.predict(xx.reshape(-1,1)), color='green', label='dt')   # plotting
example model
plt.legend()
plt.show()
```

The figure generated is

[3] In your implementation, you may make use of `sklearn.tree.DecisionTreeRegressor`, but all other code must be your own. You may use `NumPy` and `matplotlib`, but do not use an existing implementation of the algorithm if you happen to find one.



Your task is to generate a 5 x 2 figure of subplots showing the predictions of your fitted gradient-combination model. There are 10 subplots in total, the first should show the model with 5 base learners, the second subplot should show it with 10 base learners, etc. The last subplot should be the gradient-combination model with 50 base learners. Each subplot should include the scatter of data, as well as a plot of the true model (basically, the same as the plot provided above but with your fitted model in place of `dt`). Comment on your results, what happens as the number of base learners is increased? You should do this two times (two 5x2 plots), once with the adaptive step size, and the other with the step-size taken to be $\alpha = 0.1$ fixed throughout. There is no need to split into train and test data here. Comment on the differences between your fixed and adaptive step-size implementations. How does your model perform on the different x-ranges of the data?

What to submit: two 5 x 2 plots, one for adaptive and one for fixed step size, some commentary, and a screen shot of your code and a copy of your code in your .py file.

(d) Repeat the analysis in the previous question but with depth 2 decision trees as base learners instead. Provide the same plots. What do you notice for the adaptive case? What about the non-adaptive case? *What to submit: two 5 x 2 plots, one for adaptive and one for fixed step size, some commentary, and a copy of your code in your .py file.*

(e) Now, consider the classification setting where y is taken to be an element of $\{-1, 1\}$. We consider the following classification loss: $\ell(y, \hat{y}) = \log(1 + e^{-y\hat{y}})$. For round t of the algorithm, what is the expression for $r_{t,i}$? Write down an expression for the optimization problem in step (GC3) that is specific to this setting (you don't need to actually solve it).

What to submit: your working out, either typed or handwritten.

(f) Using the same setting as in the previous part, write down an expression for α_t using the adaptive approach in (SS2). Can you solve for α_t in closed form? Explain.

What to submit: your working out, either typed or handwritten, and some commentary.

(g) In practice, if you cannot solve for α_t exactly, explain how you might implement the algorithm. Assume that using a constant step size is not a valid alternative. Be as specific as possible in your answer. What, if any, are the additional computational costs of your approach relative to using a constant step size ?

What to submit: some commentary.

Question 2. Test Set [illegible]

Recall that the rM[illegible]thesis[4] h on a **test** set $\{(x_i, y_i)\}_{i=1}^n$, where x_i is a p-dimensional vector and y_i is a [illegible] as

$$[illegible](h) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2}.$$

For all parts, assume that the x_i's are known to you, but the y_i's are not. Suppose however that you are permitted to query rMSE(h). What this means is that you can query, for any hypothesis h, the rMSE of h on the test set. Suppose further that you know that rMSE$(z) = c_0$, where z is the hypothesis that returns zero for any input, i.e. $z(x_i) = 0$ for each $i = 1, \ldots, n$, and c_0 is some arbitrary positive number.

(a) Assume you have a set of T hypotheses $\mathcal{H} = \{h_1, h_2, \ldots, h_T\}$. You are told that for each i, there is a hypothesis h in $\mathcal{H}$, such that $h(x_i) = y_i$. In other words, for any point in the test set, there is at least one hypothesis in $\mathcal{H}$ that predicts that point correctly[5]. Suppose that you are permitted to blend predictions of different hypotheses in $\mathcal{H}$[6]. Design a naive brute-force algorithm that constructs a hypothesis g from the elements of $\mathcal{H}$ such that rMSE$(g) = 0$. How many queries of rMSE do you need to make? How does your algorithm scale (in the worst case) with the test size n? Describe your algorithm in detail.

What to submit: a description of your algorithm and the number of queries required, either typed or handwritten.

(b) We now consider a better approach than brute-force. For a given hypothesis h, define

$$h(X) = (h(x_1), h(x_2), \ldots, h(x_n))^\top$$
$$y = (y_1, y_2, \ldots, y_n)^\top.$$

We can always compute $h(X)$, but we do not know y. What is the smallest number of queries required to compute $y^\top h(X)$? Describe your approach in detail.

What to submit: a description of your approach and the number of queries required, either typed or handwritten.

(c) Given a set of K hypotheses $\mathcal{H} = \{h_1, \ldots, h_K\}$, use your result in the previous part to solve

$$\min_{\alpha_1,\ldots,\alpha_K} \text{rMSE}\left(\sum_{k=1}^{K}\alpha_k h_k\right),$$

and obtain the optimal weights $\alpha_1, \ldots, \alpha_K$. Describe your approach in detail, and be sure to detail how many queries are needed and the exact values of the α's, in terms of X, y and the elements of $\mathcal{H}$.

[4] The term hypothesis just means a function or model that takes as input x and returns as output a prediction $h(x) = \hat{y}$.

[5] Note that this does not imply that there is some hypothesis in $\mathcal{H}$ that predicts all points correctly.

[6] For example, you could construct a blended hypothesis g that returns the predictions of h_2 on test points 1 to 5, and the predictions of h_5 on points 6 to n.

What to submit: a description of your algorithm and the number of queries required, either typed or handwritten.

Question 3. Newton's Method

Note: throughout this question do not use any existing implementations of any of the algorithms discussed unless [illegible] the question. Using existing implementations can result in a grade of zero f[illegible] In homework 1 we studied gradient descent (GD), which is usually referred t[illegible]. Here, we study an alternative algorithm known as Newton's algorithm, which [illegible] as a second order method. Roughly speaking, a second order method makes u[illegible]ond derivatives. Generally, second order methods are much more accurate tha[illegible]n a twice differentiable function $g : \mathbb{R} \to \mathbb{R}$, Newton's method generates a seque[illegible]cording to the following update rule:

$$x^{(k+1)} = x^{(k)} - \frac{g'(x^{(k)})}{g''(x^{(k)})}, \qquad k = 0, 1, 2, \dots, \tag{1}$$

For example, consider the function $g(x) = \frac{1}{2}x^2 - \sin(x)$ with initial guess $x^{(0)} = 0$. Then

$$g'(x) = x - \cos(x), \quad \text{and} \quad g''(x) = 1 + \sin(x),$$

and so we have the following iterations:

$$\begin{aligned}
x^{(1)} &= x^{(0)} - \frac{x^{(0)} - \cos(x^0)}{1 + \sin(x^{(0)})} = 0 - \frac{0 - \cos(0)}{1 + \sin(0)} = 1 \\
x^{(2)} &= x^{(1)} - \frac{x^{(1)} - \cos(x^1)}{1 + \sin(x^{(1)})} = 1 - \frac{1 - \cos(1)}{1 + \sin(1)} = 0.750363867840244 \\
x^{(3)} &= 0.739112890911362 \\
&\vdots
\end{aligned}$$

and this continues until we terminate the algorithm (as a quick exercise for your own benefit, code this up, plot the function and each of the iterates). We note here that in practice, we often use a different update called the *dampened* Newton method, defined by:

$$x^{(k+1)} = x^{(k)} - \alpha \frac{g'(x_k)}{g''(x_k)}, \qquad k = 0, 1, 2, \dots. \tag{2}$$

Here, as in the case of GD, the step size α has the effect of 'dampening' the update.

(a) Consider the twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. The Newton steps in this case are now:

$$x^{(k+1)} = x^{(k)} - (H(x^{(k)}))^{-1} \nabla f(x^{(k)}), \qquad k = 0, 1, 2, \dots, \tag{3}$$

where $H(x) = \nabla^2 f(x)$ is the Hessian of f. Explain heuristically (in a couple of sentences) how the above formula is a generalization of equation (1) to functions with vector inputs. *what to submit: Some commentary*

(b) Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2.$$

Create a 3D plot of the function using `mplot3d` (see lab0 for example). Further, compute the gradient and Hessian of f. *what to submit: A single plot, the code used to generate the plot, the gradient and Hessian calculated along with all working. Add a copy of the code to solutions.py*

(c) Using `NumPy` only, implement the (undampened) Newton algorithm to find the minimizer of the function in the previous part, using an initial guess of $x^{(0)} = (1.2, 1)^T$. Terminate the algorithm when $\|\nabla f(x^{(k)})\|_2 \leq 10^{-6}$. Report the values of $x^{(k)}$ for $k = 0, 1, \ldots, K$ where K is your final iteration. *what to submit: your iterations, and a screen shot of your code. Add a copy of the code to solutions.py*