COMP9417 - Machine Learning
Homework 2: Kernel Features & Model Combinations

Introduction In this homework we first take a closer look at feature maps induced by kernels. We then explore a creative use of the gradient descent 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): 3 marks
- Question 1 b): 3 marks
- Question 1 c): 3 marks
- Question 1 d): 5 marks
- Question 2 a): 2 marks
- Question 2 b): 2 marks
- Question 2 c): 4 marks
- Question 2 d): 2 marks
- Question 2 e): 2 marks
- Question 2 f): 1 mark
- Question 2 g): 1 mark

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 your code in a notebook and then copying it into a .py file though, or using a tool such as **nbconvert** or similar.
- We will set up a Moodle forum for questions about this homework. Please read the existing questions before posting new questions. Please do some basic research online before posting questions. Please only post clarification questions. Any questions deemed to be *fishing* for answers will be ignored and/or deleted.
- Please check Moodle announcements 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 **March 27th**, 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. Kernel Power

Consider the following 2-dimensional data-set, where y denotes the class of each point.

index	x_1	x_2	y
1	1	0	-1
2	0	1	-1
3	0	-1	-1
4	-1	0	+1
5	0	2	+1
6	0	-2	+1
7	-2	0	+1

Throughout this question, you may use any desired packages to answer the questions.

(a) Use the transformation $x = (x_1, x_2) \mapsto (\phi_1(x), \phi_2(x))$ where $\phi_1(x) = 2x_2^2 - 4x_1 + 1$ and $\phi_2(x) = x_1^2 - 2x_2 - 3$. What is the equation of the best separating hyper-plane in the new feature space? Provide a plot with the data set and hyperplane clearly shown.

What to submit: a single plot, the equation of the separating hyperplane, a screen shot of your code, a copy of your code in your .py file for this question.

(b) Fit a hard margin linear SVM to the **transformed** data-set in the previous part[1]. What are the estimated values of $(\alpha_1, \dots, \alpha_7)$. Based on this, which points are the support vectors? What error does your computed SVM achieve?

What to submit: the indices of your identified support vectors, the train error of your SVM, the computed α's (rounded to 3 d.p.), a screen shot of your code, a copy of your code in your .py file for this question.

(c) Consider now the kernel $k(x, z) = (2 + x^\top z)^2$. Run a hard-margin kernel SVM on the **original** (untransformed) data given in the table at the start of the question. What are the estimated values of $(\alpha_1, \dots, \alpha_7)$. Based on this, which points are the support vectors? What error does your computed SVM achieve?

What to submit: the indices of your identified support vectors, the train error of your SVM, the computed α's (rounded to 3 d.p.), a screen shot of your code, a copy of your code in your .py file for this question.

(d) Provide a **detailed** argument explaining your results in parts (i), (ii) and (iii). Your argument should explain the similarities and differences in the answers found. In particular, is your answer in (iii) worse than in (ii)? Why? To get full marks, be as detailed as possible, and use mathematical arguments or extra plots if necessary.

What to submit: some commentary and/or plots. If you use any code here, provide a screen shot of your code, and a copy of your code in your .py file for this question.

Question 2. 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 as in the previous question.

The Gradient-Combination (GC) algorithm can be described as follows: Let $\mathcal{F}$ be a set of base learning algorithms[2]. The idea is to combine the base learners in $\mathcal{F}$ in an optimal way to end up with a good

[1] If you are using the `SVC` class in `sklearn`, to get a hard-margin svm, you need to set the hyper parameter `C` to be very large.

[2] 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.

learning algorithm. Let $\ell(y, \hat{y})$ be a loss function, where y is the target, and $\hat{y}$ is the predicted value.[3] Suppose we have data (x_i, y_i) for $i = 1, \dots, n$, which we collect into a single data set D_0. We then set the number of desired base learners to T and proceed as follows:

(I) Initialize $f_0(x) = 0$ (i.e. f_0 is the zero function.)

(II) For $t = 1, 2, \dots, T$:

(GC1) Compute:

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

for $i = 1, \dots, n$. 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 $i = 1, \dots, n$.

(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:

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

(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 same setting as in the previous part, derive the step-size expression according to the adaptive approach (SS2).

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

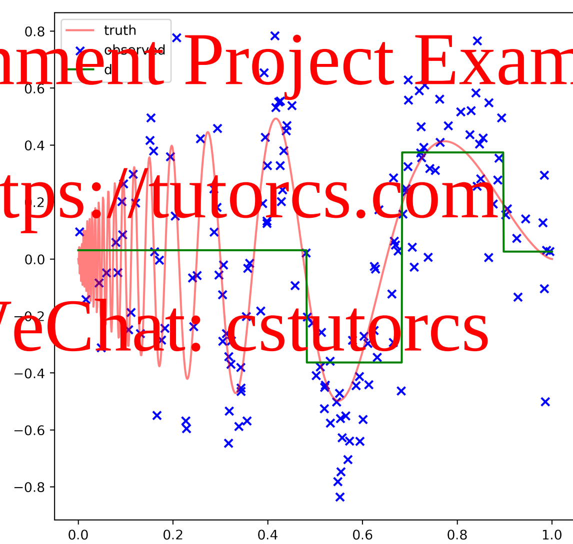
[3] Note that this set-up is general enough to include both regression and classification algorithms.

(c) We will now implement the gradient-combination algorithm on a toy dataset from scratch, and we will use the class of decision stumps (depth 1 decision trees) as our base class ($\mathcal{F}$), and squared error loss as in the previous parts.[4]. The following code generates the data and demonstrates plotting the predictions of a fitted decision tree (more details in `q2.py`):[5]

```
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



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

[4]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.

[5]Although we will not cover decision trees until week 4, we are treating the decision tree as a black box algorithm that can be called using the sklearn implementation. For more on using sklearn models, see Lab 1.

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.