# DATA7703, Assignment 2

**2022 Semester 2, due 5pm 7 Oct**

**Instructions**.

(a) Submit your solutions as a **single PDF** file on Blackboard. Go to Assessment, Assignment 2 to submit. If you don't know how to convert your file to a PDF, please search for a guide online. You can submit as many times as you want before the deadline. The last submission will be graded.

(b) Write down your **name and student number** on the first page of your solution report, and write down the **question numbers** for your solutions. For programming questions, you are welcome to submit your code files or output files in a separate zip file, but you *must* include both your code and relevant output in your submitted PDF file. Excessive code output may be penalised.

(c) Follow integrity rules, and provide citations as needed. You can discuss with your classmates, but you are required to write your solutions independently, and specify who you have discussed with in your solution. If you do not know how to solve a problem, you can get 15% of the mark by writing down "I don't know".

You are encouraged to keep your solutions concise — these questions require thoughts, not long answers.

**1.** (20 marks) This question concerns some theoretical aspects of ensemble methods.

**(a)** (5 marks) Consider a problem with a single real-valued feature $x$. For any $a < b$, consider the threshold classifiers or decision stumps $c_1(x) = I(x > a)$, $c_2(x) = I(x < b)$, and $c_3(x) = I(x < +\infty)$, where the indicator function $I(\cdot)$ takes value $+1$ if its argument is true, and -1 otherwise.

What is the set of real numbers classified as positive by $f(x) = I(0.1c_3(x) - c_1(x) - c_2(x) > 0)$? If $f(x)$ a threshold classifier? Justify your answer.

**(b)** (5 marks) Explain why OOB error is a preferred generalization performance measure for bagging as compared to the generalization performance measures estimated using the validation set method and cross-validation.

**(c)** (10 marks) Bob is a very creative data scientist. He proposes a variant of the standard bagging algorithm, called Wagging (Weighted Aggregating), and claims that it works better than standard bagging.

Wagging is used for regression. As in bagging, Wagging first trains a certain number of $m$ models $f_1, \ldots, f_m$ on $m$ bootstrap samples. Unlike bagging, Wagging assigns weights $w_1 = \frac{1}{2}, w_2 = \frac{1}{2^2}, \ldots, w_{m-1} = \frac{1}{2^{m-1}}, w_m = \frac{1}{2^{m-1}}$ to the models. If $Y_i$ is the prediction of $f_i$, then Wagging predicts $\bar{Y} = \sum_i w_i Y_i$.

We assume that $Y_1, \ldots, Y_m$ are identically distributed with $\mathrm{Var}(Y_i) = \sigma^2$ for all $i$, and $\mathrm{cov}(Y_i, Y_j) = \rho\sigma^2$ for all $1 \le i \ne j \le m$.

**i.** (5 marks) Bob claims that Wagging has a smaller bias than bagging. True or false? Justify your answer.

**ii.** (5 marks) Bob also claims that Wagging has a smaller variance than bagging. True or false? Justify your answer.
Hint: show that $\text{Var}(\sum_{i=1}^m w_i Y_i) = \sum_{i=1}^m w_i^2(1-\rho)\sigma^2 + \rho\sigma^2$ for any $w_i$'s such that $\sum_{i=1}^m w_i = 1$.

A3 - 3

**2.** (30 marks) In this question, you will perform some experiments to examine the effect of the hyperparameter $m$ used in random forest. You will investigate how $m$ affects the correlation between the trees and the generalization performance of random forests.

Recall that $m$ is the number of random features used in choosing the splitting point in the decision trees. When constructing a decision tree in a random forest, at each node, instead of choosing the best split from all $d$ given features, we can first choose $1 \leq m \leq d$ features, and then choose the best split among them. This randomization trick *decorrelates* the trees and makes the *generalization performance* of random forests better than bagging with decision trees.

**(a)** (5 marks) Load the California housing dataset provided in `sklearn.datasets`, and construct a random 70/30 train-test split. Set the random seed to a number of your choice to make the split reproducible. What is the value of $d$ here?

**(b)** (5 marks) Train a random forest of 100 decision trees using default hyperparameters. Report the training and test MSEs. What is the value of $m$ used? 去年是accuracy， m=8

**(c)** (5 marks) Write code to compute the pairwise (Pearson) correlations between the test set predictions of all pairs of distinct trees. Report the average of all these pairwise correlations.

You can retrieve all the trees in a `RandomForestRegressor` object using the `estimators_` attribute.

**(d)** (5 marks) Repeat (b) and (c) for $m = 1$ to $d$. Produce a table containing the training and test MSEs, and the average correlations for all $m$ values. In addition, plot the training and test MSEs against $m$ in a single figure, and plot the average correlation against $m$ in another figure.

**(e)** (5 marks) Describe how the average correlation changes as $m$ increases. Explain the observed pattern.

**(f)** (5 marks) A data scientist claims that we should choose $m$ such that the average correlation is smallest, because it gives us maximum reduction in the variance, thus maximum reduction in the expected prediction error. True or false? Justify your answer.

A4 - 1

**3.** (15 marks) In lecture, we discussed the architecture and representational power of neural nets, some training objectives and training algorithms. These are important design decisions when building your own neural nets. In this question, you will explore some questions on neural network learning.

**(a)** (5 marks) In lecture, we discussed training a neural net $f_{\mathbf{w}}(\mathbf{x})$ for regression by minimizing the MSE loss

$$L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2,$$

where $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ are the training examples. However, a large neural net can easily fit irregularities in the training set, leading to poor generalization performance.

One way to improve generalization performance is to minimize a *regularized* loss function

$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \frac{1}{2}\lambda\|\mathbf{w}\|^2,$$

where $\lambda > 0$ is a user-specified constant. The regularizer $\frac{1}{2}\lambda\|\mathbf{w}\|^2$ assigns a larger penalty to $\mathbf{w}$ with larger norms, thus reducing the network's flexibility to fit irregularities in the training set. We can also interpret the regularizer as a way to encode our preference for simpler models.

Show that a gradient descent step on $L_\lambda(\mathbf{w})$ is equivalent to first multiplying $\mathbf{w}$ by a constant, and then moving along the negative gradient direction of the original MSE loss $L(\mathbf{w})$.

**(b)** (10 marks) In lecture, we described how we can convert an output vector $(o_1, \ldots, o_C) \in \mathbf{R}^C$ for a classification network to a probability distribution. The conversion operation is known as the softmax function, that is

$$\text{softmax}(o_1, \ldots, o_C) = (e^{o_1}/Z, \ldots, e^{o_C}/Z),$$

where $Z = e^{o_1} + \ldots + e^{o_C}$ is the normalization constant.

We can generalize the softmax function to the scaled softmax function

$$\text{softmax}_\beta(o_1, \ldots, o_C) = (e^{\beta o_1}/Z_\beta, \ldots, e^{\beta o_C}/Z_\beta),$$

where $\beta > 0$ is a user-specified constant, and $Z_\beta = e^{\beta o_1} + \ldots + e^{\beta o_C}$ is the normalization constant.

Consider applying the scaled softmax to an output vector in which the elements are not all identical. Show that when $\beta$ increases, the probability of the class with the largest output value increases.

Intuitively, the above result implies that when training to maximize the likelihood, we can give the classifier a larger incentive to be correct using a larger $\beta$.

A4 - 2

**4.** (35 marks) In lecture, we demonstrated how to implement a neural net in PyTorch using the OLS model as an example. In this question, you will implement another basic neural net, a multi-class logistic regression model, and explore how to train a good model.

Recall that in a multi-class logistic regression model, the probability that an input $\mathbf{x} \in \mathbf{R}^{d+1}$ belongs to class $y \in \{1, \ldots, C\}$ is given by

$$p(y \mid \mathbf{x}, \mathbf{w}_{1:C}) = \frac{e^{o_y}}{\sum_{y'} e^{o_{y'}}},$$

where $o_y = \mathbf{x}^\top \mathbf{w}_y$, and $\mathbf{w}_{1:C} = (\mathbf{w}_1, \ldots, \mathbf{w}_C)$ are the parameters of the logistic regression model. Note that as in lecture, here $\mathbf{x}$ has a dummy feature with value 1 in addition to $d$ given features.

We can train a logistic regression model by minimizing the log-loss

$$\min_{\mathbf{w}_{1:C}} -\frac{1}{n} \sum_{i=1}^{n} \ln p(y_i \mid \mathbf{x}_i, \mathbf{w}_{1:C})$$

or equivalently, maximizing the log-likelihood.

**(a)** (0 marks) You are given a file `logistic_regression.py`. The file contains code to load the `covtype` dataset, create a train-test split, and train a `LogisticRegression` model from sklearn. Run the code and play with the code to understand it.

In the documentation of predict_prob, "y: predicted class labels" should be "y: predicted class distributions". The file `logistic_regression.py` has been updated.

**(b)** (5 marks) Implement the `predict` function and the `predict_proba` in `logistic_regression.py`.

A naive way to calculate the class distribution is to compute $e^{o_i}$ values first, then normalize them to a distribution. This approach often suffers from numerical overflow in practice. To avoid this problem in your implementation, you can first subtract all $o_i$ values by their maximum, and then applying softmax to turn them into a probability distribution.

**(c)** (10 marks) Implement the `fit` function in `logistic_regression.py` to support training a logistic regression model by using gradient-descent to minimize the log-loss. Your implementation should allow users to specify the learning rate and number of iterations as written in the documentation for the `fit` function.

**(d)** (5 marks) Tune the learning rate and number of learning iterations to minimize the log-loss as much as possible. Describe how you do this. Report the training log-loss of the model that you obtain, and the training and test accuracies.

**(e)** (5 marks) Gradient descent can converge very slowly in practice, and various more efficient variants have been proposed. One variant is the momentum method.

Recall that when minimizing a general loss function $L(\mathbf{w})$, gradient descent with a constant learning rate $\eta$ updates the $t$-th iterate $\mathbf{w}_t$ to $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t$, where $\mathbf{g}_t = \nabla L(\mathbf{w}_t)$. Gradient descent momentum further moves along the previous direction $\mathbf{w}_t - \mathbf{w}_{t-1}$, and has an update rule of the form

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t + \beta(\mathbf{w}_t - \mathbf{w}_{t-1}), \tag{1}$$

where $\beta > 0$ is a constant. Intuitively, the momentum method tries to keep the 'momentum' by moving along a previous direction.

Show that if $\mathbf{w}_1 = 0$, $\mathbf{w}_2 = \mathbf{w}_1 - \eta \mathbf{g}_1$, and we apply the momentum method to obtain $\mathbf{w}_t$ for $t \geq 3$, then for any $t \geq 2$, we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{g}_t + \beta \mathbf{g}_{t-1} + \ldots + \beta^{t-1}\mathbf{g}_1). \tag{2}$$

**(f)** (5 marks) Modify your `fit` function to further support the momentum trick and tune the momemtum constant $\beta$ to try to make the log-loss as small as possible. Describe how you do this. Report the training log-loss of the model that you obtain, and the training and test accuracies.

You may find the `torch.optim.SGD` optimizer helpful.

**(g)** (5 marks) Another useful trick to speed up convergence is to normalize all features to have mean 0 and unit variance first. Implement this, and repeat (d) to try to use gradient descent to find a good model. Comment on the effectiveness of this trick.

You may find `sklearn.preprocessing.StandardScaler` helpful.