CS689: Machine Learning - Spring 2023

Homework 2

Getting Started: This assignment consists of two parts. Part 1 consists of written problems, derivations and coding warm-up problems, while Part 2 consists of implementation and experimentation problems. You should first complete the derivation and written problems in Part 1. You should then start on the implementation and experimentation problems in Part 2 of the assignment. The implementation and experimentation problems must be coded in Python 3.8+. Download the assignment archive from Moodle and unzip the file. The data files for this assignment are in the `data` directory. Code templates are in the `code` directory. The only modules you are allowed to use in your implementation are those already imported in the code templates.

Submission and Due Dates: Part 1 is due Monday, Mar 27. Part 2 is due Friday, Mar 31. Sorry for the acceleration but it's to give some time before the midterm. For Part 1, please submit a PDF with your solutions to Gradescope. (If you handwrite, please write neatly.) For Part 2, please submit both a PDF and code (as zip file) to Gradescope. Late work will only be accepted in accordance with the course's late homework policy.

Academic Honesty Reminder: Homework assignments are individual work. Being in possession of another student's solutions, code, code output, or plots/graphs for any reason is considered cheating. Sharing your solutions, code, code output, or plots/graphs with other students for any reason is considered cheating. Copying solutions from external sources (books, web pages, etc.) is considered cheating. Collaboration indistinguishable from copying is considered cheating. Posting your code to public repositories like GitHub (during or after the course) is not allowed. Manual and algorithmic cheating detection is used in this class. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

Part 1: Written Problems and Derivations

1. (*8 points*) Consider the regression loss function $L_\delta(r)$ for $\delta > 0$ shown below when answering the following questions. Note that r denotes the residual $r = y - f_\theta(\mathbf{x})$.

$$L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{for } |r| \leq \delta, \\ \delta(|r| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

a. (*4 pts*) Is $L_\delta(r)$ a differentiable function of $r \in \mathbb{R}$? Explain your answer.

b. (*4 pts*) Why might we consider using this regression loss function instead of the squared loss? Why might we consider using this regression loss function instead of the epsilon insensitive loss?

2. (*3 points*) Let's examine a trivial but illustrative version of L2 regularization. Consider the L2-regularized risk minimization problem for learning linear regression when there's only one feature ($x \in \mathbb{R}$)

and no bias term, so there is only one parameter w:

$$\hat{w} = \arg\min_{w} \left(\left(\frac{1}{N} \sum_{n=1}^{N} (y_n - w x_n)^2 \right) + \frac{\lambda}{2N} w^2 \right)$$

a. (*1 pt*) What is the derivative of the regularized risk? Show your work.
b. (*1 pt*) What is the closed-form solution for the optimal parameter? Show your work.
c. (*1 pt*) Based on this, give an intuitive explanation for what λ does. λ is called the regularization hyperparameter or regularization constant.

3. (*8 points*) OK, let's generalize to full linear regression. Consider the L2-regularized risk minimization problem shown below for learning the linear regression model.

$$\hat{\theta} = \arg\min_{\theta} \left(\left(\frac{1}{N} \sum_{n=1}^{N} (y_n - (\mathbf{x}_n \mathbf{w} + b))^2 \right) + \frac{\lambda}{2N} \|\mathbf{w}\|_2^2 \right)$$

a. (*4 pts*) Derive the gradient of the regularized risk for this problem. Show your work.

b. (*4 pts*) Like the unregularized version of the problem, this minimization problem has a closed-form solution for the optimal parameters $\hat{\theta}$. Derive this closed-form solution. Show your work.

4. (*5 points*) Consider the function $f(x) = x^2$. Using a direct application of the definition of convexity, show that $f(x)$ is convex.

5. (*6 points*) In lecture we discussed the perceptron loss,

$$L(z) = \begin{cases} -z & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$

It is a neat way to justify the classic perceptron algorithm as SSGD, it's convex, and it looks a bit like hinge loss. But it has odd properties from the perspective of global optimality.

a. (*2 pts*) Consider one-feature classification without a bias term: $h(x) = \text{sign}(wx)$. (This is a pretty poor model: $x = 0$ is always the decision boundary.) Consider the following training dataset $\mathcal{D}^{tr} = \mathcal{R} \cup \mathcal{L} \cup \{(+1, -1)\}$:

- $\mathcal{R}$ = A zillion points where $x > 10$ and $y = +1$
- $\mathcal{L}$ = A zillion points where $x < -10$ and $y = -1$
- $x = +1$, $y = -1$

What is the optimal w under perceptron loss? Is it a desirable model?

b. (*2 pts*) Consider the general case ($\mathbf{x} \in \mathbb{R}$) again. When the training data is linearly separable, that means there exists a classifier θ where for all training points $(\mathbf{x}_n, y_n)$, $y_n \theta^T \mathbf{x}_n > 0$. Now consider the case where the training data is not linearly separable. Part (a) was a special case of this. What is the set of optimal θ? Explain why.

c. (*2 pts*) What is the subdifferential of the perceptron loss, $\partial L(z)$?

6. (*20 points*) In addition to the use of basis expansion to achieve non-linearity, many models based on inner products can be made non-linear through the application of kernel functions. A kernel function $\mathcal{K}$ can often compute the inner product between two basis expanded vectors $\mathcal{K}(\mathbf{x}, \mathbf{x}') = [\phi(\mathbf{x})]^T \phi(\mathbf{x}')$ without computing the basis expansion explicitly, making them more computationally efficient. This also allows for the use of non-finite dimensional basis expansions.

The linear classifier can be "kernelized" using a discriminant function of the form $g_\theta(\mathbf{x}) = b + \sum_{n=1}^{N} \alpha_n \mathcal{K}(\mathbf{x}_n, \mathbf{x})$ where the parameters are $\theta = [\alpha_1, ..., \alpha_N, b]$. The terms $\mathbf{x}_n$ represent the training data cases. Note that unlike other models that we have seen to date, this model needs access to both the learned parameters and the training data used to learn the parameters to make predictions since the discriminant function depends on the training data cases $\mathbf{x}_n$. Also note that the number of parameters in the model does not vary with the dimension of the training data D. Instead, it varies with number of trainging data cases N.

We can form a regularized risk function for this model by combining the kernelized linear classifier with the logistic loss and a kernelized regularizer as shown below. This model is referred to as *kernel logistic regression* or KLR.

$$R(g_\theta, \mathcal{D}) = \left(\frac{1}{N} \sum_{n=1}^{N} \log\left(1 + \exp(-y_n g_\theta(\mathbf{x}_n))\right) \right) + \left(\frac{\lambda}{2N} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m \mathcal{K}(\mathbf{x}_n, \mathbf{x}_m) \right)$$

a. (*12 pts*) Derive the gradient of the regularized risk function shown above. Show your work.

b. (*4 pts*) One of the most widely used kernels is the radial basis function (RBF) kernel defined as $\mathcal{K}_{RBF}(\mathbf{x}, \mathbf{x}') = \exp(-\beta \|\mathbf{x} - \mathbf{x}'\|_2^2)$ where β is a kernel hyper-parameter. Using the first two data cases in the supplied training data set, compute and report the values $\mathcal{K}_{RBF}(\mathbf{x}_1, \mathbf{x}_1)$, $\mathcal{K}_{RBF}(\mathbf{x}_1, \mathbf{x}_2)$, and $\mathcal{K}_{RBF}(\mathbf{x}_2, \mathbf{x}_2)$ using $\beta = 0.1$.

c. (*2 pts*) Using the first two data cases in the supplied training data set, the RBF kernel with $\beta = 0.1$, the regularization strength $\lambda = 0.1$, and the parameter vector $\theta = [1, 1, 1]$, compute and report the value of the regularized risk.

d. (*2 pts*) Using the first two data cases in the supplied training data set, the RBF kernel with $\beta = 0.1$, the regularization strength $\lambda = 0.1$, and the parameter vector $\theta = [1, 1, 1]$, compute and report the gradient of the regularized risk.

Part 2: Implementation and Experimentation

7. (*50 points*) In this problem, you will implement learning for the RBF kernel logistic regression classifier introduced in Question 5. We will use the model to build and test a heart disease prediction model.

To begin, implement a Python class for the model starting from the code in `klr.py`. This code includes a constructor that stores the values of β, λ and the training data set Xtr. Your class must implement the methods indicated below. You can include any additional methods that you need, but please add them after the required methods. You can change the provided function signatures, so long as the required methods are consistent with the descriptions bellow. Note that for your implementation to be computationally efficient, you will need to cache the computation of kernel values during training. How you implement this functionality is up to you.

- `compute_kernel`: Takes two input matrices $\mathbf{X}$ and $\mathbf{X}'$ as input. Computes the kernel matrix $\mathbf{K}$ such that $\mathbf{K}_{nm} = \mathcal{K}_{RBF}(\mathbf{x}_n, \mathbf{x}'_m)$ using the stored value of β.
- `discriminant`: Takes an array of parameter values θ and an array of inputs $\mathbf{X}$. Returns the value of the discriminant function $g_\theta(\mathbf{x})$ using the stored value of β and the stored training data.
- `predict`: Takes an array of parameter values θ and an array of inputs $\mathbf{X}$. Returns an array of predicted outputs $\hat{\mathbf{Y}}$ computed using the stored value of β and the stored training data.
- `risk`: Takes an array of parameter values θ, an array of inputs $\mathbf{X}$, an array of outputs $\mathbf{Y}$. Computes the value of the risk using the stored values of β and λ.
- `risk_grad`: Takes an array of parameter values θ, an array of inputs $\mathbf{X}$, an array of outputs $\mathbf{Y}$. Computes the gradient of the risk using the stored values of β and λ.
- `fit`: Takes an array of inputs $\mathbf{X}$ and an array of outputs $\mathbf{Y}$. Fits the model using the stored values of β and λ. The optimizer we will use is `scipy.optimize.minimize` with the L-BFGS-B method and tol=1e-6. Use the stored values of β and λ during training. Use the $\theta = [0, ..., 0]$ as the starting point for the optimization.

a. (*5 pts*) Using $\beta = 0.1$ and your implementation of the `compute_kernel` method, compute and report the kernel matrix between the first three training data cases, and the first four test data cases. The result will be a 3×4 matrix.

Answer

[0.09357018 0.02918977 0.1500139 0.09995688
[0.06968701 0.01661554 0.242886 0.12241315] [0.05922661 0.0204159 0.15276502 0.28253882]]

b. (*5 pts*) Using $\beta = 0.1$, compute and report the value of the `discriminant` function for the first four test cases using a training data set consisting of the first three training cases. As the parameter vector, use $\theta = [1, 1, 1, -0.5]$. The result will be a length 4 vector.

Answer

[-0.27751619 -0.43377879 0.04566492 0.00490885

]

c. (*5 pts*) Using $\beta = 0.1$, compute and report the value of the `predict` function for the first four test cases using a training data set consisting of the first three training cases. As the parameter vector, use $\theta = [1, 1, 1, -0.5]$. The result will be a length 4 vector.

Answer
[-1. -1. 1. 1.
]

d. (*5 pts*) Using $\beta = 0.1$ and $\lambda = 0.1$, compute and report the value of the `risk` function using a training data set consisting of the first three training cases. As the parameter vector, use $\theta = [1, 1, 1, -0.5]$. The result will be a single real number.

Answer
1.6451804988265282

e. (*5 pts*) Using $\beta = 0.1$ and $\lambda = 0.1$, compute and report the value of the `risk_grad` function using a training data set consisting of the first three training cases. As the parameter vector, use $\theta = [1, 1, 1, -0.5]$. The result will be a length 4 vector.

Answer

[0.48213003, 0.56994837, 0.56219413, 0.78801513436508
]

f. (*5 pts*) Using $\beta = 0.1$ and $\lambda = 0.1$, use the `fit` function to learn the model using the first 10 training data cases. Report the value of the optimal model parameters $\hat{\theta}$. The result will be a length 11 vector.

Answer

[-1.09578631 -0.64489728 -0.7214171 -0.71084364 -1.13701259 -1.06009207 2.15855564 2.40217648 1.96209868 -1.15250871 -0.97151541
]

g. (*5 pts*) Using $\beta = 0.1$ and $\lambda = 0.1$, use `fit` to learn the model on all of the training data. Using the optimal model parameters θ, compute and report the classification error on the training data set and the test data set.

Answer
training Classification Error:3.864734299516903 Test Classification error:12.222222222222223

h. (*5 pts*)This model has two hyper-parameters β and λ. Describe in words (not code) a validation set-based procedure to select the values of these hyper-parameters. Make sure to mention the following: (1) your approach to using the available data including the split percentages used, (2) the range of values that you searched over for both β and λ and how you selected this range, (3) the criterion you used to select the best

hyper-parameter values. Also explain why you selected this procedure.

Answer
There are two methods to do hyperparameter selection and that is via random search and other is grid search. For this problem, grid search would be a better option to perform hyperparameter selection and tuning. Grid search was chosen over random search due to the time complexity issue. The grid search takes a range of continuous/discrete values for the hyperparameter and compares the model based on a chosen criterion. Having pre-defined values for the grid search helps perform better time complexity as it has better defined ranges. However the random search method , it is very difficult to tightly control the hyperparameters in a reasonable time frame especially with low level of computer architecture.

1) The length of training data set was 207, so usually a good split percentage is 75 percent for training data and 25 percent for the validation dataset. This would yeild us with 52 length of validation set. This is an acceptable range to do hyperparameter testing and accomplishing a healthy balance. I chose the first 52 elements for validation because it was a balanced class set.

2) The range of values that was chosen for both the parameters were 0.0001,0.001,0.01,0.1,1,10. This range was selected to determine the power and magnitude of the hyperparameter. We wanted to see for which power of 10, the loss would be the lowest. it is generally a good idea to search over different exponents of 10, to check where the robusticity of hyperparameter lies. As a further thing , we can then again run grid search over the chosen range of exponential.
3) The criterion was selected as the Validation classification error rate. This was chosen to see how well the hyperparameters were able to generalize on the validation set by checking the classification error rate

i. **(***5 pts***)** Implement and run your hyper-parameter selection procedure. Produce one or more plots showing how the training set error and the validation set error vary as a function of β and λ. Explain your plots. (Hint: for a well implemented model, your hyper-parameter experiment should take no more than a few minutes to complete.)

Answer

We did a grid search on the Beta and lambda values in the grid range of [0.0001,0.001,0.01,0.1,1,10]. Then we plotted two graphs of hyperparameter values lambda and beta vs the classification error. The two graphs corresponds to the classification error on the training set and validation set.
From the below graphs

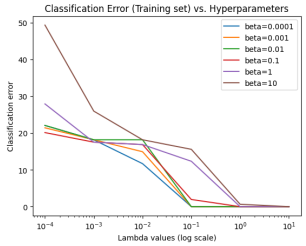

Figure 1: Classification error rate on Training set

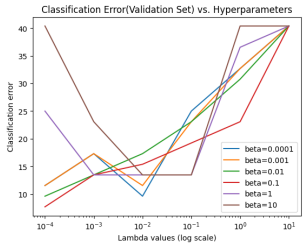

Figure 2: Classification error rate on Validation set

As evident from the last figure, we are getting the lowest classification error when we have lambda value equal to 0.0001 and among them the lowest validation classification error was reached by beta=0.1 as per the graph.
We can also see for the training classification error, for every beta , at lambda=10 we were massively over-fitting the data. hence it is not a very good choice for lambda.

j. **(***5 pts***)** Lastly, report the values of β and λ that you found to be optimal. Also report the value of the heart disease classification error on the training and test sets when the model is learned using the hyper-parameters you found to be optimal.
Answer
Best Beta is:0.1 Best lambda is: 0.0001
Training classification error:16.90821256038647 Test Classification error :11.111111111111116