3 Classification & Model Selection

Christopher Nota Edited 03/16/2021
Comments will be open until March 29, 5:00 PM

Christopher Nota 03/29/2021
Comments have been closed.

Course: COMPSCI 589 Machine Learning, Spring, 2021

Instructor: Justin Domke

Assignment: 3

Group work policy: You are allowed to complete this homework in teams of at most 3 students. You can use the Group Finder to help you find other students to work with. However, each student must submit their own individual .pdf and .zip files to Gradescope, and submit your own solutions to Kaggle. List your team members at the beginning of your `report.pdf`. You may verbally discuss the assignment with course staff or students outside your group. Please also list any students or course staff (separately) at the beginning of your `report.pdf`. However, you may not *look, copy, or show* any part of another student's assignment. Copying any part of another assignment — even a single sentence or line of code — from anyone outside your team is considered plagiarism. We use sophisticated tools to detect this. Please do not do it.

2

Due date: Mar 31, 5:00 pm

Submission instructions:

- For this assignment, you should prepare your solutions in one of three formats:
  - Latex (any style)
  - Markdown
  - Jupyter notebook
- Regardless of how you prepared the solutions, you should export a single .pdf file that you upload to Gradescope. The .pdf should be submitted to the Assignment 3: Classification assignment. Most coding question will ask you to include your code as text in the solution .pdf.

- Additionally, you **must submit a .zip file** to Assignment 3: ZIP file in Gradescope. Your .zip file should contain three things:
  - `report.pdf` - Your report.
  - `report_src/` - A directory containing all source files for the report.
  - `code/` - A directory containing Python code for all parts of the assignment.

  - `code/run_me.py` - A single Python file that will generate all figures included in your report.
- Finally, the assignment asks you to upload predictions to Kaggle. *Please create a Kaggle account using your umass.edu email address*. Otherwise, it is very difficult for us to map Kaggle submissions to students.
- If you use a Jupyter notebook, nothing changes. You still must put your code in external files, and you still must submit both a single .pdf and a .zip file containing the above components to the respective assignments in Gradescope.
- When you submit the .pdf to Gradescope, you must must mark page numbers for the different questions. We hate to do it, but we will penalize anyone who does not do this, as it creates a huge amount of difficulty for the graders.
- For the purpose of late days, the later of your two submissions will be considered the submission time for your assignment. E.g., if you submit your .pdf on time, but the .zip is two days late, the assignment will be considered two days late.

Code:

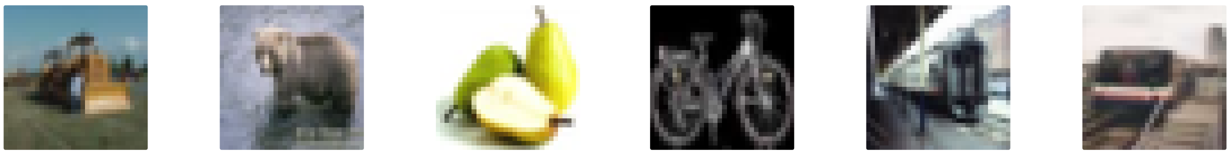
- For this assignment you **may** use the methods in `sklearn.tree`, `sklinear.linear_model`, `sklearn.svm` , `sklearn.metrics` and `sklearn.neighbors`. 6
- When using methods from `sklinear.linear_model` and `sklearn.svm` , after training them you can call them via `decision_function()` only. Do **not** use `predict` or `score` or `predict_proba`.
- You may also use `sklearn.model_selection.KFold` — but **not** any other methods in `sklearn.model_selection`.

- If the assignment asks you to implement a particular function, you are expected to implement it yourself. If you find that the function is implemented somewhere within `sklearn` or `np` but not specifically banned above, your implementation should *not* consist of a call to that function.

Preliminaries

Dataset

In this assignment you are given a set of 32x32 RBG images. There are four possible labels. Your goal will be to train a predictor to recognize what is in the image. Here are the first few elements of the training data, shown as images:



You are given a file `data.npz`.

data.npz 30013.1KB

The data can be loaded as follows:

```
stuff=np.load("data.npz") X_trn = stuff["X_trn"] y_trn =
stuff["y_trn"] X_tst = stuff["X_tst"] # no Y_tst !
```

There are a total of 6000 training examples, and 1200 test examples, each with 3072 dimensions. Those dimensions correspond to 32x32x3 RBG images (32*32*3=3072). If you like, you can plot an example with the following code:

```
from matplotlib import pyplot as plt def show(x): img =
x.reshape((3,32,32)).transpose(1,2,0) plt.imshow(img)
plt.axis('off') plt.draw() plt.pause(0.01) show(X_trn[7])
```

Kaggle

Kaggle is a very popular platform for creating and running machine learning competitions. It allows the creators of competitions to evaluate submissions against a secret test set, ensuring that competitors cannot "cheat" by fine-tuning their model against the test set.

The makers of Kaggle also created a set of features for creating an "InClass" competition, perfect for classes such as 589! We created a competition for Assignment 3 in which you are required to participate. However, don't worry! It's not truly a "competition" as much as it is a way to automatically evaluate your submissions and to familiarize you with the Kaggle platform. Again, to make it easier for us to grade, *please create a Kaggle account using your umass.edu email address*.

The "competition" has two leaderboards: A public leaderboard and private leaderboard. The test set is split into two sets: A "public" set containing about 30% of the data and a "private" set containing the remaining 70%. In a normal competition, you can see how well your submission is performing against the public set. In theory, one could use brute force to find all of the correct answers. For this reason, in most competitions submissions are scored against the "private" set. This assignment will, at various points, ask you to report the performance of various solutions according to the public leaderboard.

To submit solutions to Kaggle, you will be required to submit a `.csv` file with two columns: an "Id" column and a "Category" column classifying the integer prediction for each element in `X_tst`. A sample solution using randomly predicted outputs can be generated as follows:

2

```
import numpy as np import csv def write_csv(y_pred, filename):
"""Write a 1d numpy array to a Kaggle-compatible .csv file""" with
open(filename, 'w') as csv_file: csv_writer = csv.writer(csv_file)
csv_writer.writerow(['Id', 'Category']) for idx, y in
enumerate(y_pred): csv_writer.writerow([idx, y]) data =
np.load('data.npz') X_tst = data['X_tst'] y_pred =
np.random.randint(0, 3, size=len(X_tst)) # random predictions
write_csv(y_pred, 'sample_predictions.csv')
```

You can use the write_csv helper function in your code if you find it helpful to ensure that your solution is in the correct format.

Note that the leaderboard shows *accuracy* whereas the assignment in some places asks for *classification error*. Note that these are related by

$$\text{Classification Error} = 1 - \text{Accuracy},$$

so it is easy to translate between the two.

Simple Classifiers

Question 1 (5 points) Take a very small dataset with four scalar inputs:

$$\begin{aligned} x^{(1)} &= 1.0 \\ x^{(2)} &= 2.0 \\ x^{(3)} &= 3.0 \\ x^{(4)} &= 4.0 \end{aligned}$$

There are two possible labels, as shown below:

$$\begin{aligned} y^{(1)} &= 1 \\ y^{(2)} &= 0 \\ y^{(3)} &= 1 \\ y^{(4)} &= 1 \end{aligned}$$

For each of the following split points, what is the information gain? Show your work.

- Split at $x = 0.5$
- Split at $x = 1.5$
- Split at $x = 2.5$
- Split at $x = 3.5$
- Split at $x = 4.5$

Question 2 (5 points) Consider a classification tree with a maximum depth of M, trained on data wdith D dimensions. What is the time complexity to evaluate that classification tree on a single new input? Give an answer (Something like "order of $\log(M)\sqrt{D}$") and explain in at most 3 sentences why your answer is correct. 6

Question 3 (5 points) Take a dataset with N elements each with D dimensions. What is the time complexity to train a classification stump? Give an answer and explain why it's correct in at most 3 sentences. 2

Question 4 (6 points) Train 6 different classification trees on the image data, with each of the following maximum depths: {1,3,6,9,12,14}. (Do not apply any other restriction when growing the tree.) Using 5-fold cross validation, estimate mean the out of sample (generalization) classification error, and report this as a table. You should have one row for each possible depth and one number, which is the mean estimated error. 9+

Question 5 (6 points) What depth performs best in the previous question? Using that depth, make predictions on the test data, and upload your predictions to Kaggle. For this question, you need to report: 2

- What depth you chose.
- What was your estimated generalization error using 5-fold cross-validation.
- What accuracy you observed on the public part of the leaderboard.

Question 6 (5 points) Consider a dataset with N elements, each with D dimensions. What is the time complexity to evaluate a K-nearest neighbors classifier? Give an answer and explain why it's correct in at most 3 sentences. 6

Question 7 (6 points) Do nearest-neighbor prediction for each of the following possible values of K: {1, 3, 5, 7, 9, 11}. Using 5-fold cross-validation, estimate the out of sample classification error, and report this as a table. (Warning: This question might take a significant amount of computational time. You may consider using the `n_jobs` option.) 6

Question 8 (6 points) What K performs best in the previous question? Using that K, make predictions on the test data, and upload your predictions to Kaggle. Report:

- What value K you chose.
- What was your estimated generalization error using 5-fold cross validation.
- What accuracy you observed on the public part of the leaderboard.

Question 9 (10 points) For both hinge loss and logistic loss, train linear models with ridge regularization. That is, find w to minimize

$$\sum_{n=1}^{N} L(y^{(n)}, w^\top x^{(n)}) + \lambda \|w\|^2.$$

where L is the loss. For each loss and each of the regularization constants $\lambda \in \{10^{-4}, 10^{-2}, 1, 10, 100\}$, train a model and estimate the mean out of sample loss/error using 5-fold cross-validation. Organize your errors as a 5x2 table, with one row for each value of λ and one column for each training loss.

Give 3 tables: one where you estimate 0-1 classification error, one where you estimate logistic loss, and one where you estimate hinge loss. (You will report a total of 30 numbers.)

(**Hint**: You should be aware of `sklearn.svm.LinearSVC`. Again, you are not permitted to use `predict()` or `predict_proba()`. But `decision_function()` is OK.)

(**Hint**: There has been some confusion about this question. To clarify, you have 10 different training methods, corresponding to each combination of regularization constant (5 options) and training loss (2 options). For each of these training methods, you should estimate the generalization error using 5-fold cross-validation. But you should estimate that generalization error in three ways, for 0-1, logistic, and hinge loss. Since there are 10 training methods and 3 measures of generalization error, you report a total of 30 numbers. That is all that you report for this question.)

Question 10 (6 points) Choose the training loss and λ that you think will perform best on the public leaderboard. Make predictions for the test data and upload your predictions to Kaggle. Report:

1. What training loss and λ you chose
2. What was your estimated generalization error using 5-fold cross validation.
3. What accuracy you observed on the public part of the leaderboard.

Neural Networks

You will train several neural networks, each with a single hidden layer. These neural networks can be written as

$$f(x) = c + V\sigma(b + Wx).$$

Here:

- x is the input, a vector of length D
- W is a matrix of size $M \times D$ that maps input features to a hidden space
- b is the bias term for the hidden layer, a vector of length M
- $\sigma(a) = \tanh(a)$ is the activation function. You will need that the derivative is $\frac{d\sigma(a)}{da} = 1 - \tanh(a)^2$.
- V is a matrix of size $O \times M$ that maps the hidden space to the output space
- c is the bias term for the output space, a vector of length O

Note that $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}^O$ is a function that maps a vector to a vector. We will refer to the i-the component of the output as $f(x)_i$.

For this problem, we will use the logistic loss, defined as

4

$$L(y, f) = -f_y + \log \sum_{i=0}^{3} \exp(f_i),$$

where $y \in \{0, 1, 2, 3\}$ is the *label* for the input x, and $f \in R^O$ is the output vector. Note that f_y is therefore the y^{th} component of the output vector f. Also be careful to note that here, we are indexing f from 0 instead of 1.

Question 11 (5 points) Write a function to evaluate the neural network and loss. Your function should have the following signature:

3

```
def prediction_loss(x,y,W,V,b,c): # do stuff here return L
```

This should return a scalar. Give your function directly in your report.

Question 12 (10 points) Write a function to evaluate the gradient of the neural network. Your function should have the following signature. Do not use any packages outside of numpy.

```
def prediction_grad(x,y,W,V,b,c): # do stuff here return dLdW,
dLdV, dLdb, dLdc
```

Each returned array should be the same size as the input, and contain the corresponding gradient. So, for example, `dLdW` is the derivatives $\nabla_W L(y, f(x))$. Give your function directly in your report.

Question 13 (10 points) Take the following inputs, where there are 3 hidden units and 2 outputs ($y = 0$ or $y = 1$):

$$\begin{aligned} x &= [1, 2] \\ y &= 1 \\ W &= \begin{pmatrix} 0.5 & -1 \\ -0.5 & 1 \\ 1 & .5 \end{pmatrix} \\ V &= \begin{pmatrix} -1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \\ b &= [0, 0, 0] \\ c &= [0, 0] \end{aligned}$$

Run the function from the previous question to compute the gradient with respect to W, V, b, and c. Give the results directly in your report, organized as you see above.

Autograd

The next several questions use the `autograd` toolbox, which you can install via `pip install autograd`. A short demo of `autograd` can be found here:

Group Finder

Autograd Demo

Question 14 (5 points) Write a function to evaluate the same gradient as in Question 12 using the `autograd` toolbox (Hint: You will need to import the NumPy wrapper, `import autograd.numpy as np`, and the `grad` high-order function, `from autograd import grad`).

5

```
def prediction_grad_autograd(x,y,W,V,b,c): # do stuff here return
dLdW, dLdV, dLdb, dLdc
```

Give your function directly in your report. (You do not need to give any outputs from your function, but it is suggested to check the results against Q13 since if they are different, one must be wrong!)

Question 15 (5 points) Update your function from question 11. Instead of taking a single input x and a single output y, take an 2D of inputs X (where the first dimension indexes the different examples) and a 1D array of outputs Y. Also, take a regularization constant `λ` and apply squared regularization to `W` and `V`. Do not regularize `b` or `c`. Your function should be the sum of the logistic losses for each example in the dataset, plus the regularizer loss applied to `W` and `V`. (To be explicit, the regularizer could be written as $\lambda\left(\sum_{vm} W_{vm}^2 + \sum_{mi} V_{mi}^2\right)$.)

5

```
def prediction_loss_full(X,Y,W,V,b,c,λ): # do stuff here return L #
include regularization
```

Question 16 (5 points) Update your gradient function to work on a full dataset and include regularization, as in the previous question. Again, you should use autograd.

```
def prediction_grad_full(X,Y,W,V,b,c,λ): # do stuff here return
dLdW, dLdV, dLdb, dLdc
```

Question 17 (15 points) Here is psuedo-code to optimize a function $h(w)$ by gradient descent with momentum.

```
ave_grad = 0 for iter = 1, 2, ... max_iters: ave_grad = (1 -
momentum) ∗ ave_grad + momentum ∗ ∇h(w) w = w − stepsize ∗ ave_grad
```

For each size of the hidden layer, $M \in \{5, 40, 70\}$, train your neural network on the main data for this homework. Weights for layers W and V should be initialized by sampling from $\frac{\mathcal{N}(0,1)}{\sqrt{D}}$, where $\mathcal{N}(0, 1)$ is the standard normal distribution and D is the number of input dimensions for that layer. Weight for b and c should be initialized as zeros.

Use gradient descent with momentum, with 1000 iterations, a step size of 0.0001, a momentum of 0.1, and $\lambda = 1$.

Report the following:

1. For each value of M, what is the total training time (in ms) for all iterations. (Give a table with 3 entries.)
2. Make a plot of the training objective (regularized loss) as a function of iterations. This should be a single plot with 3 curves, one for each value of M. Include the plot in your report.

Question 18 (10 points) Make a single train-validation split of the data with 50% used for training and 50% for testing. Train your neural network using the parameters above for each value of M and give the estimated generalization error. Again, using the same initial weights generated using the scheme above. Then, retrain your network on *all* the data, make predictions for the Kaggle data, and upload to Kaggle. Report your accuracy on the public leaderboard. Report:

- What value of M you chose.
- What accuracy you expected.
- What accuracy you observed on the leaderboard.