SYSC 4415: Intro to Machine Learning, Winter 2023

Assignment 1

Submission instructions: Some answers will only require text while others require text+code+results. Please use the template solution available from the course GitHub repo under "Assignments/Assignment1". For text answers, like explanations, please write no more than 3 sentences.

Q1) Gradients. Calculate the gradient of the $f(x, y, z) = 7x^2z - 2xy^3 + 5z$ at (-1, -2, 3). What does this vector represent?

Q2) Data prep. Create a python notebook (that will be opened in Google Colab) which loads in three csv files (train.csv, val.csv, and test.csv). These can be found on GitHub (https://github.com/jrgreen7/SYSC4906/releases/tag/Ass1). Each file contains 1,000 rows (i.e., samples) and 5 columns. The first three columns (named A, B, and C) are input features, the fourth column (named Y) are outputs, and the final column (named Label) contains the continuous outputs binned into 10 categories. This last column, the labels, are integers 0 through 9. This dataset will be used for all remaining assignment questions.

a) Use pandas to load train.csv, val.csv, and test.csv into three separate dataframes. Then, create three scatter plots of the train set with features A, B, and C on the x-axes and Y on the y-axes.
b) If you were using A, B, and C to predict Y (i.e., using linear regression), how well do you think these features would perform? Do you think nonlinear functions would fit the data better? Why or why not?
c) The data is already normalized, how can you tell?
d) For each of the three input features, compute its square-root to create 3 additional features. Repeat for squaring the feature values to create another 3 features. You now have 9 total input features: A, B, C, sqrt(A), sqrt(B), sqrt(C), A^2, B^2, C^2. Do this for all three data splits (train, val, and test). You can think of these as polynomial kernels used in SVMs.

Q3) Regression. Scikit-learn is pre-installed on Google Colab. Use the default hyper-parameters for this question—i.e., use `model = LinearRegression().fit(X, y)`. In all cases, you are predicting Y. The binned versions of Y, named *Label*, are not used in Q3.

a) Train a linear regression model on the train set using *only* the 3 original input features.
b) What are the mean absolute errors of the train set, val set, and test sets using this model? Did the model overfit the train set?
c) What are the values of the 4 learned parameters (3 weights and 1 bias)? Do these values make sense given your scatter plots in Q2a?
d) Train another linear regression model using *all* 9 input features.

e) What are the mean absolute errors of each dataset, using this new model?
f) How many learned parameters are there for this model, and what are their values?
g) Which model performs better? How can you explain the difference in performance?

Q4) Classification. For classification, we want to predict the *Labels*, not the continuous Y values. These labels comprise 10 classes.

a) Use scikit-learn to train a logistic regression model on the train set using *only* the 3 original features. Again, just use the default hyper-parameters.
b) What are the overall accuracies of the train, val, and test sets? Use scikit-learn's *accuracy_score* metric.
c) Train another logistic regression model on the train set using *all* 9 features. If it doesn't converge, set max_iter to 5,000.
d) Repeat *b*, but with your new logistic regression model.
e) XGBoost is pre-installed on Google Colab. One hyper-parameter is the depth of the tree (called *max_depth*). Find the optimal value of max_depth, all other hyper-parameters can be ignored for this question. That is, your model should look like: `model = XGBClassifier(max_depth=L).fit(X,y)`. Hint: the test set should *not* be used to search for hyper-parameters. Train these models using *only* the 3 original features.
f) Repeat *Step b*, but with your new XGBoost model (with the max_depth chosen from *e*).
g) Of your 3 models—two logistic regression models and one XGBoost model—which performs best on the test set? Why do you think this is the case?

Q5) Linear Classification with PyTorch.

a) Using all 9 input features, we want to train a linear model (i.e., torch.nn.Linear) to predict the class label. This is a multi-class classification problem, which of the following loss functions, available in PyTorch, best suits our task? Explain briefly.
    i) torch.nn.CrossEntropyLoss()
    ii) torch.nn.MSELoss()
    iii) torch.nn.BCELoss()
b) Your model will output 10 numbers, one for each class. These are called *logits*. Do you need to compute the softmax across these 10 logits before applying your loss function (from *a*), or does the loss function compute the softmax for you?
c) Using the loss function from *a*, train a linear model using PyTorch and stochastic gradient descent (SGD). Use a batch size of 1, a learning rate of 0.3 and train for 200 epochs. You can update your model weights "manually" or use an optimizer.
d) Use the trained model to make predictions on the val and test sets. Calculate the overall accuracy of both sets.
e) Repeat parts *c* and *d* three times. You should get different results each run. There are two main reasons for this. What are they? Strong hints:
    i) SGD updates model parameters in steps, but they must start with some values…
    ii) Think about the term "stochastic" in SGD…