CS589: Machine Learning - Fall 2017

Homework 2: Classification

Assigned: October 10^{th}, 2018 Due: October 23^{rd}, 2018

Getting Started: In this assignment, you will train and evaluate different classification models on one dataset. **Please install Python 3.6 via Anaconda on your personal machine**. Download the homework file HW02.zip via Moodle. Unzipping this folder will create the directory structure shown below,

```
HW02
--- HW02.pdf
--- Data
    |--data_train.npz
    |--labels_train.npz
--- Submission
    |--Code
```

The data files are in 'Data' directory respectively. You will write your code under the Submission/Code directory. Make sure to put the deliverables (explained below) into the respective directories.

Deliverables: This assignment has two types of deliverables:

- **Report:** The solution report will give your answers to the homework questions (listed below). Try to keep the maximum length of the report to 5 pages in 11 point font, including all figures and tables. Reports longer than five pages will only be graded up until the first five pages. You can use any software to create your report, but your report must be submitted in PDF format. Make sure to properly label the pages for each question on Gradescope: 5 points will be deducted for any assignment that isn't properly labeled.
- **Code:** The second deliverable is the code that you wrote to answer the questions, which will involve implementing classification models. Your code must be Python 3.6 (no iPython notebooks or other formats). You may create any additional source files to structure your code, but the provided files must maintain their structure in their directories. You should submit your code files in a `.zip` file that, when unzipped, produces a folder named `Code` containing at least `data_io.py`, `DecisionTrees.py`, `NearestNeighbors.py`, `NeuralNetworks.py`, and `LinearModel.py`.

Kaggle/Gradescope Submissions: We will **not** be using Kaggle for this assignment. Instead you will receive some feedback from the Gradescope autograder as soon as you submit your code. This feedback is meant as guidance: do not rely on it. Your final grade will not be made available to you after the deadline has passed. You will be able to submit any part of your assignment on Gradescope repeatedly until three days after the deadline. Your final submission for any part of the assignment will determine the number of late days you use.

Submitting Deliverables: When you complete the assignment, you will upload your report and your code using the Gradescope.com service. Place your final code in Submission/Code. If you generated any figures place them under Submission/Figures. Finally, create a zip file of your submission directory, Submission.zip (NO rar, tar or other formats). Upload this single zip file on Gradescope as your solution to the 'HW02-Classification-Programming' assignment. Gradescope will run checks to determine if your submission contains the required files in the correct locations. Finally, upload your pdf report to the 'HW02-Classification-Report' assignment. When you upload your report please make sure to select the correct pages for each question respectively. Failure to select the correct pages will result in point deductions. The submission time for your assignment is considered to be the later of the submission timestamps of your code and report.

Academic Honesty Statement: Copying solutions from external sources (books, internet, etc.) or other students is considered cheating. Sharing your solutions with other students is also considered cheating. Posting your code to public repositories like GitHub, stackoverflow is also considered cheating. Any detected cheating will result in a grade of -100% on the assignment for all students involved, and potentially a grade of F in the course.

Task: You are given a set of gray scale images (32×32 pixels) with one (and only one) of the following objects: horse, truck, frog, ship (labels 0, 1, 2 and 3, respectively). The goal is to train a model to recognize which of the objects is present in an image. Some samples of the training images are:

Figure 1: horse

Figure 2: truck

Figure 3: frog

Figure 4: ship

You will train different models (Decision Trees, Nearest Neighbors, Linear models, Neural Networks), compare their performances and training time, and perform model selection using cross-validation.

The data set provided is the training set: $20,000$ samples with 1024 features each (pixel values, in this case). We will have a separate test set that we will run your code against in Gradescope.

Calling `data_io.read_image_data()` from any of the provided files (except `data_io.py`, of course) will result in a $N \times p$ matrix, where N is the number of training examples respectively and $p = 1024$ is the number of features. This is a numpy array containing values between 0 and 256 (which are normalized, ie. divide each value by 256). It also loads the labels for each sample in that matrix: 0 through 3, representing each of the categories. For this assignment you are allowed to use `sklearn.model_selection.*` functions unless otherwise stated. The documentation can be found here.

Note: There are 110 total points on this exam. You will be graded out of 100, with a cap at 100. This means that you can lose 10 points and still get 100 on this assignment, but getting 110 points is the same as getting 100.

Questions:

1. (*10 points*) **Decision trees:**

Train 5 different decision trees using the following maximum depths $\{3, 6, 9, 12, 14\}$. Using 5-fold cross-validation, estimate the out of sample error for each model, and report them using a table and a labeled bar graph. How does the maximum depth of the tree affect the estimated accuracy? Explain in at most 4 sentences. Choose the model with lowest estimated out of sample error, and record its max depth in the dictionary in the `best_hyperparams()` function. Upload your code to Gradescope to test your code against our test dataset. Is the predicted out of sample error close to the error reported by Gradescope on the test set? Make sure that your report clearly states which model was chosen and what was the predicted out of sample error for it.

2. (*20 points*) **Nearest neighbors:**

(10) a. A labeled dataset D with N samples, each of which consists of F features, is given. Suppose that a new sample X wants to be classified using KNN, what is the time complexity of this operation in terms of K, N, and F if a brute force approach is used?

(10) b. Train 5 different nearest neighbors classifiers using the following number of neighbors $\{3, 5, 7, 9, 11\}$. Using 5-fold cross-validation, estimate the out of sample error for each model, and report them using a table and a labeled bar graph. Choose the model with lowest estimated out of sample error, and record its max depth in the dictionary in the `best_hyperparams()` function. Upload your code to Gradescope to test your code against our test dataset. Is the predicted out of sample error close to the error reported by Gradescope on the test set? Make sure that your report clearly states which model was chosen and what was the predicted out of sample error for it.

3. (*20 points*) **Neural networks:**

(10) a. Specify a neural network that implements the Boolean function:

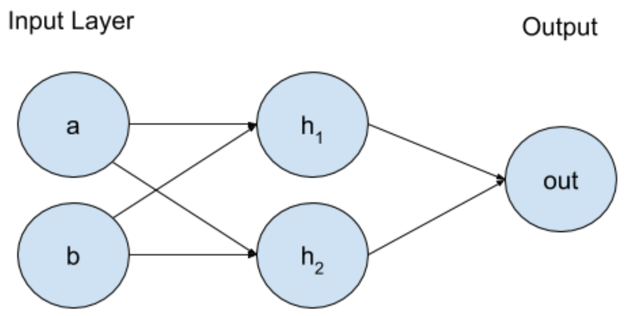
$$f(a, b, c, d) = ((\neg a \vee b) \wedge c) \vee d$$

Use only one hidden layer. Use the tanh function for non-linearities. Assume that, for the inputs, a TRUE value is represented as `1`, and a `FALSE` value is represented as `-1`. For the final output, assume a value higher than `0` counts as `TRUE`, and a value less than `0` counts as `FALSE`. Describe how you are approaching this translation problem.

Example: For the function

$$f(a, b) = a \oplus b = (a \vee b) \wedge \neg(a \wedge b)$$

your NN might look like the following:



Where $h_1 = \tanh(w_1a + w_2b + b_1)$, $h_2 = \tanh(w_3a + w_4b + b_2)$, and $out = w_5h_1 + w_6h_2 + b_3$, where $w_1 = w_2 = w_5 = w_6 = b_1 = b_2 = 1$, and $w_3 = w_4 = b_3 = -1$

(10) b. Train 3 neural networks using $\{5, 40, 70\}$ hidden-layer neurons. Using 5-fold cross-validation, estimate the out of sample error for each model, and report them using a table and a labeled bar graph. Choose the model with lowest estimated out of sample error, and record its max depth in the dictionary in the `best_hyperparams()` function. Upload your code to Gradescope to test your code against our test dataset. Is the predicted out of sample error close to the error reported by Gradescope on the test set? Make sure that your report clearly states which model was chosen and what was the predicted out of sample error for it.

4. (*10 points*) Compare the classification algorithms:

Between decision trees, nearest neighbors, and neural networks, which classification algorithm worked the best for you? Were there trade-offs between them?

For a much larger training set and an optimal choice of hyperparameters, which algorithm do you expect to preform best in terms of accuracy? In terms of efficiency? Explain why using your theoretical knowledge of the algorithms.

5. (*50 points*) Linear model:

For the previous questions, you've been using out-of-the-box `sklearn` models like `sklearn.neural_network.MLPClassifier` and `sklearn.tree.DecisionTreeClassifier`. In this question you'll be implementing parts of a logistic regression classifier on your own. Your job will be to finish the classifier outlined in `LinearModel.py`. We've made this classification problem a little easier for you than the previous ones: you won't be expected to distinguish horses from trucks from frogs anymore. The only thing you have to worry about is whether the sample is a ship or not. There's an example of how to adapt the data labels for binary classification in `main()`. As you write the classifier, please answer a few questions about the process.

(5) a. Recall from lecture that a logistic regression classifier predicts a $y_i \in \{0, 1\}$ for each sample x_i. It does so by learning β, then computing

$$p_i = P(y_i = 1|x_i) = \frac{1}{1 + e^{-\beta^\top x_i}}$$

p_i gives us a sense of the model's confidence, but when it comes time to predict $\hat{y}_i = 1$ or $\hat{y}_i = 0$, if $p_i > 0.5$, it predicts $\hat{y}_i = 1$. Otherwise it predicts $\hat{y}_i = 0$. Your first task will be to implement this behavior in `predict_prob()`, which will, upon taking an array of samples `X` with shape `n` samples by `P` features, returns `p`, an array of p_i values: one for each sample. We've provided you with `predict()`, which wraps `predict_prob()` to return $\hat{y}_i$ values instead of p_i values.

(5) b. Suppose you have a model that takes three features, and you have learned

$$\beta = \begin{bmatrix} 0.07 \\ 0.03 \\ 0.04 \end{bmatrix}$$

Hand/calculator-compute your predicted p_i (to 5 decimal places) and $\hat{y}_i$ for sample

$$x_i = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Show your work.

(10) c. Recall, once again from lecture, that the objective function for logistic regression is

$$G(\beta) = \sum_{i=1}^{n} - (y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

which comes from the following per-sample loss function:

$$l(i) = - (y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

The gradient of l is given by

$$\nabla l(i) = - (y_i - p_i) \, x_i$$

We've already implemented the loss function, `loss()`, in `LinearModel.py`. Your job is to implement `loss_grad()`, which computes the gradient function ∇l.

Note that the loss function is in terms of p_i, not $\hat{y}_i$. This is because we not only want correct predictions, but also want reasonable confidence values for some applications. This formulation gives us some trouble if the data is separable, however.

Imagine one class of samples has just one feature, whose values range from 0 to $1,000,000$, and another class of samples whose values for the feature range from $1,000,001$ to $2,000,000$. Ideally we would train a model that was less confident on samples whose feature value was close to $1,000,000.5$, and more confident on samples whose feature values are far from $1,000,000.5$. Unfortunately, since our loss function punishes low-confidence correct predictions, not just incorrect predictions, our model will end up predicting anything $\leq 1,000,000$ as being in the first category, and anything $\geq 1,000,001$ as being in the second category with arbitrarily high confidence. In other words, the learned logistic curve will be nearly horizontal up until $1,000,000$, nearly vertical between $1,000,000$ and $1,000,001$, and nearly horizontal again at and above $1,000,001$

The solution to this problem is one we've seen before: shrinkage using L2 regularization. The objective function for logistic regression with L2 regularization is as follows:

$$\begin{aligned} G_{L2}(i) &= \sum_{i=1}^{n} -\left(y_i \log p_i + (1-y_i)\log(1-p_i)\right) + \lambda ||\beta||_2^2 \\ &= \sum_{i=1}^{n} -\left(y_i \log p_i + (1-y_i)\log(1-p_i)\right) + \lambda \sum_{j=1}^{P} \beta_j^2 \end{aligned}$$

which comes from the following loss function:

$$\begin{aligned} l_{L2}(i) &= -\left(y_i \log p_i + (1-y_i)\log(1-p_i)\right) + \frac{\lambda ||\beta||_2^2}{n} \\ &= -\left(y_i \log p_i + (1-y_i)\log(1-p_i)\right) + \frac{\lambda \sum_{j=1}^{P} \beta_j^2}{n} \end{aligned}$$

where λ is a constant chosen to scale the strength of the regularization.

(5) d. What is the gradient ∇l_{L2} of the logistic regression loss function with L2 regularization? Show your work.

(5) e. Change the implementation of `loss()` and `loss_grad()` in `LinearModel.py` to use L2 regularization with regularization constant `self._lambda`. We'll grade your **5 c.** implementation by setting `self._lambda=0`.

For this problem, instead of solving analytically for the optimal β, we're going to use Stochastic Gradient Descent to learn our parameters. Recall that SGD updates β for each x_i according to the following formula:

$$\beta \leftarrow \beta - \alpha \nabla l_{L2}(i)$$

where α is a constant controlling the speed of the descent. Usually SGD is run in multiple passes over the training data, called epochs.

(5) f. Suppose that on every element i of your training data, $\nabla l(i) = \mathbf{0}$. How would SGD behave? What does this mean about the relationship between your data and the model?

(5) g. Implement `SGD_fit()` in `LinearModel.py`. Use $\beta = \mathbf{0}$ as your starting value for `self._beta`.

(10) h. Finally, it's time to run experiments with your model! Train a linear model using $\lambda \in \{0, 100\}$ and $\alpha \in \{10^{-6}, 10^{-4}, 10^{-2}, 1, 10\}$ (you will train 10 classifiers). Using 5-fold cross-validation, estimate the out of sample error for each model, and report them using a table and a labeled bar graph. Choose the model with lowest estimated out of sample error, and record its max depth in the dictionary in the `best_hyperparams()` function. Upload your code to Gradescope to test your code against our test dataset. Is the predicted out of sample error close to the error reported by Gradescope on the test set? Make sure that your report clearly states which model was chosen and what was the predicted out of sample error for it.