Assignment (3)

Artificial Intelligence and Machine Learning (24-787 Fall 2020)

Due Date: 2020/14/10 (Wed) @ 11:59 pm EST

All the files must be put into one directory **andrewID-HW3/**. Convert the jupyter notebook to a pdf file. Ensure that the submitted notebooks have been run and the cell outputs are visible - **Hint:** Restart and Run All option in the Kernel menu. Make sure all plots are visible in the pdf. Zip the directory **andrewID-HW3/** and submit the zip on canvas. You can refer to Numpy documentation while working on this assignment. **ANY** deviations from the submission structure shown below would attract penalty to the assignment score. Please use Piazza for any questions on the assignment.

andrewID-HW3/
- **q1.ipynb**
- **q1.pdf**
- **q2.ipynb**
- **q2.pdf**
- **q3.ipynb**
- **q3.pdf**

Submission file structure

PROBLEM 1

Gaussian Naive Bayes for binary classification **[35 points]**

In this problem, you will implement Gaussian Naive Bayes algorithm for binary classification. Since we are dealing with continuous dataset, the likelihood of the features is assumed to be Gaussian:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

where x_i is the i-th feature of data point x and μ_y, σ_y^2 are the mean and variance of x_i given label y. In the binary classification case, $y = 0$ or 1.

Dataset introduction In this problem, we have both train **(q1_train.csv)** and test dataset **(q1_test.csv)**. For each dataset, the first 200 columns represent the 200 features, and the last column represents the binary labels. Therefore, the input shape is (N, 200) and label shape is (N,), where N is the number of samples.

Important note In this problem, we will use some large datasets. In order to make your life easier, you only need to follow the code structure in q1.ipynb and modify the "TODO" part. You will find lots of hints there, so please start with q1.ipynb.

a) (Programming problem) First of all, let's load data **(q1_train.csv**, **q1_test.csv)** and split them into **trainX**, **trainY**, **testX**, and **testY**. (Hint: use the *Pandas* module for this part)

b) (Programming problem) Write **your own** solver and make predictions for both train and test sets. You will first finish the ***myNBSolver()*** function, and use the provided driver to test your results. When writing your ***myNBSolver()*** function, you can follow the step-by-step instructions in the template code, like computing $P(y)$, computing $P(x_i|y)$, making predictions based on $P(y|x_1, x2, ...xi)$, etc.

c) (Programming problem) Use *sklearn.naive_bayes.GaussianNB* module to validate your train and test accuracies. Like problem b), you can first modify the ***skNBSolver()*** function and then test your results.

PROBLEM 2

Linear Regression with Regularization **[25 points]**

In this problem, you'll learn the power of ***L1*** regularization (Lasso) and ***L2*** regularization (Ridge) in a linear regression example.

a) (Programming problem) First of all, let's create a data sample using the function $f(x)$ given by:

$$f(x) = \sin(2.2\pi x + 0.8)$$

From now on, $f(x)$ will be the true model we want to estimate or approximate. Now you'll need to generate $m = 30$ sample points $\{x^{(i)} \in [0, 1), i = 0, 1, \cdots, m-1\}$, using 0 random seed. To sample y's, we'll add a small Gaussian noise to $f(x)$. Specifically, $\{y^{(i)}, i = 0, 1, \cdots, m-1\}$ are obtained by

$$y^{(i)} = f(x^{(i)}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.1^2)$$

In the above function, ϵ is the Gaussian noise with standard deviation 0.1. Plot $f(x)$ together with the sampled data points within the range $[0, 1]$. Your plot should be similar to Fig. 1.

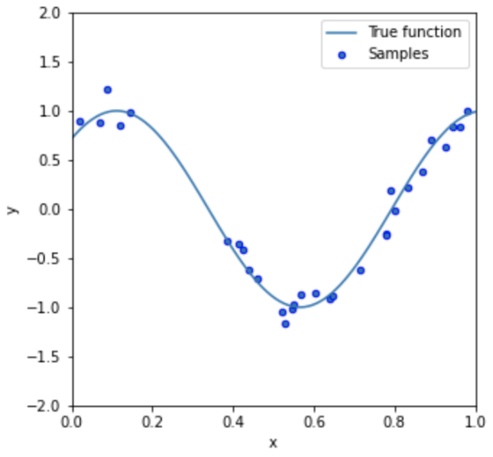

Fig. 1: The true model $f(x)$ and samples.

b) (Programming problem) Use the ***sklearn*** module to fit a linear model $y^{(i)} = w_0 + w_1 x^{(i)}$ to the sample data. Report $w = [w_0, w_1]$ and draw the fitted line on the sample data. (Hint: sklearn.linear_model.LinearRegression())

c) (Programming problem) It's obvious that a linear model is underfitting in this case. Now let's try fitting a $n = 15$ degree polynomial curve to the sampled data:

$$h(x) = \sum_{j=0}^{n} w_j x^j$$

Note this becomes a linear regression problem if we treat $[x, x^2, x^3, \cdots, x^n]$ as the transformed feature point. Now use the ***sklearn*** module to fit a linear model to the transformed sample data. Report $w = [w_0, \cdots, w_n]$ and draw the fitted curve on the sample data.

d) (Programming problem) If your implementation in **c)** is correct, you'll see the model is overfitted to the noisy data. To overcome this issue, we'll perform linear regression with L1 regularization.

With the transformed data points, the cost function becomes:

$$\frac{1}{2m}\sum_{i=0}^{m-1}||h(x^{(i)})-y^{(i)}||^2+\lambda\sum_{j=0}^{n}|w_j|$$

where λ is the hyper-parameter to regularize w. Start with $\lambda = 0.01$, then use the ***sklearn*** module to solve the Lasso regression problem (Hint: sklearn.linear_model.Lasso(), you may also need to tweak λ to find a better solution). Report the optimal values of w and corresponding λ. Again, plot the fitted curve with the sample data. Finally, briefly describe your observation of the values of w and λ. Explain your understanding of Lasso regularization. What effects do you observe when you tweak λ?

e) (Programming problem) Now implement L2 (Ridge) regularization using ***sklearn***.

With the transformed data points, the cost function now becomes:

$$\frac{1}{2m}\sum_{i=0}^{m-1}||h(x^{(i)})-y^{(i)}||^2+\lambda\sum_{j=0}^{n}(w_j)^2$$

You may need to adjust λ to find a good fit. Plot the fitted curve with the sample data.

PROBLEM 3

Logistic Regression with Non-linear Decision Boundaries and Regularization [40 points]

In homework 2, you were asked to train a binary classifier using Logistic Regression (LR). This time, let's see how LR can be used to deal with datasets that are not linearly separable.

a) (Programming problem) Load and plot the data points in **(q3_data.csv)**. There are 3 columns in the given file. The first 2 columns are data points $\{x^{(i)} = (x_1^{(i)}, x_2^{(i)}), i = 0, 1, \cdots, m-1\}$ (m is the number of data points). And the 3rd column is the label $\{y^{(i)}, i = 0, 1, \cdots, m-1\}$. Points with different labels should be in different color. Output your plot.

b) (Programming problem) You should have noticed that there is no way a linear decision boundary would separate those data points. This case is called ***not linearly separable*** in the original space. Therefore, we'll transform the original 2D data points into a higher dimension space, where the transformed data points might become linearly separable again.

Now implement a function named ***map_feature(...)*** to transform the original 2D data into 28D data using the following transformation:

$$\text{map\_feature}(x) = \begin{bmatrix}1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & \cdots & x_1x_2^5 & x_2^6\end{bmatrix}^T$$

The implemented function should take the original data matrix ($2 \times m$, not augmented with 1's) as input and return the transformed data matrix ($28 \times m$). This process is also called ***feature mapping***.

c) (Programming problem) Formulating the LR problem in the transformed space is exactly the same as before (in homework 2). The cost function is given by

$$J(w) = \frac{1}{m}\sum_{i=0}^{m-1}\left[-y^{(i)}\log\left(h(w^Tx^{(i)})\right) - (1-y^{(i)})\log\left(1 - h(w^Tx^{(i)})\right)\right]$$

and the gradient is given by

$$G(w) = \frac{\partial J(w)}{\partial w} = \frac{1}{m} \sum_{i=0}^{m-1} \left(h(w^T x^{(i)}) - y^{(i)} \right) x^{(i)}$$

where $h(\cdot)$ is the sigmoid function. Differently, the parameter w is of size 28×1 instead of 3×1 (including bias w_0).
It seems we are ready to implement a non-linear classifier. But hold on, we are not there yet!

While the feature mapping allows us to build a more expressive classifier, it is also more susceptible to overfitting. We'll use the regularization techniques taught in the lecture to combat the overfitting problem. The regularized cost function looks like this:

$$J(w) = \frac{1}{m} \sum_{i=0}^{m-1} \left[-y^{(i)} \log \left(h(w^T x^{(i)}) \right) - (1 - y^{(i)}) \log \left(1 - h(w^T x^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n-1} w_j^2$$

where λ is the hyper-parameter to regularize w and $n = 28$. In the regularized logistic regression, the bias parameter w_0 should be excluded from regularization. So in the above cost function, the summation starts at $j = 1$ (corresponding to w_1). The gradients are as follows:

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} \left(h(w^T x^{(i)}) - y^{(i)} \right) x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} \left[\left(h(w^T x^{(i)}) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \qquad \text{for } j \geq 1$$

Now implement a function named ***logistic_regression_regularized(...)*** to solve the regularized logistic regression problem. Initialize w to be zeros and $\lambda = 1$. Choose your favorite gradient descent method (GD, SGD or mini-batch) and use proper learning rate and number of iterations. You may recycle the code from homework 2. Report the optimal values of w and plot the decision boundary in the original 2D space (Hint: numpy.meshgrid and matplotlib.pyplot.contour will be your friend). Your decision boundary should be similar to Fig. 2.

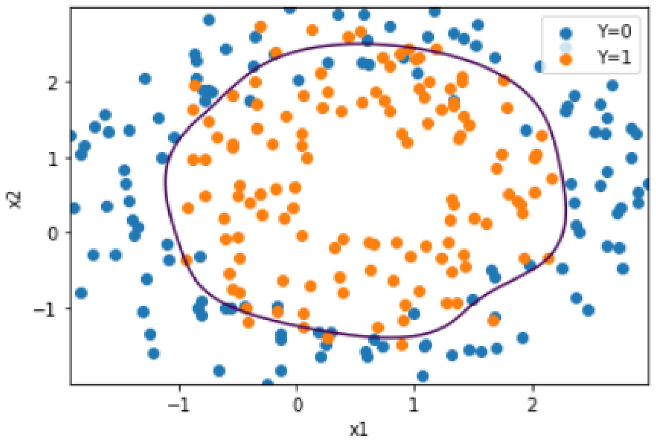

Fig. 2: Decision boundary with $\lambda = 1.0$.

d) (Programming problem) Compare the results of $\lambda = 0, 1$ and 100. How does changing λ affect the decision boundary? Why is the decision boundary affected in that way? Write your answer in the markdown cell provided.