CP8318/CPS803: Machine Learning (Fall 2020)

Assignment 1

Due: Friday October 9, 2020 (noon 12:00PM)

Instructor: Nariman Farsad

Please Read Carefully:

- Submit your assignments on D2L before the deadline. The submission will be closed after the deadline.
- Show your work and write your solutions legibly. You can use applications such as Word or Latex to type up your solutions for the written portion of the assignment or use applications such as CamScanner to create a PDF file from your handwritten solutions. For coding problems you need to submit your python source files or Jupyter notebook, and have the main plots in the written portion.
- Do the assignment on your your own, i.e., do not copy. If two assignments are found to be copies of each other, they BOTH receive 0.
- These questions require thought, but do not require long answers. Please be as concise as possible.
- If you have a question about this homework, we encourage you to post your question on our D2L discussion board.
- Students in CP8318 are required to answer all questions in the assignment. CPS803 students are not required to answer the questions marked for CP8318 students. These questions will not be graded for CPS803 students and they can just attempt them for practice.
- For instructions on setting up the python environment for the assignments read the "README.md" file.
- For the coding problems, you may not use any libraries except those defined in the provided "environment.yml" file. In particular, ML-specific libraries such as scikit-learn are not permitted.

1. (30 points) **Linear regression**

   We have learned about linear regression in the class. Here we explore what do we mean by "linear". Specifically, we will see how linear regression can be used to fit non-linear functions of the data using feature maps, and explore some of its limitations.

   1. [5 points] **Learning degree-3 polynomials of the input**

      Suppose we have a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ where $x^{(i)}, y^{(i)} \in \mathbb{R}$. We would like to fit a third degree polynomial $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the dataset. The key observation here is that the function $h_\theta(x)$ is still linear in the unknown parameters θ, even though it's not linear in the input x. This allows us to convert the problem into a linear regression problem as follows.

      Let $\phi : \mathbb{R} \to \mathbb{R}^4$ be a function that transforms the original input x to a 4-dimensional vector defined as

      $$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \tag{0.1}$$

      Let $\hat{x} \in \mathbb{R}^4$ be a shorthand for $\phi(x)$, and let $\hat{x}^{(i)} \triangleq \phi(x^{(i)})$ be the transformed input in the training dataset. We construct a new dataset $\{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^n = \{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ by replacing the original inputs $x^{(i)}$'s by $\hat{x}^{(i)}$'s. We see that fitting $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the old dataset is equivalent to fitting a linear function $h_\theta(\hat{x}) = \theta_3 \hat{x}_3 + \theta_2 \hat{x}_2 + \theta_1 \hat{x}_1 + \theta_0$ to the new dataset because

      $$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{0.2}$$

      In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \ldots, \theta_3$.

      Please write down 1) the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ and 2) the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$.

      Terminology: In machine learning, ϕ is often called the feature map which maps the original input x to a new set of variables. To distinguish between these two sets of variables, we will call x the input **attributes**, and call $\phi(x)$ the **features**. (Unfortunately, different authors use different terms to describe these two things. In this course, we will do our best to follow the above convention consistently.)

   2. [5 points] **Coding question: degree-3 polynomial regression**

      For this sub-question question, we will use the dataset provided in the following files:

      `src/featuremaps/{train,valid,test}.csv`

      Each file contains two columns: x and y. In the terminology described in the previous part, x is the attribute (in this case one dimensional) and y is the output label.

      Using the formulation of the previous sub-question, implement linear regression with **normal equations** using the feature map of degree-3 polynomials. Use the starter code provided in `src/featuremaps/featuremap.py` to implement the algorithm.

      Create a scatter plot of the training data, and plot the learnt hypothesis as a smooth curve over it. Submit the plot in the writeup as the solution for this problem.

Remark: Suppose $\widehat{X}$ is the design matrix of the transformed dataset. You may sometimes encounter a non-invertible matrix $\widehat{X}^T\widehat{X}$. For a numerically stable code implementation, always use `np.linalg.solve` to obtain the parameters directly, rather than explicitly calculating the inverse and then multiplying it with $\widehat{X}^T y$.

3. [5 points] **Coding question: degree-3 polynomial GD and SGD**

   Repeat the previous question but this time implement the gradient descent algorithm as well as the stochastic gradient descent algorithms. Run each of these algorithms for 100, 1000, 10000 iterations. Start with $\alpha = 0.000002$, experiment with different values of α and record your observations. Finally, plot the degree-3 polynomial for GD, SGD, and using the normal equation in one plot, using the α and number of iterations that gives you the best match to the normal equation. Were you able to train a model using GD and SGD that is similar to model from the normal equation?

4. [5 points] **Coding question: degree-k polynomial regression**

   Now we extend the idea above to degree-k polynomials by considering $\phi : \mathbb{R} \to \mathbb{R}^{k+1}$ to be

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{0.3}$$

   Follow the same procedure as the previous sub-question, and implement the algorithm with $k = 3, 5, 10, 20$. Create a similar plot as in the previous sub-question, and include the hypothesis curves for each value of k with a different color. Include a legend in the plot to indicate which color is for which value of k.

   Submit the plot in the writeup as the solution for this sub-problem. Observe how the fitting of the training dataset changes as k increases. Briefly comment on your observations in the plot.

5. [5 points] **Coding question: other feature maps**

   You may have observed that it requires a relatively high degree k to fit the given training data, and this is because the dataset cannot be explained (i.e., approximated) very well by low-degree polynomials. By visualizing the data, you may have realized that y can be approximated well by a sine wave. In fact, we generated the data by sampling from $y = \sin(x) + \xi$, where ξ is noise with Gaussian distribution. Please update the feature map ϕ to include a sine transformation as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{0.4}$$

   With the updated feature map, train different models for values of $k = 0, 1, 2, 3, 5, 10, 20$, and plot the resulting hypothesis curves over the data as before.

Submit the plot as a solution to this sub-problem. Compare the fitted models with the previous sub-question, and briefly comment about noticable differences in the fit with this feature map.

6. [5 points] **Overfitting with expressive models and small data**

   For the rest of the problem, we will consider a small dataset (a random subset of the dataset you have been using so far) with much fewer examples, provided in the following file:

   `src/featuremaps/small.csv`

   We will be exploring what happens when the number of features start becoming bigger than the number of examples in the training set. Run your algorithm on this small dataset using the following feature map

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{0.5}$$

   with $k = 1, 2, 5, 10, 20$.

   Create a plot of the various hypothesis curves (just like previous sub-questions). Observe how the fitting of the training dataset changes as k increases. Submit the plot in the writeup and comment on what you observe.

   Remark: The phenomenon you observe where the models start to fit the training dataset very well, but suddenly "goes wild" is due to what is called *overfitting*. The intuition to have for now is that, when the amount of data you have is small relative to the expressive capacity of the family of possible models (that is, the hypothesis class, which, in this case, is the family of all degree k polynomials), it results in overfitting.

   Loosely speaking, the set of hypothesis function is "very flexible" and can be easily forced to pass through all your data points especially in unnatural ways. In other words, the model explains the noises in the training dataset, which shouldn't be explained in the first place. This hurts the predictive power of the model on test examples. We will describe overfitting in more detail in future lectures when we cover learning theory and bias-variance tradeoffs.

2. (30 points) **Incomplete, Positive-Only Labels**

In real life, we may not have access to labels. In this problem, we consider binary classification and a scenario, where we have labels only for a subset of the positive examples. All the negative examples and the rest of the positive examples are unlabelled.

We formalize the scenario as follows. Let $\{(x^{(i)}, t^{(i)})\}_{i=1}^{n}$ be a standard dataset of i.i.d distributed examples. Here $x^{(i)}$'s are the inputs/features and $t^{(i)}$ are the "true" labels. Now consider the situation where $t^{(i)}$'s are not observed by us. Instead, we only observe the labels of some of the positive examples. Concretely, we assume that we observe $y^{(i)}$'s that are generated by

$$\begin{aligned}
&\forall x, \quad p(y^{(i)} = 1 \mid t^{(i)} = 1, x^{(i)} = x) = \alpha, \\
&\forall x, \quad p(y^{(i)} = 0 \mid t^{(i)} = 1, x^{(i)} = x) = 1 - \alpha \\
&\forall x, \quad p(y^{(i)} = 1 \mid t^{(i)} = 0, x^{(i)} = x) = 0, \\
&\forall x, \quad p(y^{(i)} = 0 \mid t^{(i)} = 0, x^{(i)} = x) = 1
\end{aligned}$$

where $\alpha \in (0, 1)$ is some unknown scalar. In other words, if the unobserved "true" label $t^{(i)}$ is 1, then with α chance we observe a label $y^{(i)} = 1$. On the other hand, if the unobserved "true" label $t^{(i)} = 0$, then we always observe the label $y^{(i)} = 0$.

Our final goal in the problem is to construct a binary classifier h of the true label t, with only access to the partial label y. In other words, we want to construct h such that $h(x^{(i)}) \approx p(t^{(i)} = 1 \mid x^{(i)})$ as closely as possible, using only x and y.

Real world example: Suppose we maintain a database of proteins which are involved in transmitting signals across membranes. Every example added to the database is involved in a signaling process, but there are many proteins involved in cross-membrane signaling which are missing from the database. It would be useful to train a classifier to identify proteins that should be added to the database. In our notation, each example $x^{(i)}$ corresponds to a protein, $y^{(i)} = 1$ if the protein is in the database and 0 otherwise, and $t^{(i)} = 1$ if the protein is involved in a cross-membrane signaling process and thus should be added to the database, and 0 otherwise.

For the rest of the question, we will use the dataset and starter code provided in the following files:

- `src/posonly/{train,valid,test}.csv`
- `src/posonly/posonly.py`

Each file contains the following columns: x_1, x_2, y, and t. As in Problem 1, there is one example per row. The $y^{(i)}$'s are generated from the process defined above with some unknown α.

1. [5 points] **Coding problem: ideal (fully observed) case**

   First we will consider the hypothetical (and uninteresting) case, where we have access to the true t-labels for training. In `src/posonly/posonly.py`, write a logistic regression classifier that uses x_1 and x_2 as input features, and train it using the t-labels. We will ignore the y-labels for this part. Output the trained model's predictions on the **test set** to the file specified in the code.

   Create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

2. [5 points] **Coding problem: The naive method on partial labels**

   We now consider the case where the t-labels are unavailable, so you only have access to the y-labels at training time. Extend your code in `src/posonly/posonly.py` to re-train the classifier (still using x_1 and x_2 as input features), but using the y-labels only. Output the predictions on the **test set** to the appropriate file (as described in the code comments).

   Create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ (even though we only used the $y^{(i)}$ labels for training, use the true $t^{(i)}$ labels for plotting) than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

   Note that the algorithm should learn a function $h(\cdot)$ that approximately predicts the probability $p(y^{(i)} = 1 \mid x^{(i)})$. Also note that we expect it to perform poorly on predicting the probability of interest, namely $p(t^{(i)} = 1 \mid x^{(i)})$.

In the following sub-questions we will attempt to solve the problem with only partial observations. That is, we only have access to $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, and will try to predict $p(t^{(i)} = 1|x^{(i)})$.

3. [5 points] **Warm-up with Bayes rule**

   Show that under our assumptions, for any i,

   $$p(t^{(i)} = 1 \mid y^{(i)} = 1, x^{(i)}) = 1 \tag{0.6}$$

   That is, observing a positive partial label $y^{(i)} = 1$ tells us for sure the hidden true label is 1. Use Bayes rule to derive this (an informal explanation will not earn credit).

4. [5 points] **CP8318 Only Question**

   Show that for any example, the probability that true label $t^{(i)}$ is positive is $1/\alpha$ times the probability that the partial label is positive. That is, show that

   $$p(t^{(i)} = 1 \mid x^{(i)}) = \frac{1}{\alpha} \cdot p(y^{(i)} = 1 \mid x^{(i)}) \tag{0.7}$$

   Note that the equation above suggests that if we know the value of α, then we can convert a function $h(\cdot)$ that approximately predicts the probability $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ into a function that approximately predicts $p(t^{(i)} = 1 \mid x^{(i)})$ by multiplying the factor $1/\alpha$.

5. [5 points] **CP8318 Only Question: Estimating α**

   The solution to estimate $p(t^{(i)}|x^{(i)})$ outlined in the previous sub-question requires the knowledge of α which we don't have. Now we will design a way to estimate α based on the function $h(\cdot)$ that approximately predicts $p(y^{(i)} = 1 \mid x^{(i)})$ (which we obtained in part b).

   To simplify the analysis, let's assume that we have magically obtained a function $h(x)$ that perfectly predicts the value of $p(y^{(i)} = 1 \mid x^{(i)})$, that is, $h(x^{(i)}) = p(y^{(i)} = 1 \mid x^{(i)})$.

   We make the crucial assumption that $p(t^{(i)} = 1 \mid x^{(i)}) \in \{0, 1\}$. This assumption means that the process of generating the "true" label $t^{(i)}$ is a noise-free process. This assumption is not very unreasonable to make. Note, we are NOT assuming that the observed label $y^{(i)}$ is noise-free, which would be an unreasonable assumption!

   Now we will show that:

   $$\alpha = \mathbb{E}[h(x^{(i)}) \mid y^{(i)} = 1] \tag{0.8}$$

   To show this, prove that $h(x^{(i)}) = \alpha$ when $y^{(i)} = 1$, and $h(x^{(i)}) = 0$ when $y^{(i)} = 0$.

   The above result motivates the following algorithm to estimate α by estimating the RHS of the equation above using samples: Let V_+ be the set of labeled (and hence positive) examples in the validation set V, given by $V_+ = \{x^{(i)} \in V \mid y^{(i)} = 1\}$.

   Then we use

   $$\alpha \approx \frac{1}{|V_+|} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

   to estimate α. (You will be asked to implement this algorithm in the next sub-question. For this sub-question, you only need to show equation (0.8). Moreover, this sub-question may be slightly harder than other sub-questions.)

6. [5 points] **Coding problem.**

   Using the validation set, estimate the constant α by averaging your classifier's predictions over all labeled examples in the validation set:[1]

   $$\alpha \approx \frac{1}{|V_+|} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

   Add code in `src/posonly/posonly.py` to rescale your predictions $h(y^{(i)} = 1 \mid x^{(i)})$ of the classifier that is obtained from part b, using the equation (0.7) obtained in part (d) and using the estimated value for α.

   Finally, create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ (even though we only used the $y^{(i)}$ labels for training, use the true $t^{(i)}$ labels for plotting) than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's **adjusted** predicted probability = 0.5) in red color. Include this plot in your writeup.

Remark: We saw that the true probability $p(t \mid x)$ was only a constant factor away from $p(y \mid x)$. This means, if our task is to only rank examples (*i.e.* sort them) in a particular order (e.g, sort the proteins in order of being most likely to be involved in transmitting signals across membranes), then in fact we do not even need to estimate α. The rank based on $p(y \mid x)$ will agree with the rank based on $p(t \mid x)$.

[1] There is a reason to use the validation set, instead of the training set, to estimate the α. However, for the purpose of this question, we sweep the subtlety here under the rug, and you don't need to understand the difference between the two for this question.