CS534 — Implementation Assignment 3 — Due 11:59PM Nov 14th, 2021

General instructions.

1. Please use Python 3 (preferably version 3.6+). You may use packages: Numpy, Pandas, and matplotlib, along with any from the standard library (such as 'math', 'os', or 'random' - for example).

2. You should complete this assignment alone. Please do not share code with other students, or copy program files/structure from any outside sources like Github. Your work should be your own.

3. Submit your report on Canvas and your code on TEACH following this link:
`https://teach.engr.oregonstate.edu/teach.php?type=assignment`.

4. Please follow the submission instructions for file organization (located at the end of the assignment description).

5. Please run your code before submission on the EECS servers (i.e. babylon). You can make your own virtual environment with the packages we've listed in either your user directory or on the scratch directory.

6. Be sure to answer all the questions in your report. You will be graded based on your code as well as the report. The report should be clear and concise, with figures and tables clearly labeled with necessary legend and captions. The quality of the report and is worth 10 pts. The report should be a PDF document.

7. In your report, the **results should always be accompanied by discussions** of the results. Do the results follow your expectation? Any surprises? What kind of explanation can you provide?

Perceptron and Kernels
(total points: 90 pts + 10 report pts)

Note we are using the same data as in IA2. So most of this page is a repetition from IA2.

Data. This dataset consists of health insurance customer demographics, as well as collected information related to the customers' driving situation. Your goal is to use this data to predict whether or not a customer may be interested in purchasing vehicular insurance as well (this is your "Response" variable). The dataset description (dictionary) is included. **Do not use existing code from outside sources for any portions of this assignment. This would be a violation of the academic integrity policy.**

The data is provided to you in both a training set: **IA2-train.csv**, and a validation set: **IA2-dev.csv**.

Preprocessing Information In order to train on this data, we have pre-processed it into an appropriate format. This is done for you in this assignment to ensure results are similar across submissions (easier to grade). You should be familiar with this process already from the last assignment. In particular, we have treated [**Gender, Driving_License, Region_Code, Previously_Insured, Vehicle_Age, Vehicle_Damage, Policy_Sales_Channel**] as categorical features. We have converted those with multiple categories (some that originally contained textual descriptions) into one-hot vectors. Note that we left **Age** as an ordinal numeric feature. As previously, you can normalize the numerical features using the z-score method to ensure the range of different features are consistent with one another. Additionally, the dataset is class balanced (close to the same number of +1's and -1's for Response). This was not the case in the original raw data, but we have performed down-sampling for the assignment. There are other ways to handle class imbalance, beyond the scope of this assignment, but it is a common problem in real-world data.

General comments about training. For all parts, we have specified $maxiter = 100$ as the upper limit on the number of training iterations. This may not be sufficient for convergence but the online algorithms are slower due to the fact that you can not use matrix operations to make predictions for all examples at once. If you find that your algorithm needs more than 100 iterations to converge to a good solution, feel free to use higher values if you have time. But if the run time is a concern, stick with the 100 limit. You may wonder why should one use online algorithms if the run time is so bad. There are several situations for which online algorithms are practically useful. Sometimes we receive data instances one at a time and does not allow storing. Online learning would be appropriate for such setting. In other applications, e.g., structured prediction problems in Natural language processing, the feature set can be ginormous and super sparse, it makes sense to process one example at a time.

1 Part 1 (35 pts) : Average Perceptron.

For this part you will implement and experiment with average perceptron, which is described in Algorithm 1.

Algorithm 1: Average Perceptron

Input: $\{(\mathbf{x}_i, y_i)_{i=1}^N\}$(training data), *maxiter* (maximum of iterations)
Output: online perceptron $\mathbf{w}$, average perceptron $\bar{\mathbf{w}}$

Initialize $\mathbf{w} \leftarrow 0$;
Initialize $\bar{\mathbf{w}} \leftarrow 0$;
Initialize example counter $s \leftarrow 1$;
while *iter* $<$ *maxiter* **do**
  for *each training example* $\mathbf{x}_i$ **do**
    if $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
    end
    $\bar{\mathbf{w}} \leftarrow \frac{s\bar{\mathbf{w}}+\mathbf{w}}{s+1}$;
    $s \leftarrow s + 1$;
  end
end

Algorithm 1 returns two solutions, $\mathbf{w}$ is the solution of the vanilla online perceptron and $\bar{\mathbf{w}}$ is the solution of average perceptron. Note that the running average $\bar{\mathbf{w}}$ always gets updated every time an example is processed, regardless whether the current $\mathbf{w}$ correctly classifies it or not. If you are interested, there is a slightly different but equivalent version of this algorithm that only updates $\bar{\mathbf{w}}$ when $\mathbf{w}$ is updated, presented in Chapter 3 of A course in machine learning (Algorithm 7). This can lead to slightly improved running time. You can choose to implement either version.

Use your implementation to perform the following experiments:

(a) Apply your implemented algorithm to learn from the training data with *maxiter* $= 100$. Plot the train and validation accuracy of $\mathbf{w}$ (online perceptron) and $\bar{\mathbf{w}}$ (average perceptron) as a function of the number of training iterations. This means that you will need to compute the accuracy of your $\mathbf{w}$ and $\bar{\mathbf{w}}$ on both training data and validation data at the end of each while loop.

> Question: comparing the training/validation accuracy curves of the average perceptron with those of the online perceptron, what do you observe? What are your explanation for the observation?

(b) For both average and online perceptron, use the validation accuracy to decide the best stopping point and report the iteration number and resulting validation accuracy.

> Question: which algorithm is more sensitive to the stopping point, (i.e., choosing different stopping point can lead to strong performance fluctuations) online or average perceptron? What are some practical implications of such sensitivity?

2 Part 2 (55 pts). Perceptron with Polynomial Kernel.

The online/average perceptron in Algorithm 1 are linear models. In this part we will consider kernelized perceptron, as described in Algorithm 2.

Algorithm 2: Kernelized Perceptron

Input: $\{(\mathbf{x}_i, y_i)_{i=1}^N\}$(training data), *maxiter* (maximum iterations), κ (kernel function)
Output: $\alpha_1, \ldots, \alpha_N$

Initialize $\alpha_i \leftarrow 0$ for $i = 1, ..., N$;
for $i = 1, ..., N$, $j = 1, ..., N$ **do**
  $K(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$; // Compute the Gram Matrix, please avoid for loops when implementing.
end
while *iter* $<$ *maxiter* **do**
  for *each training example* $\mathbf{x}_i$ **do**
    $u \leftarrow \sum_j \alpha_j K(i, j) y_j$; // make prediction for $\mathbf{x}_i$. Similarly, avoid for loop at all cost
    if $uy_i \leq 0$ **then**
      $\alpha_i \leftarrow \alpha_i + 1$; // Mistake on example i, increment the counter
    end
  end
end

In this assignment, we will implement the kernelized perceptron with polynomial kernel with degree p:

$$\kappa(x_1, x_2) = (x_1^T x_2)^p \tag{1}$$

Note that we have $(1 + x_1^T x_2)^p$ in the slides, but since the $\mathbf{x}$'s we provided for the data already include the dummy variable (intercept) 1, we don't need to add 1 here.

Part 2a. (40 pts) With your implementation of Algorithm 2, perform the following experiments:

(a) Apply the kernelized perceptron with different p values in [1, 2, 3, 4, 5] with $maxiter = 100$. Note that $p = 1$ will return to the vanilla online perceptron, so you should expect similar behavior compared to part 1. For each p value, at the end of each training iteration (the while loop) use the current model (aka the current set of α's) to make prediction for both the training and validation set. Record and plot the train and validation accuracy as a function of training iterations. You should create a figure for each p value and plot both the training accuracy and validation accuracy on the same figure with different colors to differentiate the two.

> Question: as p changes, do you see different trends for how training and validation accuracies vary with the number of training iteration? Do you see the risk of overtraining (i.e., when training for too many iterations leads to overfitting) for some p value? Please explain why this happens or does not happen for different p values?

(b) Report the best training and validation accuracy achieved for each p value (over all iterations).

> Question: which value of p produces the best validation accuracy? How do you think p is affecting the train and validation accuracy?

(c) Make your implementation as efficient as possible (by eliminating as many for loops as possible). For $p = 1$, plot the empirical runtime of your algorithm (with a fixed number of iterations) as a function of N, the training data size. You can do this by measuring the runtime of your algorithm on training data of different sizes ranging from 10, 100, 1000 to 10000 (you can use the subset of the validation data for the last size).

Question: what is the asymptotic runtime of your algorithm (in big O notation[a]) in terms of the number of training examples n? Does your empirical runtime match up with the asymptotic analysis?

[a]If you are not familiar with big-O notation and asymptotic runtime analysis, there are many online resources like this one https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/. This particular algorithm is a really basic one to analyze, you simply need to figure out how many nested loops there are in the code.

Part 2b. (15 pts) Algorithm 2 implements the online kernelized perceptron. Now please modify it to implement batch kernelized perceptron and perform the following experiments.

(a) Apply your batch kernel perceptron with the p value that achieved the best validation accuracy in part 2. Record and plot the training accuracy and validation accuracy in a single figure as a function of the iterations.

Question 1: Comparing the curves for batch kernel perceptron with the ones acquired with the same p value in part 2a, do you observe any differences? What are your explanations for them?

Question 2: Perceptron uses a fixed learning rate of 1 for the subgradient descent. What would be the impact if we use a different learning rate?

(b) Make your implementation as efficient as possible. Follow the same procedure as in part 2a, plot the empirical runtime of your algorithm (with fixed number of iterations) as a function of the training data size.

Question: Provide the psudocode for your batch kernel perceptron algorithm. What is the asymptotic runtime of the batch kernel perceptron as a function of the number of training examples n? How does your empirical runtime match up with the asymptotic analysis? Do you observe a significant difference in runtime compared to the online algorithm? Provide an explanation for your observation.

Submission instructions. Your submission should include the following:
1) Your source codes with a readme file to specify the required packages, zipped in a single file submitted on TEACH. We require **One file for each Part**.
2) You do not need to generate plots in the submission code, please remember to include those in your report. You need to print out your code result of each question in the console since it's easier to check the correctness of your code. Generally speaking, we check the code by console outputs and plots by report.
3) Please do **not** upload your python virtual environment. You can include the **data** in the same directory of the code (unless you think the data file is too big to upload), which can be tested with "python your_code.py" directly.
4) Your report (see general instruction items 6 and 7 on page 1 of the assignment), which should begin with a general introduction section, followed by one section for each part of the assignment; The report should be in PDF, submitted on Canvas.
5) Please always put your name on the first page of the report. All graphs must be properly labeled, i.e. Graphs must contain: Title, labels along both axis. It's a good habit to number pages, sections of the report.