Homework Assignment # 3

Due: Wednesday, March 29, 2017, 11:59 p.m.
Total marks: 100

Question 1. [60 MARKS]

In this question, you will implement several binary classifiers: naive Bayes, logistic regression and a neural network. An initial script in python has been given to you, called `script_classify.py`, and associated python files. You will be running on a physics dataset, with 8 features and 100,000 samples (called `susysubset`). The features are augmented to have a column of ones, in `dataloader.py` (not in the data file itself). Baseline algorithms, including random predictions and linear regression, are used to serve as sanity checks. We should be able to outperform random predictions for this binary classification dataset.

(a) [15 MARKS] Implement naive Bayes, assuming a Gaussian distribution on each of the features. Try including the columns of ones and not including the column of ones in the predictor. What happens? Explain why.

(b) [15 MARKS] Implement logistic regression.

(c) [20 MARKS] Implement a neural network with a single hidden layer, with the sigmoid transfer.

(d) [10 MARKS] Briefly describe the behavior of these classification algorithms you have implemented. You do not need to make claims about statistically significant behavior, but report average error and standard error. You do not need to run on the whole dataset.

Question 2. [40 MARKS]

In this question, you will implement kernel logistic regression. Kernel logistic regression can be derived using the kernel trick, where the optimal solution $\mathbf{w}$ is always a function of the training data $\mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha}$ for $\mathbf{X}^\top \in \mathbb{R}^{n \times d}$ and $\boldsymbol{\alpha} \in \mathbb{R}^n$. Therefore, we could instead learn $\boldsymbol{\alpha}$, and whenever we predict on a new value $\mathbf{x}$, the prediction is $\mathbf{x}^\top \mathbf{w} = \mathbf{x}^\top \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{i=1}^n k(\mathbf{x}, \mathbf{x}_i)\alpha_i$ with $k(\mathbf{x}, \mathbf{x}_i) = \langle \mathbf{x}, \mathbf{x}_i \rangle$ in this case. In general, we can extend to other feature representations on $\mathbf{x}$, giving $\phi(\mathbf{x})$ and so a different kernel $k(\mathbf{x}, \mathbf{x}_i) = \langle \mathbf{x}, \mathbf{x}_i \rangle$.

The kernel trick is useful conceptually, and for algorithm derivation. In practice, when implementing kernel regression, we do not need to consider the kernel trick. Rather, the procedure is simple, involving replacing your current features with the kernel features and performing standard regression or classification. For learning, we replace the training data with the new kernel representation:

$$\mathbf{K}_{\text{train}} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{c}_1) & k(\mathbf{x}_1, \mathbf{c}_2) & \dots & k(\mathbf{x}_1, \mathbf{c}_k) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_n, \mathbf{c}_1) & k(\mathbf{x}_n, \mathbf{c}_2) & \dots & k(\mathbf{x}_n, \mathbf{c}_k) \end{bmatrix} \in \mathbb{R}^{n \times k}$$

for some chosen centers (above those chosen centers were the training data samples $\mathbf{x}_i$). For example, for the linear kernel above with $k(\mathbf{x}, \mathbf{x}_i) = \langle \mathbf{x}, \mathbf{x}_i \rangle$, the center is $\mathbf{c} = \mathbf{x}_i$. Notice that the number of features is now k, the number of selected centers, as opposed to the original dimension d. Once your've transformed your data to this new representation, then you learn $\mathbf{w}$ with logistic regression as usual, such that $\mathbf{K}_{\text{train}}\mathbf{w}$ approximates $\mathbf{y}_{\text{train}}$. As before, you can consider adding

regularization. The prediction is similarly simple, where each new point is transformed into a kernel representation using the selected centers.

(a) [25 MARKS] Implement kernel logistic regression with a linear kernel and run it on `susysubset`. Compare the performance in one sentence to the performance of the algorithms from the first question.

(b) [15 MARKS] Using the same implementation, change the linear kernel to a Hamming distance kernel and run the algorithm on the dataset `Census`. In one or two sentences, summarize your performance, compared to the random predictor.

Homework policies:

Your assignment will be submitted as a single pdf document and a zip file with code, on canvas. The questions must be typed; for example, in Latex, Microsoft Word, Lyx, etc. or must be written legibly and scanned. Images may be scanned and inserted into the document if it is too complicated to draw them properly. All code (if applicable) should be turned in when you submit your assignment. Use Matlab, Python, R, Java or C.

Policy for late submission assignments: Unless there are legitimate circumstances, late assignments will be accepted up to 5 days after the due date and graded using the following rule:

on time: your score 1

1 day late: your score 0.9

2 days late: your score 0.7

3 days late: your score 0.5

4 days late: your score 0.3

5 days late: your score 0.1

For example, this means that if you submit 3 days late and get 80 points for your answers, your total number of points will be $80 \times 0.5 = 40$ points.

All assignments are individual, except when collaboration is explicitly allowed. All the sources used for problem solution must be acknowledged, e.g. web sites, books, research papers, personal communication with people, etc. Academic honesty is taken seriously; for detailed information see Indiana University Code of Student Rights, Responsibilities, and Conduct.

Good luck!