CS452, Spring 2018, Problem Set # 3
Ashesi University

March 13, 2018

Due: March 29th 2018, 11:59pm

This problem set requires you to provide written answers to a few technical questions. In addition, you will need to implement some learning algorithms in Python and apply them to the enclosed data sets. The questions requiring a written answer are denoted with **W**, while the Python programming questions are marked with **P**.

Both the written answers as well as the code must be submitted electronically via Courseware.

For the written answers, you must provide a single PDF containing all your answers. Upload the PDF files trough the appropriate link in Courseware. The PDF file can be generated using editing software (e.g., Latex) or can be a scanned copy of your hand-written answers. If you opt for scanned copies, please make sure to use a scanner, *not* your phone camera. Familiarize yourself with the process of scanning documents a few days before the due date. Late submissions due to scanning problems will be treated and penalized as any other form of late submission, as per the rules stated on the Syllabus. Also, make sure your scanned submission is readable. If we cannot read it clearly, you will receive 0 points for that problem.

Your Python code must also be submitted via Courseware. We have provided Python function files with an empty body for all the functions that you need to implement. Each file specifies in detail the input and the output arguments of the function. A few of the functions include some commands to help you get started with the implementation. The function stubs include also *import* commands for all the libraries that are needed for your implementation (e.g., numpy, math). Do not add any other import command, i.e., do not use any other libraries except those already specified in the function stubs. If in doubt, please ask. We have also provided scripts that check the sizes of the inputs and outputs of your functions. Do not modify these scripts. In order to allow us to test your code, it is imperative that you do not rename the files or modify the syntax of the functions. Make sure to include all the files necessary to run your code. **If we are unable to run one of your functions, you will receive 0 points for that question.** Please place all your files (data, .png files automatically generated by the provided scripts, and function files) in a single folder (**without subfolders**). Zip up the entire folder, and upload the compressed file to Courseware. Note that we provided **two** separate links for your homework submission: one for your written answers, one for your code.

1. **[P, 15 points]** In this problem you will have a chance to develop your own spam filter using a Naïve Bayes classifier. The data set that you will be using ("spamdata.mat") was derived from spam and non-spam emails collected a few years ago at Hewlett Packard Research Labs to test different spam detection algorithms. Each email is represented as a vector of 48 binary features indicating whether or not particular words occur in the email (the complete list of words can be found in the file "spambase_names.txt"). The label y takes on two values, one corresponding to "ham" (i.e., valid email) and the other to "spam". It should be easy for you to figure out whether "spam" is denoted by 0 or 1 (hint: look at the frequency of occurrence of certain cue words in examples of class 0 and class 1). There are 1500 training examples and 3101 test examples.

   (a) **[P, 12 points]** Implement the Maximum Likelihood Naïve Bayes classifier for *binary* features with Laplacian smoothing illustrated in class and train it on the training set. Report the misclassification rate (i.e., the fraction of examples misclassified) obtained on the training set as well as on the test set. In order to do this, you need to implement the following functions:

   - [**P**, 8 points] `q1_nb_train.py` trains Naïve Bayes given a training set. The function should learn the class prior and the class conditional probabilities for every feature.
   - [**P**, 4 points] `q1_nb_predict.py` predicts the class labels for a given set of examples using a trained model. **Note:** Consider these tips to implement this function: first, because $p(x)$ is a constant for a given x, instead of computing exactly $p(y = 0|x)$ and $p(y = 1|x)$, you just need to find $\arg\max_y p(x|y)p(y)$; second, to avoid the numerical issues arising from the product of many small numbers in $p(x|y)$, use the logarithm function, which being a monotonically increasing function, will not change the result of the arg max operator.

   After you are done coding the two functions above, run the script `q1a.py` to compute the training and the test errors of your Naïve Bayes classifier.

   (b) **[P, 3 points]** Using the learned Naïve Bayes model, compute $p(y = 0|x_j = 1)$ for all features $j = 1, \ldots, 48$. Based on these probabilities, list the 6 words that are most indicative of a message being "spam", and the 6 words most indicative of a message being "ham" (the words associated to the binary features can be found in file "spambase_names.txt"). To accomplish this task, you must simply implement the function `q1_top_words.py`. Then run the script `q1b.py` to output the top 6 most indicative words for both classes.

2. **[25 points]** In class we have seen that learning a Support Vector Machine (SVM) requires solving a constrained quadratic optimization problem. In this problem, you will implement linear SVMs first for the linearly separable case, and later for the non-separable case. To computer the optimal parameters you will be using the Python `CVXOPT` solver for constrained quadratic optimization [1]The only issue is that it is not as efficient as the SMO method discussed in class so we can only use it on small training sets.

[1]Please note that `CVXOPT` does not come standard in most installations of Python. Make sure to download `CVXOPT` and install it before you get started on this problem.

(a) **[W, 4 points]** Specifically, the `CVXOPT` solver can solve arbitrary constrained quadratic optimization problems of the type

$$\begin{aligned} & \arg\min_{x} \frac{1}{2} x^\top P x + q^\top x \\ \text{s.t.} \quad & Gx \leq h. \end{aligned} \tag{1}$$

To get familiar with `CVXOPT`, start by reading the following gentle introduction to this package: `https://courses.csail.mit.edu/6.867/wiki/images/a/a7/Qp-cvxopt.pdf`

Now consider the SVM primal formulation in the linearly separable case:

$$\begin{aligned} & \arg\min_{w,b} \frac{1}{2} ||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \ \ \forall i = 1, \ldots, m. \end{aligned} \tag{2}$$

Your first task is to figure out how to convert this formulation into the format required by `CVXOPT`. To do this, you must describe in full detail how to set the arguments P, q, G, h of solver `solvers.qp` in `CVXOPT` in terms of the training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$. Also, what does the vector x (the output of `solvers.qp`) contain in this case?

(b) **[P, 0 points]** No need to code anything here, just run the provided script `q2b.py`. The script loads the file `iris_subset.mat` which contains a subset of the Iris dataset from the UC Irvine Machine Learning Repository. Each example represents an instance of Iris flower described by 2 features (corresponding to measurements *sepal length* and *sepal width*). In this subset there are two Iris classes (denoted with labels $1, -1$). The script generates a scatter plot of the training examples using two different markers for the two classes. Note how the training set is linearly separable, as a straight line can successfully separate the two classes.

(c) **[P, 4 points]** Fill the body of the function `q2_train_svm_primal_separable.py`. You will use `CVXOPT` to train the linear SVM in the formulation given by eq. 2 on the 70 training examples. The function must also determine which examples are the support vectors of the optimal solution by inspecting the Lagrange multipliers returned by `CVXOPT`. After you have coded this function, run the given script `q2c.py`, which will report the values of w and b that you obtained, print the number of SVs identified by your function, and provide a nice visualization of the learned SVM model. In particular the visualization will plot for you the decision boundary $\{x \ \ s.t. \ \ w^\top x + b = 0\}$, the margin lines (see illustration in the lecture slides) on top of the scatter plot of the data, and also mark the support vectors found by your function. Use solver `solvers.qp` from the package `CVXOPT` to compute the optimal solution of the quadratic program. Note that you will first need to use the function `matrix` to convert numpy arrays into the format required by `CVXOPT` for the arguments of the solver. You can then convert the solution from the `matrix` format to numpy arrays using the command `x = np.array(sol['x'])` where `sol` is the solution returned by `solvers.qp`. In order to determine the support vectors you will need to inspect the Lagrange multipliers returned by the solver in the variable `sol['z']`. Use a small positive threshold (e.g., `1e-5`) to find the support vectors.

Note that with SVMs there is no need to add the constant 1 as extra feature for the examples, as we have an explicit bias term b. So, leave the feature vectors as they are without adding the feature 1.

(d) **[W, 3 points]** Now consider the following quadratic optimization problem, which contains slack variables:

$$\begin{aligned} & \arg\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \ \ \forall i \\ & \xi_i \geq 0, \ \ \forall i. \end{aligned} \tag{3}$$

Again, describe in detail how to set the arguments P, q, G, h of solver `solvers.qp` in `CVXOPT` so as to implement this formulation.

(e) **[P, 3 points]** Fill the body of the function `q2_train_svm_primal.py` which implements the optimization problem with slack variables that you just derived. Code also the function `q2_predict_svm_primal.py` which, given a set of unlabeled examples and an SVM model as input, predicts the labels and outputs the SVM classification scores for the examples. After you have done so, run `q2e.py`. The script will train many SVM models on the same data as before, for different values of C. For each model, it also prints the training/test error, and produces a plot like before. The data is linearly separable, but you should see the effect of the slack variables for certain values of C when you inspect the automatically-generated plots.

(f) **[W, 1 point]** For which values of C do the margin lines look like as before? For which values of C do we have some violations of the margin constraint?

(g) **[W, 3 points]** We will now move to the dual formulation of the SVM, which enables the use of kernels and thus nonlinear classification. You will use again the package `CVXOPT`, but now you need to consider a slightly more general case. The solver also works on arbitrary constrained quadratic optimization problems involving *equality* constraints:

$$\begin{aligned} & \arg\min_{x} \frac{1}{2}x^\top P x + q^\top x \\ \text{s.t.} \quad & Gx \leq h \\ & Ax = b. \end{aligned} \tag{4}$$

Using this variant of the solver we want to implement the SVM dual formulation for the non-separable case:

$$\arg\max_{\alpha} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}k(x^{(i)}, x^{(j)}) \tag{5}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \\ & 0 \leq \alpha_i \leq C, \ \ \forall i. \end{aligned} \tag{6}$$

Here $k(x^{(i)}, x^{(j)})$ denotes some kernel function applied to example i and j. Again, describe how you need to set the arguments of `solvers.qp` to implement this optimization.

Note that lower and upper bounds on the variables can be implemented using inequality constraints of `solvers.qp` through matrix G and vector h.

(h) **[P, 6 points]** Implement the following Matlab functions:

- [1 point] `q2_kernel.py` which calculates the kernel distance between two examples. You will need to implement both the linear kernel $k(x^{(1)}, x^{(2)}) = {x^{(1)}}^\top x^{(2)}$, as well as a polynomial kernel of degree 3, i.e. $k(x^{(1)}, x^{(2)}) = ({x^{(1)}}^\top x^{(2)})^3$.
- [3 points] `q2_train_svm_dual.py` which trains a SVM model using the dual formulation that you derived in the previous point. The function returns the indices of the training examples that are support vectors, and the associated alpha coefficients. Note that although an example i that is *not* a support vector should have value α_i equal to zero, in practice due to numerical errors its coefficients α_i may be slightly different from 0 (albeit close to it). Use the positive threshold $\tau = 1e-5$ to determine if an input vector is a support vector, i.e, check if $\alpha_i \geq \tau$.
- [1 points] `q2_calculate_bias_svm_dual.py` which computes the bias term of the model from the dual parameters. For computational efficiency, you should only sum over the support vectors. The offset (bias) b in this case is given as

$$b = \frac{1}{N_{SV}} \sum_{i \,\text{s.t.}\, \alpha_i > \tau} \left(y^{(i)} - \sum_{j \,\text{s.t.}\, \alpha_j > \tau} \alpha_j y^{(j)} k(x^{(i)}, x^{(j)}) \right).$$

  where N_{SV} is the number of support vectors.
- [1 point] `q2_predict_svm_dual.py` which performs the prediction for a given test example. The function takes as input the bias term, since it is independent from the test examples. Again, to compute the SVM score $s(x)$, you only need to sum over the support vectors:

$$s(x) = \sum_{i \,\text{s.t.}\, \alpha_i > \tau} \alpha_i y^{(i)} k(x^{(i)}, x) + b.$$

You should now be able to run the given script `q2h.py`, which trains an SVM with the linear kernel and another one with the polynomial kernel, using $C = 100$. The script will tell you now many support vectors were found by your function, and it will plot the decision boundary as well as the margin lines on top of the scatter plot of the data.

(i) **[W, 1 point]** How many support vectors did you get for the linear case? How many for the polynomial (nonlinear) case? Does the decision function for the polynomial-kernel case look truly nonlinear?

3. **[P, 7 BONUS points]** This problem requires you to implement the k-Nearest Neighbors classifier (kNN) based on Euclidean distance and evaluate it on the Parkinsons dataset *parkinsons.mat* already encountered in HW2.

(a) **[5 points]** Implement the following functions:

- [4 points] `q3_predict.py` predicts the class label of an input test example using kNN.
- [1 point] `q3_test_error.py` computes the misclassification rates of kNN on a test set for different choices of hyperparameters k. [Hint: reuse the function `q2_error.py` that you wrote for the previous problem].

You now should be able to run the script `q3a.py` to plot the test errors for different parameters $k \in \{1, 3, 5, 7, 9, 11, 13\}$.

(b) **[2 points]** Implement `q3_leave_one_out_error.py` to evaluate the validation errors of *leave-one-out* for different parameters k. After you complete this function, run the script `q3b.py` to plot the leave-one-out errors for different parameters $k \in \{1, 3, 5, 7, 9, 11, 13\}$.