Exercise 4

AMTH/CPSC 445a/545a - Fall Semester 2016

October 30, 2017

Compress your solutions into a single zip file titled `<lastname and initials>_assignment4.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment4.zip`. Include a single PDF titled `<lastname>_assignment4.pdf` and any MATLAB or Python scripts specified.

Please include your name in the header of all submitted files (on every page of the PDF) and in a comment header in each of your scripts.

Your homework should be submitted to Canvas before Monday, November 20, 2017 at 5:00 PM.

Programming assignments should use built-in functions in MATLAB or Python; In general, Python implementations may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of data mining algorithms - if a function exists that solves an entire problem (either in the MATLAB standard library or in the `scipy` stack), please consult with the TA before using it.

Problem 1

1. Formulate a nonlinear soft-margin SVM with feature map $\Phi(x)$, and derive a soft-margin kernel SVM quadratic program using the kernel trick with $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$.

2. One could argue that the kernel trick is unnecessary since, given a $N \times N$ Mercer (i.e., symmetric and positive semi-definite) kernel matrix $K_{ij} = k(x_i, x_j)$, $i, j = 1, \ldots, N$, one can easily find feature vectors $\vec{u}_i = \Phi(x_i)$, $i = 1, \ldots, N$, such that $K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle$. Then, the SVM can be trained with these vectors to find appropriate $\vec{w}$ and b without directly using the kernel.

   - Explain how to find such feature vectors $\vec{u}_i$.
   - Is this claim correct and the kernel trick is unnecessary for nonlinear SVM classification? Explain your answer. (*Hint:* remember that a classifier is trained on collected labeled data and applied to new unlabeled data)

Problem 2

1. Compute and compare the time complexity of k-means and PAM-based k-medoids for clustering N data points in $\mathbb{R}^n$ into k clusters based on Euclidean distances. Your complexity can be phrased in terms of N, n, k, and the number of iterations until convergence. Justify your calculations.

2. Prove the "shake & bake" dissimilarity is a distance metric.

3. Prove the attribute-wise mode is indeed the centroid w.r.t the mismatch dissimilarity in terms of minimizing SAE in each cluster.

Problem 3

In this problem you will write a basic kernel SVM function. Specific instructions for MATLAB and Python are included following the problem description.

1. Implement a training kernel SVM function with the following calling sequence.

   ```
   Md = svm_train(c,x,kh)
   ```

   where

   - c is an $n \times 1$ vector of class labels $c[i] \in \{-1, 1\}$.
   - x is an $n \times m$ real-valued attribute matrix
   - kh is an anonymous function which takes two arguments x_1, x_2 which are both $1 \times m$ real-valued vectors and returns the kernel function value
   - Md is a data structure (designed as you see fit) which encodes the trained model

2. Implement a classification kernel SVM function with the following calling sequence.

   ```
   [chat,d] = svm_classify(Md,y)
   ```

   where

   - Md is the data structure output by your `svm_train` function.
   - y is an $\ell \times m$ real-valued attribute matrix
   - $chat$ is an $\ell \times 1$ vector of predicted class labels $c \in \{-1, 1\}$
   - d is an $\ell \times 1$ vector of the SVM confidence level

3. Test your kernel SVM code using the standard (linear SVM) inner product kernel

   - Train your SVM algorithm on the `simple_iris.mat` data set using the standard inner product kernel
     $$K(x, y) = \langle x, y \rangle$$
   - Produce the following figure and add it to the submitted PDF:
     - Plot each of the class -1 data points as a red dot
     - Plot each of the class 1 data points as a blue dot
     - Plot each of the support vector data points as a large black dot
     - Plot the three linear-SVM hyperplanes
       $$\{x : w^T x - b = -1\}, \quad \{x : w^T x - b = 0\}, \quad \{x : w^T x - b = 1\}$$

       Notice: the SVM topic slides have details on how to compute w

4. Test your implementation using the quadratic kernel.

   - Train your kernel SVM algorithm on the `simple_nonlinear.mat` data set using the quadratic kernel
     $$K(x, y) = (\langle x, y \rangle + 1)^2$$

- Create a 20×20 grid of points y which covers your training data x
- Classify the grid of points y using your SVM classification function.
- Create the following two figures and add them to the submitted PDF:
  - In the first figure, plot the grid points y where class -1 points are colored red, class 1 points are colored blue, and support vector points are colored black
  - In the second one, plot the grid of points y where points are colored by the vector d output by your SVM classify function.

Problem 3 Matlab specific instructions

- Complete Problem 3 only using the core MATLAB language, basic built in functions at your discretion (e.g., `zeros, size, plot, scatter,` etc.), and the following special purpose command for Quadratic Programming (QP):
  - `lambda = quadprog(H,f,A,b,Aeq,beq,LB,UB)`
- Implement kernel SVM by stating an appropriate Quadratic Program (QP) and using `quadprog` to find a solution.
- Base your code of the template `script3.m` which loads two example data sets and specifies function calling sequences.

Problem 3 Python specific instructions

- Install the following packages:
  - lapack, e.g., using yum by `yum install lapack-devel`
  - cvxopt, e.g., using pip by `pip install cvxopt`
  - See netlib.org/lapack and cvxopt.org for additional information.
- Complete Problem 3 only using the scipy stack and the following import statements
  - `from scipy.io import loadmat`
  - `from scipy.optimize import minimize`
  - `from cvxopt import matrix, solvers`
- Implement kernel SVM by stating an appropriate Quadratic Program (QP) and using `solvers.qp()` to find a solution.
- Base your code of the template `script3.py` which loads two data sets and specifies function calling sequences.

Problem 4

In this problem you will use cross validatation to tune LIBSVM models for two different data set. Specific MATLAB and Python instructions are listed following the problem description.

1. Linear SVM with Slack
   - Load the data set `mixed.mat`.
   - Create a scatter plot of the data x colored by the class labels c, and add it to the submitted PDF.

- Use cross validation to determine the best slack parameter nu for a Linear Classifier on the data. Specifically,
  - Test `svmtrain` with arguments
    $$\texttt{-s 1 -t 0 -n } \nu$$
    where $\nu \in \{0.1 + j/100 : j = 0, \ldots, 49\}$.
  - For each parameter value ν, perform 4-fold cross validation and compute the classification accuracy acc.
  - Create a blue line plot of nu vs acc with a red star at the maximum value of acc, and add it to the submitted PDF.
- Visualize the descision region
  - Using the optimal slack paramter `nu` from cross validation train a Linear SVM model on the entire `mixed.mat` data set.
  - Create a 30×30 grid of data points y, which cover a bounding box of the original data x
  - Classify the points y using your model
  - Create a scatter plot of the points y colored by their class, and add it to the submitted PDF.

2. RBF Kernel SVM
   - Load the data set `target.mat`.
   - Create a scatter plot of the data x colored by the class labels c, and add it to the submitted PDF.
   - Use cross validation to determine the best value of the bandwidth parameter γ in the RBF kernel. Specifically,
     - Run `svmtrain` with arguments
       $$\texttt{-s 1 -t 2 -n 0.5 -g } \gamma$$
       where $\gamma \in \{2^j : j = -15, -14, \ldots, 14, 15\}$.
     - Preform 4-fold a cross validation to compute the classification accuracy acc for each value of γ
     - Plot a blue line plot of acc vs γ with a red star the maximum value, and add it to the submitted PDF.
   - Using the optimal value of γ from cross validation train a RBF Kernel SVM model on the entire `mixed.mat` data set.
   - Create a 30×30 grid of data points y, which cover a bounding box of the original data x
   - Classify the points y using your model
   - Create a scatter plot of the points y colored by their class, and add it to the submitted PDF.

Problem 4 Matlab specific instructions

- Install LIBSVM
  - Download the zip file http://www.csie.ntu.edu.tw/~cjlin/libsvm/#download into your working directory
  - Unzip the file and see the README for system specific instructions, e.g., on linux run `make` in `/libsvm-3.21/matlab/`, and add `addpath('libsvm-3.21/matlab/',0)` to your code.
- Complete Problem 3 only using the core MATLAB language, basic built in functions at your discretion, and the following two functions from LIBSVM:
  - `Md = svmtrain(c1,x1,args)`
  - `chat = svmpredict(c0, x0, Md)`
- Base your code on the template `script4.m`.

Problem 4 Python specific instructions

- Install LIBSVM
- Download the zip file http://www.csie.ntu.edu.tw/~cjlin/libsvm/#download into your working directory
- Unzip the file and see the README for system specific instructions, e.g., on linux run `make` in `libsvm-3.12/python/`, and add `path.append('./libsvm-3.21/python/')` to your code.
- Complete Problem 3 only using the core Python langage, the scipy stack, and the following import statements:
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
  - `from scipy.io import loadmat`
  - `from sys import path`
  - `path.append('./libsvm-3.21/python/')`
  - `from svmutil import *`
  - Note: you may modify the last three lines as needed for your system.
- In order to test your LIBSVM installation, you can train a SVM model on c, x, and then test the prediction on c, x using the following commands:
  - `prob = svm_problem(c,x)`
  - `args = svm_parameter(str)` where `str` is the argument string
  - `Md = svm_train(prob,args)`
  - `p_label, p_acc, p_val = svm_predict(c, x, Md)`
  - Note: see the README file in `libsvm-3.21/python` for additional information.
- Base your code on the template `script4.py`.

Problem 5

For this problem, you will implement a basic k-means clustering algorithm. Specific MATLAB and Python insturctions are given following the problem description.

- Implement k-means in a function `[C,M,C0,M0] = KmeansClustering(X,k)`
  - The function gets as an input a $N \times n$ data matrix X and a number $1 < k < N$ of clusters
  - It returns a $N \times 1$ vector C (with $1 \leq C[i] \leq k$, $i = 1, \ldots, N$) that contains the cluster number for each data point, a $k \times n$ matrix M with the cluster centroids as its rows
  - Additinally, it should return the initialization (clusters & centroids) of the algorithm as $C0$ and $M0$.
  - Your implementation should initialize the centroids by choosing k random data points, and should iterate until no data point changes its cluster assignment.
- Illustrate clustering on a two-dimensional dataset:
  - Load the Iris dataset `iris_num.mat` and use the first two coordinates in the data matrix as an input to k-means with $k = 3$.

  - Create a 2D scatter plot of the data colored by initial cluster numbers in $C0$, where data points are marked as circles.
  - Add to this plot the initial centroids from $M0$, where these three points are marked using + markers.
  - Create a 2D scatter plot of the data colored by the final cluster number C, where data points are marked as circles.
  - Add to this plot the final centroids from $M0$, where these three points are marked using + markers.
  - Include the two produced plots to the PDF

- Test clustering quality on the full iris data:
  - Cluster the full four dimensional Iris data into three clusters using k-means and assess the accuracy of your clustering using the class labels. *Notice* that you first need to find the optimal mapping between cluster indices and class labels, since they might be permuted.
  - Repeat this process three times in your script and report the three accuracy numbers in the submitted PDF.
  - Repeat the above process three more times, but this time use RandIndex to assess accuracy w.r.t. class labels. Report the three RandIndex numbers in the PDF. *Notice* that this time you don't need to deal with permuted class/cluster indices.

- Compare accuracy, RandIndex, and SSE as cluster quality measures:
  - Run the classification 50 times, and for each iteration record the RandIndex, clustering accuracy, and the total SSE over the three clusters.
  - Plot a histogram with four bins showing the distribution of accuracies over the 50 runs
  - Plot a histogram with four bins showing the distribution of total SSE over the 50 runs
  - Produce a scatter plot of accuracy vs. SSE (i.e., with 50 points - one for each run)
  - Include the three produced plots to the PDF

- What conclusions can you draw from these tests regarding the total SSE and its usage for assessing clustering quality? Include your answer in the submitted PDF.

Problem 5 MATLAB specific instructions

1. Complete the problem only using the core MATLAB language and basic MATLAB functions (e.g. `zeros, size, plot, figure`) at your discretion. (Please do not just call a built k-means clustering routine).
2. Base your code on `script5.m`

Problem 5 Python specific instructions

1. Complete the problem only using the core Python language and the scipy stack
2. Base your code on `script5.py`

Problem 6

- Implement DBScan in a function `[C,point_type]=dbscan(X,min_pts,epsilon,dist)`
  - The function gets as an input a $N \times n$ data matrix `X`, the parameters `min_pts` & `epsilon` of the algorithm, and a callable distance function `dist`.
  - It returns a $N \times 1$ vector C with cluster number for each data point, and the point types (encoded as -1 for noise point, 0 for border point, and 1 for core point) in another $N \times 1$ vector `point_type`.
  - The `D = dist(y,Y)` function comuptes the distance between a row `y` and the rows of the matrix `Y` and returns them as a vector `D`.
  - In this exercise you will only need to use `pdist2` in Matlab (`cdist` Python) as the distance function.
- Test DBScan on unlabeled data:
  - Load the data from `clusters1.mat`
  - Scramble the data points by using a random permutation of the rows in the loaded data matrix.
  - Set `epsilon` to be the first percentile of distances in the data.
  - Set `min_pts` to be the 10th percentile of ε-neighborhood sizes (i.e., by considering the number of points in $N_\varepsilon(x)$ for all x in the data).
  - Use DBScan to cluster the data.
  - Produce a scatter plot of the data colored by cluster number.
  - Produce a scatter plot of the data colored by point_type.
  - Repeat the above process, but this time use the 0.3% percentile for `epsilon`.
  - Add the four produced plots to the submitted PDF.
- Test DBScan on labeled data:
  - Load the data from `clusters2.mat`
  - Test three values of `epsilon`, set by the 3rd, 5th, and 7th percentiles of distances in the data.
  - For each of these values, test three values of `min_pts`, set by the 10th, 20th, and 30th percentiles of ε-neighborhood sizes.
  - For each configuration of `epsilon` and `min_pts`, test three random permutations of the data matrix rows.
  - In each of these 27 iterations, use DBScan to cluster the data and assess the quality of the clustering using RandIndex with the provided labels.
  - Add to the submitted PDF the best and worst RandIndex, together with the configurations that achieved them.
  - Produce a scatter plot of the data colored by cluster number provided by the best RandIndex configuration.
  - Produce a scatter plot of the data colored by cluster number provided by the worst RandIndex configuration.
  - Add the two produced plots to the submitted PDF.

Problem 6 MATLAB specific instructions

1. Complete the problem only using the core MATLAB language and basic MATLAB functions (e.g. `zeros, size, plot, figure`) at your discretion.
2. Base your code on `script6.m`

Problem 6 Python specific instructions

1. Complete the problem only using the core Python language and the scipy stack
2. Base your code on `script6.py`

References

[1] "The scipy stack specification." [Online]. Available: https://www.scipy.org/stackspec.html