PROBLEM SET II

MAT 6480 / STT 6705 - Fall Semester 2019

Your solutions must be in a single zip file titled `ps2.zip`. The zip file should include a single PDF titled `ps2.pdf` and MATLAB or Python scripts as specified. Your homework should be submitted via StudiUM before Friday, Nov. 8, 2019 at 23:59.

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 $\vec{u}_i = \Phi(x_i)$, $i = 1, \ldots, N$, in a Euclidean space, such that $K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = \vec{u}_i^T \vec{u}_j$ (when treated as column vectors). 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

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 that 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 2 Matlab specific instructions

- Complete Problem 2 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 `script2.m` which loads two example data sets and specifies function calling sequences.

Problem 2 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 2 only using the core Python language and the following import statements
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
  - `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 `script2.py` which loads two data sets and specifies function calling sequences.

Problem 3

Using the template `script3.py` or `script3.m` complete the following problem. Generate a toy example to test Mahalanobis distances & PCA and visualize them using scatter and quiver plots, based on the following steps:

Part A: data generation

- Sample 1000 points uniformly from the interval $[0, 10]$ on the horizontal axis in $\mathbb{R}^2$(i.e., x-coordinate in $[0, 10]$ and y-coordinate being 0);
- Choose an angle θ and rotate your points about the origin using a rotation matrix R_θ formed using this angle;
- Add 2-dimensional Gaussian noise to generate the toy example data.

Part B: Mahalanobis and principal components

- Find mean μ and covariance Σ of the data;
- Compute the Mahalanobis distance of each data point x from the mean μ of the data (i.e., $D(x) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)}$);
- Compute the two principal components ϕ_1 & ϕ_2 (i.e., covariance eigenvectors) and associated eigenvalues λ_1 & λ_2.

Part C: Visualization

- Produce a 2D scatter plot of the generated data, where each data point is colored by $D(x)$ from part B (i.e., its Mahalanobis distance from the data mean). For this plot use filled markers (using the flag 'filled') of size of 9 (using the parameter 'S');
- Use the quiver command to add to this plot the two vectors $\sqrt{\lambda_1}\phi_1$ & $\sqrt{\lambda_2}\phi_2$ whose directions is determined by the principal components and their length is determined by the corresponding eigenvalues. For this plot set the color of the vectors to be black (using the flag 'k') and the line width to 4 (using the 'LineWidth' option).
- Save the produced plot as a PNG file with the name `mahal_pca_illustration.png`.

Include the produced plot in `ps2.pdf`.

Notice: Use the template `script3.m` or `script3.py` depending on if you are using MATLAB or Python.

If you're using MATLAB, complete the exercise only using core MATLAB language (arithmetic operations, loops, arrays, etc.) and the following functions as needed: `rng, cos, sin, rand, randn, mean, ones, repmat, inv, sqrt, svd, figure, scatter, title, colormap, hold on, axis equal, quiver`

If you're using Python, complete the exercise only using the core Python language and the following imported functions included in the template.

```
from numpy.random import seed as seed
from numpy.random import rand as rand
from numpy.random import randn as randn
from numpy import pi as pi
from numpy import cos as cos
from numpy import sin as sin
from numpy import zeros as zeros
from numpy import ones as ones
from numpy import mean as mean
from numpy import mat as mat
from numpy import array as array
from numpy import transpose as transpose
from numpy import sqrt as sqrt
from numpy import diag as diag
from numpy import dot as dot
from numpy.linalg import inv as inv
from numpy.linalg import svd as svd
from numpy.matlib import repmat as repmat
from matplotlib.pyplot import figure as figure
from matplotlib.pyplot import title as title
from matplotlib.pyplot import plot as plot
from matplotlib.pyplot import scatter as scatter
from matplotlib.pyplot import axis as axis
from matplotlib.pyplot import quiver as quiver
from matplotlib.pyplot import savefig as savefig
from matplotlib.pyplot import show as show
```

Problem 4

For this problem you will use an implementation of CART to build a decision tree model for a leaf data set. You will evaluate your model using several different methods of cross validation. In particular, you should

evaluate you model using leave-one-out cross validation, 2-fold cross validation, and 17-fold cross validation. Each cross validation method will make a total of 340 predictions (one for each sample). For example, for leave-one-out cross validation you should make 340 splits (corresponding to the number of samples) such that each sample in excluded once from training, and its predicted value is determined by this split. For each cross validation method, summarize your predictions in a 30×30 confusion matrix (where 30 is the number of classes).

Plot each of the resulting confusion matrices (using `imagesc` in MATLAB or `imshow` in Python) and include the plots in your assignment PDF. Additionally, the classification accuracy for each method should be computed and included in your PDF. Finally, construct a decision tree on the entire dataset and visualize the resutling tree. The method of visualization is detailed below. Save the resulting image as a PNG, and include the PNG file in your assignment ZIP file (because the resulting tree may be very large, there is no need to include this image in your PDF).

Your code should be contained in a single file based on the template `script4.m` or `script4.py` depending on if you're using MATLAB or Python. The template script will load preprocessed data `leaf.mat`. The data consists of an m-dimensional vector c of classes represented as positive integer (1-30), and an $m \times n$ floating point attribute matrix. A key for the classes and a description of the attributes is provided in `leaf_key.txt`.

If you're using MATLAB, complete the exercise only using the core MATLAB language and the following functions as needed: `load, rng, size, zeros, randsample, randperm, figure, imagesc, title, xlabel, ylabel, print, trace, fitctree, predict, view.` The function `fitctree` fits a decision tree based on the CART algorithm. The model the trained by

$$M = \text{fitctree}(x, c)$$

and classification is performed by

$$c\_hat = \text{predict}(M, y).$$

The decision tree M can be visualized in MATLAB by

$$\text{view}(M, 'mode', 'graph')$$

If you're using Python, you will need to install several packages. First, install `scikit-learn` for your Python distribution. For example, the package can be installed using pip by `pip install -U scikit-learn`. for additional information. Second, install `pydotplus` for your Python distribution. For example, the package can be installed using pip by `pip install -U pydotplus`. Third, install the GraphViz binaries. For example, the binaries can be installed using yum by `yum install graphviz`. You will need to determine the appropriate way to install these packages on your system. For additional information, see scikit-learn.org/stable/install.html. After installing the appropriate packages, complete the exercise only using the core Python programming language the the following functions as needed:

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from scipy.io import loadmat as load
from numpy import argsort, reshape, transpose, array, zeros
from matplotlib.pyplot import imshow, xlabel, ylabel, title, figure, savefig
from numpy.random import permutation, seed
from pydotplus import graph_from_dot_data
```

The function `DecisionTreeClassifier` can be used to train a CART decision tree by

$$M = \text{DecisionTreeClassifier}()$$

$$M = M.\text{fit}(x, c)$$

and then classification can be performed by

$$c\_hat = M.\text{predict}(y).$$

The tree can be visualized by

$$\text{dot\_data} = \text{export\_graphviz}(Md, \text{out\_file} = \text{None})$$

$$\text{graph} = \text{graph\_from\_dot\_data}(\text{dot\_data})$$

$$\text{graph.write\_pdf}('\text{cart\_tree.png}', \text{f='png'})$$

For additional information see scikit-learn.org/stable/modules/tree.html.

Problem 5

In this problem you will use cross validation 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 ν 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 ν 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 ν from cross validation train a Linear SVM model on the entire `mixed.mat` data set.
     - Create a 30×30 grid of data points y that 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 that 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 5 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-3.24/matlab/`, and add `addpath('libsvm-3.24/matlab/',0)` to your code.
- Complete Problem 5 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 `script5.m`.

Problem 5 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-3.24/python/`, and add `path.append('./libsvm-3.24/python/')` to your code.
- Complete Problem 5 only using the core Python langage 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-3.24/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-3.24/python` for additional information.
- Base your code on the template `script5.py`.