Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: August 27, 2017
Total: 40 marks

ASSIGNMENT 1

Instructions

1. Only electronic submissions to this assignment will be accepted. No handwritten, scanned or photographed submissions would be accepted.
2. Submissions are to be made separately in two parts - theory part and code part. Instructions for each of these parts are given below.
3. Submissions will be accepted till September 10, 2017, 2359 hrs, IST.
4. Your submission will be considered complete only when both parts are submitted. If both parts are not submitted by the above deadline, your submission will be considered late.
5. Late submissions will be accepted till September 12, 2017, 2359 hrs, IST.
6. Late submissions will incur a penalty – the maximum marks for a late submission will be 80% of the total marks, even if solutions to all questions are absolutely correct.
7. We will be closely checking your code and theory submissions for instances of plagiarism.
8. You may be penalized if you do not follow the formatting and code submission instructions given below very carefully. We will be using automated scripts to perform part of the evaluation. If your file names, format etc are wrong, the scripts will not detect your answer and you will get a penalty.

Theory Part Submission

1. Your submission should be a single PDF file. No zip/tar/png/jpg files will be accepted.
2. The PDF file should have been compiled using the LaTeX style file provided to you.
3. A sample submission has been include to demonstrate the use of this style file.
4. Your answer to every question should begin on a new page. The style file is designed to do this automatically. However, if it fails to do so, use the `\clearpage` option in LaTeX before starting the new question, to enforce this.
5. Submission for this part should be made on Gradescope `https://gradescope.com`.
6. An account has been created for you on this website. Use your IITK CC ID (not GMail, CSE etc IDs) to login and use the "Forgot Password" option to set your password initially.
7. While submitting your assignment on this website, you will have to specify on which page(s) is question 1 answered, on which page(s) is question 2 answered etc. To do this properly, first ensure that the answer to each question starts on a different page.
8. Be careful to flush all your floats (figures, tables) corresponding to question n before starting the answer to question $n+1$ otherwise graders might miss your figures and award you less points. Again, the style file should do this automatically but be careful.

9. Your solutions must appear in proper order in the PDF file i.e. your solution to question n must be complete in the PDF file (including all plots, tables, proofs etc) before you present a solution to question $n + 1$.

10. We may impose a penalty on submissions that significantly deviate from the style file or which do not following the formatting instructions.

Installing the Anaconda Environment for Python

- Please note that the packages we will require for this assignment are not readily available in standard form for Windows platforms. We will recommend a Linux platform for the following. The IITK Computer Center provides Linux workstations to all students.

- Python will be the language used in all assignments in this course. If you are unfamiliar with the language, we recommend going to a source like (`https://www.codecademy.com/learn/learn-python`) for the basic syntax. All assignments will assume a passing familiarity with the language, and a certain level of comfort with programming in general.

- Feel free to contact the mentors with any queries you may have with regards to assignments. Python has two major versions, 2.7 and 3.5. There aren't many differences between the two, and our assignments should work on both. We will assume 3.5 as the default version but things should work on 2.7 without any additional changes.

- Anaconda is an "environment" for Python. The benefit of using this environment is to have all the packages we might need for work in one go, along with a separate copy of the basic Python backend.

- Please install Anaconda from `https://docs.continuum.io/anaconda/install/`. The package comes in a heavier (nearly 500MB) version which contains most packages an ML practitioner would ever need (and then some). If you feel this takes up too much time/space, download the light version, very creatively named "Miniconda". You will find both versions on the website given above as well as local mirrors of certain versions hosted at `http://web.cse.iitk.ac.in/users/govindg/temp/`.

Installing the Shogun Environment

- For this assignment, you will need the LMNN algorithm. It has an implementation in the Shogun machine learning library. Installation steps for Shogun are given below.

- Install Anaconda or Miniconda (https://docs.continuum.io/anaconda/install/)

- Open a terminal and execute the command: `conda install -c conda-forge shogun`

- Create a conda environment using the command `conda create --prefix ~/cs771`, replacing `~/cs717` with whatever path you wish to have.

- Activate the environment by executing the command `source activate ~/cs771`

- Initialize the Python environment by executing the command `python`

- Verify the Shogun installation by executing the command `import modshogun`

- If all goes well, the import will be successfull, and no error messages will be displayed

- Please note that alternate implementations of the LMNN algorithm do exist e.g. `https://all-umass.github.io/metric-learn/metric_learn.lmnn.html` which may run on Windows platforms but the Shogun implementation is highly optimized for speed and accuracy (it actually runs in C++).
- Also note that Shogun does not have readily available binaries for Windows unless you build the thing from scratch using MS Visual Studio etc. Hence, working with Linux environments is preferred.
- If you wish to use non-Shogun or Windows-based implementations of the LMNN algorithm, do realize that the accuracies you achieve may be different from those that your classmates obtain using the Shogun implementation.
- If you are able to get Shogun running with LMNN on Windows then all is fine. However, in that case, please teach the instructor how to do this as well :)

Code Part Submission

1. This submission should be a single ZIP file. No PY/PYC/M files will be accepted.
2. The name of the ZIP file should be your roll number. Eg. 17001.zip. If your submission is wrongly named, we may be unable to link it to you and you may lose credit.
3. You may resubmit but do not resubmit more than twice – you may incur a penalty for excessive resubmission. We will simply accept your latest submission.
4. Submissions for this part should be made to the following URL Submissions will be accepted at the following URL
   `https://www.dropbox.com/request/7s7l6UtgbuFv4hZ8jIwD`
5. The ZIP file should be no more than 500KB in size.
6. Do not include your PDF file from the theory part in this submission.
7. **Do not include any data files (training features etc) in your ZIP archive**. Your archive should only contain code and model files.
8. Your code must be well commented and must execute/compile without need for special packages or installations. If we are unable to execute your code, you may be asked to put up a demonstration and incur a penalty too.

Problem 1.1 (V for Voronoi)**.** Recall the learning with prototypes problem. Consider a two class problem where the prototypes are the points $(1, 0)$ (green) and $(0, 1)$ (red). Calculate the decision boundary when we use the learning with prototypes rule but with the following *Mahalanobis* metrics. In the following, $\mathbf{z}^1, \mathbf{z}^2 \in \mathbb{R}^2$ denote two points on the real plane

1. $d(\mathbf{z}^1, \mathbf{z}^2) = \langle \mathbf{z}^1 - \mathbf{z}^2, U(\mathbf{z}^1 - \mathbf{z}^2) \rangle$, where $U = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$
2. $d(\mathbf{z}^1, \mathbf{z}^2) = \langle \mathbf{z}^1 - \mathbf{z}^2, V(\mathbf{z}^1 - \mathbf{z}^2) \rangle$, where $V = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

Figure 1 pictorally depicts the prototypes as well as the sample solution if we had used the standard Euclidean metric to compute distances. In your submission, for each of the two parts above, you must include the following details

1. The mathematical expression for the decision boundary. For example, in the Euclidean case, it is the line $y = x$.
2. An image shading the red and green decision boundaries for the above cases similar to the figure on the right in Figure 1.

To aid you, both figures in Figure 1 have been included in your assignment package (`proto_blank.png` and `proto_euclid_sample.png`). Note that your images must be embedded in your PDF file and not sent separately. Use the `\includegraphics` command in LaTeX to embed images in your submission PDF file. (5+5=10 marks)

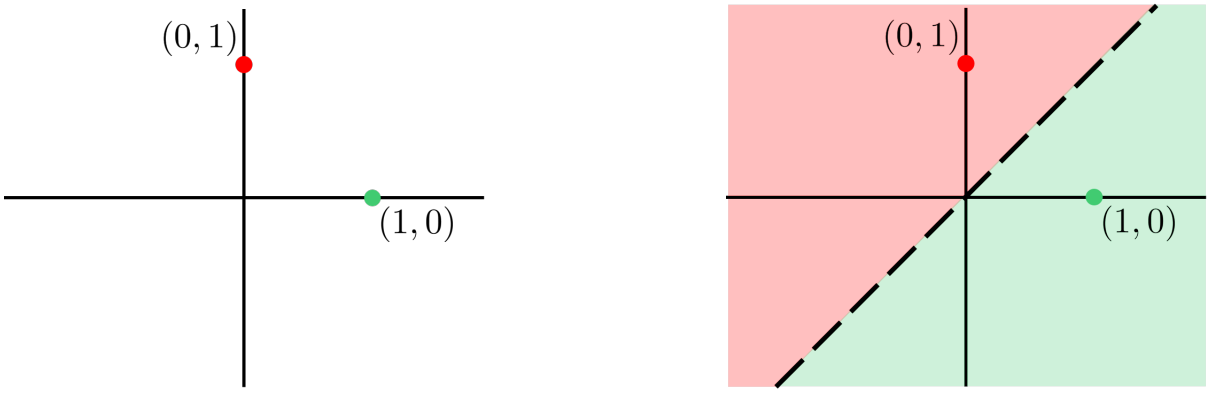

Figure 1: Learning with Prototypes: the figure on the left shows the two prototypes. The figure on the right shows what the decision boundary if the distance measure used is $d(\mathbf{z}^1, \mathbf{z}^2) = \|\mathbf{z}^1 - \mathbf{z}^2\|_2$, for any two points $\mathbf{z}^1, \mathbf{z}^2 \in \mathbb{R}^2$. The decision boundary in this case is the line $y = x$.

Problem 1.2 (PML For Constraints)**.** Consider the following constrained least-squares regression problem on a data set $(\mathbf{x}^i, y^i)_{i=1,\ldots,n}$, where $\mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in \mathbb{R}$.

$$\hat{\mathbf{w}}_{\text{cls}} = \arg\min \sum_{i=1}^{n} (y^i - \langle \mathbf{w}, \mathbf{x}^i \rangle)^2$$
$$\text{s.t. } \|\mathbf{w}\|_2 \leq r.$$

Design a likelihood distribution (on the responses, conditioned on the data covariates $\mathbf{x}$) and prior distribution (on the parameter) such that $\hat{\mathbf{w}}_{\text{cls}}$ is the MAP estimate for your model. Give explicit forms for the density functions of your likelihood and prior distributions. The above shows that PML approaches can also lead to constrained optimization problems. (5 marks)

Problem 1.3 (Fun with Features). Consider the following *feature-regularized* least-squares regression problem on a data set $(\mathbf{x}^i, y^i)_{i=1,\ldots,n}$, where $\mathbf{x}^i \in \mathbb{R}^d$, $y^i \in \mathbb{R}$, and $\alpha_j > 0$ for $j \in [d]$.

$$\hat{\mathbf{w}}_{\text{fr}} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} (y^i - \langle \mathbf{w}, \mathbf{x}^i \rangle)^2 + \sum_{j=1}^{d} \alpha_i (\mathbf{w}_i)^2$$

Design a likelihood and prior distribution such that $\hat{\mathbf{w}}_{\text{fr}}$ is the MAP estimate for your model. Give explicit forms for all distributions. It turns out that just as there exists a closed form expression for the solution to the L_2-regularized least-squares problem, one exists for this problem too. Find a closed-form expression for $\hat{\mathbf{w}}_{\text{fr}}$. (5+5=10 marks)

Problem 1.4 (Break Free from Constraints). Recall the OVA approach to multi-classification. Let us use a dataset $(\mathbf{x}^i, y^i)_{i=1,\ldots,n}$, where $\mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in [K]$ i.e. there are K classes. Denote using $\mathbf{W} = [\mathbf{w}^1, \ldots, \mathbf{w}^K] \in \mathbb{R}^{d\times K}$, the set of K linear models that make up the OVA classifier. The Crammer-Singer formulation $(P1)$ for a single machine learner for multi-classification is

$$\begin{aligned} \left\{\widehat{\mathbf{W}}, \{\hat{\xi}_i\}\right\} = \arg\min_{\mathbf{W},\{\xi_i\}} & \sum_{k=1}^{K} \left\|\mathbf{w}^k\right\|_2^2 + \sum_{i=1}^{n} \xi_i \\ \text{s.t. } & \left\langle \mathbf{w}^{y^i}, \mathbf{x}^i \right\rangle \geq \left\langle \mathbf{w}^k, \mathbf{x}^i \right\rangle + 1 - \xi_i, \forall i, \forall k \neq y^i \qquad (P1) \\ & \xi_i \geq 0, \text{ for all } i \end{aligned}$$

Show that $(P1)$ is equivalent to the following unconstrained formulation $(P2)$

$$\widehat{\mathbf{W}} = \arg\min_{\mathbf{W}} \sum_{k=1}^{K} \left\|\mathbf{w}^k\right\|_2^2 + \sum_{i=1}^{n} \ell_{\text{cs}}(y^i, \boldsymbol{\eta}^i) \qquad (P2),$$

where $\boldsymbol{\eta}^i = \langle \mathbf{W}, \mathbf{x}^i \rangle$ and

$$\ell_{\text{cs}}(y^i, \boldsymbol{\eta}^i) = [1 + \max_{k\neq y} \boldsymbol{\eta}^i_k - \boldsymbol{\eta}^i_y]_+$$

To show equivalence, you will have to show that if $\left\{\mathbf{W}^0, \{\xi^0_i\}\right\}$ are an optimum for (P_1) then $\mathbf{W}^0$ must be an optimum for $(P2)$, as well as if $\mathbf{W}^1$ is an optimum for $(P2)$ then there must exist $\left\{\xi^1_i\right\} \geq 0$ such that $\left\{\mathbf{W}^1, \{\xi^1_i\}\right\}$ are an optimum for (P_1). (15 marks)

Problem 1.5 (Sub-gradient Computation). Consider the following function, where $(\mathbf{x}^i, y^i)_{i=1,\ldots,n}$, where $\mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in \{-1, 1\}$ i.e. binary Rademacher labels.

$$f(\mathbf{w}) = \sum_{i=1}^{n} [1 - y^i \langle \mathbf{w}, \mathbf{x}^i \rangle]_+$$

Suppose I construct a vector $\mathbf{g}$ as follows $\mathbf{g} = \sum_{i=1}^{n} \mathbf{h}^i$, where

$$\mathbf{h}^i = \begin{cases} -y^i \cdot \mathbf{x}^i & \text{if } \; y^i \langle \mathbf{w}, \mathbf{x}^i \rangle < 1 \\ \mathbf{0} & \text{if } \; y^i \langle \mathbf{w}, \mathbf{x}^i \rangle \geq 1, \end{cases}$$

then show that $\mathbf{g} \in \partial f(\mathbf{w})$ i.e. $\mathbf{g}$ is a member of the subdifferential of f at $\mathbf{w}$. Recall that to show this, you have to show that for every $\mathbf{w}' \in \mathbb{R}^d$, $f(\mathbf{w}') \geq f(\mathbf{w}) + \langle \mathbf{g}, \mathbf{w}' - \mathbf{w} \rangle$. (5 marks)

Problem 1.6 (Metric Learning for NN classifiers)**.** Recall that we commented that the nearest neighbor (or more generally, the k-NN) algorithm can be made more powerful by learning an application specific metric instead of using a fixed metric like Euclidean (L_2) or Manhattan (L_1). In this exercise, you will use the LMNN method [1] to learn a Mahalaobis metric and then use the k-NN algorithm with this learnt metric to perform classification.

You have been supplied with a training data set with 60K data points and a test set with 20K data points, each point being 100 dimensional. Download these datasets from the URL

`http://web.cse.iitk.ac.in/users/purushot/courses/ml/2017-18-a/material/assn1data.zip`

This is a supervised multi-classification problem with 3 classes. You may use the training data provided to you in any way to tune your parameters (split into validation sets in any fashion (e.g. held out, k-fold), as well as use any fraction of data for validation, etc) but your job is to do well on your test data points (as well as a secret test set which we have with us and will not reveal to you).

Execute the following experiments using your data

1. Use the k-NN algorithm with the Euclidean metric to perform classification. Report your test error (on the 20K sized dataset) for different values of $k = 1, 2, 3, 5, 10$. Plot a graph showing test accuracies (fraction of the 20K points that were correctly classified) vs k. Explain your observations.

2. Tune a good value of k using your favorite validation technique. Do not touch your test set while performing validation. Report which value of k you found to work best.

3. Fixing the above value of k, learn a good Mahalanobis metric for the classification problem using the LMNN package. Note that your value of k was chosen for the Euclidean metric and not for the metric learnt by LMNN. We may do a joint optimization of the k value and the metric but let us leave that for now. Report your test accuracies using the learnt metric and the value of k chosen in step 2.

4. LMNN learns a Mahalanobis metric corresponding to a positive semi-definite matrix M but does not return M. Instead, it returns a linear transformation L such that $M = L^\top L$. Compute and store M in a serialized file called `model.npy`. Include this file inside your submission ZIP file.

5. Submit all your training code inside your submission ZIP file. Do not have include subdirectories inside the ZIP file. All files should be present directly inside the ZIP file. If you have used multiple files in the entire process of training, submit all of them.

6. Write a single Python script called `test.py` to allow others to use your learnt metric as well as your tuned value of k to perform classification on new points. The file must be able to accept the training and testing file names as system input, load the metric from the file `model.npy`, and save the predictions on the test set to a file called `testY.dat`. Include the file `test.py` inside your submission ZIP file.

Please note that your submissions to parts (1, 2, 3) above above go in the PDF file to Gradescope, while your submissions to parts (4, 5, 6) go in the ZIP file to Dropbox.

For you convenience, a sample training (`train.py`) and testing (`test.py`) file have already been included in the assignment package which adhere to the above instructions. Please make

[1] Kilian Q. Weinberger and Lawrence K. Saul, Distance Metric Learning for Large Margin Nearest Neighbor Classification, Journal of Machine Learning Research, 10:207-244, 2009.

changes to those files as per your wish but be careful not to change the way the test file accepts input and writes output. You may also refer to the following Python notebook for a quick explanation on how to invoke the LMNN routine `http://nbviewer.jupyter.org/gist/iglesias/6576096`.

You may use a brute force search or sorting operations to identify the k-nearest neighbor and need not invest in faster NN data structures like k-d trees etc. However, you should use fast sorting operations offered by packages such as `NumPy` and not try to implement sorting etc operations yourself – chances are your implementation will be really slow since Python is by-and-large, an interpreted language.

Extra Credit: the ITML metric learning technique [2] is another excellent method for learning Mahalanonbis metrics. This method won the best paper award at the ICML 2007 conference. You may obtain a Python implementation of ITML from `https://all-umass.github.io/metric-learn/metric_learn.itml.html` or else a Matlab implementation by the original authors from `http://www.cs.utexas.edu/users/pjain/itml/`.

The Github website also hosts several other metric learning techniques like SDML and LSML. Use ITML (and optionally the other methods) to learn Mahalanobis metrics and see if you can get superior accuracies or not. Report your findings on your test set. **Do not submit `test.py` or `model.npy` files for extra credit experiments** Just include your observations in the PDF file and the training code in the ZIP file.

Instructions

1. All plots must be generated electronically - no hand-drawn plots would be accepted. All plots must have axes titles and a legend indicating what the plotted quantities are.

2. All plots must be embedded in the PDF file – no stray image files will be accepted. Use the `\includegraphics` command in LaTeX to embed images in your submission PDF file.

3. Your submission must describe neatly what the plotted quantities are as well as the main inference that can be drawn from the plot. E.g. if varying k changes the accuracy, what changes do you observe?

4. If a file name has been specified in the instructions above, please stick to it and do not use any other file name. Our automated scripts will not be able to evaluate your code otherwise and you will incur a penalty. For training, you may use file names as per your wish but of course avoid the names we have specified above.

5. **Do not include any data files (.dat etc) inside your submission**. Only your code files (in .py format) and your model file (`model.npy`) should be there inside your submission ZIP file. We have the training files with us already and can easily include it at our own end to test your code. Your total submission size should not exceed 500KB.

(35+extra marks)

[2] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra and Inderjit S. Dhillon, Information Theoretic Metric Learning, ICML 2007.