# Homework 2
## Kernel SVM and Perceptron

CMU 10-701: Machine Learning (Fall 2014)
https://piazza.com/cmu/fall2014/1070115781/home
OUT: Sept 25, 2014
DUE: Oct 8, 11:59 PM

## START HERE: Instructions

- **Late days:** The homework is due Wesnesday Oct 8th, at 11:59PM. You have five late days to use throughout the semester, and may use at most three late days on any one assignment. Once the allowed late days are used up, each additional day (or part of a day) will subtract 1 from the normalized score for the assignment. View the full late days policy on Piazza.
- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get inspiration (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators and resources fully and completely (e.g., "Jane explained to me what is asked in Question 3.4" or "I found an explanation of conditional independence on page 17 of Mitchell's textbook"). Second, write up your solution independently: close the book and all of your notes, and send collaborators out of the room, so that the solution comes directly from you and you alone.
- **Programming:**
  - **Octave:** You must write your code in Octave. Octave is a free scientific programming language, with syntax identical to that of MATLAB. Installation instructions can be found on the Octave website. (You can develop your code in MATLAB if you prefer, but you *must* test it in Octave before submitting, or it may fail in the autograder.)
  - **Autograding:** All programming problems are autograded using the CMU Autolab system. The code which you write will be executed remotely against a suite of tests, and the results used to automatically assign you a grade. To make sure your code executes correctly on our servers, you should avoid using libraries which are not present in the *basic* Octave install.
- **Submitting your work:** All answers will be submitted electronically through the submission website: https://autolab.cs.cmu.edu/10701-f14.
  - Start by downloading the submission template. The template consists of directory with placeholders for your writeup ("problem1.pdf", "problem2.pdf"), and a single sub-directory for the programming parts of the assignment. *Do not modify the structure of these directories or rename these files.*
  - **IMPORTANT:** When you download the template, you should confirm that the autograder is functioning correctly by compressing and submitting the directory provided. This should result in a grade of zero for all programming questions, and an unassigned grade (-) for the written questions.
  - **Writeup:** Replace the placeholders with your actual writeup. Make sure to keep the expected file names ("problem1.pdf"), and to submit one PDF per problem. To make PDFs, we suggest pdflatex, but just about anything (including handwritten answers) can be converted to PDF using copier-scanners like the ones in the copier rooms of GHC.
  - **Code:** For each programming sub-question you will be given a single function signature. You will be asked to write a single Octave function which satisfies the signature. In the handout linked above, the "code" folder contains stubs for each of the functions you need to complete.
  - **Putting it all together:** Once you have provided your writeup and completed each of the function stubs, compress the top level directory *as a tar file* and submit to Autolab online (URL

above). You may submit your answers as many times as you like. You will receive instant feedback on your autograded problems, and your writeups will be graded by the instructors once the submission deadline has passed.

# Problem 1: SVM decision boundaries [Zichao - 30pts]

Support Vector Machines are powerful tools for classifications. SVM can perform non-linear classification using the kernel trick. In this question, you are to examine the decision boundaries of SVM with different kernels.

1. Recall that the soft-margin primal SVM problem is

$$\min \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{i=1}^{n}\xi_i \tag{1}$$

$$\text{s.t.} \forall i = 1, \cdots, n: \tag{2}$$

$$\xi_i \geq 0 \tag{3}$$

$$(\mathbf{w}\cdot\mathbf{x_i} + b)y_i - (1-\xi_i) \geq 0. \tag{4}$$

For hard-margin primal SVM, $\xi_i = 0, \forall i$. We can get the kernel SVM by taking the dual of the primal problem and then replace the product of $\mathbf{x_i}\cdot\mathbf{x_j}$ by $k(\mathbf{x_i}, \mathbf{x_j})$, where $k(.,.)$ can be any kernel function.

Figure 1 plots SVM decision boundaries resulting from using different kernels and/or different slack penalties. In Figure 1, there are two classes of training data, with labels $y_i \in \{-1, 1\}$, represented by circles and squares respectively. The SOLID circles and squares represent the support vectors. Label each plot in Figure 1 with the letter of the optimization problem below and explain WHY you pick the figure for a given kernel. (Note that there are 6 plots, but only 5 problems, so one plot does not match any of the problems.)

   (a) A soft-margin linear SVM with $C = 0.1$. **[4 pts]**

   (b) A soft-margin linear SVM with $C = 10$. **[4 pts]**

   (c) A hard-margin kernel SVM with $\mathbf{K}(\mathbf{u}, \mathbf{v}) = \mathbf{u}\cdot\mathbf{v} + (\mathbf{u}\cdot\mathbf{v})^2$. **[4 pts]**

   (d) A hard-margin kernel SVM with $\mathbf{K}(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{1}{4}\|\mathbf{u}-\mathbf{v}\|^2\right)$. **[4 pts]**

   (e) A hard-margin kernel SVM with $\mathbf{K}(\mathbf{u}, \mathbf{v}) = \exp\left(-4\|\mathbf{u}-\mathbf{v}\|^2\right)$. **[4 pts]**

2. You are given a training dataset, as shown in Fig 2. Note that the training data comes from sensors which can be error-prone, so you should avoid trusting any specific point too much. For this problem, assume that we are training an SVM with a quadratic kernel.

   (a) Where would the decision boundary be for very large values of $C$ (i.e., $C \to \infty$)? Draw on figure and justify your answer. **[3 pts]**

   (b) For $C \approx 0$, indicate in the figure where you would expect the decision boundary to be? Justify your answer. **[3 pts]**

   (c) Which os the two cases above would you expect to work better in the classification task? Why? **[4 pts]**

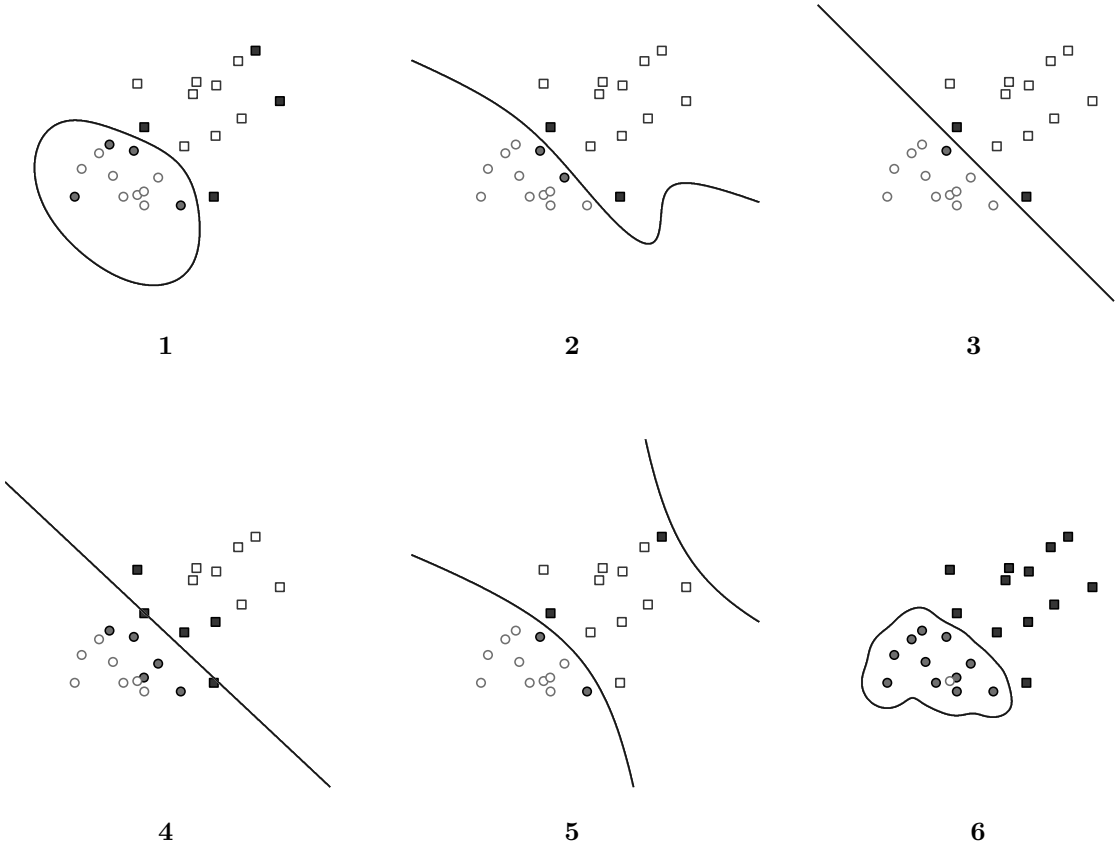


Figure 1: Induced Decision Boundaries

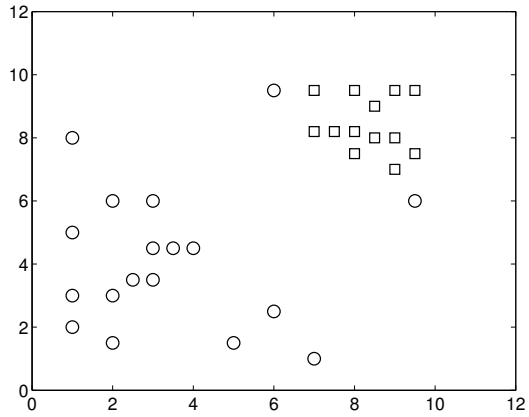


Figure 2: Training dataset

# Problem 2: Understanding the Likelihood Function, Bit by Bit (Ben, 30 points)

You are receiving a random stream of 0 or 1 bits, and you would like to know the probability that the next received bit is 1. This can be thought of as flipping a (possibly unfair) coin and finding the probablity of the coin being heads.
Let

$$H_i = \begin{cases} 1 & \text{if the } i\text{th bit received is 1} \\ 0 & \text{if the } i\text{th bit received is 0} \end{cases}$$

Let $P(H_i = 1) = p_H$. Then, $P(H_i = 0) = 1 - p_H$. Let $N_H = \sum_{i=1}^{N_{\text{bits}}} H_i$ be the number of received 1s and $N_{\text{bits}}$ be the total number of bits received. We observe $H_1, \ldots, H_{N_{\text{bits}}}$.

1. (a) (1 point) Give the name of the distribution and list its parameters for the random variable $H_i$ (i.e., $H_i \sim$ Distribution(parameters)). Give the name of the distribution and list its parameters for the random variable $N_H$.

   (b) Since we do not know the true parameter $p_H^*$, we would like to estimate it from observed data. One estimate is the maximum likelihood estimate (MLE). This maximizes the likelihood function of $p_H$ given the data.

      i. (1 point) Give the form of the likelihood function, $\mathcal{L}(H_1, \ldots, H_{N_{\text{bits}}}; p_H)$. Use the symbols $N_H$ and $N_{\text{bits}}$ where appropriate.

      ii. (2 points) Let $N_{\text{bits}} = 3$, and say we observed four different sequences of bits: 111, 110, 001, 000. Plot $\mathcal{L}(H_1, \ldots, H_{N_{bits}}; p_H)$ (y-axis) vs. $p_H$ (x-axis) for each sequence on the same axes. (You will have four curves on the same plot. Sample $p_H$ from 0 to 1 in 0.01 increments.) On each curve, put a star at the $p_H$ that has the highest likelihood. These are the maximum likelihood estimates. Do they make intuitive sense given the data? Why?
      [For your convenience, you can use the given hw2_ques2_1_b_ii.m as a starting script to make the plot. Simply fill in the missing code where it instructs you. Include a figure of the plot in your write-up. Do not include extra .m files in your homework submission.]

      iii. (2 points) Using calculus, calculate the area under the curve for each sequence. You can check the approximate correctness of these answers by summing up the curves in 2.1.b.ii. Is the likelihood function a valid probability distribution over $p_H$? Explain.

   (c) (2 points) Now compute $\hat{p}_H$, the MLE, for any $N_{\text{bits}}$. Write out and label each step clearly, providing movitation for the steps (e.g., why are you taking the derivative?). Just writing down the MLE form will not be given full credit. You should only need to use $N_{\text{bits}}$, $H_1, \ldots, H_{N_{\text{bits}}}$, $N_H$, $p_H$, and $\hat{p}_H$ as symbols.

2. We now consider the case where we receive the stream of bits with additive noise (i.e., channel corruption). The noise model can be written:

$$O_i = H_i + \epsilon_i$$

   where $i = 1, \ldots, N_{\text{bits}}$, $O_i$ is the observed bit with noise, $H_i$ is defined in the same manner as in 2.1, and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, where $\sigma^2$ is given.

   (a) (3 points) Give the form of the likelihood function, $\mathcal{L}(O_1, \ldots, O_{N_{\text{bits}}}; p_H)$. Determine if the log of this function is concave. Why is concavity important? (One way of checking for concavity is to see if the second derivative is negative for all values of $p_H$.)

   (b) (3 points) As in 2.1.c, compute the MLE of $p_H$, $\hat{p}_H$, for the likelihood function, $\mathcal{L}(O_1, \ldots, O_{N_{\text{bits}}}; p_H)$. Write out the analytical form, if possible. Otherwise, describe how to obtain the MLE without a closed form.

(c) (4 points) Let the true parameter $p_H^* = 0.6$, $\sigma^2 = 0.1$. Generate 4 datasets from the model with different trial sizes (i.e., vary $N_{\text{bits}}$): $\{100, 500, 1000, 2500\}$. Plot $\log \mathcal{L}(O_1, \ldots, O_{N_{\text{bits}}}; p_H)$ vs. $p_H$ for each dataset on the same axes. For each curve, place a star at the $p_H$ with the maximum log-likelihood. (There will be 4 curves on the same plot. Again, sample $p_H$ from 0 to 1 with 0.01 increments.) What happens when you increase $\sigma^2$? [You can use the given hw2_ques2_2_c.m as a starting script. Include the plot as a figure in your write-up.]

(d) (3 points) Let us define another estimator, $\bar{p}_H$, as the number of $O_i$ greater than 0.5 divided by $N_{\text{bits}}$. This estimator takes a threshold of the observed data, and is not the MLE. To compare $\hat{p}_H$ with $\bar{p}_H$, plot $\hat{p}_H$ vs. trial size (the same trial sizes as in 2.2.c) and $\bar{p}_H$ vs. trial size on the same axes. (There will be two curves on the same plot.) Comment on which estimator does a better job. What happens when you increase $\sigma^2$? [You can use the given hw2_ques2_2_d.m as a starting script. Include the plot as a figure in your write-up.]

3. Assume the same model in 2.2, $O_i = H_i + \epsilon_i$, $i = 1, \ldots, N_{\text{bits}}$, but now $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$, where the variance grows for each consecutively received bit.

(a) (3 points) Write out the form of the likelihood function $\mathcal{L}(O_1, \ldots, O_{N_{\text{bits}}}; p_H)$. Is the log-likelihood function concave?

(b) (3 points) Repeat 2.2.c with the new model, where $p_H^* = 0.6$ and $\sigma_i^2 = \frac{i}{N_{\text{bits}}}$. Plot $\log \mathcal{L}(O_1, \ldots, O_{N_{\text{bits}}}; p_H)$ vs. $p_H$ for each trial size. (There will be 4 curves on the same plot.) [You can use the given hw2_ques2_3_b.m as a starting script. Include the plot as a figure in your write-up.]

(c) (3 points) Repeat 2.2.d with the new model. Plot $p_H$ vs. trial size for both estimators. (There will be two curves on the same plot.) Comment on the differences between $\hat{p}_H$ and $\bar{p}_H$. [You can use the given hw2_ques2_3_c.m as a starting script. Include the plot as a figure in your write-up.]

# Problem 3: Kernels and Features [Adona; 35 pts + 5 Bonus]

In this question you will implement *Perceptron Classification*, and *Kernel Perceptron Classification*. Remember to follow the detailed coding and submission instructions at the top of this file!

- **Data:** All questions will use the following datastructures:
  - $XTrain \in \mathbb{R}^{n \times f}$ is a matrix of training data, where each row is a training point, and each column is a feature.
  - $XTest \in \mathbb{R}^{m \times f}$ is a matrix of test data, where each row is a test point, and each column is a feature.
  - $yTrain \in \{-1, +1\}^{n \times 1}$ is a vector of training labels
  - $yTest \in \{-1, +1\}^{m \times 1}$ is a (hidden) vector of test labels.
- **SUBMISSION CHECKLIST**
  - Submission executes on our machines in less than **10 minutes**.
  - Submission is smaller than 15MB.
  - Submission is a `.tar` file.
  - Submission returns matrices of the *exact* dimension specified.

## Perceptrons

1. **Perceptron Weights [5 pts]**
   Complete the function $pWeights(XTrain, yTrain, nIter)$ which takes as input the training data $XTrain$, the training labels $yTrain$, and the number of iterations $nIter$ (number of times the algorithm goes through the entire dataset) and returns a $f \times 1$ weight vector $w$. pWeights() should implement the iterative perceptron weight learning algorithm described in class.

2. **Perceptron Classifier [2pts]**
   Complete the function $pClassify(XTest, w)$ which takes as input the test data, and the weight vector, and returns a $m \times 1$ vector of class predictions $p \in \{-1, +1\}^m$.

## Kernel Perceptrons

In this question you will implement the kernel perceptron classification algorithm.

At an abstract level kernel perceptrons work by projecting the features into a high dimensional feature space, then doing normal perceptron learning in the high dimensional feature space. i.e. $\forall x$ we replace $x$ with $\phi(x)$, then run perceptron learning using $\phi(x)$ as our features.

Unfortunately it is computationally infeasible to do this explicitly. Instead we work in the high dimensional space implicitly using inner products. We note that given a data point $\phi(x)$ and a weight vector $w$, we classify $\phi(x)$ by calculating $\langle \phi(x), w \rangle$. Furthermore we know $w$ is a linear combination of the points $\phi(x_1), ..., \phi(x_n)$ (i.e. $w = \sum_i \alpha_i \phi(x_i)$ for some $\alpha_1, ..., \alpha_n$. Therefore $\langle \phi(x), w \rangle = \sum_i \alpha_i \langle \phi(x), \phi(x_i) \rangle$. Therefore all we really need to do is calculate the inner products between points, and keep track of the $\alpha$ values.

***Note that in this question we use a $d$th degree polynomial kernel with inner product*** $\langle \phi(x_i), \phi(x_j) \rangle = \langle x_i, x_j \rangle^d$.

1. **Kernel Weights [10 pts]**
   Complete the function $kpWeights(XTrain, yTrain, nIter, d)$ which takes as input the training data $XTrain$, the training labels $yTrain$, the number of iterations to run for $nIter$, and the degree of the polynomial kernel $d$. kpWeights() should output a $n \times 1$ vector $\alpha$, whose elements are the multipliers for each example learned by the kernel perceptron algorithm. Note that the perceptron weight vector $w$ is implicitly represented through these $\alpha$ values.

2. **Kernel Perceptron Classifier [8 pts]**
   Complete the function $kpClassify(XTrain, XTest, \alpha, d)$ which takes as input the training data $XTrain$, the test data $XTest$, the weight vector $\alpha$, and the degree of the polynomical kernel $d$. kpClassify() should output a $m \times 1$ vector of class predictions $p \in \{-1, +1\}^m$.

## CHALLENGE: SPAM Feature Engineering [10 pts + 5 bonus]

This is a challenge question, and is significantly more difficult than previous questions. Furthermore this question is a Competition! There is a class leaderboard for this question on the Autolab website. The 10 basic points will be awarded to all students passing a basic accuracy threshold. Bonus points will be awarded to students who have top 10 classification accuracy on the class leaderboard.

- In this question you will engineer a list of features for SPAM email classification. The idea is to convert an email into a list of real numbers, which can then be used train a machine learning classification algorithm. The performance of the resulting classifier will depend ***Heavily*** on how good your features are, so select them carefully!
- Specifically your task is to complete the function $mail2Feat(Mail)$ which converts a set of emails into a matrix of real numbers. Suppose we have $e$ emails, where the $i$th email has $w_i$ words. $Mail$ is a $e \times 1$ cell array of emails, each of which is $w_i \times 1$ cell array of words. You can produce as many output features as you like, the only caveat is that all emails must have the same set of features. If you decide to output $f$ features, then $mail2Feat()$ should output a $e \times f$ matrix of real numbers.
- In addition to the $mail2Feat()$ function you may also use the *dictionary.csv* file. This file allows you to include abitrary data (such as a dictionary of words) with your submission. Use this file to store and retrieve preprocessed results in your submission. An example of how to use this file is included in the $mail2Feat.m$ template file.
- Your will be graded based on how well your features support classification. Specifically your $mail2Feat()$ function will be used to convert two hidden sets of emails into features. We will then train a perceptron using one set of emails, and test the performance of the perceptron on another. Your grade will be directly proportional to the classification accuracy obtained.

HINTS: I have included code in the $mail2Feat.m$ template file which demonstrates an incredibly basic feature map. This code simply takes a few words and counts how many times each word appears in each email. These counts are the features. For more advanced (and successful) feature maps, TF-IDF is a good place to start. (http://en.wikipedia.org/wiki/Tf%E2%80%93idf).