MACHINE LEARNING COMS 4771, HOMEWORK 3
Assigned October 15, 2015. Due November 3, 2015 before 10:00am.

1 Problem 1 (5 points)

Suppose you work at a casino and you are responsible for making sure no one is cheating. Specifically, you are interested in making sure the dice used in the craps tables are fair.

Craps is a game that involves throwing two 6-sided dice each round. For this exercise let's consider a simplified version of this game:

- Each round a player throws a pair of dice.
- If the sum of the outcomes of the dice is 7, the player wins, otherwise the player loses.

The casino has noticed that some players are secretly replacing the regular (fair) dice with a pair of loaded (unfair) dice. When you throw an unfair die there is 1/3 probability the outcome being a 3, 1/3 probability of being a 4, and the other outcomes are all equally likely. Historic data supports that there is a 1/1000 probability that a craps table has unfair dice at any point in time. The plan of action of the casino is to replace both dice of a table when there is more than 50% probability that they are unfair.

The casino manager asks you two things:

1. A mathematical function that receives the number of rounds played in a table, and outputs the minimum number of wins of a player, such that the dice are most likely unfair.
2. What is the minimum number of rounds they have to observe so they can infer a pair of dice is more likely to be unfair? What are the necessary outcomes of the dice that result in the minimum number of observations needed for the inference?

2 Problem 2 (10 points)

For this problem, suppose we have a coin that when tossed, has probability μ of landing on heads (and probability $1-\mu$ of tails). However, we do not know μ.

Consider two possible prior distributions: (A) $\mu \sim \text{Uniform}[0,1]$; (B) we have some reason to think the coin is likely to be fair so assume μ has a probability distribution which has the form of a concave parabola with its maximum at $\frac{1}{2}$ and falls to 0 at 0 and 1.

Along with 2 possible realizations: (1) $\mathcal{D}_1 = \{H, T\}$; (2) $\mathcal{D}_2 = \{T, T, T\}$.

For each of the 4 combinations of priors and realizations, derive with proof:

- $p(H)$ given the prior
- the posterior distribution $p(\mu|\mathcal{D})$ [ensure this is properly normalized]
- the maximum likelihood estimate μ_{ML} given the data $\mathcal{D}$

- the MAP estimate μ_{MAP} given the data $\mathcal{D}$
- $p(H|\mathcal{D})$, i.e. the full Bayesian probability that the next toss is H
- the variance of the posterior distribution $p(\mu|\mathcal{D})$

Put the final result of all items for all combination of priors and realizations in a single table.

Give one reason why the maximum likelihood estimate might not be very reliable in this context. Use the results you calculated to support your argument.

3 Problem 3 (5 points): Conditional Independence

Prove or disprove with a counterexample. For random variables a, b, c:

1. $a \perp\!\!\!\perp b|c \rightarrow a \perp\!\!\!\perp b$.
2. $a \perp\!\!\!\perp b \rightarrow a \perp\!\!\!\perp b|c$
3. $(a \perp\!\!\!\perp b) \wedge (b \perp\!\!\!\perp c) \rightarrow a \perp\!\!\!\perp c$

4 Problem 4 (10 points): Maximum Likelihood

The Gamma distribution is a probability density over non-negative scalars, $x \geq 0$. One particular form is as follows

$$p(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x^{\alpha-1} e^{-\lambda x}$$

The density has two scalar parameters $\lambda, \alpha > 0$.

What is the maximum likelihood estimate of λ from a dataset $x_1, \cdots, x_n$ of n iid samples, given that α is fixed?

NOTE: make sure you show that your answer is a maximum point of the likelihood, and not a minimum.

5 Problem 5 (20 points): Kernel Logistic Regression

Load dataset 'data1.mat'. Implement kernel logistic regression with an ℓ^2 regularizer using the empirical kernel map. In other words, minimize:

$$J(w) = -\frac{1}{N} \sum_{i=1}^{N} \log(\sigma(y_i w^\top k_i)) + \lambda w^\top w$$

to get w. Here k_i is a column vector such that $k_i = [k(x_i, x_1), \cdots, k(x_i, x_j), \cdots, k(x_i, x_N)]^\top$. Here $y_i \in \{-1, +1\}$ is the given label for each input $x_i \in \mathbb{R}^d$. The function σ is defined as $\sigma(v) = 1/(1 + e^{-v})$.

For this exercise, use the RBF kernel $\exp(-\|x_i - x_j\|^2/\kappa^2)$, where the parameter κ is $\kappa^2 = \frac{1}{N^2}\sum_{i,j=1}^{N} \|x_i - x_j\|^2$.

After w is obtained, for any test input x, compute $p(y = 1|x) = \sigma(w^\top k_x)$, where $k_x = [k(x, x_1), \cdots, k(x, x_j), \cdots, k(x, x_N)]^\top$. If $p(y = 1|x) > 0.5$, the predicted label $y = 1$, otherwise, $y = -1$.

To optimize $J(w)$, use three different methods: gradient descent and two different flavors of stochastic gradient descent (SGD). For SGD, use p random points to estimate the gradient in each round. Experiment with $p = 1$ and $p = 100$.

For each of the three methods, experiment with the step size and choose the one that makes the algorithm converge faster while maximizing the final accuracy. Stop the iterations when the gradient becomes smaller than epsilon (say, $1e - 2$). For SGD, since depending on the step size convergence might take a long time, also add a time limit to the training loop. Make sure you set a maximum time that is at least as long as the time it takes for the gradient descent to converge, so we can make a fair comparison between the methods. Notice also that the final model in the case of SGD will not be the model of the last training iteration, but rather the model found to have lowest cost during the training.

It is not necessary to cross-validate to choose regularization factor, use a fixed value (say, $1e - 2$).

For the each of the three optimization methods, report:

1. Test accuracy
2. Chosen step size
3. Number of training iterations (until convergence or time limit)
4. Total training time (in seconds)
5. One plot of the cost $J(w)$ as a function of time (in seconds) since the beginning of the training. Plot the first 5-10 seconds, depending on your computer speed. To do this plot, simply record the cost at every iteration for the first 5-10 seconds and then plot this over a linear space from 0 to 5-10 seconds (assuming all iterations took the same amount of time)

Finally, create a single plot overlaying the cost functions over time of the three methods. Which methods seem faster in the first seconds of training?

HINT: use the "tic" function before you begin training and the "toc" function inside your loop to check how long the training has been running.