# CS 475 Machine Learning (Spring 2017): Assignment 3

Due on April 7th, 2017 at 3:00PM

*Raman Arora*

**Instructions**: Please read these instructions carefully and follow them precisely. Feel free to ask the instructor if anything is unclear!

1. Please submit your solutions electronically via Gradescope.
2. Please submit a PDF file for the written component of your solution including derivations, explanations, etc. You can create this PDF in any way you want: typeset the solution in LATEX (recommended), type it in Word or a similar program and convert/export to PDF, or even hand write the solution (legibly!) and scan it to PDF. We recommend that you restrict your solutions to the space allocated for each problem; you may need to adjust the white space by tweaking the argument to `\vspace{xpt}` command. Please name this document <firstname-lastname>-sol3.pdf.
3. Submit the empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures) in a Jupyter notebook file. In addition, we require that you save the Python notebook as a pdf file and append it to the rest of the solutions.
4. In addition, you will need to submit your predictions on a sentiment classification task to `Kaggle`, as described below, according to the competition rules.
5. **Late submissions:** You have a total of 72 late hours for the entire semester that you may use as you deem fit. After you have used up your quota, there will be a penalty of 50% of your grade on a late homework if submitted within 48 hours of the deadline and a penalty of 100% of your grade on the homework for submissions that are later than 48 hours past the deadline.
6. **What is the required level of detail?** When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers). When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important. When submitting code, please make sure it's reasonably documented, and describe succinctly in the written component of the solution what is done in each py-file.

**Name:** Tony Melo

## I. Support Vector Machines

In this problem, we will consider some details of the dual formulation of SVM, the one in which we optimize over the Lagrange multipliers $\alpha_i$. In class we saw how to derive a constrained quadratic program, which can then be "fed" to an off-the-shelf quadratic program solver. These solvers are usually constructed to handle certain standard formulations of the objective and constraints.

Specifically, the canonical form of a quadratic program with linear constraints is, mathematically:

$$\operatorname*{argmin}_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^\top \boldsymbol{\alpha}, \tag{1}$$

$$\text{such that: } \mathbf{A}\cdot\boldsymbol{\alpha} \le \mathbf{a}, \tag{2}$$

$$\mathbf{B}\cdot\boldsymbol{\alpha} = \mathbf{b}. \tag{3}$$

The vector $\boldsymbol{\alpha} \in \mathbb{R}^N$, where $N$ is the number of training examples, contains the unknown variables to be solved for. The matrix $\mathbf{H} \in \mathbb{R}^{N\times N}$ and vector $\mathbf{f} \in \mathbb{R}^N$ specify the quadratic objective; the matrix $\mathbf{A} \in \mathbb{R}^{k_{ineq}\times N}$ and vector $\mathbf{a} \in \mathbb{R}^{k_{ineq}}$ specify $k_{ineq}$ inequality constraints. Similarly, $\mathbf{B} \in \mathbb{R}^{k_{eq}\times N}$ and vector $\mathbf{b} \in \mathbb{R}^{k_{eq}}$ specify $k_{eq}$ equality constraints. Note that you can express a variety of equality constraints by adding rows to $\mathbf{A}$ and elements to $\mathbf{a}$; think how you would do it to express, e.g., a "greater or equal" constraint.

**Problem 1 [20 points]** Describe in detail how you would compute $\mathbf{H}$, $\mathbf{f}$, $\mathbf{A}$, $\mathbf{a}$, $\mathbf{B}$, and $\mathbf{b}$, to set up the dual optimization problem for the kernel SVM

$$\operatorname*{argmin}_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max\left[0, 1 - y_i\left(\mathbf{w}^\top \phi(\mathbf{x}_i) - w_0\right)\right]\right\}$$

given a kernel function $k(\cdot,\cdot)$ corresponding to the dot product in $\phi$ space, and $N$ training examples $(\mathbf{x}_i, y_i)$

Here we start with the question of how to optimize the margin of a linear classifier, such that for the distance from a correctly classified $(x,y)$ to the boundary.

$$\frac{1}{\|w\|} y(w\cdot x + w_0)$$

We want to maximize $\frac{1}{\|w\|}$ s.t. $y_i(w\cdot x + w_0) \ge 1$ or minimize $\|w\|^2$.

We can associate the loss function:

$$\max_{\alpha_i \ge 0} \alpha_i [1 - y_i(w_0 + w\cdot x_i)] = \begin{cases} 0 & \text{if } y_i(w_0 + w\cdot x_i) - 1 \ge 0 \\ \infty & \text{otherwise} \end{cases}$$

and reformulate our problem as $\min\{\frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \max_{\alpha_i\ge 0} \alpha_i [1 - y_i(w_0 + w\cdot x_i$

After substitution of derivatives, we arrive at

$\min -\{\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\alpha_i\alpha_j y_i y_j x_i\cdot x_j$ so, $H = y_i K(x_i, x_j) y_j$, $f = -1$

subject to $\sum \alpha_i y_i = 0$, $\alpha_i \ge 0$ $A = -1$, $a = 0$, $b = 0$, $B = y$

## II. Sentiment analysis

In this problem, we will develop a sentiment analysis tool. We have provided a data set containing short customer reviews (or snippets of reviews) for products. Each has been labeled as a positive or negative review. For instance, below is an example of a positive review

```
i downloaded a trial version of computer associates ez firewall and antivirus and
fell in love with a computer security system all over again .
```

and a negative one

```
i dont especially like how music files are unstructured ; basically they are
just dumped into one folder with no organization , like you might have in windows
explorer folders and subfolders .
```

from the training set. We will use Support Vector Machines to learn to classify such review sentences into positive and negative classes.
We will use the word occurrence features: if a particular word (or more generally, a token, which may include punctuation, numbers, etc.) $w$ occurs in an example, the corresponding feature is set to 1, otherwise to 0.

**Problem 2 [40 points]** Fill in the missing pieces of code to fully implement the SVMs. This includes fleshing out the input to the optimization, and calculation of the model predictions. Using the provided `dev` (development) set as a validation set, tune an SVM predictor you think is best, and use it to compute and submit predictions on the `test` set to Kaggle: `https://inclass.kaggle.com/c/cs475-sentiment-analysis`.

You are free to experiment with various aspects of SVMs, but at the minimum please do the following:

1. Run linear SVM
2. Run at least one non-linear SVM (with some non-linear kernel)
3. Evaluate a range of values of $C$ (regularization parameter) and the kernel-specific parameter(s) and discuss your findings – how do these values affect the performance of the classifier on training/validation data?

The value of C changes our accuracy on the training/val data where C being too large will have too large a margin and lead to poor validation while C being too small means the training accuracy is worse since our margin is much smaller.

Some ideas for additional (optional) exploration:

- Consider various scaling on the input features, for instance, z-scoring or unit length normalization of each example.
- Using bigram features. Consider pairs of consecutive words, instead of, or in addition to, the unigram features (individual word occurences).
- Experiment with the frequency cutoff for including a word (or bigram) in the dictionary used to extract features. The default value for this we recommend is 5 (i.e., if a word appear less than 5 times in the data set it is ignored), but perhaps you will get better results with other values. This and the previous points are already possible with the code (see arguments in `utils.preprocess`).
- Once you identify good setting for your hyperparameters ($C$ etc.) you could retrain the classifier on the combined `train` and `dev` sets, and then test it on `test`.

Note: you will need to install a convex optimization package `cvxopt` which is likely not included by default with your Python installation. If you are using Anaconda, you may be able to install is with the simple command

```
conda install -c omnia cvxopt
```

or

```
pip2 install cvxopt
```

You can find other instructions on how to install it here:
`https://anaconda.org/anaconda/cvxopt`
or here:
`http://cvxopt.org/install/`
Start working on this early, and ask course staff for help with the software issues, if you need it!

## III. Graphical Models

**Problem 3 [10 points]** Consider the Bayesian Network given in Figure 1.

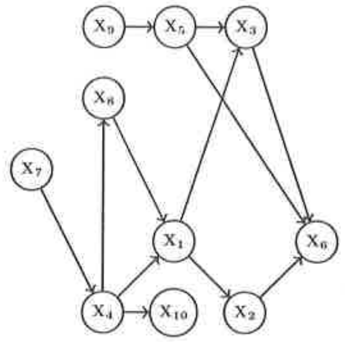


**Figure 1:** A directed graph

Are the sets **A** and **B** d-separated given set **C** for each of the following definitions of **A**, **B** and **C**? Justify each answer.

a. $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_5, x_4\}$

Yes, $X_3$ is a collider along the only path from $X_1$ to $X_9$ so $A$ is d-separated from B by C since $X_3$ has no descendants in C.

b. $\mathbf{A} = \{x_2\}$, $\mathbf{B} = \{x_7\}$, $\mathbf{C} = \{x_1, x_9\}$

Yes, even though there is a collider-free path from $X_2$ to $X_7$ it traverses $X_1 \in C$.

c. $\mathbf{A} = \{x_4\}$, $\mathbf{B} = \{x_5\}$, $\mathbf{C} = \{x_6, x_3\}$

No, since $X_6 \in C$ and is also a collider, the path from $X_4$ to $X_5$ becomes unblocked meaning C no longer blocks A from B.

d. $\mathbf{A} = \{x_4, x_8\}$, $\mathbf{B} = \{x_6, x_3\}$, $\mathbf{C} = \{x_1, x_7, x_9\}$

Yes, every path from $a \in A$ to some $b \in B$ is collider-free but traverses a vertex $c \in C$

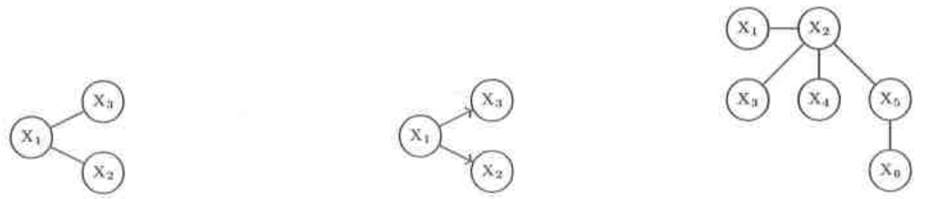
e. $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_2, x_3, x_4, x_5, x_8\}$, $\mathbf{C} = \{x_6, x_7, x_9, x_{10}\}$

No, $X_1$ is not d-separated from $X_2$, thus A cannot be d-separated from B since every $a \in A$ is not d-separated from every $b \in B$

Now assume that Figure 1 is a Markov Random Field, where each edge is undirected (just drop the direction of each edge.) Re-answer each of the above questions with justifications for your answers.

If we change Figure 1 to an MRF, every answer except e is yes, and e is no. The same justifications apply, except for c, there are no longer any directions to make $X_6$ a collider, so C does block A from B.

**Problem 4 [10 points]** A Markov Random Field usually cannot help us with the factorization of the distribution function. However, some MRFs can be converted to Bayesian Networks. For example, consider the graph structure in Figure 2a.



(a) Original undirected graph (b) Converted directed graph (c) An Undirected Graph

**Figure 2**: Graphs for Problem 4

From this graph, we know that $X_2$ and $X_3$ are conditionally independent given $X_1$. We can draw the corresponding directed graph as Figure 2b. This suggests the following factorization of the joint probability:

$$P(X_1, X_2, X_3) = P(X_3|X_1)P(X_2|X_1)P(X_1)$$

Now consider the graphical model in Figure 2c. As before, we can read the conditional independence relations from the graph.

(a) Following the example above, show a factorization of the joint distribution.

$$\begin{tikzcd}
X_1 \arrow[r] & X_2 \arrow[dl] \arrow[d] \arrow[dr] & \\
X_3 & X_4 & X_5 \arrow[d] \\
& & X_6
\end{tikzcd}$$

$p(X_1, X_2, \ldots, X_6) = p(X_6 \mid X_5, \ldots, X_1) \cdot p(X_5 \mid X_4, \ldots, X_1) \cdot \ldots \cdot p(X_1)$

From the conditional indepencies in our graph, this factors down to

$p(X_6 \mid X_5) p(X_5 \mid X_2) p(X_4 \mid X_2) p(X_3 \mid X_2) p(X_2 \mid X_1) \cdot p(X_1) = p(X_1, \ldots, X_6)$

(b) Is this factorization unique, meaning, could you have written other factorizations that correspond this model?

No, this factorization is not unique

(c) If the factorization is unique, show why it is unique. If it is not unique, provide an alternate factorization.

$$\begin{tikzcd}
X_1 & X_2 \arrow[l] \arrow[dl] \arrow[d] \arrow[dr] & \\
X_3 & X_4 & X_5 \arrow[d] \\
& & X_6
\end{tikzcd}$$

$p(X_1, \ldots, X_6) =$

$p(X_2) p(X_1 \mid X_2) p(X_3 \mid X_2) p(X_4 \mid X_2) p(X_5 \mid X_2) \cdot p(X_6 \mid X_5)$

## Problem 5 [20 points]

In this problem we will derive belief propagation for parameter estimation and prediction in a generative model with continuous variables.

Let $(X_i)_{i=1}^N$ be a sequence of real valued random variables whose values we observe as data. Let the sequence $\mathcal{D} = (x_i)_{i=1}^N$ represent the observed values of these random variables, i.e. $\mathcal{D}$ is our data. Our task is to predict the value of $X_{N+1}$. Imagine that we know special domain knowledge about the dynamics of the process that generated $\mathcal{D}$ in the sense that we know that the observed values are actually an interpolation of a sequence of hidden random variables $(H_i)_{i=0}^N$ whose values $(h_i)_{i=0}^N$ are generated as follows: $h_0$ comes from the gaussian distribution $\mathcal{N}(\mu_0/\lambda_0, \sigma^2/\lambda_0^2)$. $h_i$ comes from the conditional gaussian disrtibution $\mathcal{N}(\kappa h_{i-1}, \sigma^2)$ and $x_i|h_{i-1}, h_i$ comes from $\mathcal{N}(\lambda h_{i-1} + (1-\lambda)h_i, \sigma^2)$. This process can be modelled via the factor graph in figure 3 with the following factors:

$$\xi_0(h_0) = \frac{\lambda_0^2}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\lambda_0 h_0 - \mu_0)^2}{2\sigma^2}\right) \tag{4}$$

$$\xi_i(x_i, h_i, h_{i-1}) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_i - (\lambda h_{i-1} + (1-\lambda)h_i))^2}{2\sigma^2}\right) \exp\left(-\frac{(h_i - \kappa h_{i-1})^2}{2\sigma^2}\right) \tag{5}$$

$$\forall i \in [1, \dots, N]$$

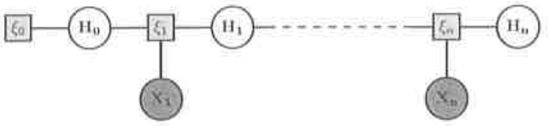


**Figure 3**: The Factor Graph of Observations

(a) Prove that $p(x_i|h_{i-1}) = \mathcal{N}((\lambda + \kappa(1-\lambda))h_{i-1}, \sigma^2(1 + (1-\lambda)^2))$.

You may find the following identities useful:

$$\int_{\mathbb{R}} \exp\left(-\frac{ax^2 + 2bx + c}{2\sigma^2}\right) dx = \sqrt{\frac{2\pi\sigma^2}{a}} \exp\left(-\frac{c - b^2/a}{2\sigma^2}\right), \qquad a > 0 \tag{6}$$

$$\int_{\mathbb{R}} \exp\left(-\frac{(ax-b)^2 + (cx-d)^2}{2\sigma^2}\right) dx = \sqrt{\frac{2\pi\sigma^2}{a^2 + c^2}} \exp\left(-\frac{(bc - ad)^2}{2\sigma^2(a^2 + c^2)}\right) \tag{7}$$

$$\int_{\mathbb{R}} \exp\left(-\frac{(ax-b)^2}{2\sigma_1^2} - \frac{(cx-d)^2}{2\sigma_2^2}\right) dx = \sqrt{\frac{2\pi}{a^2/\sigma_1^2 + b^2/\sigma_2^2}} \exp\left(-\frac{(bc - ad)^2}{2(a^2\sigma_2^2 + c^2\sigma_1^2)}\right) \tag{8}$$

$\xi_i(x_i, h_i, h_{i-1}) = p(x_i, h_i, h_{i-1}) = p(x_i | h_i, h_{i-1}) p(h_i | h_{i-1}) p(h_{i-1})$

$= p(x_i | h_i, h_{i-1}) p(h_i | h_{i-1})$ where $p(h_{i-1}) = 1$

So we integrate over $h_i$ to get that:

$\int_{h_i} p(x_i | h_i, h_{i-1}) p(h_i | h_{i-1}) dh_i = \int_{h_i} \xi_i(x_i, h_i, h_{i-1}) dh_i$

$$\int_{h_i} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_i-(\lambda h_{i-1}+(1-\lambda)h_i))^2}{2\sigma^2}\right) \exp\left(-\frac{(h_i-kh_{i-1})^2}{2\sigma^2}\right) dh_i$$

Now let $a=(1-\lambda)$, $b=\lambda h_{i-1}-x_i$, $c=1$, $d=kh_{i-1}$, we can use identity (7) to yield the result

$$\frac{1}{2\pi\sigma^2}\cdot\sqrt{\frac{2\pi\sigma^2}{a^2+c^2}} \exp\left(-\frac{(bc-ad)^2}{2\sigma^2(a^2+c^2)}\right) = \sqrt{\frac{2\pi\sigma^2}{(1-\lambda)^2+1}} \exp\left(-\frac{((\lambda h_{i-1}-x_i)-(1-\lambda)kh_{i-1})^2}{2\sigma^2((1-\lambda)^2+1}\right)$$

$$= \frac{1}{\sqrt{2\pi\sigma^2((1-\lambda)^2+1)}} \exp\left(-\frac{(x_i-\lambda+k(1-\lambda))h_{i-1})^2}{2\sigma^2(((1-\lambda)^2+1)}\right)$$

Which is $\mathcal{N}(\lambda+k(1-\lambda)h_{i-1}, \sigma^2(((1-\lambda)^2+1))$

(b) Prove that $p(h_i|x_i) = \mathcal{N}(\frac{\kappa x_i}{\lambda+(1-\lambda)\kappa}, \frac{\sigma^2(\lambda^2+\kappa^2)}{(\lambda+(1-\lambda)\kappa)^2})$.

$$p(h_i|x_i) = \frac{\int_{h_{i-1}} p(x_i, h_i|h_{i-1})p(h_{i-1})}{\int_{h_{i-1}} p(x_i|h_{i-1})p(h_{i-1})} = \frac{p(x_i,h_i)}{p(x_i)} = \frac{\int_{h_{i-1}} p(x_i,h_i,h_{i-1})}{\int_{h_{i-1}} p(x_i|h_{i-1})}$$

NUMERATOR: $\int \exp\left(-\frac{(x_i-(\lambda h_{i-1}+(1-\lambda)h_i)^2}{2\sigma^2}\right) \exp\left(\frac{h_i-kh_{i-1})^2}{2\sigma^2}\right) dh_{i-1}$

once again use (7) with $a=\lambda$, $b=x_i+(1-\lambda)h_i$ $c=k$, $d=h_i$ to get

$$\sqrt{\frac{2\pi\sigma^2}{\lambda^2+k^2}} \exp\left(-\frac{(kx_i+k(1-\lambda)h_i-\lambda h_i)^2}{2\sigma^2(\lambda^2+k^2)}\right)$$

DENOMINATOR: Integrate the result from part a and use identity (7) w/ $a=-k(1-\lambda)+\lambda$, $b=-x_i$, $c=d=0$, $\sigma^2=\sigma^2((1-\lambda)^2+1)$ to get $\sqrt{\frac{2\pi\sigma^2}{(-k(1-\lambda)+\lambda)^2}}$

$p(h_i|x_i) = \sqrt{\frac{(-k(1-\lambda)+\lambda)^2}{\lambda^2+k^2}} \exp\left(-\frac{(kx_i+h_i[k(1-\lambda)-\lambda])^2}{2\sigma^2(\lambda^2+k^2)}\right)$ which simplies to the above

(c) Prove that $p(x_{i+1}|x_i) = \mathcal{N}(\kappa x_i, \sigma^2(1+\lambda^2+(1-\lambda)^2+\kappa^2))$.

Now consider the scenario where the values of $\lambda_0, \mu_0, \sigma$ are known to us apriori but we do not know the multiplicative rate of change $\kappa$ or the interpolation factor $\lambda$. A practical scheme that we can use in such a situation – where we want to make a prediction based on a model but we do not know all the system parameters – is the following maximum likelihood based method.

**Step 1** Formulate the probability of the data $p(\mathcal{D}) = \{X_i\}_{i=1}^N$ as a function of the unknown parameters $\kappa, \lambda$.

**Step 2** Maximize the probability of data to compute $\widehat{\kappa}^{\mathrm{MLE}}, \widehat{\lambda}^{\mathrm{MLE}}$.

**Step 3** Use $\widehat{\kappa}, \widehat{\lambda}$ (omitting the MLE superscript) to compute the distribution of $H_n$ and $X_{N+1}$ and ultimately predict $X_{N+1}$.

Let $\mathcal{H}$ denote a sequence of values of the hidden random variables. As we saw in the class the sum-product algorithm (also called Belief Propagation), is an algorithm for efficiently computing $p(\mathcal{D})$. According to our model:

$$p(\mathcal{D}) = \int p(\mathcal{D}, \mathcal{H}) = \frac{1}{Z} \int_{h_0} \cdots \int_{h_N} \xi_0(h_0) \prod_{i=1}^{N} \xi_i(x_i, h_i, h_{i-1})$$

In the following questions you will compute the messages $\{m_i\}_{i=0}^N$, where message $m_i$ is sent from factor $\xi_i \to H_i$ as shown in figure 4. Assume that $H_N$ is chosen as the root node, and you run the belief propagation algorithm to compute messages $(m_i)_{i=0}^N$.

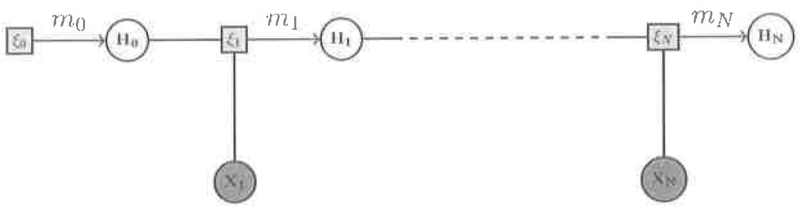


**Figure 4**: The Meessages from $\xi_i$ to $H_i$.

(d) Express the probability of data, $p(\mathcal{D})$, in terms of $m_N$.

$$p(D) = \int p(D,H) = \tfrac{1}{2} \int_{H_0} \cdots \int_{H_N} \mathcal{E}(h_0) \prod \mathcal{E}_i (x_i, h_i, h_{i-1})$$

$$= \tfrac{1}{2} \left[ \int_{h_N} \cdots \left[ \int_{h_0} \mathcal{E}(x_i, h_i, h_0) dh_0 \right] \right]$$

$$= \tfrac{1}{2} \left[ \int_{h_N} \cdots \text{message } m_N \right.$$

$$= \tfrac{1}{2} \int_{h_n} \left[ \int_{h_{N-1}} \mathcal{E}_n (x_n, h_n, h_{n-1}) \cdots \left[ \int_{h_0} \mathcal{E}_1 (x_1, h, h_0) dh_0 \right] \right.$$
$$dh_{n-1}$$

$$= \tfrac{1}{2} \int_{h_N} m_N dh_N$$

(e) What is $m_0 : \mathbb{R} \to \mathbb{R}_+$ ?

$m_0 = 1$

(f) Prove that

$$m_i = a_i \exp -\frac{(\lambda_i h_i - \mu_i)^2}{2\sigma^2}, \qquad \forall i \in [0, \dots, N].$$

Is $m_N$ a function of $\lambda, \kappa$? You do not need to fully simplify the expression.

(g) Assume that we maximized the value of $p(\mathcal{D})$ with respect to $\lambda, \kappa$ to get the MLE estimates $\widehat{\kappa}, \widehat{\lambda}$. What will be the mean of the distribution of $H_n$ given $\widehat{\kappa}, \widehat{\lambda}$ and $\mathcal{D}$? How will you compute the distribution of $X_{N+1}$ from $m_N$?

(h) Can you think of a way to use the expression for $p(x_{i+1}|x_i)$ to estimate $\lambda, \kappa$ from data? Is this method different from the maximum likelihood estimator that you derived above?

# PS3-sentiment

April 10, 2017

```
In [1]: from __future__ import absolute_import
        from __future__ import division
        from __future__ import print_function

        from utils import *

        import numpy as np
        import cvxopt
        from cvxopt import matrix

        cvxopt.solvers.options['show_progress'] = False

        %matplotlib inline

In [2]: """
        Get the preprocessed data:
        X and y are dicts, has keys train, val and test. keys gives you what word

        You can also get bigram features if you call preprocess(bigram = True)`
        Another optional argument mincount specifies the frequency cutoff for inclu
        dictionary.

        """

        X, y, keys = preprocess()

Feature size:  1464


In [3]: class Kernel(object):
            """
            A class containing all kinds of kernels.
            Note: the kernel should work for both input (Matrix, vector) and (vecto
            """
            @staticmethod
            def linear():
                def f(x, y):
                    return np.dot(x, y)
                return f

            @staticmethod
            def gaussian(gamma):  # we use the commonly used name, although it's no
                def f(x, y):
                    exponent = - gamma * np.linalg.norm((x-y).transpose(), 2, 0) **
                    return np.exp(exponent)
                return f

            @staticmethod
            def _poly(dimension, offset):
                def f(x, y):
                    return (offset + np.dot(x, y)) ** dimension
                return f

            @staticmethod
            def inhomogenous_polynomial(dimension):
                return Kernel._poly(dimension=dimension, offset=1.0)

            @staticmethod
            def homogenous_polynomial(dimension):
                return Kernel._poly(dimension=dimension, offset=0.0)

            @staticmethod
            def hyperbolic_tangent(kappa, c):
                def f(x, y):
                    return np.tanh(kappa * np.dot(x, y) + c)
                return f

In [4]: class SVM(object):
            def __init__(self, kernel, c):
                """
                Build a SVM given kernel function and C

                Parameters
                ----------
                kernel : function
                    a function takes input (Matrix, vector) or (vector, vector)
                c : a scalar
                    balance term

                Returns
                -------
                """
                self._kernel = kernel
                self._c = c

            def fit(self, X, y):
```

```
        """
        Fit the model given data X and ground truth label y

        Parameters
        ----------
        X : 2D array
            N x d data matrix (row per example)
        y : 1D array
            class label

        Returns
        -------
        """
        # Solve the QP problem to get the multipliers
        lagrange_multipliers = self._compute_multipliers(X, y)
        # Get all the support vectors, support weights and bias
        self._construct_predictor(X, y, lagrange_multipliers)

    def predict(self, X):
        """
        Predict the label given data X

        Parameters
        ----------
        X : 2D array
            N x d data matrix (row per example)

        Returns
        -------
        y : 1D array
            predicted label
        """
        result = np.full(X.shape[0], self._bias) # allocate

        #YOUR CODE HERE
        N = X.shape[0]
        """
        func = (self._weights) * self._support_vector_labels
        prediction = [np.sum(func) * [self._kernel(X[i], sv) for sv in sel
                      + self._bias for i in range(N)]

        result = np.sign(prediction)
        """
        result = [np.sum(self._weights * self._support_vector_labels * [sel
                       for sv in self._support_vectors]) + self._bias for
        return np.sign(result)

    def _kernel_matrix(self, X):
        """
        Get the kernel matrix.

        Parameters
        ----------
        X : 2D array
            N x d data matrix (row per example)

        Returns
        -------
        K : 2D array
            N x N kernel matrix
        """
        N, d = X.shape
        K = np.zeros((N, N))
        for i, x_i in enumerate(X):
            for j, x_j in enumerate(X):
                K[i, j] = self._kernel(x_i, x_j)
        return K

    def _construct_predictor(self, X, y, lagrange_multipliers):
        """
        Given the data, label and the multipliers, extract the support vect

        Parameters
        ----------
        X : 2D array
            N x d data matrix (row per example)
        y : 1D array
            class label
        lagrange_multipliers: 1D array
            the solution of lagrange_multiplier

        Fills in relevant variables: model bias and weights (alphas), and c

        -------
        """
        support_vector_indices = \
            lagrange_multipliers > 1e-5

        print("SV number: ", np.sum(support_vector_indices))

        support_multipliers = lagrange_multipliers[support_vector_indices]
        support_vectors = X[support_vector_indices]
        support_vector_labels = y[support_vector_indices]

        """
        Get the bias term (w_0)
        """
        #YOUR CODE HERE
        bias = 0
        for i in range(len(support_vectors)):
            bias += support_vector_labels[i] - np.sum(support_multipliers *

        self._bias=bias/len(support_vectors) if bias != 0 else 0
        self._weights=support_multipliers
        self._support_vectors=support_vectors
        self._support_vector_labels=support_vector_labels

    def _compute_multipliers(self, X, y):
        """
        Given the data, label, solve the QP program to get lagrange multip

        Parameters
        ----------
        X : 2D array
            N x d data matrix (row per example)
        y : 1D array
            class label

        Returns
        lagrange_multipliers: 1D array
        -------
        """
        N, d = X.shape
        K = self._kernel_matrix(X)
        """
        The standard QP solver formulation:
        min 1/2 x^T H x + f^T x
        s.t.
        Ax <=  a
        Bx = b
        """
        N = X.shape[0]
        H = matrix(np.dot(np.dot(np.diag(y), K), np.diag(y).T), tc='d')
        f = matrix(np.array([-1]*N), tc='d')

        A = matrix(np.array([[-1]*N + [1]*N,]*N), tc='d').trans()
        A = matrix(np.vstack((-np.identity(N), np.identity(N))), tc='d')
        a = matrix(np.array([0]*N+[self._c]*N), tc='d')

        B = matrix(np.array([y,]*N), tc='d').trans()
        B = matrix(np.reshape(y, (1, N)), tc='d')
        b = matrix(np.array([0]*N), tc='d')
        b = matrix(0, tc='d')
```

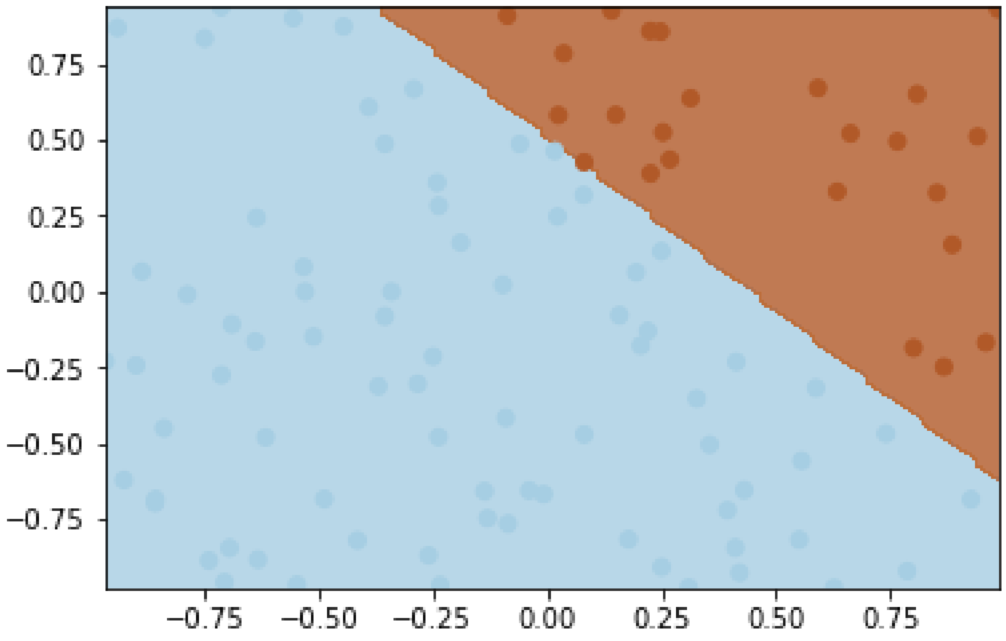
```
                # call the QP solver
                solution = cvxopt.solvers.qp(H, f, A, a, B, b)

                # Lagrange multipliers (the unknown vector 'x' is our alphas)
                return np.ravel(solution['x'])
```

Test the trainer. The following code would generate data which the grounth truth split is x+y = 0.5.

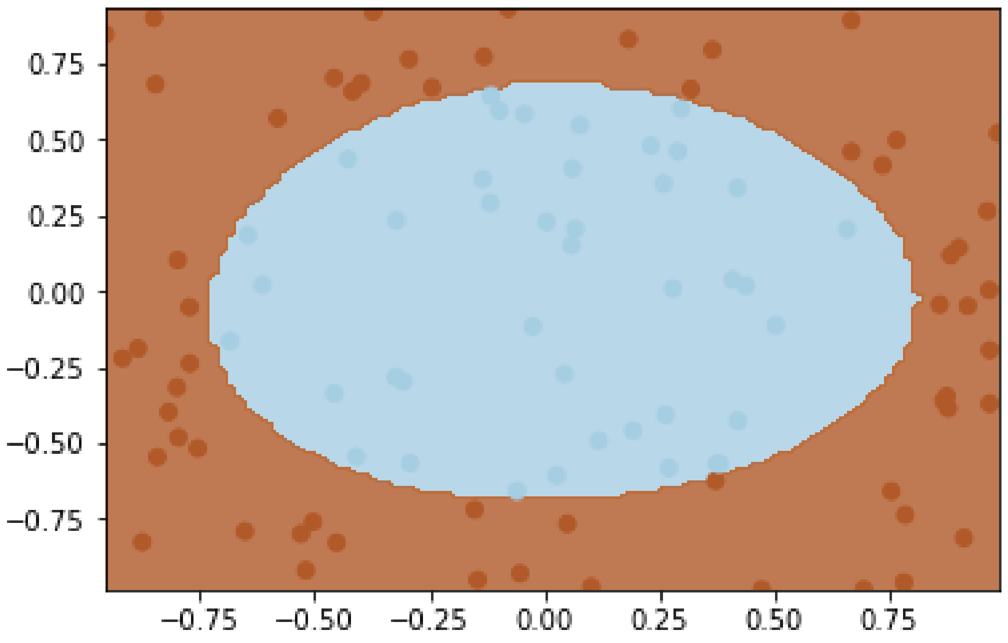
```
In [5]: clf = SVM(Kernel.linear(), 100)
        test_linear_SVM(clf, 100)
```

```
SV number:  5
```



```
In [6]: clf = SVM(Kernel.gaussian(1), 100)
        test_rbf_SVM(clf, 100)
```

```
SV number:  12
```



If you think your code is correct, then we can move to the real problem. Below are some code examples; you will need to fill in some details, and extend these to any experiments you want to run.

```
In [7]:  for C in [0.01, 0.1, 0.5, 1, 10]:
             clf = SVM(Kernel.linear(), C)
             clf.fit(X['train'], y['train'].astype('double'))
             print("C = ", C)
             y_hat = clf.predict(X['train'])
             print("Acc on train: ", np.mean(y_hat == y['train']))
             y_hat = clf.predict(X['val'])
             print("Acc on val: ", np.mean(y_hat == y['val']))
```

```
SV number:  2052
C =  0.01
Acc on train:  0.797477477477
Acc on val:  0.722
SV number:  1641
C =  0.1
Acc on train:  0.895855855856
Acc on val:  0.732
SV number:  1366
C =  0.5
Acc on train:  0.942342342342
Acc on val:  0.718
SV number:  1268
C =  1
```

```
Acc on train:  0.956036036036
Acc on val:  0.724
SV number:  1040
C =  10
Acc on train:  0.995315315315
Acc on val:  0.71
```

Choose the best C, and predict the label for test data.

```
In [8]: C = 0.1
        clf = SVM(Kernel.linear(), C)
        clf.fit(X['train'], y['train'].astype('double'))
        y_hat = clf.predict(X['test'])
        save_submission('sub_linear.csv', y_hat)
```

```
SV number:  1641
```

RBF (Gaussian) kernel SVM

```
In [8]: for C in [0.001, 0.01, 0.1, 0.5, 10, 100]:
            for gamma in [0.1, 0.5, 10]:
                clf = SVM(Kernel.gaussian(gamma), C)
                clf.fit(X['train'], y['train'].astype('double'))
                print("C = ", C)
                print("gamma = ", gamma)
                y_hat = clf.predict(X['train'])
                print("Acc on train: ", np.mean(y_hat == y['train']))
                y_hat = clf.predict(X['val'])
                print("Acc on val: ", np.mean(y_hat == y['val']))
```

```
SV number:  2352
C =  0.001
gamma =  0.1
Acc on train:  0.636396396396
Acc on val:  0.67
```

```
        ---------------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last

        <ipython-input-8-c36f67467956> in <module>()
          2     for gamma in [0.1, 0.5, 10]:
          3         clf = SVM(Kernel.gaussian(gamma), C)
    ----> 4         clf.fit(X['train'], y['train'].astype('double'))
```

```
      5         print("C = ", C)
      6         print("gamma = ", gamma)


    <ipython-input-4-1ee906a878f9> in fit(self, X, y)
     32         """
     33         # Solve the QP problem to get the multipliers
---> 34         lagrange_multipliers = self._compute_multipliers(X, y)
     35         # Get all the support vectors, support weights and bias
     36         self._construct_predictor(X, y, lagrange_multipliers)


    <ipython-input-4-1ee906a878f9> in _compute_multipliers(self, X, y)
    141         """
    142         N, d = X.shape
--> 143         K = self._kernel_matrix(X)
    144         """
    145         The standard QP solver formulation:


    <ipython-input-4-1ee906a878f9> in _kernel_matrix(self, X)
     83         for i, x_i in enumerate(X):
     84             for j, x_j in enumerate(X):
---> 85                 K[i, j] = self._kernel(x_i, x_j)
     86         return K
     87


    <ipython-input-3-1aa570b6ff1f> in f(x, y)
     13     def gaussian(gamma):  # we use the commonly used name, although it'
     14         def f(x, y):
---> 15             exponent = - gamma * np.linalg.norm((x-y).transpose(), 2, (
     16             return np.exp(exponent)
     17         return f


    C:\Users\tmelo1\Anaconda2\lib\site-packages\numpy\linalg\linalg.pyc in norm
   2157         elif ord is None or ord == 2:
   2158             # special case for speedup
-> 2159             s = (x.conj() * x).real
   2160             return sqrt(add.reduce(s, axis=axis, keepdims=keepdims))
   2161         else:


    KeyboardInterrupt:
```

```
In [ ]: clf = SVM(Kernel.gaussian(BEST GAMMA), BEST C)
```

```
        clf.fit(X['train'], y['train'].astype('double'))y_hat = clf.predict(X['test
        save_submission('sub_rbf.csv', y_hat)
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```