6.034 Quiz 4 Review
Fall 2009

Topics:

- **SVMS: Classifier & Kernels**
- **Boosting**
- **Bayes Nets (Probability)**

SVMs:

The problem: given a set of training points, find the *maximum margin classifier* which will assign a value to any new point: $h(\vec{x}) = \{+1, -1\}$
Constraints: maximize the "margin" or the "width of the road" separating positive (label = +1) and negative (label = -1) training points.

Tools: (Remember $K(\vec{u}, \vec{v}) = \vec{\phi}(\vec{u}) \cdot \vec{\phi}(\vec{v})$)

Formulas:	If you know α's	If you know $\vec{w}$
Classifier	$h(\vec{x}) = sign(\sum_i [\alpha_i y_i K(\vec{x_i}, \vec{x})] + b)$	$h(\vec{x}) = sign(\vec{w} \cdot \vec{x} + b)$
Decision boundary	$\sum_i [\alpha_i y_i K(\vec{x_i}, \vec{x})] + b = 0$	$\vec{w} \cdot \vec{x} + b = 0$
Positive gutter	$\sum_i [\alpha_i y_i K(\vec{x_i}, \vec{x})] + b = 1$	$\vec{w} \cdot \vec{x} + b = 1$
Negative gutter	$\sum_i [\alpha_i y_i K(\vec{x_i}, \vec{x})] + b = -1$	$\vec{w} \cdot \vec{x} + b = -1$
Width of the road	n/a	$\frac{2}{\Vert\vec{w}\Vert}$ where $\Vert\vec{w}\Vert = \sqrt{\sum_i w_i^2}$

The LaGrangian:

- To maximize the margin, we minimize $\frac{1}{2}\Vert\vec{w}\Vert^2$ subject to the constraints $y_i(\vec{w} \cdot \vec{x} + b) \geq 1$
- The resulting LaGrange multiplier equation to optimize is: $L = \frac{1}{2}\Vert w\Vert^2 - \sum_i \alpha_i (y_i(\vec{w} \cdot \vec{x} + b) - 1)$
- Sum of all alphas (support vector weights) with their signs should add to 0. This equation comes from $\frac{\partial L}{\partial b} = 0$

Equation 1: $\sum_i \alpha_i y_i = 0$

Note that $y_i \in \{-1, +1\}$ and $\alpha_i = 0$ for non-support vectors.

- Sum of the vector product of α_i, y_i and $\vec{x_i}$ is equal to $\vec{w}$. This equation comes from solving $\frac{\partial L}{\partial w} = 0$

Equation 2: $\sum_i \alpha_i y_i \vec{x_i} = \vec{w}$

Methods:

- Linear kernel or no kernel?
    - Eyeball decision boundary (remember that your equation could be a scalar multiple of the actual h(x))
    - Scale using either width of the road or an equation for a gutter
    - Given scaled $\vec{w}$, use Equation 1 and 2 derived from LaGrangian to solve for alphas.

  OR
  - Calculate the kernel values for all pairs of points
  - Use the alpha version of h(x) to set up system of equations to solve for alphas and b
  - Use the LaGrangian equations (Equations 1, 2) to find the weight vector w
- Complicated kernel?
    - Find $\phi(x)$ given the kernel
    - Use $\phi(x)$ to transform the training data into a different space
    - Find the (linear) decision boundary in the new space
    - Convert the decision boundary into regular space

Practice Problem: 2006 final

Kernels:

- Linear Kernel: $K(\vec{u},\vec{v}) = \vec{u} \cdot \vec{v}$
- Polynomial Kernel: $K(\vec{u},\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^n$
- Radial Basis Kernel: $K(\vec{u},\vec{v}) = exp\left(-\frac{\|\vec{u} - \vec{v}\|^2}{2\sigma^2}\right)$

Changing n:
Changing sigma:

Practice Problem: 2001 final

Boosting:

The problem: given a set of training points, find a *strong classifier* that classifies all of the training data correctly as a *weighted sum of weak base classifiers.*

- $H(\vec{x}) = sign(\sum_{i}^{s} \alpha_i h_i(\vec{x})) = \{+1,-1\}$
- $h(\vec{x}) = sign($ a **weak** classifier - e.g. a horizontal or vertical decision line $) = \{+1,-1\}$

Method: Adaboost:

1. Recalculate the weights $\vec{w_i}$ on the data points

- Initialize to $\vec{w_i} = \frac{1}{\#samples}$
- Use the following formula for weight updates:

  $w_i^{s+1} = \frac{1}{2} \cdot \frac{w_i^s}{1-\epsilon^s}$ for correctly classified examples.

  $w_i^{s+1} = \frac{1}{2} \cdot \frac{w_i^s}{\epsilon^s}$ for misclassified samples.
- To check your answers: $\sum_{wrong} w_i^{s+1} = \frac{1}{2}$

2. Find a weak base classifier $h_i(\vec{x})$ that minimizes the error: $\epsilon^s = \sum_{incorrect} w_i$
3. Calculate the alpha for this classifier: $\alpha_s = \frac{1}{2} \ln\left(\frac{1-\epsilon^s}{\epsilon^s}\right) = \ln\left(\frac{1-\epsilon^s}{\epsilon^s}\right)^{\frac{1}{2}}$
4. Are you done? If not, go back to step 1. Otherwise, calculate the strong classifier (you did remember all the alpha's and h's, right?)

Practice Problem: 2004f, part B

Optional: Mark's **Keep-the-numerator-constant** method, a faster method for simulating Boosting weight updates:

1. Keep all weights in the form of $w_i^s = \frac{n_i}{d}$ where the denominator d is the same to all weights.
2. Circle the data points that are misclassified.
3. Compute the new denominator for (the circled) misclassified points:

$$d'_{wrong} = 2 \sum_{wrong} n_i$$

or sum of the numerators times two.

4. Compute the new denominator for (uncircled) correct points:

$$d'_{correct} = 2 \sum_{correct} n_i$$

5. New weights w_i^{s+1} are simply the old numerator divided by the updated denominators found in 3, and 4.
6. Alter the weight: w_i^{s+1} numerator and denominators such that the denominator is again the same for all weights.
7. Repeat all above steps until done.

CAUTION do not attempt this method unless you absolutely know what you are doing!

Bayes nets:

- A compact way to represent the Joint Distribution of a set of Random Variables (Random variables can take on values with different probabilities.)
- A way to model the independence relationships between Random Variables.

Tools and Formulas:

Next to each node, you have conditional probability tables (**CPT**), these represent the conditional probability of the underlying R.V. conditioned on its parents. The probabilities in each row must sum to **1** (though we may leave off a column if its value is implied from the other column(s)).

How to compute a **specific variable setting** or JOINT probability? For any Bayes Net:

$$p(V_1 \ldots V_n) = \prod_{i=1}^{n} p(V_i | Parents(V_i))$$

Forward Inference: (assuming the following network: Z -> X <- Y):

$$P(X|Y) = P(X|Y,Z=T)P(Z=T) + P(X|Y,Z=F)P(Z=F)$$

Backward Inference: Using Bayes Rule when the network configuration is Y -> X:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

The general three variable Bayes Rule:

$$P(X|Y,Z) = \frac{P(Y,Z|X)P(X)}{P(Y,Z)} = \frac{P(Y|Z,X)P(X|Z)}{P(Y|Z)}$$

What about the probability of **one random variable**? Just sum over all the variables we don't care about ("marginalization" / "exact inference").

$$P(V_i) = \sum_{V_{i \neq j}} P(V_1, V_j, \ldots, V_n) = \sum_{V_{j \neq i}} P(V_i | parent(V_i))$$

Here we can use a summation trick. If summation produces terms like: $\sum_{V_i} P(V_i|V_j) = 1$ (after reshuffling, you can eliminate them).

Short-cut: ***Any node that is not an "ancestor" of a variable of interest (anything in the probability we are computing) we can ignore in the summation.***

Explaining Away: From the recitation example, we know that a sculpture **S** can be caused by two events, a hack on campus **H** or an art exhibit **A**. We showed in tutorial that if we know that an art-exhibit is occurring, and we see a statue, then the probability of there being a hack declines versus not knowing about the art-exhibit.
i.e: P(H=T | S=T, A=T,) < P(H=T | S=T).

As an exercise, try computing the probability of seeing a hack given that we see a statue, and knowing that an art-exhibit is NOT occuring i.e: What is P(H=T | S=T, A=F)? is it bigger or smaller than P(H=T|S=T)? This effect due to competing causes in Bayes Nets is commonly known as "EXPLAINING AWAY".

Rules for independence (optional, good to know but not required):
D-Separation (arrows correspond to node connections in the Bayes Net):

X -> Y <- Z   X and Z are independent if we **know nothing** about the value of Y.
X <- Y -> Z   X and Z are independent if we **know** the value of Y.
X -> Y -> Z   X and Z are independent if we **know** the value of Y.

Independence implies that
P(X | Y) = P(X) if X is independent of Y.   Also   Independence implies: P(X, Y) = P(X)P(Y)

Naive Bayes: *a CLASSIFICATION method*

- There is a Root node (class variable **C**).
- It generates n feature (**F_i**) nodes (observed feature variables).
- The root node fans out to each of the feature nodes, via an outgoing arrow.

Basic idea:
1) Find good estimates of the **likelihood** of a class generating a features P(f | c).
2) Then use Bayes Rule to compute the reverse: i.e. **P(c | features)**

$$P(C|F_1...F_n) = \frac{P(F_1 \cdot\cdot F_n|C)P(C)}{P(F_1..F_n)}$$

3. Assume that features given the class are independent. Namely:

$$P(F_1...F_n|C) = P(F_1|C)P(F_2|C)...P(F_n|C) = \prod_{i=1}^{n} P(F_i|C)$$

Generally, our goal is to find the most probable class, or argmax C. So we can ignore the denominator (the normalization constant P(F_1...F_n), because it is common to all classes C.

$$\arg\max_c P(C|F_1...F_n) = \arg\max_c \frac{P(C)\prod_{i=1}^{n} P(F_i|C)}{P(F_1..F_n)} = \arg\max_c P(C)\prod_{i=1}^{n} P(F_i|C)$$

We estimate P(C) and P(F|C) from data we observe/collect.

Practice Problem: Naive Bayes

	Pyro	ForeignLang	GoodShape	# surveyed
East Campus	8/10	1/10	3/10	10 10/30
West Campus	3/10	6/10	3/10	10 10/30
FSILG	1/10	3/10	8/10	10 10/30

Question: Where would a new student who loves foreign languages (and abhors everything else) be classified if they filled in their incoming survey as follows:
Pyro = False
ForeignLang = True
GoodShape = False