CS5350: Machine Learning *Due December 16, 2011*

HW5: Assorted Topics

1. Consider the following semi-supervised variant of a support vector machine:

$$\min_{\boldsymbol{w},b} \quad \tfrac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 \tag{1}$$

$$\text{s.t.} \quad y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad (1 \leq n \leq L) \tag{2}$$

$$\left|\boldsymbol{w}^\top \boldsymbol{x}_n + b\right| \geq 1 \quad (L+1 \leq n \leq L+U) \tag{3}$$

where we have L labeled examples $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_L, y_L)$ and U unlabeled examples $\boldsymbol{x}_{L+1}, \dots, \boldsymbol{x}_{L+U}$.

(a) Construct the Lagrange formulation (yes, again!) for this model. (*10 points*)

Begin by putting it in primal form:

$$\frac{||\boldsymbol{w}||^2}{2} + \sum \alpha_m(1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)) + \sum \beta_n(1 + \boldsymbol{w}^\top \boldsymbol{x}_n + b)$$

Convert to dual:

$$\frac{||\boldsymbol{w}||^2}{2} + \frac{\partial}{\partial \boldsymbol{w}} \sum \alpha_m(1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)) + \frac{\partial}{\partial \boldsymbol{w}} \sum \beta_n(1 + \boldsymbol{w}^\top \boldsymbol{x}_n + b)$$

$$\sum \alpha_m(y_n x_n) + \sum \beta_n x_n = 0$$

$$w + \sum \alpha_m(y_n x_n) + \sum \beta_n x_n = 0$$

$$\sum_1^L \alpha_m(y_n x_n) - \sum_{L+1}^{L+U} \beta_n x_n = w$$

$$L_D = \frac{1}{2}\sum_1^L \alpha_m(y_n x_n) - \frac{1}{2}\sum_{L+1}^{L+U} \beta_n x_n = w$$

(b) In the Lagrange formulation, there is a term for the hinge loss on the labeled data points. There is also some term for a loss-like-thing on the unlabeled data points. What does this loss look like (i.e., draw a picture); it should be clear from this picture that this is non-convex. (*10 points*)

It's going to have two local minima, looking sort of like a valley with a bulge in the middle. I'm not going to draw it, but Chappelle, Sindhwani, and Keerthi showed that you can actually reduce smoothing, and make it convex.

(c) (6350 only) What is the gradient of this formulation, as a function of $\boldsymbol{w}$ (don't bother doing the case for b)? If you were to run gradient descent, what (intuitively) would it do? (*10 points*)

2. Both semi-supervised learning and (pool-based) active learning make use of of unlabeled data, but in different ways. Explain how unlabeled data helps in each of these learning paradigms? (*10 points*)

The main idea behind pool-based **active learning** is that we can use some notion of relevance (*e.g.*, the mean squared error of the maximum likelihood estimates for different configurations of the weight vector $\boldsymbol{w}$) to rank data points, from which we select the top n to train on. Our goal is to use few labeled examples as we can *without* sacrificing the accuracy of our estimations. This is known in statistics as *optimal experimental design*.

Semi-supervised learning, on the other hand, depends on using some classifier to label the unlabeled data, with the n most confident classifications integrated into the training set. Various algorithms attempt this in different ways, but it is critically different in the approach it uses to pick the unlabeled data that is incorporated into the training set; in particular, SSL depends on the notion that the answers the classifier is most confident in are probably correct, where AL does not.

3. How are pool-based active learning and stream-based active learning different from each other? What are their pros and cons? In particular, compare them in terms of how well they can assess the informativeness of an example, and how efficient you expect their implementations to be?

   In the class, we also looked at density based methods that try to deal with the issue of outlier examples which may *seem* to be the informative points but they aren't actually. Do you think stream-based active learning can make use of density based methods? Why or why not? (*20 points*)

Stream-based active learning (SBAL) attempts to find the top n example points $\{\boldsymbol{x}_n\}$ by simply looking at each test point in turn, selecting them based on what is most useful given what we already have chosen. **Pool-based active learning** (PBAL) has access to all the unlabeled examples at a given point in time, and will rank the test points according to some sort of relevance.

SBAL will usually be pretty fast, since it considers one example at a time, and decides whether to include it or not. Some types of PBAL are slower than others, but it usually will not be extremely fast, since it tends to consider the context of the test points somewhat carefully (being that this is its big advantage over SBAL).

SBAL of course trades speed for certain empirical constraints—there are just some things that are either impossible or very difficult to know when considering one point at a time. As a very simple example, let's consider a linear model $h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\top \boldsymbol{x}^{(i)} + \epsilon$, where $\epsilon\ N(0, \sigma^2)$. Even if we assume no constraints whatsoever in our ability to pick the next test point to assimilate under SBAL (which is a *huge* assumption), when the test points are very close to each other, this noise will almost certainly have a large impact on the weights we develop for $\boldsymbol{w}$. This is something that SBAL just has no defense against in the general case. In contrast, PBAL has a number of things at its disposal that can help it against such issues. There are a number of situations in which SBAL will almost certainly do poorly simply because of a lack of data; PBAL, in contrast, has access to the entire pool of unlabeled data, and therefore has a distinct advantage in this regard.

As for density models in the SBAL case, the general answer is no. Density weighting mechanisms in the general case require data about the entire pool in order to produce a good sense of the informativeness of a test point; SBAL does not have access to the entire pool, and therefore in the general case is unable to use it.

4. What are the things that a Naive Bayes classification model needs to estimate? How is it different from discriminative models such as logistic regression? Also explain the difference between the estimation equations in the case when the features in the data are discrete valued vs when they are continuous valued? In both cases, also explain the intuitive meaning of the estimation equations? (*20 points*)

For some Y that determines a set of features $\boldsymbol{F}$, the Naive Bayes (NB) classifier estimates $P(Y|\boldsymbol{F})$. The first step is to estimate the parameters of the distribution of features, either using an assumed distribution, or a nonparametric model. In this class we really only care about the first type: in the discrete case, $P(Y|\boldsymbol{F} \propto P(Y)\prod_n P(F_n|Y)$; in the continuous case, it might be the case that (depending on the distribution) $P(Y|\boldsymbol{F}) = \frac{1}{\sqrt{2\pi\sigma_F^2}}e^{-(\frac{Y-\mu_F)^2}{2\sigma_F^2}}$. We do this by approximating the distribution that maximizes the likelihood of our observed $\boldsymbol{F}$: in the continuous setting, we find the variance σ^2 and mean μ_F of the data; in the discrete sense, we might estimate set each $P(F_n|Y)$ with $\frac{P(F_n,Y)}{P(Y)}$, which intuitively is the probability that some F_n occurs in a Y-labeled document divided by the probability that a given document is labeled Y.

The intuitive reason why we're doing this is that we don't actually know these distributions. To compensate, we simply model them in a way that seems reasonable. More precisely, one option is to model the empirical observations that we have. In the case of the discrete model, we do this by simply assigning observed probabilities to the likelihood $P(\boldsymbol{F}|Y)$; in the case of the continuous case, we might simply take the mean μ and variance σ^2 of the observed data and us that as parameters to, say, a Gaussian distribution.

One difference between Logistic Regression (LR) and NB classifiers is that LR actually trains on the data to estimate the conditional probability distribution $P(Y|X)$; in contrast, NB simply estimates the joint distribution of the data. LR is almost inherently more flexible because it models the conditional relationship of X and Y, where NB is an empirical model of the data's current manifestation as a joint distribution of X and Y. In practice, it tends to have a higher bias and lower variance than LR, which in particular means that LR tends to do better than NB on problems that do not require formulation of the joint distribution (*i.e.*, classification).

5. Structured Prediction Basics:

   (a) Structured Prediction algorithms such as MEMM, CRF, Structured-Perceptron/SVM, etc. use a feature function that depends on the outputs y as well (i.e. $\phi(x, y)$). What's the reason for using such a feature function?

   One of the major downsides to NB is the extremely strong independence assumption. Unfortunately, not everything we want to model is made up of completely independent random variables. By conditioning on y as well as x, we are able to supplant this strong independence assumption with a *conditional independence* assumption.

   (b) Explain how the Maximum Entropy Markov Model (MEMM) is different from the Hidden Markov Model (HMM)? In particular, how is the label distribution estimated in both of these? Why is the MEMM more suitable than HMM in capturing feature dependencies? Which one has the larger number of parameters to be estimated (and why)? What are the single-label analogue models for HMM and MEMM?

The basic idea behind Maxent is, out of a series of distributions, to pick the one that has the highest entropy. MEMMs track state using a binary vector, settling on the feature distribution with the highest entropy. Maxent does not assume independence of features, so they can overlap and even be correlated. Further, in HMMs, the current state depends only on the current observation, where in an MEMM, the current state may depend on both the previous and the current state. Combined, these things make MEMM definitively better at capturing feature dependencies.

The MEMM is trained pretty much exactly like the Maxent model, since the state transition function is given by the Maxent model: assuming we have a set of possible features and a method for telling whether a datum has a given feature, we can set the initial markov model with initially arbitrary transition probabilities. Eventually they will be proportional to the product of all features exhibited by the given state (according to the Maxent model). Then the model is tested against the training data to get the likelihood of the paths in the model. HMMs are trained similarly, but without the Maxent details (obviously).

Obviously MEMMs have more features to keep track of, on account of the binary vector that Maxent operates on. A single-label MEMM is linear regression, which forms the basis of Maxent. A single-label HMM is Naive Bayes, since it finds the set of parameters that maximizes the likelihood on some set of data.