CS771A: Machine Learning *Your Name & Roll no:*

CS771A Final Exam (Semester-I, 2016)

Instructions:

- This question paper is worth a total of 120 marks.
- Total time is 4 hours.
- Write your name and roll number at the top of this sheet as well as on the provided answer sheet.
- Important: The **true/false** and **fill-in-the-blanks** questions have to be answered on this question sheet itself. This question sheet along with the answer sheet must be submitted.
- To answer the other parts of the question paper, please use the provided answer sheet.
- For rough work, please use pages at the back of answer sheet. Extra sheets can be provided if needed.

Questions begin from the next page

1 True/False Problems (10 questions, 10 marks)

Please read each statement below carefully and then mark T/F in the square bracket. There are no partial marks for this section, so please do not explain your answer.

1. [**T**] As data dimensionality (number of features) increases, overfitting becomes more and more likely.
2. [**F**] 1-nearest neighbors classification/regression models have good generalization (accuracy on test data) since the learned functions are smooth.
3. [**F**] Probabilistic linear regression with Gaussian likelihood and a Laplace prior $p(\boldsymbol{w}) \propto \exp(-\gamma||\boldsymbol{w}||_1)$ on the weights is more robust to outliers than ridge regression. (Since the likelihood is still Gaussian, it won't help. The role of Laplace prior on the weight will only be in getting a sparse $\boldsymbol{w}$.)
4. [**T**] The VC dimension of 1-NN is larger than that of an RBF kernel SVM.
5. [**F**] Changing the hard margin SVM condition $y_n\boldsymbol{w}^\top\boldsymbol{x}_n \geq 1$ to $y_n\boldsymbol{w}^\top\boldsymbol{x}_n \geq \gamma$ for some positive scalar γ will change the learned decision boundary. (Only the scale will change; not the direction).
6. [**T**] Low model complexity can cause the model to have a large bias.
7. [**T**] Clustering using GMM with EM, in general, is computationally more expensive than K-means. (Note that GMM has to estimate many more parameters.).
8. [**F**] Unlike PCA, Kernel PCA is a method to *increase* the data dimensionality.
9. [**T**] A good classifier should pass through regions where the data density is low.
10. [**F**] Consider the recurrent neural net (RNN) defined as $\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$. Setting the transition matrix $\mathbf{U}$ to an identity matrix identity makes it equivalent to a standard feedforward neural net. (Setting $\mathbf{U} = 0$ will make that happen.).

2 Fill-in-the-blanks Problems (12 questions, 24 marks)

Please only write the final answer in the blank space. There are no partial marks for this section, so please do not show the steps.

1. For a weight vector $\boldsymbol{w} = [0.5, 0, 0, 0.5, 0, 0.5, 0.5, 0, 0, 0]$, the values of ℓ_0, ℓ_1, and ℓ_2 norms will be $\underline{4,\ 2,\ 1}$.
2. The prediction rule of a binary linear classification model with weights $\boldsymbol{w}$ for an input $\boldsymbol{x}$ is written as $y = \underline{\text{sign}(\boldsymbol{w}^\top\boldsymbol{x})}$.
3. Given a training set of N examples, the number of distance computation required at test time of KNN will be $\underline{N}$ for $K = 1$ and $\underline{N}$ for $K = 10$. (assuming a single test example; if you assumed more than one test example, the N will simply get scaled by that number)
4. Given training data $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, the SGD update for the weights of a logistic regression model, with an ℓ_2 regularizer $\lambda||\boldsymbol{w}||^2$ on the weights, will be $\underline{\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \eta(2\lambda\boldsymbol{w}_{old} - (y_n - \sigma(\boldsymbol{w}_{old}^\top\boldsymbol{x}_n))\boldsymbol{x}_n)}$.

5. Assuming D dimensional data, the total number of parameters to be learned in the M step of the EM algorithm for a K component Gaussian mixture model, with each Gaussian having a different spherical covariance matrix, will be $\underline{DK + K + (K-1) = DK + 2K - 1}$.

6. Number of parameters required to model a feedforward neural network which takes 100 dimensional data as input, has two hidden layers with 5 hidden units each, and learns a 20 class softmax classifier in the output layer, will be 100x5 + 5x5 + 5x20 = 625.

7. Consider PCA and Kernel PCA on an $N \times D$ data matrix $\mathbf{X}$ with $N > D$. On this data, between these two algorithms, PCA will be more efficient when computing the eigenvectors, and PCA will be more efficient when computing the embedding of each data point.

8. Convex loss functions are nice because local solution = global solution $\Rightarrow$ unique minima

9. Given the ground truth $\boldsymbol{y} = [1\ 0\ 0\ 1\ 1\ 0]$ and prediction $\hat{\boldsymbol{y}} = [1\ 0\ 1\ 0\ 1\ 1]$ of a model, the precision and recall scores of the model will be $2/4$, and $2/3$

10. For a linear regression model $y = \boldsymbol{w}^\top \boldsymbol{x}$, suppose S_{nm} denotes the graph based similarity between a pair of points $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$. For semi-supervised learning with graph-based regularization, assuming $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ are unlabeled, the *additional* regularizer term associated with this pair of points (assuming you are solving a minimization problem w.r.t. $\boldsymbol{w}$) will be $\underline{\lambda S_{nm}(\boldsymbol{w}^\top \boldsymbol{x}_n - \boldsymbol{w}^\top \boldsymbol{x}_m)^2}$.

11. ReLU based nonlinearity is cheaper to compute as compare to sigmoid/tanh based nonlinearity because computing the exponential in the sigmoid and tanh functions is expensive.

12. As per PAC analysis, for the non-zero training error case, to reduce the test error from 30% to 15%, the number of training examples needs to be increased roughly by at least 4 times.

3 Short Answer Problems (10 questions, 30 marks)

1. For two models with the same model complexity, but one trained using 5,000 training examples and the other trained using 100,000 training examples, which one is likely to have a lower training error? Which one is likely to have a lower test error? Also explain why.
Answer: 5,000 training examples: more likely to overfit $\Rightarrow$ lower training error. 100,000 training examples: less likely to overfit $\Rightarrow$ may have a higher training error BUT likely to have lower test error.

2. List 3 benefits of a Gaussian Mixture Model (GMM) over K-means clustering.
Answer: Produces soft cluster assignments, allows non-spherical clusters, allows unequal cluster sizes, can handle missing data, easy to extend to fully Bayesian formulations leading to many other benefits.. Okay, I should stop now! :)

3. Why is there no notion of generalization error for online learning algorithms? How can we then quantify the "true" (i.e., expected) error of an online learning algorithm?
Answer: Simply because there is no separate test set. The performance of online learning algorithms is measured in terms of the "regret" (in terms of cumulative mistakes, how much worse it is as compared to the "optimal" forecaster (the best "expert") in hindsight.

4. Why cross-validatation cannot be used for doing model selection in unsupervised learning? How do we then perform model selection for unsupervised learning? Give names of at least two methods.

Answer: Because unsupervised learning problems have no ground truth label (the "gold standard"). For doing model selection in these cases, we can use quantities that measure a trade-off between how well the model describes the (unlabeled) data, e.g., in terms of the negative log-likelihood or distortion, vs how many parameters the model has. Examples include AIC, BIC, etc.

5. In each iteration of the EM, the incomplete data log likelihood can only increase in the M step. Why?
Answer: Because the incomplete data log likelihood $\log p(\mathbf{X}|\Theta)$ only depends on the parameters Θ, which only change in the M step (E step only changes $\mathbf{Z}$ and Θ remains unchanged during this step).

6. For a mixture model, can we use a Gaussian distribution to model the mixture component ids $\boldsymbol{z}_n$ of each data point $\boldsymbol{x}_n$? If yes, why? If no, why not?
Answer: No, because the cluster ids are discrete variables, for which a discrete distribution such as multinomial/multinoulli) is more appropriate. Gaussian models real values and thus not appropriate.

7. For a probabilistic PCA or factor analysis model, is it possible to specify the number of latent factors K to be *larger* than the data dimension D? If yes, why? If no, why not?
Answer: Yes. Note that PPCA/FA does a linear transformation of the latent factors and parameter estimation doesn't rely on eigendecomposition (unlike standard PCA), so K can be larger than D. In fact, such models are used quite a lot in "overcomplete" dictionary learning or sparse coding.

8. Despite single hidden layer neural nets being capable of learning any function, why do we often prefer using deep neural nets with more than one hidden layer?
Answer: A single hidden layer neural net usually requires too many hidden nodes and may be hard to learn in practice. In contrast, having multiple hidden layer offers many benefits, e.g., (1) it can better, hierarchical, semantically meaningful, nonlinear representations, (2) the deep structure allows efficient reuse of computation and consequently general requires much fewer nodes overall.

9. If adding more training data is not helping you in getting the test error down, what does it tell you about your model? What should you be doing then to get a lower test error?
Answer: It suggests that the model is perhaps too simple and we should try going for a richer or more complex model, or add more features (which effectively makes the model richer).

10. Give an example of a pairwise loss function that can be used to handle class imbalance (name is fine if you don't remember the expression). Why this specific loss function helps in handling class imbalance?
Answer: Example include the AUC based loss function, which is sort of a pairwise 0-1 loss. In practice, this is optimized using pairwise hinge loss (refer the slides). The reason why such loss functions help in handling class imbalance is because we are no longer looking each example (positive/negative) independently in the loss function, but rather in pairs, and each pair contains one positive and one negative example. Therefore, the majority class doesn't overwhelm the minority class.

4 Medium Length Answer Problems (5 questions, 40 marks)

1. Recall that the final training error rate of AdaBoost after T rounds is $\text{err}(H_{final}) \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2)$, where $\epsilon_t = \frac{1}{2} - \gamma_t$ denotes error of the model learned in round t. Also assume that $\forall t$, we have $\gamma_t \geq \gamma$.
Given a total of N training examples, what will be the minimum number of rounds T AdaBoost needs to be run to achieve zero training error? Your answer needs to be in terms of N and γ only.
(**Hint:** Think of the smallest *non-zero* error the model can achieve on this training data)
Answer: Since $\gamma_t \geq \gamma$, we will have $\text{err}(H_{final}) \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2) \leq \exp(-2T\gamma^2)$. Now, the smallest non-zero training error is $1/N$ (making just a single mistake) and in order to achieve zero training error, we must have $\exp(-2T\gamma^2) < 1/N$. Taking log of both sides, we get $T > \frac{\log N}{2\gamma^2}$

2. Consider the following: (1) Kernel based supervised learning methods, e.g., SVM,; (2) Boosting methods, e.g., AdaBoost; and (3) Deep neural networks, e.g., feedforward nets. All of these can be seen

as learning a nonlinear model by learning a ***linear*** model on a ***new feature representation*** of the data. Explain briefly (using appropriate equations if needed) why this is true for each of these cases and, in each case, what is this new feature representation? Also, for each case, state whether the new feature representation is pre-defined or *learned*.

Answer: Kernel methods have prediction rules of the form $y_* = \text{sign}(\sum_{n=1}^N \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}_*))$ where k is the predefined kernel function and gives us N features of the form $k(\boldsymbol{x}_n, \boldsymbol{x}_*)$ for any example $\boldsymbol{x}_*$. Boosting methods have prediction rules of the form $y_* = \sum_{t=1}^T \alpha_t h_t(\boldsymbol{x}_*)$, where h_t are the learned weak hypotheses, each of which gives a feature $h_t(\boldsymbol{x}_*)$. Deep nets have prediction rule in the form $y_* = f(\boldsymbol{v}^\top \boldsymbol{h}_*)$ where $\boldsymbol{h}_*$ is the nonlinear feature representation of $\boldsymbol{x}_*$ learned by the deep neural net. So we can see that all of these are essentially learning linear models on different types of feature representations. Also note that, for kernel methods, the feature representations are pre-defined (by the choice of kernel) which is akin to using a pre-defined set of "basis functions", whereas for boosting and deep nets, the feature representations are learned from data (which is why these methods are also called "adaptive" basis function methods)

3. Suppose you applied the general Gaussian Mixture Model (GMM) and the K-means algorithm to cluster the same data and both algorithms ended up learning the same cluster means. You then converted the soft assignments of GMM to hard assignments (i.e., for each point, setting the largest cluster membership probability to 1 and setting the rest to 0). Will the resulting clusterings produced by both algorithms (based on point to cluster assignments) be the same? If yes, why? If no, why not? You may explain your answer using words/equations/figures (or a mix of all).

   Answer: Even though both may have the same cluster means, when we compute the cluster membership probability vector $\boldsymbol{z}_n$ for each point $\boldsymbol{x}_n$, the value of each z_{nk} not just depends on the distance from μ_k but also on π_k and Σ_k (recall the expression of the posterior probability of z_{nk} being 1). Therefore which entry of $\boldsymbol{z}_n$ will be the largest also depends on the size of the each cluster (as governed by the values of π_k's) as well as the shape (as defined by Σ_k's). Therefore, this one-hot vector conversion will not give us the same cluster assignments that K-means produced. I am lazy to draw figures now but I think that's a fairly reasonable and formal enough explanation. :)

4. Assume you have a binary matrix $\mathbf{X}$ of size $N \times M$ and you want to perform matrix factorization to learn latent factor matrix $\mathbf{U}$ of size $N \times K$ and $\mathbf{V}$ of size $M \times K$. Since each entry of $\mathbf{X}$ is binary, the commonly used squared loss for matrix factorization (that we saw in the class) may not be appropriate.

   - Suggest an alternate loss function that is more appropriate for modeling each binary observation in $\mathbf{X}$ and write down the full expression for the corresponding ℓ_2 regularized version of the loss function for this matrix factorization problem. For simplicity, you may assume that all the entries of the binary matrix $\mathbf{X}$ are observed.
   - For this new matrix factorization model which uses the above loss function, explain briefly (with or without equations) how you would estimate the latent factor matrices $\mathbf{U}$ and $\mathbf{V}$? In particular, what would the estimation problems for the latent factors $\{\boldsymbol{u}_n\}_{n=1}^N$ and $\{\boldsymbol{v}_m\}_{m=1}^M$ look like?

   Also, does this alternate loss function make estimation of latent factors harder or easier than in the squared loss case? If yes, why? If no, why not?

   Answer: A natural choice would be to model each binary-valued X_{nm} using a sigmoid function as $p(X_{nm} = 1|\boldsymbol{u}_n, \boldsymbol{v}_m) = \sigma(\boldsymbol{u}_n^\top \boldsymbol{v}_m) = \frac{1}{1+\exp(-\boldsymbol{u}_n^\top \boldsymbol{v}_m)}$ This makes intuitive sense: it basically says that if $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ are very similar (high dot product) then the probability of X_{nm} being 1 is high. The loss function will be very similar to the squared loss case we saw in the class, except that the squared loss will be replaced by a logistic regression like loss.

$$\mathcal{L} = \sum_{(n,m)\in\Omega} (X_{nm} \log p_{nm} + (1 - X_{nm}) \log(1 - p_{nm}) + \sum_{n=1}^N \lambda_U ||\boldsymbol{u}_n||^2 + \sum_{m=1}^M \lambda_V ||\boldsymbol{v}_m||^2$$

   where $p_{nm} = \sigma(\boldsymbol{u}_n^\top \boldsymbol{v}_m)$.

Now estimating each $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ basically reduces to solving a logistic regression problem (whereas in the squared loss case, we had to solve a least squares regression problem). That's the only basic difference. Of course, logistic regression doesn't have a closed-form solution, so each subproblem of estimating $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ will have to be solved in iterative fashion (just as we solve logistic regression). The lack of a closed form solution is a bit of a disadvantage as compared to the squared loss case (but each subproblem, being a logistic regression problem, is convex nevertheless)

Of course, other loss function suitable for modeling binary data (e.g., use a hinge loss) are also possible, but the logistic loss is perhaps the most widely used for such problems.

5. Recall the K-means objective function: $\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}||\boldsymbol{x}_n - \mu_k||^2$. As we have seen, the K-means algorithm minimizes this objective by taking a greedy iterative approach of assigning each point to its closest center (finding the z_{nk}'s) and updating the cluster means $\{\mu_k\}_{k=1}^{K}$.

   The standard K-means algorithm is a batch algorithm and uses all the data in every iteration. It however can be made online by taking a random example $\boldsymbol{x}_n$ at a time, and then (1) assigning $\boldsymbol{x}_n$ "greedily" to the "best" cluster, and (2) updating the cluster means using SGD on the objective $\mathcal{L}$.

   Assuming you have initialized $\{\mu_k\}_{k=1}^{K}$ randomly and are reading one data point $\boldsymbol{x}_n$ at a time,

   - How would you solve step 1?
   - What will be the SGD-based cluster mean update equations for step 2? Intuitively, why does the update equation make sense?
   - Note that the SGD update requires a step size. For your derived SGD update, suggest a good choice of the step size (and mention why you think it is a good choice).

   Answer: For step 1, just as in standard batch K-means, we simply have to assign $\boldsymbol{x}_n$ greedily to its closest center based on the smallest Euclidean distance. If $\boldsymbol{x}_n$ happens to be the closest from μ_ℓ, we will set $z_{n\ell} = 1$. Since all other $z_{nk} = 0$, as far as $\boldsymbol{x}_n$ is concerned, when computing the stochastic gradient, we only have to look at the term $z_{n\ell}||\boldsymbol{x}_n - \mu_\ell||^2 = ||\boldsymbol{x}_n - \mu_\ell||^2$ in the objective. Its gradient w.r.t. μ_ℓ will simply be $-2(\boldsymbol{x}_n - \mu_\ell)$. Therefore the SGD update for μ_ℓ will be (ignoring the factor of 2 and absorbing it in the step size η)

   $$\mu_\ell^{new} = \mu_\ell^{old} + \eta(\boldsymbol{x}_n - \mu_\ell^{old})$$

   Note that the other cluster means do not have to be updated.

   Intuitively, the above update makes sense because if $\boldsymbol{x}_n$ get assigned to cluster ℓ, we are moving its mean towards $\boldsymbol{x}_n$ (by adding $\eta(\boldsymbol{x}_n - \mu_\ell^{old})$).

   Choosing the step size: Note that the SGD update may cause the mean to be influenced too much by a single data point $\boldsymbol{x}_n$ which may not be a good thing. A small step size η can control it to some extent but a better way would be to control it based on how populated cluster ℓ is. If it has many points assigned to it then we probably should't move it by too much. One way to achieve this effect will be to set the step size to $1/N_\ell$ where N_ℓ is the number of points in cluster ℓ. So if N_ℓ is large then we won't move the mean too much (note: you can add 1 to N_ℓ to handle empty clusters).

5 Long Answer Problems (1 question, 16 marks)

1. Consider the following generative model that generates pairs $\{x_n, y_n\}_{n=1}^{N}$ as follows

   $$\begin{aligned} y_n &\sim \text{Bernoulli}(p) \\ x_n|y_n &\sim \text{Normal}(\mu_{y_n}, \sigma_{y_n}^2) \end{aligned}$$

   For simplicity, assume $x_n \in \mathbb{R}$ (drawn from a univariate normal). Consider the following cases:

- $\{x_n, y_n\}_{n=1}^N$ are observed. Give the MLE estimates for the model parameters $\{p, \mu_0, \sigma_0^2, \mu_1, \sigma_1^2\}$. You are not required to derive the expressions. If you *know* what these will be (or can *guess*, but your guess better be fully correct! :)), you can just write down the final expressions.
- Only $\{x_n\}_{n=1}^N$ are observed. Is MLE estimation now easier/harder than the above case, or equally easy/hard? Explain briefly, how would you do it in this case (no need to write equations)?
- $\{x_n, y_n\}_{n=1}^M$ and $\{x_n\}_{n=M+1}^N$ are observed. What about the ease/hardness of parameter estimation in this case relative to the above two cases, and how would you perform MLE in this case?

Also answer the following questions:

- For each of the above three cases, are you guaranteed to find a unique solution for the parameters? Briefly justify why or why not.
- Having estimated the model parameters, suppose you get a new input x_*. How would you find the corresponding y_* in each of the above cases? Write down the expression to find y_*.
- Explain briefly why might you prefer this way of doing classification over a method like logistic regression?

Answer: (Case 1) If $\{x_n, y_n\}_{n=1}^N$ are observed, the MLE estimation problem is very easy. Since each $y_n \in \{0,1\}$, the Bernoulli probability will simply be the fraction of 1s in $\{y_n\}_{n=1}^N$, i.e., $p = \frac{\sum_{n=1}^N y_n}{N}$. Since all y_n's are known, we know which x_n comes from which Gaussian. Therefore estimating the Gaussian means and variances $\mu_0, \sigma_0^2, \mu_1, \sigma_1^2\}$ is also very easy, and these estimates are empirical means and variances, as given below

$$\begin{aligned} \mu_0 &= \frac{1}{N_0}\sum_{n:y_n=0} x_n, \quad \sigma_0^2 = \frac{1}{N_0}\sum_{n:y_n=0}(x_n-\mu_0)^2 \\ \mu_1 &= \frac{1}{N_1}\sum_{n:y_n=1} x_n, \quad \sigma_1^2 = \frac{1}{N_1}\sum_{n:y_n=1}(x_n-\mu_1)^2 \end{aligned}$$

where we have defined $N_1 = \sum_{n=1}^N y_n$ to be the number of points that have $y_n = 1$ and $N_0 = N - N_1$ to be the number of points that have $y_n = 0$.

Note: Since $y_n\{0,1\}$, an equivalent way to write the above equations will be as follows

$$\mu_0 = \frac{\sum_{n=1}^N (1-y_n)x_n}{\sum_{n=1}^N (1-y_n)}, \quad \sigma_0^2 = \frac{\sum_{n=1}^N (1-y_n)(x_n-\mu_0)^2}{\sum_{n=1}^N (1-y_n)} \tag{1}$$

$$\mu_1 = \frac{\sum_{n=1}^N y_n x_n}{\sum_{n=1}^N y_n}, \quad \sigma_1^2 = \frac{\sum_{n=1}^N y_n(x_n-\mu_1)^2}{\sum_{n=1}^N y_n} \tag{2}$$

(This equivalent form, as we will see, is more amenable to use in EM where we would like to replace the y_n directly by the corresponding expectations)

(Case 2) When only $\{x_n\}_{n=1}^N$ are observed, the problem is a bit harder because the y_n's are now unknown and must be inferred. This kind of becomes like a GMM problem with 2 clusters (the y_n's can be thought of as the latent cluster assignments). However, we can use EM to estimate y_n's in the E step by computing

$$\begin{aligned} \mathbb{E}[y_n] &= 0 \times p(y_n=0|x_n) + 1 \times p(y_n=1|x_n) \\ &= p(y_n=1|x_n) \\ &= \frac{p(y_n=1)p(x_n|y_n=1)}{p(y_n=0)p(x_n|y_n=0) + p(y_n=1)p(x_n|y_n=1)} \end{aligned} \tag{3}$$

These expectations can then be used in the M step to compute the parameters $\{p, \mu_0, \sigma_0^2, \mu_1, \sigma_1^2\}$ (simply replace every y_n in Eq (1) and (2) above by the corresponding expectations).

(Case 3) When $\{x_n, y_n\}_{n=1}^M$ and $\{x_n\}_{n=M+1}^N$ are observed, this basically is a "semi-supervised" learning problem (some y_n's are known and some are unknown). The same EM based procedure as above can be used to estimate the unknown y_n's, and the parameters can be estimated in the M step using the known y_n's and the estimated y_n's (for those that are missing).

Only in case (1) we are guaranteed to find a unique solution (available in closed form). Note that when y_n's are observed, we are basically solving simple estimation problems (estimating a Bernoulli parameter, or estimating mean/variance of Gaussian). In the other two cases, we are relying on EM to guess y_n's which we then use to estimate the parameters. As we know, EM is only guaranteed to find a local optima.

Given a new x_*, we can use Eq (3) to compute the posterior probability of y_* being 1. This is essentially the generative classification approach because to compute $p(y|x)$ we used $p(y)$ and $p(x|y)$ and used Bayes rule.

One reason why this way of doing classification may be preferable over something lile logistic regression is that it easily allows us to perform semi-supervised learning (as we saw above; also recall our discussion in class on this topic), which is not as straightforward for discriminative models like logistic regression. So we can learn a reliable model even with small amounts of labeled data and possibly a lot of cheaply available unlabeled data.