CS4780 Final

Spring 2017

NAME:	
Net ID:	
Email:	

1 [??] General Machine Learning

Please identify if these statements are either True or False. Please justify your answer **if false**. Correct "True" questions yield 1 point. Correct "False" questions yield two points, one for the answer and one for the justification.

1. (**T/F**) Random forests is one of the few machine learning algorithms that makes no assumptions on the data.
   False, every machine learning algorithm makes assumptions. RF assumes that similar inputs have similar labels.

2. (**T/F**) One implication of the curse of dimensionality is that if you sample n data points uniformly at random within a hyper cube of dimensionality d, all pairwise distances converge to 0 as $n \to \infty$.
   False, they become concentrated around the average distance as $d \to \infty$.

3. During training, in a linearly separable data set, the perceptron algorithm never misclassifies the same input twice. (**T/F**)
   False, it can iterate many times over the data set and get the same points wrong repeatedly.

4. (**T/F**) You have a biased coin and toss it n times. The MAP estimate with $+1$ smoothing of the probability of getting "head" is $\frac{n_H+1}{n+1}$, where n_H is the number of occurrences of "head" amongst your n throws.
   False, it is $\frac{n_H+1}{n+2}$.

5. (**T/F**) The multinomial Naive Bayes algorithm is a linear classifier.
True.

6. (**T/F**) MAP inference maximizes $P(\mathbf{w}|Data)$ whereas MLE maximizes $P(Data; \mathbf{w})$, where $\mathbf{w}$ represents the model parameters.
True.

7. (**T/F**) Newton's Method diverges only if the Hessian matrix is not invertible.
False, it can also diverge with invertible Hessian matrices.

8. (**T/F**) Linear (ordinary least squares) regression can be solved in closed form, although sometimes that is computationally impractical or even infeasible.
True.

9. (**T/F**) SVMs maximize the margin between the training and testing data.
False, they maximize the margin between the training data and the separating hyperplane.

10. (**T/F**) In order for gradient descent to converge, the loss function has to be convex and differentiable everywhere.
False, if it is not convex it will still converge, but to a local minimum.

11. (**T/F**) The bias variance trade-off decomposes the error obtained by a classifier into (squared) bias, variance, and noise. The noise term cannot possibly be addressed, even by changing the feature representation of the data.
False, changing the feature representation of the data will affect the noise. For example, if all features are removed the error is only noise (which would be very large).

12. (**T/F**) In a setting of high bias, a great remedy is to add more training data.
False, more training data does not help with bias.

13. (**T/F**) Bagging reduces variance.
True.

14. (**T/F**) Boosting reduces noise.
False, it reduces bias (and sometimes even variance a little).

15. (**T/F**) Learning with kernels is expensive, because the data is mapped into a very high dimensional space and therefore storing the transformed data consumes a lot of storage.
False, the mapping is performed implicitly.

16. (**T/F**) The mean prediction of Gaussian processes is identical to kernelized linear regression.
True.

17. (**T/F**) One popular application of Gaussian Processes is to find hyper-parameters of machine learning algorithms.
True.

18. (**T/F**) Ball-Trees are a data structure to speed up the perceptron algorithm.
False, they can speed up nearest neighbor searchers, but that is never performed in the Perceptron.

19. (**T/F**) Decision Trees stop splitting when the impurity function can no longer be improved with a single split.
False, e.g. in the XOR data set the first split does not improve the impurity function. The splitting stops if the maximum depth (or number of nodes) is reached, or all inputs are identical.

20. (**T/F**) Random Forests are bagged decision trees with one additional modification: Each splitting dimension is chosen completely uniformly at random.

    False, the best splitting dimension is selected amongst k random dimensions.

21. (**T/F**) Provided each weak learner can classify a weighted version of the training data set with better than 0.5 accuracy, in AdaBoost the training error reduces exponentially.
    True.

22. (**T/F**) Deep neural networks are great on many data sets, but do not work competitively on image classification tasks.
    False, they are particularly good at image classification tasks.

23. (**T/F**) The optimization of deep neural networks is a convex minimization problem.
    False, it is non-convex because of the non-linear transition functions.

2 [21] Bias Variance / Model Selection

1. (3) Write down the bias (squared), variance, noise decomposition of the expected test error $\mathbb{E}_{x,y,D}\left[(h_D(x) - y)^2\right]$.
   Variance: $E_{x,D}[(h_D(x) - \bar{h}(x))^2]$
   Bias squared: $E_x[(\bar{h}(x) - \bar{y}(x))^2]$
   Noise: $E_{x,y}[(\bar{y}(x) - y(x))^2]$

2. (5) Describe how to detect settings with high bias and provide three approaches that could help reduce the bias.
   Detect high bias if traning error is above goal error (plot training and testing error vs number of data points)
   Reduce bias by decreasing model complexity, using boosting, and 2 points for correct detection
   3×1 point for correct remedies

3. (13) For each of the following scenarios, determine if the model has low/high bias and variance. Explain your choice.

   (a) Logistic regression on *linearly* separable data and *non-linearly* separable data.
   Linearly separable: low bias, low variance
   Non-linearly separable: high bias, low variance

   (b) kNN with small k and large k.
   Small k: low bias, high variance
   Large k: high bias, low variance

(c) Uniform random labeling.
low bias, high variance

3 [21] Kernel Methods

1. Suppose you are using a kernel SVM with the RBF kernel $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|_2^2}{\sigma^2}\right)$ to do classification. Recall that the kernel SVM is trained by solving the dual optimization problem:

$$\min_{\alpha_1,\ldots,\alpha_n} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{K}_{ij} - \sum_i \alpha_i$$
$$\text{s.t. } 0 \le \alpha_i \le C$$
$$\sum_i \alpha_i y_i = 0$$

   (a) (5) Assume you can either set C and σ^2 to a very large value ($\gg 0$) or a very small value (ϵ). Provide a setting with high bias and one with high variance. *Briefly* explain your answers.

   (b) (3) $\mathbf{x}_1$ turns out to be a support vector. What can you say about its corresponding optimal value α_i^* and the margin between the hyperplane and $\mathbf{x}_1$?

   (c) (5) In order to apply the classifier to a test point, we need the hyper-plane bias b. Show how b can be recovered from $\alpha_1^*, \ldots, \alpha_n^*$ with the help of the support vector $\mathbf{x}_i$ and label $y_i \in \{-1., +1\}$.

2. (8) For this question, you will find the following rules about recursively building kernels helpful. Given kernels $k_1(\mathbf{x}, \mathbf{z})$ and $k_2(\mathbf{x}, \mathbf{z})$, the following are well-defined kernels:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top A \mathbf{z}, A \succeq 0 \tag{1}$$
$$k(\mathbf{x}, \mathbf{z}) = c k_1(\mathbf{x}, \mathbf{z}) \tag{2}$$
$$k(\mathbf{x}, \mathbf{z}) = \exp\left(k_1(\mathbf{x}, \mathbf{z})\right) \tag{3}$$
$$k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{z}) f(\mathbf{z}) \tag{4}$$
$$k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z}) \tag{5}$$

Suppose that $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$. Let $[\mathbf{x}]_1$ and $[\mathbf{x}]_2$ denote the first and second coordinates of $\mathbf{x}$, respectively. Show that

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|[\mathbf{x}]_1 - [\mathbf{z}]_1\|_2^2}{\sigma^2}\right) + \exp\left(-\frac{\|[\mathbf{x}]_2 - [\mathbf{z}]_2\|_2^2}{\sigma^2}\right)$$

is a kernel.
Hint: You may find the following two matrices helpful:

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

You can assume they are positive semi-definite (i.e. $A_1 \succeq 0, A_2 \succeq 0$).
The trick here is to define

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$
$$A_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

so that $\mathbf{x}^\top A_1 \mathbf{z} = [\mathbf{x}]_1 [\mathbf{z}]_1$ and $\mathbf{x}^\top A_2 \mathbf{z} = [\mathbf{x}]_2 [\mathbf{z}]_2$ (these matrices are psd with eigenvalues 0 and 1). The rest of the proof is identical to the proof for the RBF kernel in class:

(a) $k_1(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top A_1 \mathbf{z} = [\mathbf{x}]_1 [\mathbf{z}]_1$, rule (1)

(b) $k_2(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} k_1(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} [\mathbf{x}]_1 [\mathbf{z}]_1$, rule (2)

(c) $k_3(\mathbf{x}, \mathbf{z}) = \exp\left(k_2(\mathbf{x}, \mathbf{z})\right) = \exp\left(\frac{2[\mathbf{x}]_1 [\mathbf{z}]_1}{\sigma^2}\right)$, rule(3)

(d) $k_4(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{[\mathbf{x}]_1 [\mathbf{x}]_1}{\sigma^2}\right) \exp\left(\frac{2[\mathbf{x}]_1 [\mathbf{z}]_1}{\sigma^2}\right) \exp\left(-\frac{[\mathbf{z}]_1 [\mathbf{z}]_1}{\sigma^2}\right) = \exp(-\frac{\|[\mathbf{x}]_1 - [\mathbf{z}]_1\|_2^2}{\sigma^2})$, rule (4) with $f(\mathbf{x}) = \exp(-\frac{[\mathbf{x}]_1 [\mathbf{x}]_1}{\sigma^2})$

(e) Repeating the above with A_2, $k_5(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|[\mathbf{x}]_2 - [\mathbf{z}]_2\|_2^2}{\sigma^2})$ is a kernel.

(f) Finally, $k_4 + k_5$ is a kernel by rule (5).

4 [21] CART

1. (2) Imagine you build a KD-Tree and label each leaf with the most common label amongst all training points that fall into this leaf. Why would this not be a desirable classifier? Because many leaves would not be pure, which makes the most common label a bad estimate.

2. (4) Name two reasons why Random Forests are such popular classifiers amongst practitioners? 1. They only have two hyper-parameters (the number of trees m and the number of features K), but both are really easy to set. You can set $K = \sqrt{d}$ and m as large as you can afford. 2. RF are based on decision / regression trees and require no feature scaling or any of the typical pre-processing of the data. Features can be in completely different units and can be categorical or real valued.

3. (2) Assume you pre-process all your features in the following way: you sort each feature independently. For each feature, you then assign all those inputs that share the lowest feature value a new feature value of 1, all those with the second lowest value a 2, etc. How does this affect the trees that you construct? It doesn't.

4. (4) Under what conditions on your training set will a CART tree (with unlimited depth) obtain 0% training error. If there are no two training inputs with identical features but different labels.

5. (3) You are building a regression tree with the squared loss impurity. i.e. the labels in the leaf are $L = \{y_1, \ldots, y_m\}$ and the loss, under prediction t, is $\sum_{y \in L}(y-t)^2$. Prove that the average label $t = \frac{1}{m}\sum_{i=1}^{m} y_i$ minimizes the loss at a leaf.

$$t = \text{argmin}_t \sum_{i=1}^{n}(t - y_i)^2 \tag{6}$$

Taking the derivative and eq. with 0:

$$2\sum_{i=1}^{n}(t - y_i) = 0 \tag{7}$$

$$2nt - 2\sum_{i=1}^{n} y_i = 0 \tag{8}$$

$$t = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{9}$$

6. (6) You are now considering minimizing the absolute loss instead: $\sum_{y \in L} |y - t|$. Define $L_{\leq} = \{y \in L : y \leq t\}$ and $L_{>} = \{y \in L : y > t\}$. Prove that setting t to the median of L minimizes this loss. To simplify things you can assume you have an odd number of samples (i.e. $m = 2r+1$) and that all $y_i \in L$ are distinct (i.e. $y_i \neq y_j$ for any $y_i, y_j \in L$). (Without loss of generality it is sufficient to show there is no better splitting value t' that is *larger* than the median.)

5 [21] Boosting / Bagging

1. (3) Name two algorithms, for which boosting will be ineffective. Briefly justify why.

   e.g. k-NN classification, unlimited depth decision trees, kernel SVMs. They have high variance and essentially zero bias.
   Rubrics: one point for each algorithm, one point for correct justification. Maximum 3 points.
   special cases: (1) Linear classifiers also gain 1 point since if the data set isn't linearly separable there is not much used to ensemble linear classifier.
   special cases: (2) Naive Bayse doesn't get point.
   special cases: (3) Random Labling get 1 point since it's not a week learner (since it doesn't learn)
   special cases: (4) Mode Labling didn't get points

2. (5) Describe what happens in AdaBoost if two training inputs (in a binary classification problem) are identical in features but have different labels?

   Both points will obtain very high weights and eventually will dominate the training data set. Weak learners will no longer be able to classify the weighted data set with better than 50% accuracy and the algorithm will stop.
   minimum 1 point if say something and show effort. (+1) if state that the algorithm will stop. (+2) if state that these two data points will gain weights. (+1) if state that the weak learner won't be able to distinguish these two data points eventually. maximum 5 points.

3. (3) In neural networks bagging can be performed *without random subsampling of the data*. i.e., one trains m neural networks independently and ensembles their results. Can you explain why the subsampling is unnecessary in this case?

   The random initialization and non-convexity of neural networks ensures that independently trained models will end up in different local minima and obtain different results. The effect is similar to training on slightly different data sets.

   minimum 1 point if say something and show effort. (+2) if state that NN has random initialization. (+1) if state NN convergences to local minimum due to non-convexity.
   special case (1) : if mentioned that Stochastic Gradient Descend randomly sample training data and lead to different weights, get 2 points.
   special case (2) : if mentioned that layers such as dropout is some embedded randomness, get also 2 points.

4. Assume you have weak learners $h \in \mathcal{H}$ s.t. $h(\mathbf{x}) \in \{+1, -1\}$ for any $\mathbf{x}$. You are trying to apply boosting with the logistic loss function

$$\mathcal{L}(H) = \sum_{i=1}^{n} \ln\left(1 + e^{-y_i H(\mathbf{x}_i)}\right). \tag{10}$$

(remember, ln here refers to the natural logarithm)

(a) (4) Compute the derivative $\frac{\partial \mathcal{L}(H)}{\partial H(\mathbf{x}_i)}$.

$$\frac{\partial \mathcal{L}(H)}{\partial H(\mathbf{x}_i)} = \frac{-y_i}{1 + e^{-y_i H(\mathbf{x}_i)}} e^{-y_i H(\mathbf{x}_i)} = -\frac{y_i}{1 + e^{y_i H(\mathbf{x}_i)}}. \tag{11}$$

$$\tag{12}$$

minimum 1 point if show effort and write something. (+3) if the answer is correct. (+2) if the answer has only minor mistake (i.e. flip the sign, etc).

(b) (6) Let $w_i = \frac{1}{1+e^{y_i H(\mathbf{x}_i)}}$ and let $\epsilon(h) = \sum_{i:h(\mathbf{x}_i)\neq y_i} w_i$ be the weighted error of the training set. For simplicity assume we are using a fixed step-size of 1. Show that the next classifier to be added to the ensemble H in order to minimize the loss function is $h = \text{argmin}_h \mathcal{L}(H + h) = \arg\min_h \epsilon(h)$.

$$h = \text{argmax}_h \sum_{i=1}^{n} h(\mathbf{x}_i) \frac{\partial \mathcal{L}(H)}{\partial H(\mathbf{x}_i)} \tag{13}$$

$$= \text{argmin}_h \sum_{i=1}^{n} w_i y_i h(\mathbf{x}_i) \tag{14}$$

$$= \text{argmin}_h \sum_{i:h(\mathbf{x}_i)=y_i} w_i - \sum_{i:h(\mathbf{x}_i)\neq y_i} w_i \tag{15}$$

$$= \text{argmin}_h \epsilon(h) - (1 - \epsilon(h)) \tag{16}$$

$$= \text{argmin}_h \epsilon(h) \tag{17}$$

6 [21] Deep Learning

Assume you are given a neural network with L layers to minimize a loss function $\mathcal{L}$

$$h(\mathbf{x}) = \mathbf{w}^\top \phi_1(\mathbf{x}) \tag{18}$$
$$\phi_1(\mathbf{x}) = \sigma(\mathbf{U}_1 \phi_2(\mathbf{x})) \tag{19}$$
$$\vdots \tag{20}$$
$$\phi_\ell(\mathbf{x}) = \sigma(\mathbf{U}_\ell \phi_{\ell+1}(\mathbf{x})) \tag{21}$$
$$\vdots \tag{22}$$
$$\phi_L(\mathbf{x}) = \sigma(\mathbf{U}_L \mathbf{x}) \tag{23}$$

(*Note that the subscript of ϕ starts at 1 at the end of the network, and increases to L as we make our way back to the start*)

1. (5) Let us define $a_\ell = \mathbf{U}_\ell \phi_{\ell+1}(\mathbf{x})$ such that $\phi_\ell = \sigma(a_\ell)$. Let $\delta_\ell = \frac{\partial \mathcal{L}}{\partial a_\ell}$. Express $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_\ell}$ in terms of δ_ℓ. (assume $1 < \ell < L$)

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{U}_\ell} &= \frac{\partial \mathcal{L}}{\partial a_\ell} \frac{\partial a_\ell}{\partial \mathbf{U}_\ell} \\ &= \delta_\ell \phi_{\ell+1}(\mathbf{x})^T\end{aligned}$$

2. (5) Assume that the derivative of $\sigma(z)$ is given as $\sigma'(z)$. Define $\delta_{\ell+1}$ as a function of δ_ℓ. (assume $1 < \ell < L$)

$$\begin{aligned}\delta_{\ell+1} &= \frac{\partial \mathcal{L}}{\partial a_{\ell+1}} \\ &= \frac{\partial \mathcal{L}}{\partial \phi_{\ell+1}} \frac{\partial \phi_{\ell+1}}{\partial a_{\ell+1}} \\ &= \frac{\partial \mathcal{L}}{\partial a_\ell} \frac{\partial a_\ell}{\partial \phi_{\ell+1}} \frac{\partial \phi_{\ell+1}}{\partial a_{\ell+1}} \\ &= \sigma'(a_{\ell+1}) \odot \mathbf{U}_\ell^T \delta_\ell \\ &= \sigma'(\mathbf{U}_{\ell+1} \phi_{\ell+2}) \odot \mathbf{U}_\ell^T \delta_\ell\end{aligned}$$

where $x = \phi_{L+1}$.

3. (3) Provide one reason why stochastic gradient descent can be better than traditional (batch) gradient descent when applied to neural networks. SGD can jump out of local minima more easily, since it's more noisy. Alternatively, you can note that as you increase your batch size, your update gradient asymptotically approaches the true gradient. Thus, you can split your batch into n parts, yielding n updates with generally better than $1/n$ accuracy relative to the true gradient, yielding more progress per computation time. SGD takes this to an extreme. Both answers are correct, but not equivalent.

4. (4) Assume you make all transition functions the identity (i.e. $\sigma(z) = z$). Prove that the final classifier is simply a linear classifier of the form $h(\mathbf{x}) = \hat{\mathbf{w}}^\top \mathbf{x}$ for some vector $\hat{\mathbf{w}}$.

$$h(\mathbf{x}) = \mathbf{w}^T (\prod_{\ell=L}^{1} \mathbf{U}_\ell)\mathbf{x}$$
$$= \hat{\mathbf{w}}^T \mathbf{x}$$

5. (4) ML-practitioners tend to drop the learning rate during training. Explain why and what effect it has. Starting out with a large learning rate has two advantages: 1. it prevents you from getting trapped in sharp local minima, because the weights "jump around" too much with each step; and 2. it moves you quickly "down-hill" because you take larger steps. Then switching to a smaller learning rate allows the network to converge to the local minimima closest to the current weight position. (One correct answer is to draw the picture from class.)

This page is left blank for scratch space.

This page is left blank for jokes. (Nothing too dirty.)

Please do not write on this page. For administrative purposes only.

T/F	
BV	
Kernels	
CART	
ENSEMBLE	
DL	
TOTAL	