CS 189 Introduction to Machine Learning
Spring 2018 EXAMFINAL

After the exam starts, please write your student ID (or name) on every page. There are **6** questions for a total of **38** parts. Two parts are bonus and can give extra credit points. On the last question (multiple choice), you will be graded on your best fifteen of eighteen parts. You may consult your two sheets of notes. Calculators, phones, computers, and other electronic devices are not permitted. There are **27** single-sided pages on the exam. **Notify a proctor immediately if a page is missing.** You may, without proof, use theorems and lemmas that were proven in the notes and/or in lecture, unless we explicitly ask for a derivation. **You have 180 minutes**.

Exam Location: Sahai's Office :O . **SID (Last Digit)**: -1

PRINT and SIGN Your Name: _____, _____, _____
(last) (first) (signature)

PRINT Your Student ID: _____

Person before you: _____, _____
(name) (SID)

Person behind you: _____, _____
(name) (SID)

Person to your left: _____, _____
(name) (SID)

Person to your right: _____, _____
(name) (SID)

Row (front is 1): _____ Seat (leftmost is 1): _____ (*Include empty seats/rows.*)

1 Pre-exam Questions (4 points)

(a) **(2 points)** What is your favorite machine learning topic covered in the course?

(b) **(2 points)** Describe some aspect of machine learning that you understand better having worked hard in this course.

Do not turn this page until your instructor tells you to do so.

2 When testing is "unfair" (35 points)

In this problem we explore what happens when we train an estimator on a data set which has a different distribution than the test set.

For concreteness, let us consider the problem of linear regression where we want to learn something close to the true linear model $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}^\star$. Our training set consists of a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ with rows $\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top$ and noisily observed target vector $\mathbf{y} = \mathbf{X}\mathbf{w}^\star + \boldsymbol{\epsilon}$ with Gaussian observation noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$, where $\mathbf{w}^\star \in \mathbb{R}^d$ is the true weight vector.

After fitting a linear regression model to $(\mathbf{X}, \mathbf{y})$ which results in the estimator $\widehat{\mathbf{w}}$, we want to know how well this learnt model will perform on a test set, where inputs have been drawn from a different distribution than the training points, but still generating true outputs in the same manner (namely with $y_{\text{test}} = \mathbf{x}_{\text{test}}^\top \mathbf{w}^\star$, for the same true $\mathbf{w}^\star$). This is sometimes referred to as "covariate shift" in the machine learning and statistical literature.

In the following we assume that $\mathbf{X}$ is full rank and has compact singular value decomposition $\mathbf{X} = \mathbf{U}\boldsymbol{\Lambda}^{1/2}\mathbf{V}^\top$ with diagonal matrix $\boldsymbol{\Lambda}$ which has entries $\lambda_1, \ldots, \lambda_d$ on the diagonal and matrices $\mathbf{U} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{d \times d}$. We refer to the matrix $\boldsymbol{\Sigma} = \mathbf{X}^\top \mathbf{X}$ as the empirical covariance matrix of $\mathbf{X}$.

(a) (5 pts) **Write down the least squares estimator $\widehat{\mathbf{w}}$ you obtain by minimizing the ordinary least squares (OLS) objective, i.e. $\widehat{\mathbf{w}} := \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$, in terms of $\mathbf{w}^\star, \mathbf{V}, \boldsymbol{\Lambda}$ and $\widetilde{\boldsymbol{\epsilon}} = \mathbf{U}^\top \boldsymbol{\epsilon}$.** You do not need to derive the standard OLS solution but can start with that if you choose.

Solution: Note that the eigendecomposition of $\boldsymbol{\Sigma} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$ reads and therefore

$$\begin{aligned}
\widehat{\mathbf{w}} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \\
&= \boldsymbol{\Sigma}^{-1} \mathbf{X}^\top (\mathbf{X}\mathbf{w}^\star + \boldsymbol{\epsilon}) \\
&= \mathbf{w}^\star + \mathbf{V}\boldsymbol{\Lambda}^{-1}\boldsymbol{\Lambda}^{1/2}\mathbf{U}^\top \boldsymbol{\epsilon} \\
&= \mathbf{w}^\star + \mathbf{V}\boldsymbol{\Lambda}^{-1/2}\widetilde{\boldsymbol{\epsilon}}
\end{aligned}$$

(b) (5 pts) **What is the mean and covariance matrix of the Gaussian random vector $\widetilde{\boldsymbol{\epsilon}} = \mathbf{U}^\top \boldsymbol{\epsilon}$?**

Solution: Use the fact that

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\epsilon}}[\widetilde{\boldsymbol{\epsilon}}\widetilde{\boldsymbol{\epsilon}}^\top] &= \mathbb{E}_{\boldsymbol{\epsilon}}[\mathbf{U}^\top \boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top \mathbf{U}] \\ &= \mathbf{U}^\top \mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top]\mathbf{U} \\ &= \mathbf{I}_d\end{aligned}$$

and mean is zero by linearity of expectation.

(c) (15 pts) We are now given a different test set $\mathbf{X}_{\text{test}}$ with empirical covariance $\mathbf{\Sigma}_{\text{test}}$, where $\mathbf{X}_{\text{test}}$ has the same eigenbasis (and hence implicity, the same number n of points) for simplicity, i.e. $\mathbf{X}_{\text{test}} = \mathbf{U}\mathbf{\Lambda}_{\text{test}}^{1/2}\mathbf{V}^\top$. Use parts (a), (b) to **derive the following expected average prediction error in terms of the eigenvalues λ_i of $\mathbf{X}^\top\mathbf{X}$ and eigenvalues λ_i^{test} of $\mathbf{X}_{\text{test}}^\top\mathbf{X}_{\text{test}}$:**

$$\frac{1}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\|\mathbf{X}_{\text{test}}\widehat{\mathbf{w}} - \mathbf{X}_{\text{test}}\mathbf{w}^\star\|^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{\lambda_i^{\text{test}}}{\lambda_i}.$$

Recall that these eigenvalues, λ_i^{test}, correspond to the same i-th eigenvector in $\mathbf{V}$, for $i = 1, \ldots, d$. Note that the expectation above is taken over the only randomness in the problem, the training noise $\boldsymbol{\epsilon}$.

Hint: Depending on how you do it, you might find these facts useful: (i) $\mathbf{u}^\top\mathbf{v} = \text{tr}(\mathbf{v}\mathbf{u}^\top)$ and (ii) $\mathbb{E}[\text{tr}(A)] = \text{tr}(\mathbb{E}[A])$ and (iii) $\text{tr}(A) = \sum_i \lambda_i$ where λ_i are the eigenvectors of matrix A. But there are other ways to get to the answer as well.

Solution: Noting that $\mathbf{\Sigma}_{\text{test}} = \mathbf{V}\mathbf{\Lambda}_{\text{test}}\mathbf{V}^\top$, the error on the new $\mathbf{X}_{\text{test}}$ with covariance matrix $\mathbf{\Sigma}_{\text{test}}$ is

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\epsilon}}\|\mathbf{X}_{\text{test}}\widehat{\mathbf{w}} - \mathbf{X}_{\text{test}}\mathbf{w}^\star\|^2 &= \mathbb{E}(\widehat{\mathbf{w}} - \mathbf{w}^\star)^\top\mathbf{\Sigma}_{\text{test}}(\widehat{\mathbf{w}} - \mathbf{w}^\star) \\
&= \mathbb{E}\widetilde{\boldsymbol{\epsilon}}^\top\mathbf{\Lambda}^{-1}\mathbf{\Lambda}_{\text{test}}\widetilde{\boldsymbol{\epsilon}} \\
&= \text{tr}(\mathbb{E}[\widetilde{\boldsymbol{\epsilon}}\widetilde{\boldsymbol{\epsilon}}^\top]\mathbf{\Lambda}^{-1}\mathbf{\Lambda}_{\text{test}}) \\
&= \text{tr}(\mathbf{\Lambda}^{-1}\mathbf{\Lambda}_{\text{test}}) = \sum_{i=1}^{n}\frac{\lambda_i^{\text{test}}}{\lambda_i}
\end{aligned}$$

where the third equality follows from the linearity of expectation and the hint, and the fourth equality follows from (b).

(d) (5 pts) In practice, we sometimes have a choice of training sets. Let's consider a concrete scenario with $d = 2$. We assume for simplicity that $\mathbf{V} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ for all covariance matrices to follow. Assume that we can choose to obtain noisy observations $\mathbf{y} = \mathbf{X}^\top \mathbf{w}^\star + \boldsymbol{\epsilon}$ for three possible training feature matrices $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$ whose covariance matrices have the following diagonal eigenvalue matrices

$$\boldsymbol{\Lambda}_1 = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix} \qquad \boldsymbol{\Lambda}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 100 \end{pmatrix} \qquad \boldsymbol{\Lambda}_3 = \begin{pmatrix} 100 & 0 \\ 0 & 0.1 \end{pmatrix}.$$

First, we are given a test set $\mathbf{X}_{\text{test}}$ (with the same singular vectors as $\mathbf{X}$) with diagonal eigenvalue matrix $\boldsymbol{\Lambda}_{\text{test}} = \begin{pmatrix} 0.01 & 0 \\ 0 & 100 \end{pmatrix}$ and are asked to **minimize the average expected prediction error $\frac{1}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\|\mathbf{X}_{\text{test}}\widehat{\mathbf{w}} - \mathbf{X}_{\text{test}}\mathbf{w}^\star\|_2^2$. Which training feature matrix do you choose and why?**

Solution: For this test set, the best choice is feature matrix number 2. One can see this by looking at the formula in the previous subpart; the sum of the ratio is minimized among the different diagonal matrices for $\boldsymbol{\Lambda}_2$ (first has sum ~ 10, second has sum ~ 1, third has sum ~ 100). We can also see this intuitively because the second eigenvector is weighted heavily in the test set, and thus needs to be better "explored" in the training set.

(e) (5 pts) Now you are asked to make a choice of training data from the previous part's $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$ which **minimizes the worst-case expected error under any possible $\mathbf{x}_{\text{test}}$ with unit norm. Which feature matrix do you choose and why?**

Solution:

First we should consider what $\mathbf{x}$ to choose to test given that the predictor knows both $\widehat{\mathbf{w}}$ and $\mathbf{w}^\star$. Then we can consider what the expected worst case average prediction error is, i.e.,

$\mathbb{E}_{\boldsymbol{\epsilon}} \sup_{\|\mathbf{x}\|_2 \leq 1} (\mathbf{x}^\top \widehat{\mathbf{w}} - \mathbf{x}^\top \mathbf{w}^\star)^2$ as a function of $\lambda_1, \ldots, \lambda_d$, before concluding which feature matrix to choose.

Using question (a) we obtain

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\epsilon}} \|\widehat{\mathbf{w}} - \mathbf{w}^\star\|^2 &= \mathbb{E} \widetilde{\boldsymbol{\epsilon}}^\top \boldsymbol{\Lambda}^{-1/2} \mathbf{V}^\top \mathbf{V} \boldsymbol{\Lambda}^{-1/2} \widetilde{\boldsymbol{\epsilon}} \\
&= \operatorname{tr}(\mathbb{E}[\widetilde{\boldsymbol{\epsilon}} \widetilde{\boldsymbol{\epsilon}}^\top] \boldsymbol{\Lambda}^{-1}) \\
&= \operatorname{tr}(\boldsymbol{\Lambda}^{-1})
\end{aligned}$$

Then we note that $\sup_{\|\mathbf{x}\|_2 \leq 1} \mathbf{x}^\top \mathbf{v} = \|\mathbf{v}\|_2$ for all $\mathbf{v}$ and thus,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\epsilon}} \sup_{\|\mathbf{x}\|_2 \leq 1} (\mathbf{x}^\top \widehat{\mathbf{w}} - \mathbf{x}^\top \mathbf{w}^\star)^2 &= \mathbb{E}_{\boldsymbol{\epsilon}} \mathbb{E}_{\boldsymbol{\epsilon}} \|\widehat{\mathbf{w}} - \mathbf{w}^\star\|^2 \\
&= \sum_{i=1}^{d} \lambda_i^{-1}
\end{aligned}$$

Thus the worst case error only depends on the sum of inverse eigenvalues. Thus the feature matrix number 1 is the best because the first has error ~ 0.1, second has ~ 1, third has ~ 10.

3 Coin tossing with unknown coins (35 points)

This question is about adapting EM and the spirit of k-means to a discrete problem of tossing coins.

We have a bag that contains two kinds of coins that look identical. The first kind has probability of heads p_a and the other kind has probability of heads p_b, but we don't know these. We also don't know how many of each kind of coin are in the bag; so the probability, α_a, of drawing a coin of the first type is also unknown (and since $\alpha_a + \alpha_b = 1$, we do not need to separately estimate α_b, the probability of drawing a coin of the second type).

What we have is n pieces of data: for each data point, someone reached into the bag, pulled out a random coin, tossed it ℓ times and then reported the number h_i which was the number of times it came up heads. The coin was then put back into the bag. This was repeated n times. The resulting n head-counts $(h_1, h_2, h_3, \ldots, h_n)$ constitute our data.

Our goal is to estimate p_a, p_b, α_a from this data in some reasonable way.

For this problem, the binomial distribution can be good to have handy:

$$P(H = h) = \binom{\ell}{h} p^h (1-p)^{\ell - h}$$

for the probability of seeing exactly h heads having tossed a coin ℓ times with each toss independently having probability p of turning up heads. Also recall that the mean and variance of a binomial distribution are given respectively by ℓp and $\ell p(1-p)$.

(a) (10 pts) **How would you adapt the main ideas in the k-means algorithm to construct an analogous approach to estimating $\widehat{p}_a, \widehat{p}_b, \widehat{\alpha}_a$ from this data set? Give an explicit algorithm, although it is fine if it is written just in English.**

Solution: They key ideas of k-means are: (1) introduce some notion of each point belonging to each "cluster"–here the analogue to cluster from k-means is which coin the data came from, either a or b, (2) introduce some parametric way to define each cluster, such as a "centroid"–here one analogue to the centroid vector could be the scalar representing the probability of heads for each coin, p_a and p_b, (3) introduce some notion of distance between each observed data point and each "cluster", such as the Euclidean distance between a data point and a centroid center—here the analogue could be something like $d(h_i, \widehat{p}_a; l) = |\frac{h_i}{l} - \widehat{p}_a|$, (4) an iterative procedure that goes back and forth between assigning each point fully to one cluster (coin) or the other and updating the "centroid" parameters.

Thus, one example of a solution would have been:

(a) Randomly initialize the values $\widehat{p}_a$ and $\widehat{p}_b$ between 0 and 1. (Equivalently could instead have initialized the cluster assignments, in which case reverse the next two steps).

(b) Assign each data point to coin a if $|\frac{h_i}{l} - \widehat{p}_a| >= |\frac{h_i}{l} - \widehat{p}_a|$, and otherwise coin b. Denote this assignment, z_i, where $z_i \in \{a, b\}$.

(c) Update the parameter $\widehat{p}_a$ using $\widehat{p}_a = \frac{1}{|\{i|z_i=a\}|} \sum_{h_i|z_i=a} \frac{h_i}{l}$.

(d) Repeat the two steps above until convergence, where convergence could be that the parameters have stopped moving more than T away from where they were at the last iteration.

(e) Set the parameter α_a using $\alpha_a = \frac{1}{n} \sum_{h_i | z_i = a} 1$. Similarly for α_b.

(b) (8 pts) Suppose that the true $p_a = 0.4$ and the true $p_b = 0.6$ and $\alpha_a = 0.5$, and $\ell = 5$. For $n \to \infty$, **will your "k-means" based estimates (those from the preceding question) for $\widehat{p}_a$ and $\widehat{p}_b$ yield the correct parameter estimates (namely, $\widehat{p}_a = 0.4$ and $\widehat{p}_b = 0.6$)? Why or why not?**

Hint: Draw a sketch of the typical histograms of the number of heads of each coin on the same axes.

Solution: The binomials have means of 2 and 3 for class a and b respectively and variance of 1.2. Thus if one draws out the histograms on the same plot, we see that their distributions overlap by a substantial amount. Because "K-means" does "hard" assignment of each point to each "cluster" (coin), then this overlap will cause problems since it is not clear which coin each data point was generated from. To properly handle this we would need probabilistic assignments, but "k-means" uses hard assignments. Thus, because of this ambiguity, some data which were truly generated from class b will get assigned to class a, and vice-versa, even as the algorithm progresses–it is inherent to the hard assignments. Thus the final estimates for $\widehat{p}_a$ and $\widehat{p}_b$ will be biased (and thus incorrect even with infinite data). In particular, $\widehat{p}_a$ will be too low and $\widehat{p}_b$ will be too high.

(c) (17 pts) How would you adapt the EM for Gaussian Mixture Models that you have seen to construct an EM algorithm for estimating $\widehat{p}_a, \widehat{p}_b, \widehat{\alpha}_a$ from this data set?

You don't have to solve for the parameters in closed form, but (i) **write down the E-step update equations (i.e. write down the distributions that should be computed for the E-step — not in general, but specifically for this problem) and (ii) the objective function that gets maximized for the M-step and also what you are maximizing with respect to (again, not just the general form, but specific to this problem).** If you introduce any notation, be sure to **explain what everything means. Explain in words what the E- and M-steps are doing on an intuitive level.**

Solution:

Intuitively, in English: The E-step fills in the "missing" data (the assignment of data point to the coin it was generated from) in a probabilistic manner. The M-step uses this "probabilistically filled in data" to perform maximum likelihood on the data.

Formally now. First, let $z_i \in \{a, b\}$ represent which class data item i was generated from.

Then, in the E-step, we compute the posterior over this hidden variable for each data point. In particular we compute

$$p(z_i = a|\hat{p}_a, \hat{p}_b, \hat{\alpha}_a l, h_i) = \frac{r_i^a}{r_i^a + r_i^b}$$

where $r_i^a \equiv \alpha_a \text{Binomial}(H = h_i; \hat{p}_a, l)$. Similarly, compute the analogue for class b.

Then, in the M-step, we introduce a short-hand notation, $q_n^a \equiv p(z_i = a|\hat{p}_a, \hat{p}_b, \hat{\alpha}_a l, h_i)$ and similarly for q_n^b. Using this, we then need to maximize the likelihood of each binomial distribution, using the "soft" assignment of data to each binomial:

$$\hat{p}_a = \arg\max_{p_a} \sum_i q_i^a \log \text{Binomial}(H = h_i; \widehat{p}_a, l)$$

$$\hat{p}_a = \arg\max_{p_b} \sum_i q_i^b \log \text{Binomial}(H = h_i; \widehat{p}_b, l)$$

$$\hat{\alpha_a} = \frac{\sum_i q_i^a}{n}$$

Implicitly, it is also true that $\hat{\alpha_b} = 1 - \hat{\alpha_a}$.

One could instead have written down the objective function for $\hat{\alpha}_a$ as requested:

$$\hat{\alpha}_a = \arg\max_{\alpha_a} \sum_n q_n^a \log(\alpha_a) + q_n^a \log \text{Binomial}(H = h_n; p_a, l) + \ldots$$
$$\ldots + q_n^b \log(1 - \alpha_a) + q_n^b \log \text{Binomial}(H = h_n; p_b, l)$$

4 Convergence of Boosting (30 + 10 bonus points)

This question is about how boosting can reduce training error with each iteration.

Recall the Adaboost set-up: Given a set of training points $(\mathbf{x}_i, y_i)_{i=1,\ldots,n}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{-1, 1\}$, we build a classifier, F_t out of "simpler" classifiers G from the class of models, $\mathcal{G}$ (where G returns $+1$ or -1 given input $\mathbf{x}$). We do so iteratively using the following updates:

Initialize sample weights, $w_i^0 = 1/n$ for all $i \in \{1, \ldots, n\}$, and $F_0 = 0$, and iterate for $t = 1, \ldots, T$:

1. set $G_t = \arg\min_{G \in \mathcal{G}} \sum_{i=1}^{n} w_i^{t-1} \mathbb{I}\left[y_i \neq G(\mathbf{x}_i)\right]$,
2. compute $\epsilon_t = \sum_{y_i \neq G_t(\mathbf{x}_i)} w_i^{t-1}$ and then compute $\alpha_t = \frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t}$,
3. update $F_t = F_{t-1} + \alpha_t G_t$,
4. compute $w_i^t = \frac{\exp\left(-y_i F_t(\mathbf{x}_i)\right)}{\sum_{j=1}^{n} \exp\left(-y_j F_t(\mathbf{x}_j)\right)}$,
5. set $t = t + 1$ and repeat steps $1 - 5$,

where T is suitably chosen. The final classification rule for any point with feature $\mathbf{x}$ is given by $\hat{y} = \text{sgn}\left(F_T(\mathbf{x})\right)$.

Also recall that classification error and exponential loss for classifier F on data $\{(\mathbf{x}_i, y_i)\}$ are given by

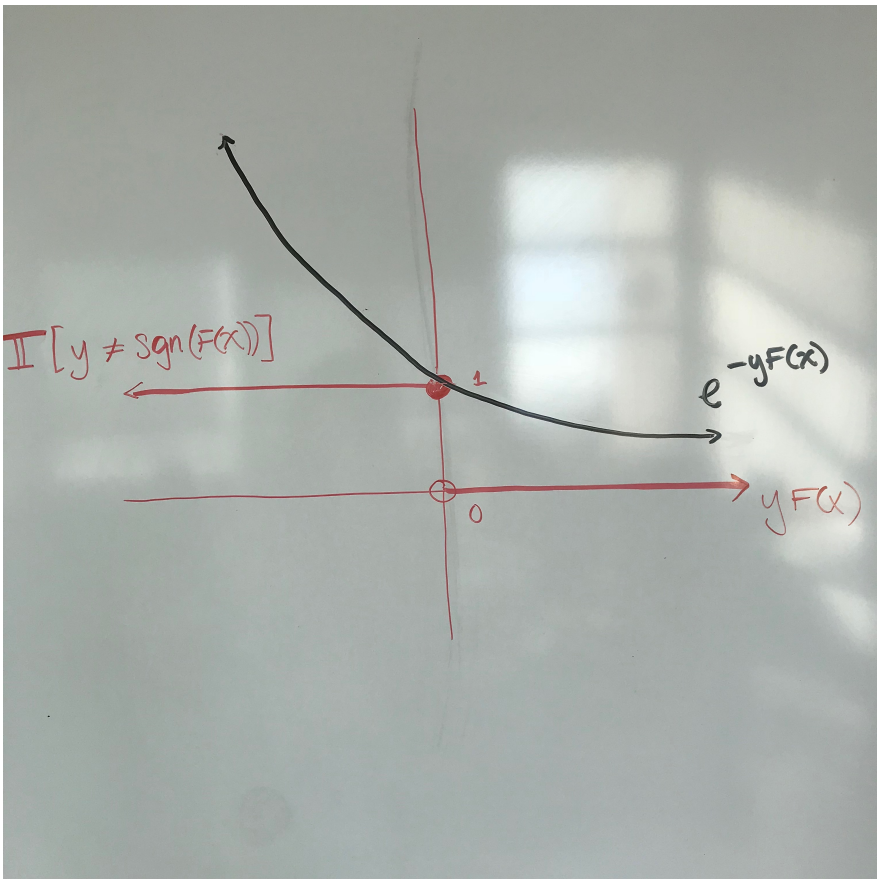
$$\mathcal{L}_{\text{class}}(F, \{(\mathbf{x}_i, y_i)\}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[y_i \neq \text{sgn}(F(\mathbf{x}_i))\right], \quad \text{and} \tag{1}$$

$$\mathcal{L}_{\text{exp}}(F, \{(\mathbf{x}_i, y_i)\}) = \frac{1}{n} \sum_{i=1}^{n} e^{-y_i F(\mathbf{x}_i)}. \tag{2}$$

(a) (5 pts) **Make a plot of $\mathbb{I}\left[y \neq \text{sgn}(F(\mathbf{x}))\right]$ and $e^{-yF(\mathbf{x})}$, with these losses on the vertical axis and $yF(\mathbf{x})$ on the horizontal axis. Explain briefly why can we conclude that**

$$\mathcal{L}_{\text{class}}(F) \leq \mathcal{L}_{\text{exp}}(F).$$

Solution: The plot of $\mathbb{I}\left[y \neq \text{sgn}(F(\mathbf{x}))\right]$ and $e^{-yF(\mathbf{x})}$ looks as follows:



We have

$$\mathbb{I}\left(y_i \neq \text{sgn}(F(\mathbf{x}))\right) = \begin{cases} 1 & \text{if} y \neq \text{sgn}(F(\mathbf{x})) \\ 0 & \text{if} y_i = \text{sgn}(F(\mathbf{x})) \end{cases}$$

When $y \neq \text{sgn}(F(\mathbf{x}))$, then clearly $y\text{sgn}(F(\mathbf{x})) < 0$ and consequently $yF(\mathbf{x}) < 0$ as well. Therefore, clearly $e^{-yF(\mathbf{x})} > 1$. So the exponential loss is above the classification loss in these cases for these kinds of points.

When $y = \text{sgn}(F(\mathbf{x}))$, then clearly $y\text{sgn}(F(\mathbf{x})) > 0$ and consequently $yF(\mathbf{x}) > 0$ as well. Therefore, clearly $0 < e^{-yF(\mathbf{x})} < 1$. So the exponential loss is above the classification loss for these kinds of points as well.

This is what the figure illustrates.

Since $\mathcal{L}_{\text{exp}}(F)$ and $\mathcal{L}_{\text{class}}(F)$ are just **positively** weighted sums of these quantities, we can just add together weighted inequalities for each of the $(\mathbf{x}_i, y_i)$ pairs to obtain that

$$\mathcal{L}_{\text{class}}(F) \leq \mathcal{L}_{\text{exp}}(F).$$

(b) (**Bonus 10 pts**) It turns out that some algebra shows that the exponential loss for the Adaboost updates satisfies

$$\mathcal{L}_{\text{exp}}(F_t) = \frac{1}{n}\left(e^{\alpha_t} \sum_{y_i \neq G_t(\mathbf{x}_i)} e^{-y_i F_{t-1}(\mathbf{x}_i)} + e^{-\alpha_t} \sum_{y_i = G_t(\mathbf{x}_i)} e^{-y_i F_{t-1}(\mathbf{x}_i)} \right)$$

Building on this, show the following (hints below):

$$\mathcal{L}_{\textbf{exp}}(F_t) = \mathcal{L}_{\textbf{exp}}(F_{t-1}) \cdot 2\sqrt{\epsilon_t(1-\epsilon_t)},$$

where F_t and ϵ_t are as defined earlier in this problem. [*Hints: (i) write the losses in terms of the weights:* $e^{-y_i F_{t-1}(\mathbf{x}_i)} = w_i^{t-1} \sum_{j=1}^n \exp\left(-y_j F_{t-1}(\mathbf{x}_j)\right)$*, (ii) use the definition of the exponential loss in* (2)*, (iii) expand* e^{α_t} *in terms of* ϵ_t].

Since the exponential loss at time 0 is clearly $\mathcal{L}_{\text{exp}}(F_0) = \frac{1}{n}\sum_{i=1}^n e^{-y_i 0} = 1$, induction immediately will tell us that the final exponential loss $\mathcal{L}_{\text{exp}}(F_T)$ is thus given by (no need to show this)

$$\mathcal{L}_{\text{exp}}(F_T) = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}.$$

.

Solution:

For your convenience, we are including parts of the derivation that we told you to assume in the problem so that you can see the whole story.

Since $F_0 = 0$, we have $e^{-y_i F_0(\mathbf{x}_i)} = 1$ and hence $\mathcal{L}_{\text{exp}}(F_0) = 1$.

Note that

$$y_i G_t(\mathbf{x}_i) = \begin{cases} 1 & \text{if} \quad y_i = G(\mathbf{x}_i) \\ -1 & \text{if} \quad y_i \neq G(\mathbf{x}_i). \end{cases}$$

Consequently, we have

$$\begin{aligned}
\mathcal{L}_{\text{exp}}(F_t) = \frac{1}{n}\sum_{i=1}^n e^{-y_i F_t(\mathbf{x}_i)} &= \frac{1}{n}\sum_{i=1}^n e^{-y_i(F_{t-1}(\mathbf{x}_i) + \alpha_t G_t(\mathbf{x}_i))} \\
&= \frac{1}{n}\sum_{i=1}^n e^{-y_i F_{t-1}(\mathbf{x}_i) - \alpha_t y_i G_t(\mathbf{x}_i)} \\
&= \frac{1}{n}\left(\sum_{y_i \neq G_t(\mathbf{x}_i)}^n e^{\alpha_t} e^{-y_i F_{t-1}(\mathbf{x}_i)} + \sum_{y_i = G_t(\mathbf{x}_i)}^n e^{\alpha_t} e^{-y_i F_{t-1}(\mathbf{x}_i)} \right) \\
&= \frac{1}{n}\left(e^{\alpha_t} \sum_{y_i \neq G_t(\mathbf{x}_i)} e^{-y_i F_{t-1}(\mathbf{x}_i)} + e^{-\alpha_t} \sum_{y_i = G_t(\mathbf{x}_i)} e^{-y_i F_{t-1}(\mathbf{x}_i)} \right)
\end{aligned}$$

At this point is where you were asked to proceed. Following the hint and substituting in the expression for the weight:

$$\mathcal{L}_{\text{exp}}(F_t) = \frac{1}{n}\left(e^{\alpha_t}\sum_{y_i \neq G_t(\mathbf{x}_i)} w_i^{t-1} + e^{-\alpha_t}\sum_{y_i = G_t(\mathbf{x}_i)} w_i^{t-1}\right)\cdot\left(\sum_{i=1}^{n} e^{-y_i F_{t-1}(\mathbf{x}_i)}\right)$$

Now note that $e^{\alpha_t} = \sqrt{(1-\epsilon_t)/\epsilon_t}$ and $\sum_{y_i \neq G_t(\mathbf{x}_i)} w_i^{t-1} = \epsilon_t$ and hence

$$\begin{aligned}
&\frac{1}{n}\left(e^{\alpha_t}\sum_{y_i \neq G_t(\mathbf{x}_i)} w_i^{t-1} + e^{-\alpha_t}\sum_{y_i = G_t(\mathbf{x}_i)} w_i^{t-1}\right)\cdot\left(\sum_{i=1}^{n} e^{-y_i F_{t-1}(\mathbf{x}_i)}\right) \\
&\quad= \left(\sqrt{\frac{(1-\epsilon_t)}{\epsilon_t}}\cdot\epsilon_t + \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}\cdot(1-\epsilon_t)\right)\cdot\underbrace{\frac{1}{n}\sum_{i=1}^{n} e^{-y_i F_{t-1}(\mathbf{x}_i)}}_{\mathcal{L}_{\text{exp}}(F_{t-1})} \\
&\quad= 2\sqrt{\epsilon_t(1-\epsilon_t)}\cdot\mathcal{L}_{\text{exp}}(F_{t-1}),
\end{aligned}$$

as claimed. This is as far as you were asked to go in the problem.

Recursing this update from $t = 1$ upto $t = T$, we find that

$$\mathcal{L}_{\text{exp}}(F_T) = \mathcal{L}_{\text{exp}}(F_0)\prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

since $\mathcal{L}_{\text{exp}}(F_0) = 1$.

(c) (10 pts) Suppose that at each step t, the chosen G_t has a weighted classification error that is strictly better than pure guessing, i.e., $\epsilon_t \leq 1/2 - \beta$ for some $\beta > 0$. By exploiting the fact that $\sqrt{1-x} \leq e^{-x/2}$ for all $x \in [0,1]$, it is easy to show that the exponential loss of the boosted classifier, $F_T(\mathbf{x})$, on the training set, decreases exponentially fast with number of boosting iterations T. In particular,

$$\mathcal{L}_{\text{exp}}(F_T) \leq e^{-2T\beta^2}.$$

Show that we achieve *zero classification error on the training set* of size n, for $T > \frac{\ln n}{2\beta^2}$ boosting iterations.

The following hint may or may not be useful: If we have 4 data points and the average classification error over them is less than $1/5$, how many data points can be incorrectly classified?

Solution:

Again, to help you get to the start of the problem, let us just assume that the β can vary with time for now. Substituting into the expression for exponential loss from the previous part, we find that

$$\begin{aligned}
\mathcal{L}_{\text{exp}}(F_T) &= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&= \prod_{t=1}^{T} 2\sqrt{\left(\frac{1}{2} - \beta_t\right)\left(\frac{1}{2} + \beta_t\right)} \\
&= \prod_{t=1}^{T} 2\sqrt{\frac{1}{4} - \beta_t^2} \\
&= \prod_{t=1}^{T} \sqrt{1 - 4\beta_t^2} \\
&\leq \prod_{t=1}^{T} e^{-2\beta_t^2} \\
&\leq e^{-2\sum_{t=1}^{T}\beta_t^2}
\end{aligned}$$

Using the fact that $\beta_t \geq \beta$, we have that

$$\mathcal{L}_{\text{exp}}(\mathbf{y}, F_T) \leq e^{-2T\beta^2}.$$

This is where you were expected to begin.

Now using part (a), we obtain that

$$\mathcal{L}_{\text{class}}(F_T) \leq \mathcal{L}_{\text{exp}}(F_T) \leq e^{-2T\beta^2}. \tag{3}$$

Note that zero classification error is equal to ensuring that

$$\mathcal{L}_{\text{class}}(F_T) < \frac{1}{n}$$

because this means that there is no data point that is wrongly classified. (If even one was wrong, we would pay at least $\frac{1}{n}$ average classification loss.)

Using the bound (3), we see that we want to find a T so that

$$\mathcal{L}_{\text{class}}(F_T) \leq e^{-2T\beta^2} < \frac{1}{n}.$$

If we plug in $\frac{\ln n}{2\beta^2}$ for T in the expression above, we see that $e^{-2T\beta^2} = \exp\left(-\frac{\ln n}{2\beta^2} 2\beta^2\right) = \frac{1}{n}$. Since $e^{-2T\beta^2}$ is clearly a decreasing function of T, the result is proved.

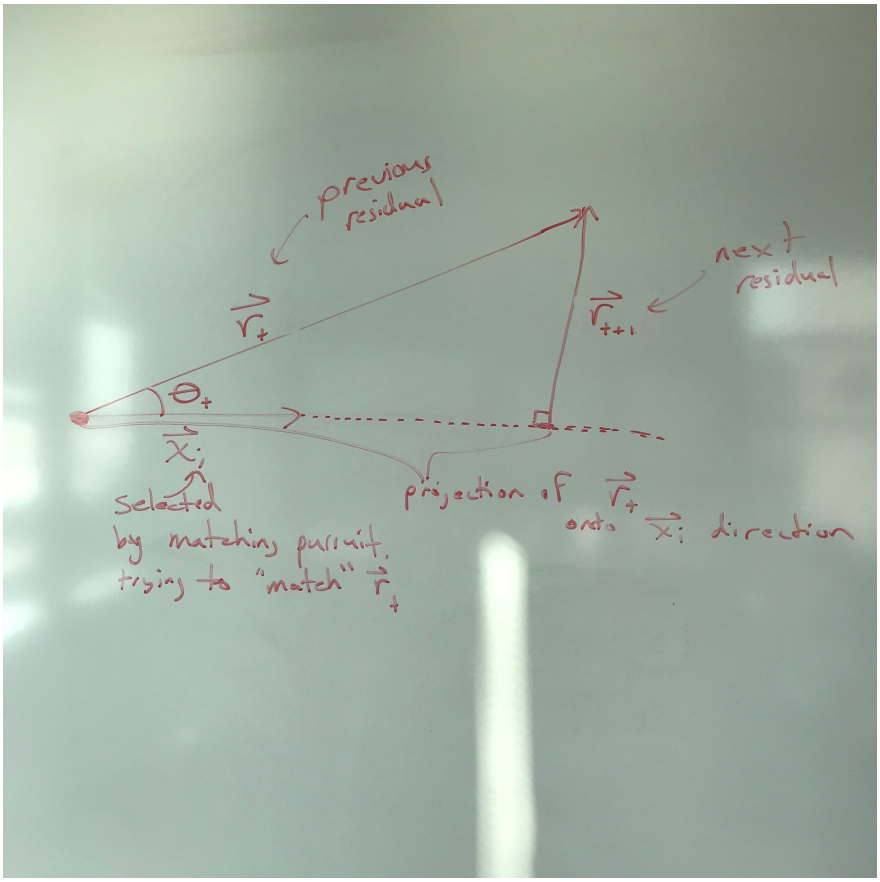
(d) (15 pts) Stepping back a bit, the condition that "the chosen G_t has a weighted classification error that is strictly better than pure guessing, i.e., $\epsilon_t \leq 1/2 - \beta$ for some $\beta > 0$" might seem a bit mysterious when applied to classification, even though its payoff of geometrically decreasing exp-loss is appealing.

Leverage the connection between matching-pursuit for least-squares problems and boosting to **state what you believe the analogous condition on the chosen columns of the feature matrix $\mathbf{X}$ could be and argue why it makes sense that this condition would induce an exponentially decreasing squared-error loss on the training data with matching-pursuit iterations.**

Hint: remember that the re-weighted points in AdaBoost correspond to the residuals in matching pursuit.

Second Hint: draw a labeled right triangle and recall the definition of $\sin\theta$ *and* $\cos\theta$*. Which sides of the triangle correspond to the previous residual, the next residual, and the direction of the feature column chosen by matching pursuit?*

Solution:



The goal in this problem is to find a condition that will allow us to guarantee that the squared-error loss for matching pursuit will converge exponentially to zero. Moreover, we want this condition to be "local" in that it is something that we want to assert about every "match" that

is found during the operation of matching pursuit. This is because the matches (best column to use) found by matching pursuit are the counterparts to the simple classifiers found during each stage of AdaBoost.

As in the figure above, let $\mathbf{x}_i$ be the feature vector chosen by matching pursuit at this iteration. The residual it was facing is $\mathbf{r}^t$. The "line search" that matching pursuit will do (corresponding to the setting of the α_t in AdaBoost) is going to involve projecting the previous residual $\mathbf{r}^t$ onto the feature vector that was just found to get an additive update to our predictor. The figure above shows that the residual $\mathbf{r}^{t+1}$ that will be left for the next iteration to start with will be what is left over — namely the component of $\mathbf{r}^t$ that is orthogonal to $\mathbf{x}_i$.

Notice that $\|\mathbf{r}^t\|^2$ represents the training error left at the t-th boosting stage. So, to get an exponentially decreasing loss analogous to what we found for AdaBoost with guaranteedly-good simple classifiers, we need to have $\frac{\|\mathbf{r}^{t+1}\|^2}{\|\mathbf{r}^t\|^2} < 1 - \beta^2 < 1$. At that point, induction would give us exponentially decreasing training losses.

So what is $\frac{\|\mathbf{r}^{t+1}\|}{\|\mathbf{r}^t\|}$? Simple trigonometry tells us that $|\sin\theta_t| = \frac{\|\mathbf{r}^{t+1}\|}{\|\mathbf{r}^t\|}$. Consequently, a condition that would work would be to assert that $|\sin\theta_t| < 1 - \beta < 1$ for some $\beta > 0$. This is equivalent, since $\sin^2\theta_t + \cos^2\theta_t = 1$ to requiring that there exists some $\beta' > 0$ so that $|\cos\theta_t| > \beta'$.

The advantage of cosines is that we can directly connect them to inner products. We want

$$\frac{\langle \mathbf{r}^t, \mathbf{x}_i \rangle}{\|\mathbf{r}^t\|\|\mathbf{x}_i\|} > \beta' > 0$$

For the vector $\mathbf{x}_i$ chosen by matching pursuit at this iteration. As long as this always happens, we are guaranteed to get exponential convergence of the training loss to zero with iterations.

In math,

$$\|\mathbf{r}^{t+1}\|^2 \leq \|\mathbf{r}^t\|^2(1 - \cos^2(\theta_t)) \leq \|\mathbf{r}^t\|^2(1-\beta^2) \leq \|\mathbf{y}\|^2(1-\beta^2)^T \leq \|\mathbf{y}\|^2 e^{-\beta^2 T}.$$

Although we did not ask for it, this will always happen as long as $\mathbf{y}$ is in the span of the features. This is also pretty clearly a very overfitting-inviting situation to be in, which should further motivate why early stopping is generally important for boosting-type algorithms.

What is less immediately obvious, although no less true, is that the arguments you saw in class when dealing with the exponential convergence of OLS also apply here. Even when $\mathbf{y}$ itself is not in the span of the features, we can look at its projection within that span. As far as matching pursuite is considered, that is all that it can see (since it just does inner products). So, it will exponentially converge to that — i.e. to the OLS solution. Once again, if you believe by now that OLS really should be regularized to be trusted in terms of generalization, the same applies for boosting. Early stopping provides one such regularization approach at the algorithmic level that favors sparsity.

5 SVMs for Novelty Detection (30 + 10 bonus points)

This problem is an SVM-variant that works with training data from only one class.

The classification problems we saw in class are two-class or multi-class classification problems. What would one-class classification even mean? In a one-class classification problem, we want to determine whether our new test sample is *normal* (not as in Gaussian), namely whether it is a member of the class represented by the training data or whether it is *abnormal*. One-class classification is also called *outlier detection*. In particular, we assume that all/most of the training data are from the normal class, and want to somehow model them, such that for new unseen test points, we can tell whether they "look like" these points, or whether they are different (i.e, abnormal).

For example, Netflix may want to predict whether a user likes a movie but only have thumbs-up data about movies that the user liked and no thumbs-down votes at all. How can we deal with learning with no negative training samples?

(a) (7 pts) Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be your training data for the one-class classification problem (all supposedly belonging to one class — the normal class). One way to formulate one-class classification using SVMs is to have the goal of finding a decision plane which goes through the origin, and for which all the training points are on one side of it. We also want to maximize the distance between the decision plane and the data points. Let the equation of the decision plane H be

$$H := \{\mathbf{x} \in \mathbb{R}^d | \mathbf{w}^\top \mathbf{x} = 0\}. \tag{4}$$

Let the margin m be the distance between the decision plane and the data points

$$m = \min_i \frac{|\mathbf{w}^\top \mathbf{x}_i|}{\|\mathbf{w}\|}. \tag{5}$$

If the convex hull of the training data does not contain the origin, then it is possible to solve the following optimization problem:

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{6}$$

$$\text{subject to} \quad \mathbf{w}^\top \mathbf{x}_i \geq 1, \quad 1 \leq i \leq n \tag{7}$$

Argue that in the above hard one-class SVM optimization (assuming that the convex hull of the training data does not contain the origin), **the resulting margin is given by** $\widehat{m} = \frac{1}{\|\widehat{\mathbf{w}}\|}$**.**

Solution: We claim that for some $j \in \{1, \ldots, n\}$, the constraint $\widehat{\mathbf{w}}^\top \mathbf{x}_j \geq 1$ holds with equality (i.e. $\widehat{\mathbf{w}}^\top \mathbf{x}_j = 1$). If this is not the case, then $\omega := \min_i \widehat{\mathbf{w}}^\top \mathbf{x}_i$ is strictly greater than 1. Thus we can make $\widehat{\mathbf{w}}$ smaller without breaking the constraints, as

$$\forall i \quad \left(\frac{\widehat{\mathbf{w}}}{\omega}\right)^\top \mathbf{x}_i = \frac{\widehat{\mathbf{w}}^\top \mathbf{x}_i}{\omega} \geq 1$$

yet

$$\left\|\frac{\widehat{\mathbf{w}}}{\omega}\right\| = \frac{\|\widehat{\mathbf{w}}\|}{\omega} < \|\widehat{\mathbf{w}}\|$$

contradicting the fact that $\widehat{\mathbf{w}}$ is optimal.

Since $\widehat{\mathbf{w}}^\top \mathbf{x}_i \geq 1$ for all i, with equality for at least one i, we have

$$\widehat{m} = \min_i \frac{|\widehat{\mathbf{w}}^\top \mathbf{x}_i|}{\|\widehat{\mathbf{w}}\|} = \frac{1}{\|\widehat{\mathbf{w}}\|}$$

as claimed.

(b) (**Bonus** 10 pts) The optimal $\widehat{\mathbf{w}}$ in the hard one-class SVM optimization problem defined by (6) and (7) is identical to the optimal $\widehat{\mathbf{w}}_{\text{two-class}}$ in the traditional two-class hard-margin SVM you saw in class using the augmented training data $(\mathbf{x}_1, 1)$, $(\mathbf{x}_2, 1)$, ..., $(\mathbf{x}_n, 1)$, $(-\mathbf{x}_1, -1)$, $(-\mathbf{x}_2, -1)$, ..., $(-\mathbf{x}_n, -1)$.

Argue why this is true by comparing the objective functions and constraints of the two optimization problems, as well as the optimization variables.

Solution: The traditional two-class hard-margin SVM problem writes

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad 1 \leq i \leq n \end{aligned}$$

Plugging our augmented training set into the above yields

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & \mathbf{w}^\top \mathbf{x}_i + b \geq 1, \quad 1 \leq i \leq n \\ & \mathbf{w}^\top \mathbf{x}_i - b \geq 1, \quad 1 \leq i \leq n. \end{aligned}$$

Note that for any value of b, the constraints of our one-class problem ($\mathbf{w}^\top \mathbf{x}_i \geq 1$) would be satisfied, since by adding up the two constraints associated with each datapoint we obtain

$$2\mathbf{w}^\top \mathbf{x}_i = (\mathbf{w}^\top \mathbf{x}_i + b) + (\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 + 1 = 2$$

Also note that the objective function, $\frac{1}{2}\|\mathbf{w}\|_2^2$, does not contain b. Thus, we are free to choose $b = 0$, which yields the optimization problem

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & \mathbf{w}^\top \mathbf{x}_i \geq 1, \quad 1 \leq i \leq n. \end{aligned}$$

which is precisely the one-class SVM problem.

(c) (3 pts) It turns out that the hard one-class SVM optimization cannot deal with problems in which the origin is in the convex hull of the training data. To extend the one-class SVM to such data, we use the hinge loss function

$$\max(0, 1 - \mathbf{w}^\top \mathbf{x}_i) \tag{8}$$

to replace the hard constraints used in the one-class SVM so that the optimization becomes

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n} \max(0, 1 - \mathbf{w}^\top \mathbf{x}_i). \tag{9}$$

Explain how the hyper-parameter $C > 0$ affects the behavior of the soft one-class SVM in (9).

Solution: When C is small, the one class SVM will optimize toward maximizing the margin while allowing more samples in the training data to be implicitly classified as outliers.

The larger C is, the harder the one class SVM will try to minimizing the number of training points implicitly classified as outliers. In the limit $C \to \infty$ we recover the hard-margin SVM, since every point must satisfy $\mathbf{w}^\top \mathbf{x}_i \geq 1$.

Finally, C controls what extent we fit (or overfit) our data. For example, if we find that the model is overfitting the training data (e.g., this could happen if there are some genuine outliers in the training data), then we could decrease C to alleviate the overfitting problem.

(d) (5 pts) Let $\widehat{\mathbf{w}}^\top \mathbf{x}$ be the score of sample $\mathbf{x}$ where $\widehat{\mathbf{w}}$ is the optimal $\mathbf{w}$ according to the training data and the soft one-class SVM in (9) with some particular C. We will classify a sample as an outlier if its score is below a threshold η. **Describe a procedure to find a threshold η so that about 5% of your training data will be classified as outliers.**

Solution: Let $\text{score}[i] \leftarrow \widehat{\mathbf{w}}^\top \mathbf{x}_i$. We sort the array score and let $\eta := \text{score}[0.05n]$, where n is the number of samples. Then, by construction, about 5% of the training data will have a score of at most η.

(e) (10 pts) The Lagrangian dual of the soft one-class SVM optimization problem is:

$$\widehat{\boldsymbol{\alpha}} = \arg\max_{\boldsymbol{\alpha}} \quad \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \mathbf{x}_i{}^\top \mathbf{x}_j \qquad (10)$$
$$0 \leq \alpha_i \leq C, \quad 1 \leq i \leq n.$$

Given a sample $\mathbf{x}_{\text{test}}$ and the $\widehat{\mathbf{w}}$ solution to the primal one-class-SVM optimization in (9), you could test whether it is an outlier by checking if $\widehat{\mathbf{w}}^\top \mathbf{x}_{\text{test}} < \eta$. **How can you test whether $\mathbf{x}_{\text{test}}$ is an outlier using the dual formulation given the training data and the optimal $\widehat{\alpha}$ from** (10)**?**

Solution: Recall that in the traditional two-class SVM problem,

$$\widehat{\mathbf{w}} = \sum_{i=1}^{n} \widehat{\alpha}_i y_i \mathbf{x}_i$$

Here there are no labels y_i, but we can derive a similar result. The Lagrangian of the one-class problem writes

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \alpha_i (1 - \mathbf{w}^\top \mathbf{x}_i)$$

By the KKT conditions, any optimal solution $(\widehat{\mathbf{w}}, \widehat{\boldsymbol{\alpha}})$ must satisfy

$$\mathbf{0} = \nabla_{\mathbf{w}} \mathcal{L}(\widehat{\mathbf{w}}, \widehat{\boldsymbol{\alpha}}) = \widehat{\mathbf{w}} - \sum_{i=1}^{n} \widehat{\alpha}_i \mathbf{x}_i$$

i.e.

$$\widehat{\mathbf{w}} = \sum_{i=1}^{n} \widehat{\alpha}_i \mathbf{x}_i$$

Thus, to check if $\widehat{\mathbf{w}}^\top \mathbf{x}_{\text{test}} < \eta$, we can test if

$$\sum_{i=1}^{n} \widehat{\alpha}_i \mathbf{x}_i{}^\top \mathbf{x}_{\text{test}} < \eta. \qquad (11)$$

(f) (5 pts) Your friend claims that linear models like the one-class SVM are too simple to be useful in practice. After all, for the example training data in Figure 1, it is impossible to find a sensible decision line to separate the origin and the raw training data. Suppose that we believe the right pattern for "normalcy" here is everything within an approximate annulus around the unit circle. **How could you use the one-class SVM to do the right thing for outlier detection with such data? Explain your answer.**

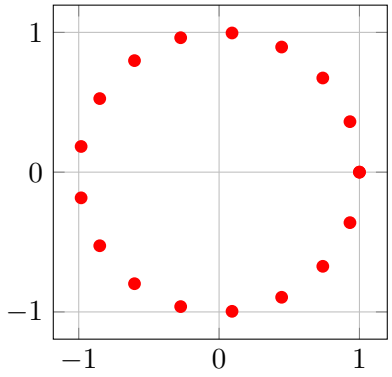


Figure 1: Counter example provided by your friend.

Solution:

The origin is in the convex hull of the data provided by your friend, so there exists no hyperplane that can separate the origin from this data. However, we can project the data into a higher dimensional space where our training data are linearly separable from the origin. We can either use the RBF/Gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\big(\gamma\|\mathbf{x}_1 - \mathbf{x}_2\|\big)$ for a suitable choice of γ, or we can add explicit features to lift the problem into a space wherein an annulus can be represented as being on one side of a hyperplane.

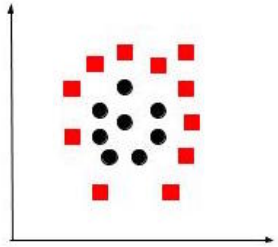
Note that it is *not* sufficient to use a quadratic kernel, although we could use a polynomial kernel of degree at least 4. We want to be able to detect both outliers within the circle of training data and those outside the circle. A quadratic kernel will not produce a boundary that separates the annulus from both its interior and its exterior.

6 Multiple Choice Questions (30 points)

For these questions, **select <u>all</u> the answers which are correct.** You will get full credit for selecting all the right answers. On some questions, real-valued partial credit will be assigned.

You will be graded on your best 15 of the 18 questions below. So feel free to skip three of them if you want.

(a) Which of the following classifiers can be used to classify the training data *perfectly*? Methods are not kernelized or with additional features unless explicitly stated so.

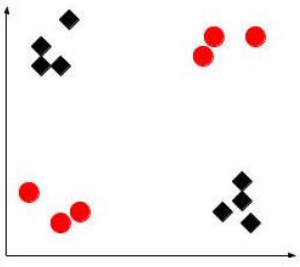


- ◯ Logistic regression
- ◯ 3-nearest neighbors
- ◯ SVM with a quadratic kernel
- ◯ Hard margin SVM
- ◯ LDA
- ◯ QDA

Solution:

We need a classifier that can express a quadratic (circular) decision boundary, and among these, the SVM with quadratic kernel and QDA are capable of doing so. 3-nearest neighbors will not work: some of the squares will be misclassified as circles. LDA has a hyperplane boundary between two classes and so will logistic regression and the hard-margin SVM.

(b) Which of the following classifiers can be used to classify the data *perfectly*? Methods are not kernelized or with additional features unless explicitly stated so.



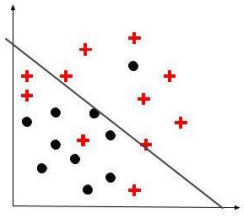
- ◯ Logistic regression without a kernel

- ◯ 3-nearest neighbors
- ◯ SVM with a quadratic kernel
- ◯ Hard margin SVM
- ◯ LDA
- ◯ QDA

Solution: 3-nearest neighbors works since the closest 2 neighbors (besides itself) for each training point are of the same class as the training point. An SVM with a quadratic kernel can express the boundary because a quadratic form can give us a hyperbola for the boundary. QDA is capable of separating the data because long diagonal ellipses can grab each class separately. (They might be tied in the middle, but we have no points there to worry about.)

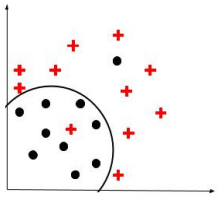
To get LDA to work, we would have to pretend that there are actually four different classes, but we only have two. Same for logistic regression and hard-margin SVMs.

(c) For each classifier below, choose whether the classifier has high "bias," high "variance," or it is good enough for the given problem. Select all that apply.



◯ High bias
◯ High variance
◯ Reasonable

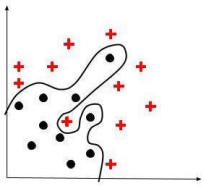
Solution: (a) The model (a straight line) has high bias but low variance. Depending on how you feel, you might also consider this reasonable given the number of data points.



◯ High bias
◯ High variance
◯ Reasonable

Solution:

The decision boundary appears very regular (a circle) — suggesting low variance — and capable of separating most of the data, suggesting low bias. This makes is reasonable. One might also say that it has high variance–it's hard to tell with so few data points.

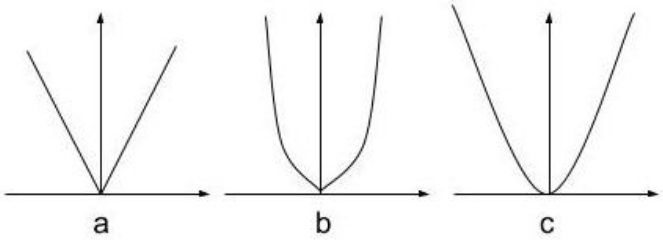


◯ High bias
◯ High variance
◯ Reasonable

Solution:

The decision boundary is irregular, following the quirks of the sampled data points, and this is a signature of "high variance". However, it classifies the points correctly, so there is no evidence of high bias. (for example, 1-NN can produce similar decision boundaries).

(d) Which of the following loss functions tends to result in a sparse solution? Select all that apply.



- ◯ a
- ◯ b
- ◯ c
- ◯ None

Solution:

(a) and (b) have a sharp corner at 0, so like the L_1 loss, these will tend to induce sparsity. (c) is smooth, and as with L_2 loss, will not induce sparsity.

(e) "Regularization" is a critical aspect of machine learning. Which of the following statements about regularization is true?

- ◯ the main function of regularization is to reduce the bias of your learned predictor
- ◯ the main function of regularization is to reduce the variance of your learned predictor
- ◯ regularization serves to combat overfitting
- ◯ regularization serves to combat underfitting
- ◯ Removing training data points is regularizing
- ◯ Adding noisy copies of training data points is regularizing
- ◯ Dropout is regularizing
- ◯ Doing dimensionality reduction as a part of the learning process has a regularizing effect.
- ◯ Adding a penalty term on the learned weights can be regularizing.
- ◯ Adding a prior for the learned weights can be regularizing.
- ◯ Averaging together the results of diverse predictors is regularizing.
- ◯ Adding features to a learning problem is regularizing.
- ◯ Using kernelized methods is always more regularizing than using direct methods.

Solution:

Regularization is fundamentally about combating overfitting. It does this by reducing the sensitivity on the training points. This manifests in a reduced variance, since the variance of the predictor is coming from the random variability in the training data. In order to do this, regularization often trades off accepting a larger bias — i.e. accepting more of a possibility of

underfitting. Having more training points reduces variance. So much so, that even adding lots of noisy copies of training points (either explicitly by augmenting our data set, or implicitly by techniques such as dropout) has a regularizing effect. Doing the opposite (removing entire training points) would have the opposite effect of making us more sensitive to the training points that remain.

Adding a prior is regularizing because it effectively makes us weigh each training point less. Adding a penalty term on the learned weights into our optimization behaves like adding a prior (namely a prior that the weights are small) and is hence regularizing. Dimensionality reduction is also regularizing (as you saw in the problem earlier connecting ridge regression to k-PCA-OLS). Adding features has the opposite effect of increasing dimensionality. Averaging together the results of diverse predictors is also regularizing for the same reason that averaging reduces variance in general.

Kernelized methods combine additional (implicit) features with an (implicit) prior. Consequently, depending on the hyperparameters, the kernelized method might be more or less regularizing than a direct method.

(f) Which of the following statements are true about step sizes for SGD and GD in order for them to eventually converge?

- ◯ both SGD and GD will always converge with a constant step size.
- ◯ both SGD and GD may require a decreasing step size to converge.
- ◯ both SGD and GD must have constantly increasing step sizes to converge.
- ◯ only GD requires a decreasing step size to converge, while SGD may converge using a constant step size.
- ◯ only SGD requires a decreasing step size to converge, while GD may converge using a constant step size.
- ◯ the convergence behavior depends on the model being used.

Solution:

This question is reasonably subtle. We saw in the homework that even straight-up gradient descent can oscillate with a constant step size if facing a function that looks like the 1-norm. In those cases, a decreasing step size is essential for convergence. Meanwhile in other cases like that arising in OLS, a constant step size can be fine. Stochastic gradient descent in general has noisy estimates for the gradient and hence typically requires decreasing step sizes in order to converge. (Although the Kaczmarz situation in which the solution to a least-squares problem can be found exactly (with zero loss) is a special case when convergence in expectation is possible with constant step sizes.)

In general, the convergence behavior does depend on the specific model of functions being optimized. However, increasing step sizes are a recipe for instability and the opposite of convergence. And typically, SGD needs smaller step sizes than GD. (Which makes sense because it needs to hedge its bets due to the lack of confidence in the gradient estimate.)

(g) The form of the decision boundary for a Linear Discriminant Analysis (LDA) based classifier

- ◯ is always linear
- ◯ is always quadratic
- ◯ is always cubic
- ◯ can take on any form
- ◯ depends on if the covariance matrices are diagonal
- ◯ depends on if the means for each class are are all the same

Solution:

LDA-based classifiers train a generative model that implicitly has a Gaussian pdf associated to each class. They learn separate means for each of the pdfs, but assume that there is a common covariance matrix. Because there is a common covariance matrix, the spectral theorem tells us that there is a linear change of coordinates to a system in which the covariances are all the identity. **There is no need for the covariance to be diagonal for this to happen.** For identity covariances, the decision boundaries are clearly hyperplanes separating the two classes. And hence, they are also hyperplanes in changed coordinates. (**There is no need to include quadratic or cubic terms, and the form of the decision boundary is quite restricted because of the strong underlying generative assumptions.**) If the means of each class are the same, then this becomes degenerate. There is no way for LDA to distinguish the two classes.

(h) K-means

- ◯ is like LDA and QDA in that it is a generative model for classification.
- ◯ is similar to EM in that during training it alternates between (i) assigning points to clusters and (ii) updating the parameters of the model – centroids for each class.
- ◯ is like Mixtures of Gaussians because it works on unsupervised data.
- ◯ always optimizes a loss which is equivalent to cross-entropy loss.
- ◯ is dissimilar to EM in that it hard assigns each training point to only one cluster during each training iteration.
- ◯ is guaranteed to find a global optimum.

Solution:

K-means is a hard-decision caricature of EM in which we alternate between assigning points to clusters and updating the centroids. **This finds a local minimum, but like all such local methods, might need to be repeatedly randomly started to gain any confidence that we are near a globally good solution.** It can work with points without any labels since the goal is not to predict but to cluster. In this way, it is very much like EM on a Gaussian Mixture Model, except that it makes hard decisions (every point belongs to exactly one cluster with no doubt) and it implicitly assumes identity covariances for all the Gaussians in the mixture. **The overall loss that k-means is optimizing is a squared-error-type loss (not a cross-entropy type)** in which the cluster centroids can be viewed as predictions of sorts for each of the points in the cluster.

The similarity to LDA/QDA type methods is only at the level of mean computation, but there is **neither any attempt to estimate a covariance (whether common to all classes like in LDA or individual to each class as in QDA)** nor any goal of recovering a generative model **to be used for classification.**

(i) Which of the following statements are true about loss functions?

- ◯ loss functions are usually learned from the data set
- ◯ cross-entropy loss and mean-squared loss are always identical.
- ◯ the lower the value of the loss function at the end of training, the better the generalization error.
- ◯ MSE loss can be derived from doing maximum likelihood on a linear regression model (with Gaussian iid noise).

Solution:

The "optimizing a (regularized) loss-function" paradigm is at the heart of modern machine learning. Typically, the loss function is chosen by hand at the beginning (**before looking at the data set at all**) and reflects a balance between computational tractability and relevance to the goals of the learning problem. Cross-entropy type losses are typically used for probability estimation (and probability estimation is the most common precursor to classification) while mean-squared type losses are common for estimation-type problems. **These are quite different from each other.** MSE loss can be understood as originating from a probabilistic inference setting when we want to do linear regression in the face of iid Gaussian noise.

The reason that we typically augment our loss function to include a parameter regularization/penalization term is precisely because overfitting is generically a problem. Beyond a point, **improved losses on training data are detrimental to generalization because those improvements reflect too much sensitivity to the peculiarities of the training data.** This causes the underylying true pattern to become somewhat obscured.

(j) Your organization wants to develop a predictor as a component of a self-driving car. It collects a lot of labeled data and randomly divides it into two parts A and B. It creates different teams to try different prediction techniques and gives every team access to A. The organization holds back B so that it can evaluate the performance of each team's predictor on B to choose which predictor to commit to for further development in the self-driving car. Your team has further divided the A data into two disjoint sets: A_V and A_T. Which of the following are true?

- ◯ If you use A_T to train your neural net predictor, you should use A_V to decide how many layers to use.
- ◯ If you use A_T to train a kernelized SVM, you should use A_V to pick the kernel parameters.
- ◯ If you use A_T to create a nearest-neighbor based predictor, you should use A_V to pick the number of neighbors.
- ◯ If you use A_T to train a boosted decision tree, you should use A_V to decide how many stages of boosting to use.

◯ Your team should try to figure out a way to get access to B so you can make the best predictor.

Solution:

Overfitting and underfitting are the bane of machine learning. One of the most fundamental techniques to deal with this is held-out data that is used to evaluate the quality of models while avoiding data incest. The set B in the problem above is what is sometimes called the "test set" but really, it is just a fair way to estimate quality and thereby compare the choice of models and parameter settings for what we really care about: the ability to generalize. **In general, any sort of access to B while making the predictor compromises its utility in allowing us to estimate the generalization performance of a predictor and is thus a very bad thing.**

We use the set A to actually arrive at a predictor. Here, we hold out some data A_V to allow for a data-driven choice for "hyperparameters" — which is a fancy word for parameters that are hard to reasonably estimate from data while also training other parameters. The reason is that unless we somehow use held-out data, the hyperparameters will tend to want to go to extreme values that represent overly complex models (that can overfit).

So, for neural nets, as the number of layers increases, the model becomes more complex and can better fit training data. So, the hold-out set A_V lets us estimate generalization performance and pick the number of layers that balances the tension between underfitting (approximation error dominates) and overfitting (estimation error dominates).

For a kernelized SVM, the kernel parameters (for example, the σ^2 of the RBF (sometimes called γ) controls how large a neighborhood around a training point feels the influence of that point) determine how sensitive the resulting decision boundary is to the individual training points. Left to itself, a high value of σ^2 would result in very low training error. Similarly, in nearest-neighbor based predictors, the number of neighbors has a similar effect. If it is set to 1, then each training point has a substantial individual influence in its local neighborhood. Increasing the number of neighbors creates an averaging effect that lessens the sensitivity to individual point variation. The hold-out set A_V lets us estimate generalization performance and pick the smoothing parameter σ^2 or the number of neighbors k that balances the tension between underfitting (non-responsive boundaries) and overfitting (boundaries that contort too much).

In boosted decision trees, every boosting stage increases the complexity of the model and allows us to better fit the training data. As you have seen, often by letting the number of stages get large enough, we can even achieve zero classification error on the training set. If the original labels were noisy, this is probably a bad thing and a symptom of overfitting. The hold-out set A_V lets us estimate generalization performance and effectively stop the boosting stages before they overfit too much.

(k) Consider using k-nearest-neighbors for the problem of classification. Which of the following are true?

- ◯ The training classification error on the training data used to create the 1-nearest-neighbor classifier is always zero.
- ◯ As the number of neighbors k gets very large and the number of training points goes to infinity, the probability of error for a k-nearest-neighbor classifier will tend to approach the probability of error for a MAP classifier that knew the true underlying distributions.
- ◯ Reducing k has a regularizing effect.
- ◯ If the training data is linearly separable with a positive margin, then the decision boundary found by 1-nearest-neighbors will always be a hyperplane.
- ◯ k-nearest-neighbors can be regularized by dimensionality reduction via PCA applied to the training data.

Solution:

Nearest-neighbor techniques are one of the most fundamental approaches to machine learning because their conceptual simplicity helps us understand why machine learning is fundamentally tied to the idea of interpolation, depends on the vagaries of what training points we get, and why overfitting is always a possibility as a result. They have a "purity" to them in that by default, there are no parameters per-se being fit from the training data. A consequence is that the decision boundaries and the like will not have any strict easy to interpret shape, even when the data comes from a nice pattern. For example, even if the training data is linearly separable with a large positive margin, the decision boundary of k-nn will "wiggle" based on where exactly the training points from each class near the margin ended up being. **It won't always be a clean hyperplane.**

Nearest neighbor techniques also help us understand why learning is possible and the importance of regularization and why the training loss doesn't necessarily predict what generalization performance will be. After all, the training classification error on the training data used to create the 1-nearest-neighbor classifier is always zero since every point by definition is its own nearest-possible neighbor. The number of neigbors k is the native regularizing parameter and **increasing it** has a regularizing effect. This can be seen most dramatically by the fact that as the number of neighbors k gets very large and the number of training points goes to infinity, the probability of error for a k-nearest-neighbor classifier will tend to approach the probability of error for a MAP classifier that knew the true underlying distributions. This is a consequence of the weak law of large numbers as applied to the mode — since the k nearest neighbors will represent a local sampling of the underlying distribution of the data very close to the query point.

Because nearest-neighbor models have a strong dependence on the dimensionality of the data (with ever increasing amount of data needed to have neighbors close enough), dimensionality reduction techniques are an important regularization approach. Doing PCA on the training data to pick out the dominant dimensions of variation is one such approach.

(l) Consider using a random forest of depth-1 trees for least-squares regression where the individual trees are formed by randomly picking a feature and then randomly chosing a value for it to split on. All random choices for the trees in the random forest are made i.i.d. Which of the following are true?

- ◯ Such individual regression trees are always going to be unbiased estimators.
- ◯ The expected (over both the training data and the random choices used to construct the trees) bias of the random forest estimator will always be the same as the expected bias of a single random tree.
- ◯ The variance (with an expectation over both the training data and the random choices used to construct the trees) of the random forest estimator will be a non-increasing function of the number of trees in the random forest.
- ◯ The random forest estimator will tend to become unbiased as the number of trees in it increases.

Solution:

Random forests for least-squares regression involve creating a set of randomized trees where at the leaves of the trees, predictions are made with a prediction rule, such as the average of data points ending at that leaf node. The splits can be made in random ways both for the effective feature used to split on as well as the threshold used to do the split. The random forest then averages together the predictions of the individual trees.

Because this depends on the vagaries of the underlying data-generating distribution and the way that random choices are made, there is **no reason to believe that the individual decision trees will be unbiased.** However, because of the linearity of expectation, the average of the forest is going to be the same as the average for an individual randomized tree and so the forest has the same bias as an individual randomized tree. The biasedness or unbiasedness of the forest won't change as we increase the number of trees k.

However, because of the behavior of the average of a bunch of independent random variables, the variance of the forest estimator will be decreasing like $\frac{1}{k}$ if the forest has k trees in it. This is the primary function of random forests — they reduce the variance of tree-based estimators, which tend to be high-variance by their nature because of their very flexible nature. (Trees are, after all, universal function approximators. With increasingly complex functions representable as the depth increases.)

(m) You have a function that is implemented as a black box executable to which you have lost the source code. (You can run it as much as you would like though.) The function has three inputs that are a True/False Boolean, a real number from $[-1, +1]$, and an integer from 1 to 10. The function returns true or false. You decide to learn an approximation to this function using machine learning and use the black box executable to generate training data. Which of the following are reasonable things to do in this situation?

- ◯ Representing the Boolean input using ± 1.

- ◯ Check our performance using a validation set that was drawn separately from the training set
- ◯ Try using AdaBoost with decision trees to fit the training data
- ◯ Try using OLS to fit the training data
- ◯ Represent the integer input using a real number but scaling and shifting it to be in the range $[-1, +1]$.
- ◯ Represent the integer input using a vector of length 10 and one-hot-encoding.
- ◯ Divide the problem into 20 sub-problems based on the Boolean and integer values, and just learn a 1-dimensional classifier for each one using a binary decision tree on the remaining real input.
- ◯ Do dimensionality reduction using random projections to regularize this problem

Solution:

This question sets up a special situation in which we want to use machine learning. What is special? Because we have black-box access to a function, there is no observation noise. After all, our goal is simply to reconstruct the function whose source code we have lost. We know that the function is representable using a combination of ALU (arithmetic logic unit) operations and branches (because the function is defined by a computer program). The other special thing is that because we have black-box access, we can get as much training data as we want. Because of this, the need to regularize is not as strong.

We still need to use a validation set to make sure that we have any confidence that we have learned the function well.

Because the function that we are learning is Boolean, this is like a binary classification problem. **For such problems, we have seen that using squared-error loss can be problematic**, with hinge-loss, exponential loss, and logistic-loss being significantly better since they do not implicitly penalize examples that are very firmly in the class.

All machine-learning situations face the issue of data representation. And in particular, we need to represent the Boolean input and the integer input. For Boolean inputs, representing as ± 1 works quite well. For integer inputs, there is always a tension. Do the numbers have semantic content in terms of how close they are to each other? If so, then representing the integer inputs to be normalized real numbers from $[-1, +1]$ is generally a useful thing. If there is no semantic content to the value per se and it is just a label, then one-hot encoding is useful. In fact, both representations can be used simultaneously as features.

Doing dimensionality-reduction by random projections is not really useful here because we do not actually have lots of natively continuous-valued features that we believe are redundant as a group. There are basically just two potential continuous dimensions here (one from the real input and the other from the integer).

When it comes to actually fitting the data and learning a function, using AdaBoost with decision trees is essentially always a good place to start for binary classification-type problems. Especially since there is no noise — the ability to drive the training error all the way to zero can be quite useful.

In this particular problem, the fact that there is a lot of potential data and only 20 different combinations of discrete variables means that just solving 20 one-dimensional classification problems is actually very doable.

(n) Suppose we have a neural network with one hidden layer and all linear activations. Then we create a second model which is similar, but with 50 hidden layers (of the same size). No parameters are tied. Which of the following are true?

- ◯ the 50-layer network has the same representational capacity as the 1-layer network.
- ◯ the 50-layer network has greater representational capacity as the 1-layer network.
- ◯ the 50-layer network is more likely to suffer from unstable gradients than the 1-layer network.
- ◯ if we let the 1-layer network be arbitrarily wide and replace the linear activations with sigmoid activations, this updated 1-layer network would have higher representational capacity (be able to model a larger class of funtions) than the original 50-layer network (with only linear activatoins).

Solution:

Neural nets using linear activations are basically just matrix multiplies. A 50-layer network can represent any function that is representable by a product of matrices — which is the same as what is representable by a single matrix. So, the representational capacity of the 50-layer network is the same as 1-layer. If however, we allow for nonlinear operations like sigmoids, then we can represent a wider class of mappings. After all, with sigmoids as approximate steps, with enough nodes in the hidden layer, we can approximate any piecewise constant function — this is a bigger set of functions than just linear.

The challenge with multiple layers is that the gradient information has to flow backwards from the final loss to the earlier layers. As it does so, it gets multiplied along the way. This repeated multiplication can become unstable and cause the gradient to grow exponentially in the number of layers.

(o) Which of the following statements about Convolutional Neural Networks (CNN) are true?

- ◯ the convolution operation is invariant to image rotation
- ◯ a CNN has fewer parameters than a fully connected network when the number of hidden units (and input size and number of outputs) is the same
- ◯ a CNN has more parameters than a fully connected network when the number of hidden units (and input size and number of outputs) is the same
- ◯ the convolutional filters must be designed by hand because they cannot be learned by gradient descent

Solution: A convolutional neural net has convolution operations that are invariant (in the sense that LTI systems are invariant to time-shifts) to image translation, but not invariant to image rotation, resizing or brightness changes. Or anything else like that. For a fixed number of input and output nodes, it has fewer parameters than a fully connected network because the weights are shared and thus it has a stronger regularization effect than fully connected networks. However, for this same reason (under the same conditions) it's a less expressive class of models than the FC networks.

As CNN weights/filters are like any other neural network weights, they can be learned from the data, and therein lies the power of both neural networks and CNNs.

(p) You designed a 3-layer, fully connected neural network with sigmoid activation functions for a classification task. The output layer is a linear layer followed by a softmax function. The whole network is trained by stochastic gradient descent with cross entropy loss. You observe that the accuracy on the training set is close to 100%, however, it performs poorly on the hold-out validation set. Which of these are likely to improve the validation performance?

- ◯ Increase the number of hidden units
- ◯ Reduce the number of hidden units
- ◯ Increase the ℓ_2 regularization weight
- ◯ Reduce the ℓ_2 regularization weight
- ◯ Add the use of dropout to the neural network training
- ◯ Train the network for more iterations/epochs

Solution: The high training accuracy and poor held-out performance is a classic signature of overfitting. Thus it would be correct to add stronger regularization or reduce model complexity. The correct answers are **Reduce the number of hidden units in the fully connected layer, Increase the ℓ_2 regularization weight, Add a dropout layer to the neural network. Trying to train the network longer on the training set will not improve the generalization performance, since we already suffer from overfitting–it is likely to make it worse, if anything.**

(q) The ReLU activation functions, $r(z)$, is less likely to contribute to vanishing gradients than the sigmoid activation function, $s(z)$, because:

- ◯ $s(z) >= r(z)$ for $z < 0$
- ◯ $r(z)$ always has a constant non-zero gradient for $z > 0$
- ◯ $r(z)$ is constant when $z < 0$
- ◯ $r(z = 0) = 0$ whereas $s(z = 0) = 0.5$

Solution:

The important thing about ReLUs from the perspective of vanishing gradients is the fact that they do not saturate in the positive regime. When an activation function saturates, it has zero gradient, and thus has higher potential to contribute to vanishing gradients. The sigmoid saturates for both high and low inputs, whereas ReLUs only saturates for low inputs. When an activation unit saturates, the gradient is zero, and gets multiplied by other factors/messages collected during backpropagation, contributing to the vanishing gradient problem. Thus the sole correct answer is $r(z)$ always has a constant non-zero gradient for $z > 0$;

(r) Backpropagation

- ◯ is a type of neural network model
- ◯ is a way of computing the gradient in neural networks
- ◯ yields better parameter estimation than using the chain rule for the same number of training iterations
- ◯ only works for cross-entropy loss
- ◯ prevents vanishing gradients by efficient local computation
- ◯ is efficient in part because it caches factors of the gradient that can be re-used
- ◯ has time complexity that is linear in the number of layers
- ◯ has time complexity that is quadratic in the number of layers

Solution:

Backpropagation is the efficient way to compute gradients with respect to lots of parameters in a computational flow graph. **It works for general loss functions (not just cross-entropy) and gives the same answer as would be obtained from doing the chain rule individually for the derivative of the loss with respect to each parameter.** Backprop is a form of "dynamic programming" that dramatically improves computational efficiency by caching factors of the gradient and reusing them in a way that exploits the structure of the computational flow graph. This has a huge effect — for example, in a network with fully-connected layers of the same width, it makes the gradient computation scale linearly with the number of layers — the same as the forward pass' complexity.

The problem of vanishing gradients is fundamental to deep computational flow graphs in which we could end up with the chain rule demanding lots of small numbers being multiplied together. **Backprop can only speed up the computation, it cannot eliminate the problem of vanishing gradients.**