Problem 1: Viterbi Algorithm [50 points]

(a) **[5 points]** By definition, $\delta_{y,t-1}$ is the probability of the most probable English sequence corresponding to the first $t-1$ Greek observations $(x_1, x_2, ..., x_{t-1})$ that end with the English character y. So, the most probable sequence including the next observation x_t corresponding to English character s is given by finding the best transition from any y to s and the emission from s to x_t. Thus for $s \in \{a, i, p, s\}$ and $2 \leq t \leq T$, the recurrence relation is

$$\delta_{s,t} = P(X_t = x_t | Y_t = s) \cdot max_{y \in \{a,i,p,s\}} P(Y_t = s | Y_{t-1} = y) \delta_{y,t-1}$$

The initial condition is

$$\delta_{s,1} = P(X_1 = x_1 | Y_1 = s) \cdot P_{START}(s) \quad (s \in \{a, i, p, s\})$$

(b) **[25 points]** In order to find the most likely English translations for each of the Greek encrypted words: a)$\eta\tau$, b) $\omega\eta\alpha$ and c) $\omega\alpha\gamma\tau$, we are going to fill the 2D dynamic programming table from left to right with memorizing the argmax state that maximizes $P(Y_t = s | Y_{t-1} = y)\delta_{y,t-1}$

- $\eta\tau \rightarrow is$

$\delta_{s,t}$	$t=1$	$t=2$
a	0.01	0.0032 (from i)
i	0.16	0.0008 (from i)
p	0.02	0.0040 (from i)
s	0.03	0.0384 (from i)

Table 1: $\delta_{s,t}$ for $\eta\tau$

- $\omega\eta\alpha \rightarrow psi$

$\delta_{s,t}$	$t=1$	$t=2$	$t=3$
a	0.02	0.01260 (from s)	0.0000945 (from s)
i	0.04	0.00540 (from s)	0.0001260 (from a)
p	0.04	0.00135 (from s)	0.0009450 (from a)
s	0.09	0.00270 (from s)	0.0008820 (from a)

Table 2: $\delta_{s,t}$ for $\gamma\alpha\omega$

- $\omega\alpha\gamma\tau \rightarrow pass$

$\delta_{s,t}$	$t=1$	$t=2$	$t=3$	$t=4$
a	0.01	0.0180 (from p)	0.000245 (from s)	0.0002646 (from s)
i	0.04	0.0045 (from p)	0.000180 (from a)	0.0000756 (from s)
p	0.10	0.0010 (from i)	0.00054 (from a)	0.0000567 (from s)
s	0.03	0.0035 (from p)	0.00378 (from a)	0.0004536 (from s)

Table 3: $\delta_{s,t}$ for $\omega\alpha\gamma\tau$

Therefore, the English decryption for Ian's password is "pass is sap".

(c) [**10 points**] Now we have m English characters and try to translate a Greek word of length T. Thus the Viterbi algorithm will decide mT entries in the 2D dynamic programming table where m comparisons are required to determine the maximum value in the recurrence relation for each entry. Assuming we memorize the argmax state in a similar fashion with part (b), the overall complexity of the Viterbi Algorithm is $O(Tm^2)$.
In contrast, the brute-force algorithm will investigate probabilities of all possible English translations of length T in which each letter varies one of m English characters. Since each sequence involves $O(T)$ multiplications, the overall complexity of the brute-force method is $O(Tm^T)$.

(d) [**5 points**] Note that the probability distribution of the marginal variables is obtained by summing over the variables being discarded. Thus the naive algorithm performs

$$P(x_1, ..., x_T) = \sum_{y_1,...,y_T} P(x_1, ..., x_T, y_1, ..., y_T).$$

after computing $P(x_1, ..., x_T, y_1, ..., y_T) = P(y_1)P(x_1|y_1)\prod_{t=2}^{t=T} P(x_t|y_t)P(y_t|y_{t-1})$. As we should sum over all possible m^T combinations of $(y_1, ..., y_T)$, it will take exponential time.

For the smarter algorithm, let's revisit the recurrence relation of the Viterbi algorithm that you found in part (a). If you switch the *max* into $\sum$, $\delta_{s,t}$ will correspond to the $P(x_1, ..., x_t, Y_T = s)$. Running the algorithm till the end, it will compute $\delta_{s,T} = P(x_1, ..., x_T, Y_T = s)$. Then summing over all possible s values,

$$P(x_1, ..., x_T) = \sum_s P(x_1, ..., x_T, Y_T = s) = \sum_s \delta_{s,T}$$

Since the Viterbi algorithm takes polynomial time[1] in computing $\delta_{s,T}$, the complexity of this sum-product algorithm will also take the polynomial time.

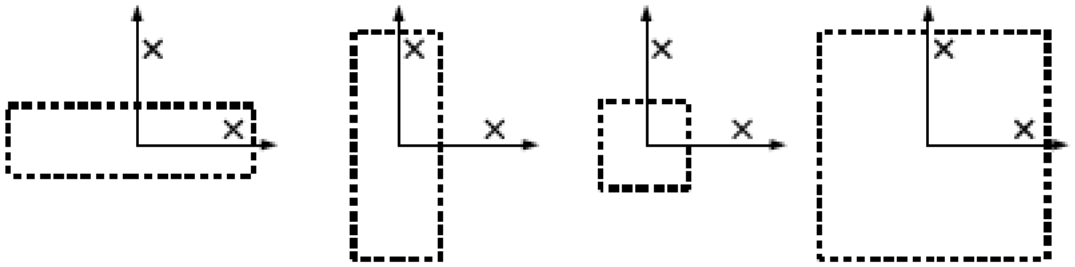
[1]Note that the only variation here is summing m numbers instead of comparing m numbers

(e) **[5 points]** Our HMM model would not work well because English sentence and Spanish sentence may have different word orders. For example, English has the object following the verb, whereas some other languages such as Korean and Japanese usually put their verbs at the very end of the sentence. In such cases, the HMM model will not perform well since it expects the same word alignment between two English and Spain sentences.

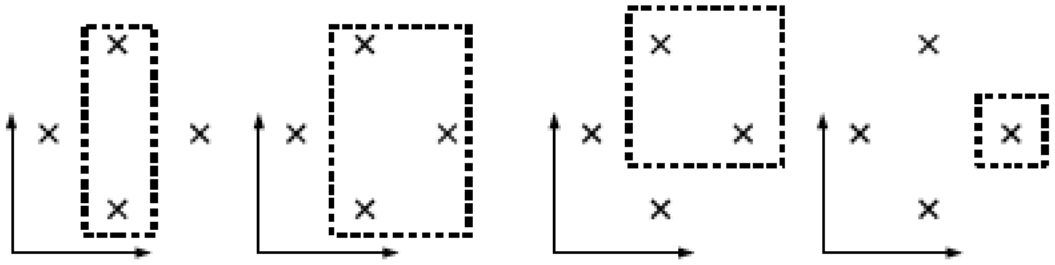
Problem 2: Statistical Learning Theory [50 points]

(a) **[20 points]** We are going to illustrate several pictures of d samples that each hypothesis class can shatter for finding the lower bound of VC dimension. Then we will roughly mention why it is unavailable to shatter $d + 1$ points.

- VC dimension ≥ 2
  Intervals on the real line can shatter any set of two points but no set of three points since the subset of the first and last points cannot be isolated. Thus, the VC-dimension of intervals is at least two (exactly two).
- VC dimension ≥ 4
  There exists a set of size four that can be shattered but no set of size five since the subset of first, third, and last point cannot be isolated. Thus, the VC-dimension of pairs of intervals is at least four (exactly four).
- VC dimension ≥ 2

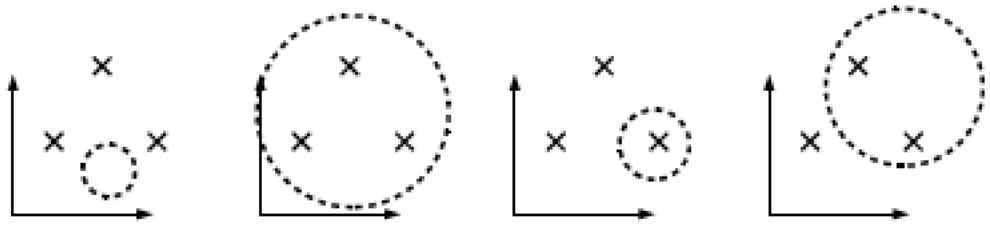


  (Note that rectangles centered at the origin cannot shatter three points. Let x^* be the farthest point (with respect to the origin). If there is a point inside the rectangle that has x^* as a vertex, label each point strictly inside as 0 and label rest of them as 1 including x^*. If no points belong to the inside of the given rectangle, label x^* as 0 and label the rest two points as 1. Since it is impossible to realize these labelings, VC dimension is at least two)
- VC dimension ≥ 4



(Note that rectangles with arbitrary origin cannot shatter five points. Imagine the minimum enclosing rectangle that has four points on each edge respectively. By construction, the the fifth point must lie either the inside or on an edge of the rectangle. Label 1 to four points on edges and label 0 to the fifth point. Since it is impossible to realize this labeling, VC dimension is at least 4)

- VC dimension ≥ 3



(Note that circles with arbitrary origin cannot shatter four points. On the one case that one point is inside the convex polygon of other three points, it is impossible to label 1 to three points outside and 0 to the one point inside. On the other case, it is impossible to realize alternating labels such as (1, 0, 1, 0) or (0, 1, 0, 1) when labeling them clockwisely. Thus VC dimension is at least 3)

(b) **[10 points]** Since H_n consists of all boolean functions mapping from $\{0,1\}^n$ to $\{0,1\}$, $|H_n| = 2^{2^n}$. See $VC(H_n) = 2^n$ because

- ($\geq$) By definition, H_n can realize all 2^{2^n} labelings on those 2^n points, which implies $VC(H_n) \geq 2^n$.
- ($\leq$) Note that the input domain is finite space consisting of 2^n distinct points. Thus $VC(H_n) \leq 2^n$.

(c) **[10 points]** We have an upper bound of the probability that the chosen classifier has at least ϵ amount of prediction error. From class we know that:

$$P(|Err_S - Err_P| \geq \epsilon) \leq 2|H|e^{-2m\epsilon^2}$$

Considering the probability of complementary events and zero training error:

$$P(Err_P < \epsilon) \geq 1 - 2|H|e^{-2m\epsilon^2} \geq 1 - \delta$$

Hence

$$\delta \geq 2|H|e^{-2m\epsilon^2} \implies \epsilon \geq \sqrt{\frac{1}{2m} \log \frac{2|H|}{\delta}}$$

Since each element of the weight vector can take one of two values (either -1 or 1), while the bias can take 31 possible values, $|H| = 31 \times 2^{100}$. Substituting the value in the above formula,

$$\epsilon \geq \sqrt{\frac{1}{2m} \log \frac{2 \cdot 31 \times 2^{100}}{\delta}}$$

Recall that ϵ is the upper bound on the prediction error (since we started with $P(Err_P \geq \epsilon)$). Thus, if we set $\delta = 2|H|e^{-2m\epsilon^2}$, we obtain an upper bound on the prediction error $\sqrt{\frac{1}{2m} \log \frac{2|H|}{\delta}}$.

Note – using the zero-error bound is incorrect when we only know that the training error is zero. However, we still gave you credit for using the other bound. Here is the argument:

Let's call all hypotheses $h \in H$ with $Err_P(h) > \epsilon$ "bad", and the others "good". Further, let's call $h \in H$ with $Err_S(h) = 0$ as "looking good", and the others as "looking bad". The key to the proof of the zero-error bound is the following line. nary

$$P(h_1 \text{ looks good OR } h_2 \text{ looks good OR} \ldots \text{ OR } h_k \text{ looks good } | h_1 \text{ bad}, h_2 \text{ bad}, \ldots, h_k \text{ bad})$$

where $h_1 \ldots h_k$ are the bad hypotheses. If we apply the zero-error bound only when we happen to draw a training set for with some h looks good (and we use some other bound when no h looks good) then we have to condition the line on that:

$$P(h_1 \text{ looks good OR } h_2 \text{ looks good OR } \ldots \text{ OR } h_k \text{ looks good } \quad | h_1 \text{ bad}, \ldots, h_k \text{bad}, \text{ some } h \text{ looks good})$$

This conditioning breaks the assumption that training error follow a binomial distribution. Why? Consider the case where ϵ is chosen so that all $h \in H$ are "bad" (one can do that if there is no h with zero prediction error). Then the probability is always 1, which is obviously not what the union bound of the binomial tells us.

(d) **[10 points]** In the case of non-zero training error, we learned

$$m \geq \frac{1}{2\epsilon^2} \ln \frac{2|H|}{\delta}$$

Substituting values we have:

$$m \geq \frac{1}{2 \times 0.1^2} \ln \frac{2 \times 31 \times 2^{100}}{0.1} \approx 3787.22$$

Hence we need at least $m = 3788$ examples.