CS 6780: Advanced Machine Learning **(Due: 09/21/23)**

Assignment #2: VC-Dimension and Perceptrons

Instructor: Thorsten Joachims *Name:* Student name(s), *Netid:* NetId(s)

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Please typeset your submissions in LaTeX. Use the template provided in `a2_template.tex` for your answers. Please include your name and NetIDs with submission.
- Assignments are due before class at 2:40 pm on the due date.
- Late assignments can be submitted on Gradescope up to Monday, Sep 25, at 5:30pm.
- You can do this assignment in groups of 2. Please submit no more than one submission per group.
- All sources of material, including whether and how you used AI tools, must be cited. The University Academic Code of Conduct will be strictly enforced.

Problem 1: VC dimensions

(15+15=30 points)

Objectives: Learn proof techniques to assert VC-Dimension for different model classes. Practice feature transformations, foreshadowing kernels.

VC-dimension is a powerful concept in statistical learning theory. Imagine we have a dataset $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$ for a binary classification problem $\mathcal{X} \mapsto \{-1, +1\}$. For a hypothesis class $\mathcal{H}$, $\text{VCDim}(\mathcal{H}) = \max\{n \text{ s.t. } \exists \mathcal{S}, |\mathcal{S}| = n, \mathcal{H} \text{ shatters } \mathcal{S}\}$. To prove that the VC-Dimension of a hypothesis class is p, we need to prove two things: $\text{VCDim}(\mathcal{H}) \geq p$, and $\text{VCDim}(\mathcal{H}) \leq p$. Proving $\text{VCDim}(\mathcal{H}) \geq p$ means that we need to prove that there exists some sample $\mathcal{S}$ of size p that is shattered by $\mathcal{H}$. In other words, we choose p points $(x_1, \ldots, x_p)$, and no matter what labeling one chooses for $(y_1, \ldots, y_p)$, there is always a hypothesis in $\mathcal{H}$ that correctly classifies that labeling. Proving the rest, that is $\text{VCDim}(\mathcal{H}) \leq p$ is usually trickier. For this we would need to prove that there is no set $\mathcal{S}$ of size $p + 1$ or larger that is shattered by $\mathcal{H}$.

(a) Consider axis-aligned ellipses in 2-D space, such that our hypothesis class $\mathcal{H}$ is $h = (w_1, w_2, w_3, w_4, b) \in \mathcal{H}$ for $w_1, w_3 > 0$ with the rule $y = +1 \iff w_1 (x_1)^2 + w_2 (x_1) + w_3 (x_2)^2 + w_4 (x_2) \geq b$. Prove that $\text{VCDim}(\mathcal{H}) \geq 4$. To prove this, you would need to choose a set of 4 points. Plot these points and show/draw the ellipse that correctly classifies each possible labeling of your chosen points (you can argue using symmetry to reduce the number of plots).

(b) For the hypothesis class of axis-aligned ellipses on the 2-D plane, prove that $\text{VCDim}(\mathcal{H}) \leq 5$.
Hint: You can use the fact that VC-Dimension of linear hyperplanes in dimension d is $d + 1$.

Problem 2: Mistake Bounds (10+15=25 points)

Objectives: Learn a useful trick to derive theorems and algorithms for biased linear classifiers from results for unbiased classifiers. Understand that perceptron mistake bounds are tight. Study extensions of perceptrons to multi-class classification.

In class, we studied the mistake bound for the unbiased perceptron. For non-trivial training sets $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{n}, x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}$, define $\mathcal{R} = \max_i \|x_i\|$. *Non-trivial* simply means there is at least one $y_i = +1$ and $y_{i'} = -1$ in $\mathcal{S}$. If there exists an unbiased optimal classifier w_{opt}, $\|w_{opt}\| = 1$ which correctly separates $\mathcal{S}$ with a margin $\delta > 0$ ($\delta = \min_i y_i(w_{opt} \cdot x_i)$), then the perceptron algorithm we saw in class makes at most $\frac{\mathcal{R}^2}{\delta^2}$ mistakes. This is a remarkable convergence bound! It is independent of:

- Dimensionality of input, d.
- Number of training examples, n.
- Order in which the examples are presented to the perceptron algorithm.
- The learning rate, η.

All that matters is how "difficult" it is to separate the training set (margin of $\|w_{opt}\|$) and the scaling of input features ($\mathcal{R}$).

(a) We run the following *bias-enabled* perceptron algorithm.

- $w_0 \leftarrow 0, b_0 = 0, t = 0$
- For each x_i, predict $y_{pred} = \text{sign}\,(w_t \cdot x_i + b_t)$.
- If $y_i \neq y_{pred}$,

$$\begin{aligned} w_{t+1} &\leftarrow w_t + y_i x_i \\ b_{t+1} &\leftarrow b_t + y_i \\ t &\leftarrow t + 1 \end{aligned}$$

Suppose we are now told that $|b_{opt}| = 1 \leq \mathcal{R}$. Prove that this bias-enabled perceptron algorithm will make at most $\frac{2(\mathcal{R}^2+1)}{\delta^2}$ mistakes.

Hint: Consider the transformation $\hat{x} = (x, 1)$ and $\hat{w} = (w, b)$. You may also use the unbiased perceptron mistake bound proved in class as a lemma.

(b) Construct a dataset $\mathcal{S}$ to show that the unbiased perceptron mistake bound is tight (i.e., there exists no better bound). To do so, derive $\mathcal{R}$ and δ for your dataset (or a suitable upper bound on $\mathcal{R}$ and lower bound on δ), and show that the unbiased perceptron algorithm makes at least $\frac{\mathcal{R}^2}{\delta^2}$ errors on that dataset, starting from $w_0 = 0$. You may use solutions to other questions in this homework for inspiration.

Problem 3: Perceptrons and SVMs (10+10+10+15=45 points)

Objectives: Understand similarities and differences between perceptrons and SVMs. Implement a practical algorithm that optimizes the hard-margin SVM objective, foreshadowing SGD. Learn tips for using linear models in practice.

You might ask this question: Is there an objective function that the perceptron is optimizing? Usually, we are directly introduced to the perceptron algorithm. In contrast, an SVM is introduced through an objective function that we optimize, and not an algorithm straightaway. However, given the similarity between perceptron and SVM that they both learn a linear hyperplane, it is interesting to see what objective a perceptron is optimizing when we run a perceptron algorithm, and how this objective compares to the SVMs primal objective (here written equivalently without slack variables ξ_i by replacing the constraints with the "max" expression):

$$F(\mathbf{w}) = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{i=1}\max\left(0, 1 - y_i\mathbf{w}\cdot\mathbf{x}_i\right)$$

Consider a modified global update perceptron learning algorithm (Algorithm 1), where instead of updating the $\mathbf{w}$ vector on each training example, you wait to update the $\mathbf{w}$ vector for all training examples until you have iterated through the entire batch. In other words, you collect all updates in a temporary variable $\Delta\mathbf{w}_{temp}$ and add it to the weight vector at the end of a complete pass through the data. Assume that we work with a training set $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.

Algorithm 1 Global Update Perceptron Learning

Require: $T, I_{\max} \in \mathbb{N}, w_{\text{init}} = (0,0), \alpha \in \mathbb{R}^+$

1: $\mathbf{w}_0 \leftarrow w_{\text{init}}$
2: $i \leftarrow 0$
3: **while** $i < I_{max}$ **do**
4: $\quad\Delta\mathbf{w}_{temp} \leftarrow \mathbf{0}$
5: $\quad$**for** $j = 1$ to n **do**
6: $\quad\quad$**if** $y_j\left(\mathbf{w}_i \cdot \mathbf{x}_j\right) \leq 0$ **then**
7: $\quad\quad\quad\Delta\mathbf{w}_{temp} \leftarrow \Delta\mathbf{w}_{temp} - y_j\mathbf{x}_j$
8: $\quad\quad$**end if**
9: $\quad$**end for**
10: $\quad\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha\Delta\mathbf{w}_{temp}$
11: $\quad i \leftarrow i + 1$
12: **end while**
13: **return** $\mathbf{w}_i$

(a) In a (batch) gradient descent algorithm, model parameters are updated using the rule that looks like line 10 in the above algorithm:

$$\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha\nabla_{\mathbf{w}}F\left(\mathbf{w}_i\right)$$

where $\mathbf{w}$ are model parameters, α is the learning rate, and $\nabla_{\mathbf{w}}F\left(w_i\right)$ is the gradient (w.r.t. $\mathbf{w}$) of the loss function (or the objective) $F\left(\mathbf{w}_i\right)$ over the training set T at $\mathbf{w}_i$.

What is the objective function $F(\mathbf{w})$ that Algorithm 1 is minimizing? Derive the gradient of this objective and verify that lines 4 to 9 are indeed computing this gradient at $\mathbf{w}_i$.

Algorithm 2 SVM Gradient Descent

Require: $T, I_{\max} \in \mathbb{N}, w_{\text{init}} = (0,0), \alpha \in \mathbb{R}^+, C \in \mathbb{R}^+$

1: $\mathbf{w}_0 \leftarrow w_{\text{init}}$
2: $i \leftarrow 0$
3: **while** $i < I_{max}$ **do**
4: $\quad \Delta\mathbf{w}_{temp} \leftarrow \boxed{\text{(i)}}$
5: $\quad$ **for** $j = 1$ to n **do**
6: $\quad\quad$ **if** $\boxed{\text{(ii)}}$ **then**
7: $\quad\quad\quad \Delta\mathbf{w}_{temp} \leftarrow \boxed{\text{(iii)}}$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad \mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha\Delta\mathbf{w}_{temp}$
11: $\quad i \leftarrow i + 1$
12: **end while**
13: **return** $\mathbf{w}_i$

(b) The primal optimization problem of an unbiased soft-margin SVM is given as:

$$F(\mathbf{w}) = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\mathbf{w} \cdot \mathbf{x}_i\right)\right)$$

As this objective looks somewhat like the one of the batched perceptron, it turns out we can simply modify Algorithm 1 to construct a gradient descent algorithm for an SVM. Derive the gradient for the SVM objective above and fill in the blanks in Algorithm 2 appropriately. Show your derivation of the gradient and the statements for blanks (i), (ii), and (iii).

(c) What is the fundamental difference between the update condition (line 6) between the two algorithms (Algorithms 1 and 2)? Typically, which of the two algorithms would you expect to include more training examples in a gradient update within one pass over the training data set?

(d) Implement both the Algorithms 1 and 2. Run them on the data in `train.txt` for $I_{max} = 2000$, for all the values of $\alpha \in \{0.006, 0.007, \ldots, 0.015\}$, $w_{init} = (1.0, 1.0)$, and $C = 0.01$ (for SVM). Report the learned weight vectors in a table for each algorithm and each value of α. Plot the data as a scatter plot on 2-D plane and plot the linear classifier each value of α as a line on the plot. Compare and discuss the separators obtained using both the algorithms.

*Submit the code for parts **(d)** in a single zip file on Gradescope.*