HW #4 CS159

Greg Stroot, Francesca-Zhoufan Li, Zhuofang Li
2135380, 2136030, 2136043

Due on April 29 2021

1 Mirror descent

1.1 The mirror world

Starting with the equation given in the problem statement:

$$\mathcal{L}(W+\Delta W) \overset{(\text{ model })}{=} \underbrace{\mathcal{L}(W) + \nabla_W \mathcal{L}(W)^T \Delta(W)}_{\text{first order Taylor expansion}} + \underbrace{h(W+\Delta W) - h(W) - \nabla_W h(W)^T \Delta W}_{\text{model of the second group of terms via } h}. \quad (1)$$

We want to find the minimum, so we take the gradient.

$$\begin{aligned}\nabla_W \mathcal{L}(W+\Delta W) &= \nabla_W \mathcal{L}(W) + \nabla_W \nabla_W \mathcal{L}(W)^T \Delta(W) + \nabla_W h(W+\Delta W) - \nabla h(W) - \nabla_W \nabla_W h(W)^T \Delta W \\ &= \underbrace{\nabla_W \mathcal{L}(W) + \nabla_W h(W+\Delta W) - \nabla h(W)}_{\text{Terms we may deal with}} + \underbrace{\nabla_W \nabla_W (\mathcal{L} - h(W))^T \Delta W}_{\text{Mismatch error between } \mathcal{L} \text{ and } h}\end{aligned}$$

So after rearranging, we are left with three terms that we may deal with and a 2nd term, which is inherent in approximating the loss function. Thus, to minimize our gradient, we wish to cancel out the 1st term. This means that we choose a ΔW, such that

$$\nabla_W h(W+\Delta W) = \nabla_W h(W) - \nabla_W \mathcal{L}(W) F. \quad (2)$$

This is the condition that was provided in the problem set. To further see this in the light of a gradient descent, we may reference the typical gradient descent formula:

$$W + \Delta W = W - \eta \mathcal{L}(W). \quad (3)$$

By applying $\nabla h(\cdot)$ to both W and ΔW and setting $\eta = 1$, we arrive at:

$$\nabla_W h(W+\Delta W) = \nabla_W h(W) - \mathcal{L}(W). \quad (4)$$

E.g. the mirror descent is a gradient descent with a step size of one and the optimization variables transformed by $\nabla h(\cdot)$

1.2 Strongly convex h

A function is *strongly convex* on a set S, when $\exists m > 0$ such that

$$\nabla^2 f(x) \succcurlyeq mI \tag{5}$$

$\forall x \in S$, where ∇^2 is the Hessian. A sufficient condition for the inverse function theorem (IFT) of a gradient of a function to hold is that the function must have a nonzero Hessian, which is the exact condition of strongly convex. Thus, strong convexity of $h(W)$ implies $\nabla h^{-1}(W)$ exists. Therefore, we may use the condition of Section 1.1, along with the aforementioned property.

$$\begin{gathered}\nabla h(W + \Delta W) = \nabla h(W) - \nabla_w \mathcal{L}(W)\\ \implies \text{(by IFT and strong convexity)}\\ W + \Delta W = \nabla h^{-1}(\nabla h(W) - \nabla_w \mathcal{L}(W))\end{gathered}$$

1.3 Entropic h

We simply start by plugging in our definition for $h(W)$ into Equation 4 listed on the homework prompt:

$$\begin{aligned}W + \Delta W &= \nabla\Big(\frac{1}{\Sigma W_i log(W_i)}\Big)\Big(\nabla(\Sigma W_i log(W_i)) - \nabla_W \mathcal{L}(W)\Big)\\ &= \frac{-\Sigma W_i \log(W_i)}{\Sigma W_i log(W_i)}\Big(\Sigma(log(W_i) + W_i\frac{1}{W_i} - \nabla_W \mathcal{L}(W)\Big)\\ &= -\Big(\Sigma(\log(W_i) + 1) - \nabla_W \mathcal{L}(W)\Big).\end{aligned}$$

2 Cover's function-counting theorem

2.1 Explain the meaning of the term *dichotomy*, as used in the paper

Given a set of vectors X in a d-dimensional Euclidean space E^d, the *dichotomy*$\{X^+, X^-\}$ is defined as dividing E^d into two sets of vectors with a *homogeneous linear threshold function* using a *weight vector* w such that one set of vectors is labelled as 1 if the inner product of w for every vector x in the set is positive and the other set of vectors is labelled as -1 if the inner product of w for every vector x in the set is negative, where the two sets are separated by the hyperplane for which the inner product of w for every vector x in the set is zero.

2.2 Argue that the total number of dichotomies of N vectors is 2^N

For each vector x in the set X, it can either be 1 or -1 so two choices each for the N vectors total and thus the total number of dichotomies is 2^N

2.3 Define the term *general position* for N vectors in d dimensions

According to the paper, *general position* means that for the set of N vectors in d-space if every subset of d or fewer vectors is linearly independent.

2.4 Plot the number of homogeneously linearly separable dichotomies of N points in general position in Euclidean d-space for $d = 25$ as a function of N. Put N on the horizontal axis, and let N range from 1 to 100. Normalise the vertical axis by the total number of dichotomies 2^N. *Hint: take care with how Cover defines the binomial coefficient*

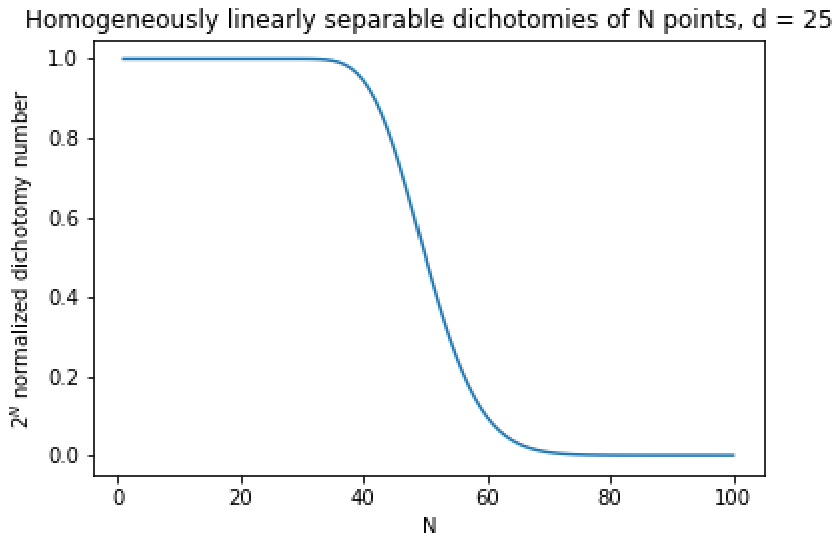

Figure 1: Cover's function-counting theorem (analysis see 2.5)

2.5 Explain the relevance (in broad terms) of Theorem 1 to the number of training points needed to learn a linear classifier under VC-style generalisation theory.

As we see from the plot, we see that normalized homogeneously linearly separable dichotomies $C(N, d)$ holds at 1 initially and sharply decreased to 0 once the number of points N gets over two times the dimension d.

This general trend aligns with what we discussed regarding the VC dimension in class, if we assume that the $N(F, X)$ is approximately equivalent to the $C(N, d)$ where the former is referring to the number of dichotomies that can be realized on a set of inputs with a function class.

Broadly, this aligns the the intuition we have as we discussed in class: if we have a classifier which is evaluated on a large fresh iid train sample, we will get a good estimation of the population error by the law of large numbers, which is the low N regime here. However, when we get to larger and larger number of function classes, the law of large numbers will fall apart, showing as the increasing N here.

2.6 Extra credit. Do you expect the number of homogeneously linearly separable dichotomies of N points in Euclidean d-space to be larger or smaller if the points are *not* in general position? Start by thinking about when $d = 2$. Can you construct a general argument for arbitrary N and d?

Now that we are considering dependence, the generalization will genera tally get worse.

3 Change of measure inequality

(a)

$$E_{x\sim P}\varphi(x) = \int_{supp(P)} \varphi(x)P(x)dx$$

$$E_{x\sim Q}\frac{P(x)}{Q(x)}\varphi(x) = \int_{supp(Q)} \varphi(x)P(x)dx$$

since $supp(Q) \subset supp(P)$ and $\varphi(x) > 0$,

$$\int_{supp(P)} \varphi(x)P(x)dx \geq \int_{supp(Q)} \varphi(x)P(x)dx$$

i.e

$$E_{x\sim P}\varphi(x) \geq E_{x\sim Q}\frac{P(x)}{Q(x)}\varphi(x)$$

(b)
Set $\varphi(x) = e^{\beta h(x)}$. By (a),

$$E_{x\sim P}e^{\beta h(x)} \geq E_{x\sim Q}\frac{P(x)}{Q(x)}e^{\beta h(x)}$$

Take log on both side,

$$ln(E_{x\sim P}e^{\beta h(x)}) \geq ln(E_{x\sim Q}\frac{P(x)}{Q(x)}e^{\beta h(x)}) \geq E_{x\sim Q}ln(\frac{P(x)}{Q(x)}) + \beta h(x) = -E_{x\sim Q}ln(\frac{Q(x)}{P(x)}) + \beta h(x)$$

the last inequality is by jensen. Note that $E_{x\sim Q}ln(\frac{Q(x)}{P(x)}) = KL(Q||P)$
Then reorder and devide by β we have

$$E_{x\sim Q}[h(x)] \leq \frac{(Q||P) + lnE_{x\sim P}[e^{\beta h(x)}]}{\beta}$$

4 Appendix

```
import numpy as np
import scipy.special

import matplotlib.pyplot as plt


class FunctionCounting(object):

    """Plot the number of homogeneously linearly separable dichotomies of N points
    in general position in Euclidean d-space for d = 25 as a function of N.
    Put N on the horizontal axis, and let N range from 1 to 100.
    Normalise the vertical axis by the total number of dichotomies 2^N.
    Hint: take care with how Cover defines the binomial coefficient

    Calculate the number of homogeneously linearly separable dichotomies
    for given number of points N in Euclidean d-space following the equation (5) in
    Geometrical and Statistical Properties of Systems of Linear Inequalities
    with Applications in Pattern Recognition, IEEE Trans.
    Electron. Comput., T. Cover, 1965.

    Since the calculated ones are very off,
    a method with the default bionomial coefficients is also included.
    """

    def __init__(self, N_start, N_end, d):
        """Get the Ns and ks"""
        self.N_start = N_start
        self.N_end = N_end
        self.N_range = range(self.N_start, self.N_end + 1)
        self.Ns = np.arange(self.N_start, self.N_end + 1)
        self.d = d
        self.ks = np.arange(d)

    def get_coeff(self, s, k):
        """Get the binomial coefficients comprising (N, d) defined for
        all real s and integer k in equation 6"""

        if k > 0:
            num = np.prod(s - np.arange(k))
            return num / np.math.factorial(k)
        elif k == 0:
            return s * (s + 1) / np.math.factorial(k)
        else:
            print("The current version is not considering k < 0")

    def get_C(self, N):
        """Calculate the homogeneously linearly separable dichotomies
        of N points in general position in Euclidean d-space
        in equation 5 with 2^N normalization"""
        return (
            2 * np.sum(np.array([self.get_coeff(N - 1, k) for k in self.ks])) / (2 ** N)
        )

    def get_Cs(self):
        """Get the array of the homogeneously linearly separable dichotomies
        of N points in general position in Euclidean d-space
        in equation 5"""
        return np.array([self.get_C(N) for N in self.Ns])
    def default_Cs(self):
        """Calculate the binomial coeffs with scipy bino"""
        return np.array(
            [
                2
                * np.sum(
                    np.array([scipy.special.binom(N - 1, k) for k in np.arange(self.d)])
                )
                / 2 ** N
                for N in self.N_range
            ]
        )

    def plot(self, ifdefault=True):
        """Plot the number of homogeneously linearly separable dichotomies of N points
        in general position in Euclidean d-space for as a function of N"""
        if ifdefault:
            Cs = self.default_Cs()
        else:
            Cs = self.get_Cs()
        plt.plot(self.Ns, Cs)
        plt.title(
            "Homogeneously linearly separable dichotomies of N points, d = {0}".format(
                self.d
            )
        )
        plt.xlabel("N")
        plt.ylabel("$2^N$ normalized dichotomy number")


FunctionCounting(1, 100, 25).plot(ifdefault=True)
# FunctionCounting(1, 100, 25).plot(ifdefault=False)
```