Supervised Learning - Coursework II

Antonios Anagnostou, Kefei Hu
{antonios.anagnostou.18, kefei.hu.18}@ucl.ac.uk

December 14, 2018

Part II

Understanding Error Decomposition

Let $h_S = ERM_H(S)$. It is the optimal solution to the training data. The risk of h_S is $L_D(h_S)$, which can be decomposed the following way:

$$L_D(h_S) = \epsilon_{app} + \epsilon_{est}$$

In the case of linear classifiers on 'just a little bit problem', as long as the algorithms predict with the first feature of x it will obtain zero approximation error.

$min_{h \in H}\big(L_D(h)\big) = L_D(sign(wx)) = 0$ where $w = k\hat{e}_0, k > 0$.

And since the approximation error is zero:

$$L_D(h_S) = \epsilon_{est}$$

The estimation error decreases with sample size m and increases with the Hypothesis complexity. The VC dimension of these linear classifiers in $\{-1, +1\}^n$ is $n + 1$. Above expression also means that $L_D(h_S)$ for this particular problem only depends on the constraint of S_m and n.

The generalization error stated in the question, $\epsilon(A_{S_m})$ refers to $L_D(h_{S_m})$.

b i)Procedure

Given a range of n, we estimate the sample complexity using random sampling and Binary Search algorithm. The following describes the procedure of finding sample complexity for a fixed value of n.

1. Set a range of m values to consider, depending on the algorithm. This step will be explained further.
2. Binary search is then applied onto the interval of m values, to find minimum m that satisfies $\epsilon_{test}(m) < 0.1$.

3. Given the pre-determined range in m, m_{left} is initialized to be the minimum value of m, m_{right} is initialized to be the maximum value of m, and m_current is initialized to be n.

4. To calculate $\epsilon_{test}(A_{S_m})$ for each value of m, the algorithm is trained on a randomly sampled training set of size (m, n), $S_m^{(i)}$, and it is tested on a randomly sampled test matrix of size (T, n). This step is repeated for α times. The mean test error calculated from α trials is used as an estimator to estimate the generalization error $\epsilon(A_{S_m})$, at m.

$$\hat{\epsilon}(A_{S_m}) = \frac{1}{\alpha}\sum_{i=1}^{\alpha}\frac{1}{T}\sum_{j=1}^{T} I[A_{S_m^{(i)}}(x_j) \neq y_j]$$

5. The generalization error is estimated like step 4 for m_{left}, m_{right}, and m respectively. Compare these errors against each other and against our goal (ϵ =0.1). For example, if $\hat{\epsilon}(A_{S_{left}}) < \hat{\epsilon}(A_{S_{right}})$, take the right half of this interval. This enables the algorithm to iteratively narrow down to the minimum value of training sample size that achieves $\hat{\epsilon}(A_{S_m}) < 0.1$.

6. The procedure above is repeated r times for a fixed n. So, applying Binary Search in the i^{th} trial gives the i^{th} estimate of sample complexity of n:

$$C_n^{(i)}(A) = min\big(\{m \in \{1, 2, ..\} : \hat{\epsilon}(A_{S_m}) \leq 0.1\}\big)$$

7. Taking the average of $C_n^{(i)}(A)$ across all 10 trials, the estimator of the sample complexity becomes:

$$\hat{C}_n(A) = \frac{1}{r}\sum_{i=1}^{r} C_n^{(i)}(A)$$

b ii) Explaining the Bias and Trade-off

First, our estimator of $\epsilon(A_{S_m})$ is an unbiased estimator of risk of h_{Sm}, such that $h_{Sm} = ERM_H(S_m)$. The risk of h_{Sm} is also $L_D(h_{Sm})$. Since the approximation error is zero, as derived at the beginning, we know that in this case $L_D(h_{Sm}) = \epsilon(A_{S_m})$. So our estimator is also an unbiased estimator of $\epsilon(A_{S_m})$.

Let $S_m^{(i)}$ be the training set generated at i^{th} trial, independently from distribution D.

Let L_i be the test error rate for the i^{th} trial, $\frac{1}{T}\sum_{j=1}^{T} I[A_{S_m^{(i)}}(x_j) \neq y_j]$, where T pairs of test data, $[x_j, y_j]$ are generated independently to $S_m^{(i)}$. The expected value of L_i across D is the $L_D(h_{Sm})$, which equates to the generalization error.

$$E(L_i) = \sum_{S_T \in \{-1,+1\}^n} \Big(\frac{1}{T}\sum_{j=1}^{T} I[A_{S_m^{(i)}}(x_j) \neq y_j]\Big) = \epsilon(A_{S_m})$$

The average test error rate $\overline{L}$ over α independent trials is used as an estimator of generalization error.

$$\overline{L} = \frac{1}{\alpha}\sum_{i=1}^{\alpha} L_i$$

Since each L_i are generated using train/test dataset that are sampled iid, $\overline{L}$ is also an estimator that predicts generalization error with zero bias. As α increases, the union of all test samples has greater coverage of D, and the variance of $\overline{L}$ as an estimator decreases.

$$Var(\overline{L}) = \frac{1}{\alpha} Var(L_i)$$

Deviation to Generalization Error due to Concentration Inequalities

Suppose having already trained our Empirical Risk Minimizer, h_S on training set S, we would like to estimate $L_D(S)$ by testing it on unseen dataset T of size m_{test}. Let Z_i be the outcome for one test example t_i, e.g. $Z_i = I[\hat{h_s}(t_i) \neq y_i]$. In theory, the probability of Z_i being 1 is p, which has an expected value as the generalization error.

In probability theory, the deviation of sample mean from the actual mean decreases as the size of sample increases.

But considering limited computational resources and time, T is set to be 10000, and α is set to be 10 for Perceptron, Winnow and Least Square, 5 for Nearest Neighbor.

Similarly, $\hat{C}_n(A)$ is an unbiased estimator for sample complexity. Although increasing the r also leads to less variance in the estimation, r is set to be 10 for Perceptron, Winnow and Least Square, 5 for Nearest Neighbor because

the computation becomes very expensive. For details on computational cost of sample complexity estimation for each algorithm, please refer to the section 'Computational time of sample complexity estimation'.

b ii) Explaining test data size

T is set to be 10000 for all n in the test. But the total number of combinations of labels = 2^n, meaning that the test set covers drastically less proportion of data points as n increases.

The maximum proportion of input space represented by the test set $\leq \frac{10000}{2^n}$, which is smaller than 0.01 when $n = 20$. So inevitably, the variance of sample complexity estimated this way in will increase with n.

b ii) Explaining choosing the range of m

The Binary Algorithm is a faster method to look for $C(A)$ ($\sim O(log(Range(m)))$) than iterating increasing values of m ($\sim O(Range(m))$). Since the algorithm looks for solution in a pre-defined range of m, it is important to ensure that the solution lies within this range.

Our choice of $Range(m)$ are based on the trends observed on sample complexity by experimenting with small values of n, e.g. $n = 1, ..., 10$. The trend helps us to give a rough prediction on where $C(A)$ could most likely exist for n up to 100.

It became obvious that both Perceptron and Least Square have sample complexity linear to n, and Winnow has sample complexity lower than n. Hence the maximum value of m for these three algorithms are set to be $O(n)$. The Sample Complexity for Nearest Neighbor appeared to increase exponentially or in polynomial with m for n up to 10. Before we could make assumptions about the exact relationship, the maximum value of m considered in the binary search is set to be $O(n^3)$ so the solution could still be found for n up to 20.

Computational time of sample complexity estimation

To estimate the sample complexity for n up to n_max, with r runs for each n, the total computational cost:

$$Comp(n_{max}) = \sum_{n=1}^{n_{max}} B(n) \times r$$

$B(n)$ is the computational cost of running binary search with averaged test errors across α random sampling trials. Let the range of m considered in the binary search be R, $R = m_{left} - m_{right}$. R is dependent on n and function choice, let R = O(M(n)).

For clarity, we first consider the computation cost per run, when both n and m are fixed. The maximum number of times we apply the function 'estimate_generalization_error' is $O(log(M))$. Let the computational time of the

chosen training algorithm to be $f_{train}(m,n)$, test algorithm to be $f_{test}(m,n)$. For a fix number of m, we train and test the algorithm for α trials to find the average test error. So the cost to estimate the generalization error for fixed m with α trials = $(f_{train}(m,n) + f_{test}(n)) \times \alpha$

Since $R \sim O(M(n))$, and the number of m where the generalization error is computed = $O(log(M(n)))$, the computational cost of $B(n)$ can be obtained as:

$$B(n) \sim O\Big(log(M(n))(f_{train}(M(n),n) + f_{test}(n))\Big)$$

Overall, the total computational time of estimating sample complexity for $n \in [n_{max}]$ is $Comp(n_{max})$:

$$Comp(n_{max}) = r \times \sum_{n=1}^{n_{max}} B(n) \Rightarrow Comp(n_{max}) \sim O\Big(n_{max}B(n_{max})\Big)$$

Perceptron

Note that T is the size of test sample, where $T >> n$.

$$\begin{aligned} M(n) &\sim O(n) \\ f_{train}(m,n) &\sim O(mn) \sim O(n^2) \\ f_{test}(n) &\sim O(n \times T) \sim O(nT) \\ B(n) &\sim O(log(n)) \times (O(n^2) + O(nT)) \sim O((n^2 + nT)log(n)) \\ \Rightarrow Comp(n_{max}) &\sim O\Big(n_{max}B(n_{max})\Big) \sim O\Big((n_{max}^3 + n_{max}^2 T)log(n_{max}))\Big) \end{aligned}$$

Winnow

Similar to Perceptron, the computational cost is bounded below by:

$$\begin{aligned} M(n) &\sim O(n) \\ \Rightarrow Comp(n_{max}) &\sim O\Big(n_{max}B(n_{max})\Big) \sim O\Big((n_{max}^3 + n_{max}^2 T)log(n_{max}))\Big) \end{aligned}$$

In practice, Winnow runs slightly slower than Perceptron as it involves more vector multiplication.

Linear Regression

$$\begin{aligned} M(n) &\sim O(n) \\ f_{train}(m,n) &\sim O(mn^2 + n^3) \sim O(n^3) \\ f_{test}(n) &\sim O(nT) \\ B(n) &\sim O(log(n)) \times (O(n^3) + O(nT)) \sim O((n^3 + nT)log(n)) \\ \Rightarrow Comp(n_{max}) &\sim O\Big(n_{max}B(n_{max})\Big) \sim O\Big(n_{max}^4 + n_{max}^2 Tlog(n_{max}))\Big) \end{aligned}$$

Linear Regression involves matrix-to-vector multiplication, which makes it a bit slower than Perceptron and Winnow. Luckily, m only grows linear to n. The sample complexity of Linear Regression could still be carried out using the same r, α under resources and reasonable time.

1NN

According to our range of m selected for 1-NN, $M(n) \sim O(n^3)$. 1-NN involves calculation of pairwise distances between training and test data, hence the cost of applying it once:

$$\begin{aligned} f(m,n) &\sim O(mn \times T) \sim O(n^4 T) \\ B(n) &\sim O(n^4 T log(n^3)) \\ \Rightarrow Comp(n_{max}) &\sim O\Big(n_{max} B(n_{max})\Big) \sim O\Big(n_{max}^5 T log(n_{max}^3))\Big) \end{aligned}$$

1NN has highest computational cost, due to the large matrix multiplications involved at the computation of pairwise Euclidean distances, and values of m considered are very large. Under computational constraint, we halved the number of runs and sampling for 1NN, and ran for n up to 20. The sample complexity of 1NN could be estimated up to 100 using fitting and extrapolation.

c) Graphs of Sample Complexity

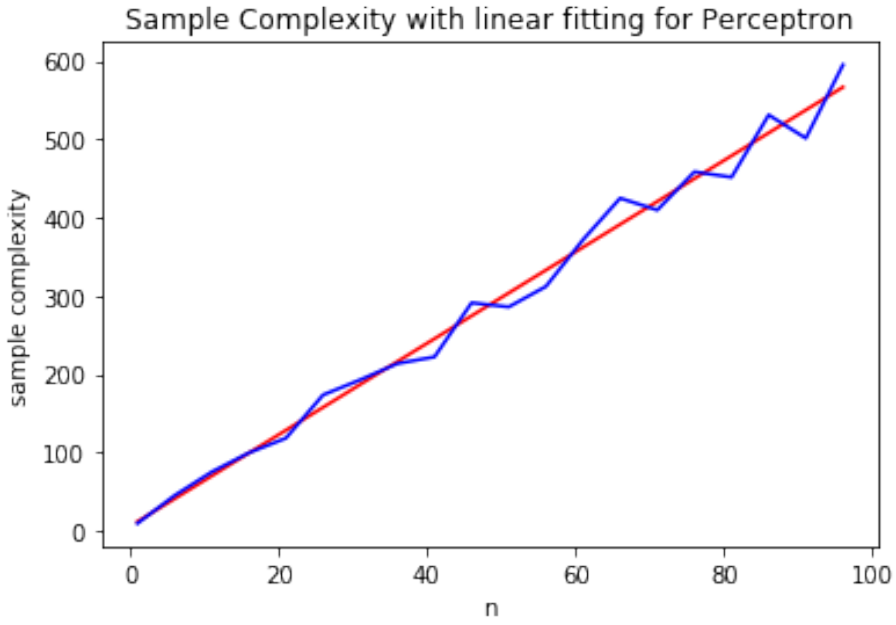

Figure 1: Perceptron : $\Theta(m(n)) = 5.79 + 5.84n \sim O(n)$

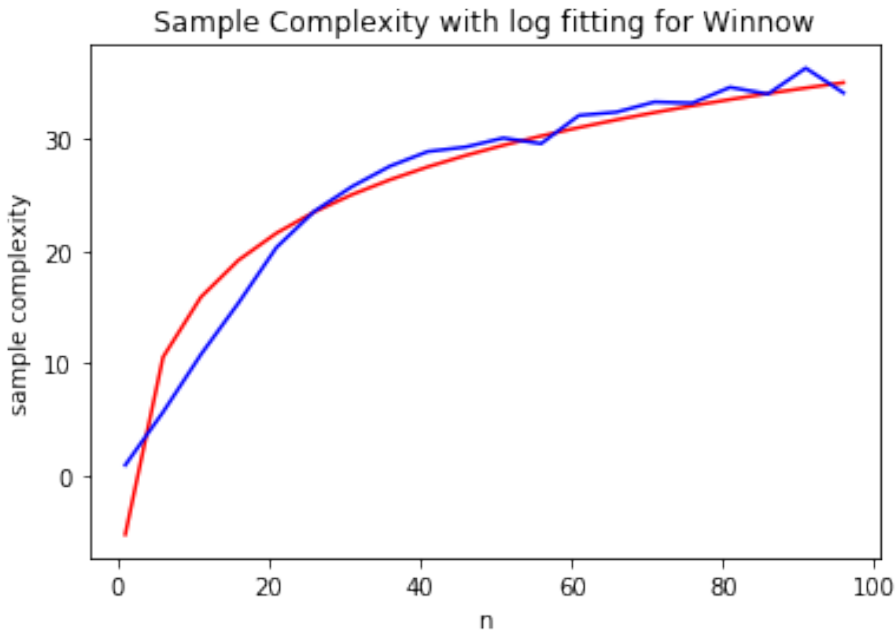

Figure 2: Winnow : $\Theta(m(n)) = -5.15 + 8.78log(n) \sim O(log(n))$

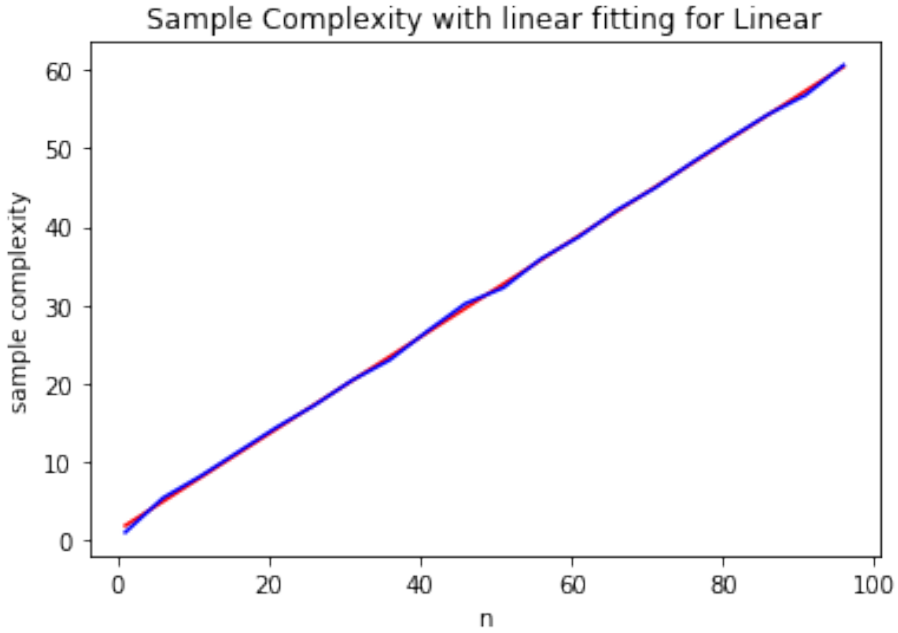

Figure 3: Linear Regression :$\Theta(m(n)) = 1.25 + 0.62n \sim O(n)$

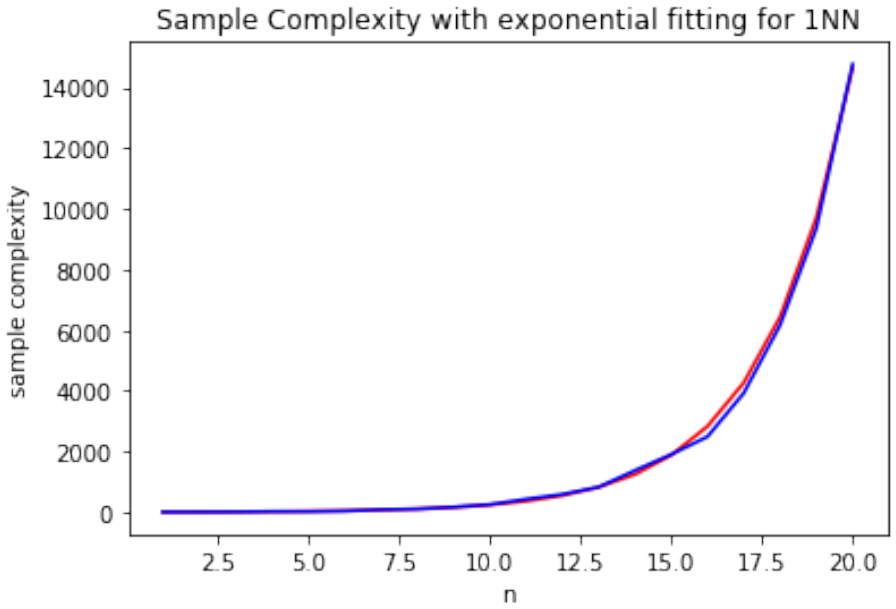

Figure 4: Nearest Neighbor :$\Theta(m(n)) = 3.95e^{0.41n} \sim O(e^n)$

c) Discussion of Observations

In the case of linear classifiers on 'just a little bit problem', as long as the algorithms predict with the first feature of x it will obtain zero approximation error.

$Argmin_{h \in H}\big(L_D(h)\big) = sign(wx)$ where $w = k\hat{e}_0, k > 0$.

The Empirical Risk Minimizer, $h_S = ERM_H(S)$ is the optimal solution constrained to the training data. The risk of h_S can be decomposed the following way:

$L_D(h_S) = \epsilon_{app} + \epsilon_{est}$

And since the approximation error is zero:

$L_D(h_S) = \epsilon_{est}$

The estimation error, which is also generalization error, decreases with m and increases with n.

Perceptron vs. Winnow

The sample complexity graphs indicate that Winnow is generally more accurate with larger number of irrelevant agents than Perceptron. This observation is consistent with the mistake bounds of online learning during training.

Suppose there are k relevant agents, or literals, $k << n$.

Perceptron mistake bound = $(4k + 1)(n + 1) \sim O(kn)$

Winnow mistake bound = $3k(log(n) + 1) + 2 \sim O(klog(n))$

The upper bound of error rate for both algorithms = $\frac{M}{m}$, where M is the mistake bound during training.

This means that given fixed n, if both Perceptron and Winnow are aiming for the same training error rate ϵ, Winnow requires less samples. More specifically, in order for the maximum error rate of either algorithm to be $< \epsilon \in (0, 1)$:
Perceptron:

$$m \geq \frac{(4k + 1)(n + 1)}{\epsilon} \sim \Omega(n)$$

Winnow:

$$m \geq \frac{3k(log(n) + 1) + 2}{\epsilon} \sim \Omega(log(n))$$

Linear Regression vs. Perceptron

Given training set S, the primal form of linear regression is derived by minimizing the empirical risk $\epsilon_{emp}(S, \omega)$ of the entire training dataset. As the first agent, x_0 is always consistent with the label, the primal form of linear regression directly computes the optimal weight, where all the entires are close to zero except for the first one. The same weight will generalize well on the test data, because the test data is also sampled with its first agent consistent to label.

On the other hand, Perceptron learns one example at a time by gradient descent with learning rate =1. This means that at each update, it could only increase the weight for the first agent by 1 at maximum. Also, since the learning rate is fixed at 1, Perceptron algorithm might 'overshoot'.

Therefore, given the same m, linear regression achieves a smaller test error than perceptron in this particular case.

Nearest Neighbors vs. all 3 Linear Classifiers

Note that VC dimension of 1-Nearest-Neighbors is infinity, hence PAC cannot be applied to bound its sample complexity, even at its worst scenarios.

Unlike linear classifiers, 1NN predicts with the closest data point instead of finding a seperating hyperplane. It will not be able to identify the strongest predictors such as x_0, without any training process.

As n increases, due to the curse of dimensionality, the euclidean distances from test point to its nearest point increases exponentially. It follows that m have to grow exponentially keep a consistent accuracy. As the nearest point becomes further away from $x_{test} \in R^n$, the probability of this nearest point having the same label as x_{test} becomes smaller.

Specifically for 'just a little bit problem', if the nearest neighbor $\in R^n$ has d features inconsistent with x_{test}, and their euclidean distances are $2\sqrt{d}$ for $d \in [n]$, the probability of x_{test} having the same x_0 as the nearest neighbor is $\frac{n-d}{n}$.

d) Bounding the probability of mistake trial in perceptron

Let B be the upper bound of total number of mistakes made by perceptron during training on any set of input.

$$M \leq B = (\frac{R}{\gamma})^2$$

Let S_m be a set of m training examples sampled independently from D. let t be drawn uniformly from {1,...,m}, where $P(t = t') = \frac{1}{m}$.

The Lowest Upper Bound, or Supremum of probability of hitting a mistake at any t^{th} trial is claimed to be $\frac{B}{m}$.

Proof by contradiction

Let p_t be the probability of hitting a mistake at t^{th} trial.

Suppose for a particular $t' \in [m]$, $p_{t'} > \frac{B}{m} + \epsilon$, where $\epsilon > 0$.

The probability of hitting mistake for rest of the trials all have the same Supremum as $\frac{B}{m}$.

Let M be the expected number of mistakes made by perceptron when it finishes training on S_m, and M_{sup} be the Maximum of expected number of mistakes, across all possible values of p_t. As the mistake bound of online learning is B, $M \leq M_{sup} \leq B$. Let's compute the actual M_{sup}.

$$\begin{aligned}
_{sup} &= Sup_{\{p_1, p_2 .. p_m\}} M \\
&= \sum_{t=1}^{m} Sup(p_t) \\
&= \sum_{t \neq t'}^{m} \frac{B}{m} + p_{t'} \\
&= (m-1)\frac{B}{m} + (\frac{B}{m} + \epsilon) \\
&= B + \epsilon
\end{aligned}$$

Now, M_{sup} becomes $B + \epsilon$, which is strictly bigger than B. This contradicts with the mistake bound on perceptron. Therefore we conclude that the Lowest Upper Bound of p_t for all $t \in [m]$ is $\frac{B}{m}$.

In 'just a little bit' problem, we know that the maximum norm of x from D is $\sqrt{n}$, giving $R = max(\| x \|_2) = \sqrt{n}$

$$\Rightarrow P(A_S(x) \neq y) \leq \frac{n}{m\gamma^2}$$

e) Lower bound of sample complexity of 1NN

Given n, the input space of 'just a little bit' problem is a n-dimension hypercube. The euclidean distances between any two possible data points, (x_1, x_2) is $2\sqrt{(d)}$, where d represents the number of features that are different between x_1 and x_2. In this question, we would like to prove that $m \sim \Omega(\frac{1}{n} e^{\frac{n}{2}})$.

Average distances between x to its nearest neighbour

If there are m training points $\{x_1, ..., x_m\}$, scattered randomly in $\{-1, +1\}^n$, the average volume of convex-hull(Voroni?)surrounding each points is expected $\frac{2^n}{m}$. The average distance from x to the boundary of its convex hull, r can be estimated as the following:

$$r = (\frac{2^n}{m})^{\frac{1}{n}} = \frac{2}{m^{1/n}}.$$

Suppose now there is a test point x_{test}, of which we would like to estimate the label. x_{test} can be inside the convex hull of any of the m training data points. The average distance between x_{test} to its nearest is approximately r.

Distance from nearest neighbour and probability of mistake

Denote the nearest point of x to be $x' \in S_m$ wlog.

The further x' is from x, smaller the probability that x' has the same label as x.

More specifically, suppose the number of features that are different between x and x' is d. To ensure the first feature (label) of x' and x are the same, the d features in x' that differs from x is restricted to be selected from $n-1$ dimensions.

Hence, the probability that x' successfully predicts x, is $\frac{(n-1)Cd}{nCd} = \frac{n-d}{n}$

Given the average Euclidean distance between $(x, x') = r$, deduce that:

$$\overline{d} = \frac{r^2}{4} = \frac{1}{m^{2/n}}$$

The average probability that x' successfully predicts and mis-classify x becomes:

$$P(y' = y) = 1 - \frac{\overline{d}}{n}$$
$$P(y' \neq y) = \frac{\overline{d}}{n} = \frac{1}{nm^{\frac{2}{n}}}$$

Deriving a lower bound of m

Now, we would like to find the minimum value of m such that the probability of misclassifying x is guaranteed to be smaller than $\epsilon \in (0, 1)$.

$$\begin{aligned} P(y' \neq y) &\leq \epsilon \\ \frac{1}{nm^{\frac{2}{n}}} &\leq \epsilon \\ m^{\frac{2}{n}} &\geq \frac{1}{n\epsilon} \\ log(m) &\geq \frac{n}{2} log(\frac{1}{n\epsilon}) \\ \Rightarrow m &\geq \frac{1}{n\epsilon} e^{\frac{n}{2}} \geq \frac{1}{n} e^{\frac{n}{2}} \end{aligned}$$

Therefore, m $\sim \Omega(\frac{1}{n} e^{\frac{n}{2}})$