CS5339 Machine Learning

Representation I

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 2, 2019/20

Representation

In this part of the course, we will study common representations in machine learning, what problems they are often used to solve, and what they can approximate.

To develop intuition, we will look at how well some of the following functions can be represented/approximated:

1. **Boolean functions:** Arbitrary Boolean functions from $\{0,1\}^d$ to $\{0,1\}$.

2. **PARITY:** The PARITY function is a Boolean function from $\{0,1\}^d$ which returns 1 if the number of the number of occurences of 1 in the input is odd and returns 0 otherwise. PARITY has some interesting properties that makes it useful for developing intuition.
    - The output of PARITY changes if the input changes by a single bit.
    - All components of the input must be used in order to compute the the output, if even one of the input is not observed, it is not possible to correctly compute the parity.

3. **Continuous functions:** Functions from $[0,1]^d$ to $\mathbb{R}$ satisfying the property: f is continuous if, for any $\epsilon > 0$, there exists a $\delta > 0$ such that for all $\mathbf{x}, \mathbf{x}' \in [0,1]^d$ where $\|x - x'\| < \delta$, we have $|f(\mathbf{x}) - f(\mathbf{x}')| < \epsilon$. Other bounded subset of $\mathbb{R}^d$ can also be used instead of $[0,1]^d$.

4. **Lipschitz functions:** A function f is c-Lipschitz for some $c > 0$ if for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $|f(\mathbf{x}) - f(\mathbf{x}')| \leq c\|\mathbf{x} - \mathbf{x}'\|$.
    - This means that if two inputs are close together, the function output is likely the same; or
    - Inputs with very different outputs can only occur far apart.

- We call a function class F a **universal approximator** if it is able to approximate any continuous function to arbitrary accuracy: for any continuous function f, there is a $g \in F$ such that $\sup_{\mathbf{x} \in [0,1]^d} |f(\mathbf{x}) - g(\mathbf{x})| < \epsilon$.

Outline

Nearest Neighbour

- Among the simplest learning algorithm:
  - Simply memorize the training set.
  - Predict with the label of the nearest neighbour in the training set.
  - Requires a distance function.
- In k-nearest neighbour, find k nearest neighbours,
  - Predict using the label with largest number of votes for classification.
  - For regression, use average of the k values.
  - Can also take weighted average or weighted votes, where the weight depends on the distance.

Representation and Approximation

- Nearest neighbour can represent any Boolean function – simply store the label for each possible input.
- Nearest neighbour can also approximate continuous functions to arbitrary accuracy: universal approximator.
    - Store the value of the function at a fine enough regular grid. This gives a piecewise constant approximation.

- To represent PARITY in d variables, all 2^d points need to be stored.
  - To see why, assume there are some points that are not stored.
  - Find one of these points, $\mathbf{x}$ that is at a distance 1 from a stored point $\mathbf{x}'$.
  - By the property of the XOR function, the label for $\mathbf{x}$ is different from the label of $\mathbf{x}'$, hence nearest neighbour will classify $\mathbf{x}$ incorrectly.

Properties

- Nearest neighbour converges to at most twice the optimal error (Bayes error), as training set size grows.

  Intuition: Consider binary classification. Assume $p(y|x)$ is the same as $p(y|x')$ at nearest neighbour x'.
  - Optimal error is $Opt = \min_{y\in\{0,1\}}(1 - p(y|x))$.
  - Under assumption,

$$\begin{aligned} E[\ell_{0-1}(NN)] &= p(y=1|x)(1-p(y=1|x)) + (1-p(y=1|x))p(y=1|x) \\ &\leq 2p(y=1|x)(1-p(y=1|x)) \\ &\leq 2\min_{y\in\{0,1\}}(1-p(y|x)) = 2Opt. \end{aligned}$$

  - $p(y|x')$ converges to $p(y|x)$ as training size grows. Convergence rate depends on data distribution.
  - Bound also hold for more than 2 classes with more complex argument.

- k-nearest neighbour converges to Bayes error.

  Intuition:

  - If all k neighbours have the same conditional distribution as k, taking label with largest num of votes approximates Bayes classifier.
  - k needs to grow slowly with increasing training set size to converge to Bayes classifier.

Curse of Dimensionality

Theorem: (SSBD Theorem 19.3) Let $\mathcal{X} = [0,1]^d$, $\mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function, $p(y|x)$, is a c-Lipschitz function. Let h_S denote the result of applying the 1-NN rule to a sample $S \sim \mathcal{D}^m$. Then,

$$E_{S\sim\mathcal{D}^m}[L_{\mathbf{D}}(h_S)] \leq 2L_{\mathcal{D}}(h^*) + 4c\sqrt{d}m^{-\frac{1}{d+1}},$$

where h^* is the Bayes decision rule.

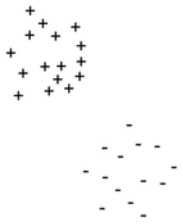
We skip the proof (see textbook if interested).

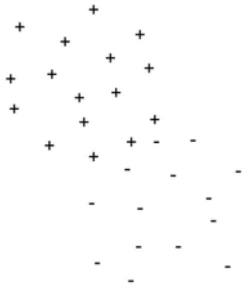
Consequences: For the last term to be smaller than ϵ, $m > (4c\sqrt{d}/\epsilon)^{d+1}$ is *sufficient*. Sample size grows exponentially with dimension d.

Theorem: (SSBD Theorem 19.4) For any $c > 1$, and every learning rule, L, there exists a distribution over $[0,1]^d \times \{0,1\}$, such that $p(y|x)$ is c-Lipschitz, the Bayes error of the distribution is 0, but for sample sizes $m \leq (c+1)^d/2$, the true error of the rule L is greater than $1/4$.

This shows that the sample size *necessarily* grows exponentially with dimension if we only impose the Lipschitz condition. We will defer the proof until we do the no-free-lunch theorem later.

Exercise 1: Consider the two figures I and II. Assume that they are plotted on the same scale. If we have to fit a function $f(x)$ to the data shown in the figures such that $f(x)$ has value $+1$ at positive points and -1 at negative points, which figure would require a function with a larger Lipschitz constant?


(I)


(II)

A. Figure I
B. Figure II

Exercise 2:
We will experiment with using nearest neighbour to do prediction for PARITY under two input distributions:

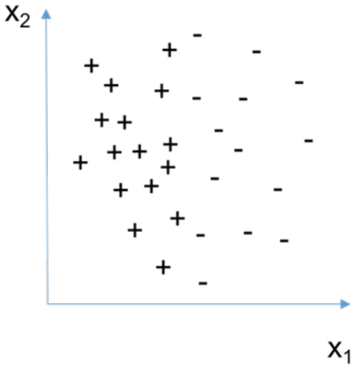
(1) Uniform distribution, and
(2) Uniform distribution over instances with at most two nonzero inputs.

Using a 20 dimensional input, we will randomly select 10,000 training examples.
Before running the experiment, predict the outcome.

A. Nearest neighbour does WELL on both (1) and (2).
B. Nearest neighbour does WELL on (1) but POORLY on (2).
C. Nearest neighbour does POORLY on (1) but WELL on (2).
D. Nearest neighbour does POORLY on both (1) and (2).

Explain the experiment results you observed.

Exercise 3: Consider the data shown in the figure. Suggest one way to improve the performance of the nearest neighbour classifier on the data.



Exercise 4:
Do you think doing feature selection would be helpful when using nearest neighbour for text classification? Consider your answer before running the experiment which uses chi square method.

Practical Properties

- NN and k-NN suffer from the *curse of dimensionality*
  - If there are irrelevant variables, removing them could be helpful.
  - Do feature selection.
- Scaling/normalization of features important as it affects distance computation.
- As suggested by the Lipschitz constant in performance bound, NN will do well if instances from the same class are clustered close together and the clusters are far from each other.

- Analysis results are worst case:
  - Bound depends on the number of discretization cells that covers all the data – this is essentially a measure of volume and grows exponentially with dimension.
    - The smallest number of cells to cover the space is sometimes called the *covering number*.
  - “Volume” of real data often much smaller than suggested by dimensionality
    - Data often occur in a small number of low dimensional subspace/manifold.

FaceNet [13]

- FaceNet, developed in 2015, uses a deep convolutional neural network to learn a 128 dimensional feature vector such that images of the same person are close together while images of different people are far apart in Euclidean distance.
- Suitable for use with k-NN classifier.
- Classification accuracy of 99.63% on whether pairs of images belong to the same person: Labeled Faces in the Wild (LFW) benchmark dataset.

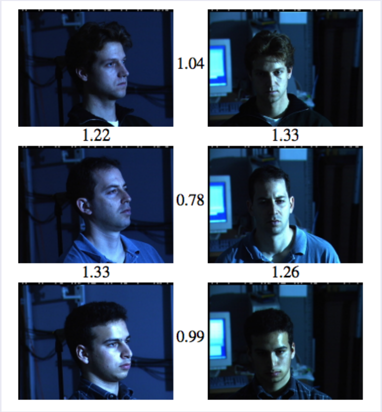


Figure: FaceNet and distances between the some image pairs – thresholding at 1.1 gives correct result for all pairs, showing good illumination and pose invariance. (Figure from [13])

Outline

Decision Trees

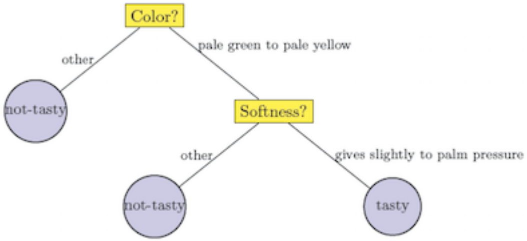


Figure: Decision tree for TASTY papaya (from SSBD)

- In a decision tree, the leaves are labeled with the class while each internal node contains a test (usually on a single variable).
- To do classification, we start from the root node and perform the test at the node, move to the child corresponding to the test output, and recurse until we reach a leaf node, which gives the class of the instance.

Disjunctions of Conjunctions

- Each leaf corresponds to a path down the decision tree.
- Instances that reach the leaf satisfies the conjunction of the tests on the path. For example, reaching the rightmost leaf in the example corresponds to
  - Conjunction 1: *(Color? = pale green to yellow) AND (Softness? = gives slightly to palm pressure)*

- All the leaves together partitions up the entire input space.
    - Every instance will be assigned to a leaf and satisfies a single rule (conjunction of tests on the path to the leaf).

- If all tests and final prediction are binary, the function represented by the decision tree can be interpreted as a DNF (disjunction (OR) of conjunctions). Generally, can also be considered as a set of rules: each conjunction implies a class.

For Boolean functions, it is useful to understand properties of decision trees in relation to standard Boolean formulae.

Disjunctive Normal Form (DNF): Disjunction (OR) of conjunctions (AND) of variables or their negations, e.g. $(x_1 \wedge \neg x_3 \wedge x_4) \vee \cdots \vee (x_2 \wedge x_5 \neg x_8)$

- k-term DNF: when the number of conjunctive terms is no more than k
- k-DNF: when the number of literals (variables or its negation), also called the width, is no more than k.

Conjunctive Normal Form (CNF): Conjunction (AND) of disjunctions (OR) of variables or their negations, e.g. $(x_2 \vee \neg x_5) \wedge \cdots \wedge (\neg x_4 \vee x6 \vee x_9)$.

- k-clause CNF: when the number of disjunctive clauses is no muore than k
- k-CNF: when the number of literals, or width, is no more than k.

De Morgan's laws are useful for manipulating boolean formulas.

- $\neg(A \vee B) = \neg A \wedge \neg B$
- $\neg(A \wedge B) = \neg A \vee \neg B$.

- For decision tree represention of a boolean function with k leaves can be represented with a k-term DNF.
  - There are at most k leaves with label 1. Represent each leaf as the conjunction of the literals representing the path from the root. Take the disjunction of the terms.
  - Furthermore, a depth d decision tree can be represented as a d-DNF

- A binary decision tree with k leaves can also be represented with a k-clause CNF.
  - There are at most k leaves with label 0. Represent each leaf as the conjunction of the literals representing the path from the root. Take the disjunction of the terms to get the formula for $\neg f$.
  - Apply De Morgan's law to convert $\neg f$ into a k-clause CNF.
- Furthermore, a depth d decision tree can be represented as a d-CNF – the tree depth corresponds to the width of the formula.

Non-Binary Functions and Variables

- Can extend the tests in decision trees to more than binary outcomes, e.g to handle categorical variables.

- Can also handle real value variables e.g. using tests such as $(x_i > a)$.
- Can also be used to do regression – predict real value at the leaf.

Function Representation and Approximation

- Trees are universal approximators
  - Can represent any Boolean function.
  - Can approximate any continuous function.
    - Partition the input space into smaller and smaller partitions of constant values.

- However, some functions require large representations.
- To represent PARITY, require 2^d leaves: exponential in the number of inputs.
  - To see this, note that to represent XOR, each instance must be tested on all d variables, otherwise, the label cannot be determined – each instance must be at depth d.
  - A conjunction of all the variables (or negation) implies that only one element can be at each leaf.
  - Hence, there must be 2^d such leaves.

Small Conjunction of Rules but Large Tree

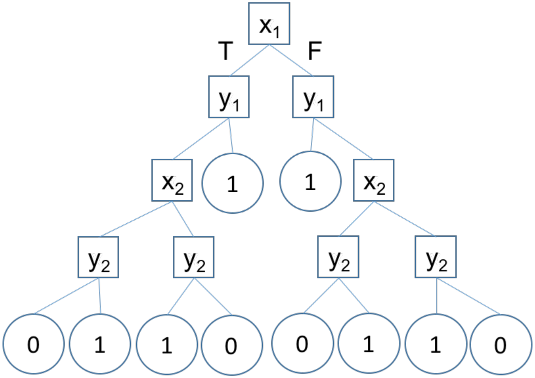
- We have argued that binary decision trees can be represented as a DNF where the number of positively labeled leaves equals the number of terms in the DNF.

- But decision trees have additional restriction of needing to recursively partition the input.
    - Does this significantly reduce the ability to represent compared to taking unions of conjunctions?

- Consider the problem $DISTINCT : \{0,1\}^{2d} \to \{0,1\}$ where

  $DISTINCT(x,y) = (x_1 \neq y_1)\ OR\ (x_2 \neq y_2) \cdots\ OR\ (x_d \neq y_d)$.

- Represented as a decision tree:



- By expanding $(x_i \neq y_i)$ into $(x_i\ AND\ \bar{y}_i)\ OR\ (\bar{x}_i\ AND\ y_i)$, we have a DNF representation of *DISTINCT* with $2d$ terms.

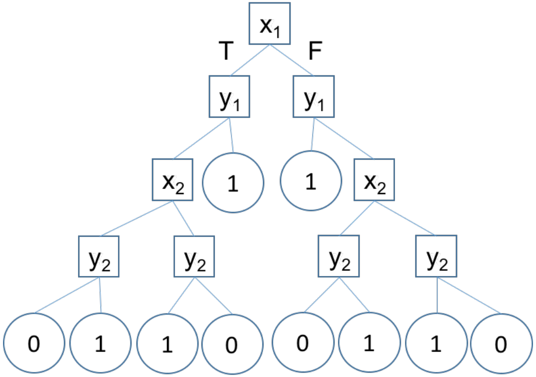
- However, it turns out that there is no small representation of *DISTINCT* as a decision tree.

Theorem: To represent *DISTINCT* as a decision tree requires at least 2^d leaves.

Consequence: Restricting the rules to take the form of a tree instead of disjunctions of them sometimes result in requiring an exponentially larger representation.

Proof in Appendix. Intuition:

- Leaves labeled 0 represent EQUAL.
- Requires all variables in the path.
- There must be exponential number of these leaves.



Exercise 5:
The Iris dataset contains 50 samples each of three types of the Iris flower: Iris Setosa, Iris Versicolor, and Iris Virginica. For each sample, the following measurements are given: sepal length, sepal width, petal width, petal length.

Run the experiment to build a decision tree for classifying Iris flowers based on their measurements.

From the visualization of the tree, write down a rule that correctly classifies all cases of Iris Setosa. How many disjunction of rules would cover all cases of positive instances of Iris Versicolor?

Exercise 6:
In this exercise, we look at the performance of learning a decision tree on DISTINCT. We will also try using ensemble (weighted average) of decision trees on the same problem.

Before running the experiment, predict which algorithm would do better and say why.

Exercise 7:

The Australian credit approval dataset illustrates multiple practical issues.

- Missing feature values: This is very common in practice. This can be handled in various ways.
    - Removing instances with missing feature values. This may be fine if the number of such instances is small. However, the issue still has to be handled when the predictor is deployed.
    - Imputing (predicting and filling in) the missing values. May affect performance if prediction is poor.
    - Encoding the missing value as a special value. This may be appropriate if the value is not missing at random and being missing actually provides some information.
    - Algorithm specific method. Decision trees have specific methods for handling missing values (e.g. averaging over both paths down the tree at the missing node) but this is not implemented in Scikit Learn. Generative models handle missing values naturally as part of probabilistic inference.

- The Australian dataset has a mix of continuous and categorical feature.
- For confidentiality purposes, feature names and values have been changed into meaningless symbols in the dataset. As a result, we cannot use our exploit knowledge of the problem to construct better predictors. One question is how to handle the categorical feature.
  - If the feature is an ordinal variable, i.e. the values are ordered, it may sometimes be useful to map the values to integers or reals, particularly if we expect the target value to change monotonically with the value of the feature.
  - If the feature is a nominal variable, i.e. the values cannot be ordered, mapping the values to integers or reals may not make sense. Some decision tree algorithms can do a multiway split at the variable with one child for each possible variable value. However, Scikit Learn simply treats all variables as integers/reals and do binary tests with $\leq$.

In this dataset, missing values are handled by imputation: mean values are used for continuous variables, and mode is used for categorical variables. For datasets like this, one strength of decision trees is that it is somewhat less sensitive to appropriate processing of variables.

Do you think the decision tree algorithm is sensitive to scaling of the variables? Yes or No.

Practical Properties

- Trees are usually more interpretable to humans (as long as it is not too large):
  - Can explain the classification of each instance by the conjunction of tests to the leaf that classifies that instance (a form of transparency).
  - Can explain the entire tree as a set of rules.
  - Can visualize tree.
- Features do not need to be scaled/normalized.

- However, decision trees learning methods can be unstable: small change in training data can result in large change in the learned tree.
- Often does not generalize as well as other learning methods – the restriction to the tree structure seems to sometimes affect performance in practice.
- But can use with ensemble methods to provide learning methods that scale well and generalize well – ensembles are able to represent unions, intersections and other functions of the base classes.
    - Multiple trees are often trained with random subsets of the training example to obtain a *random forest* and the output a combination of the outputs of the trees.
    - Weighted combinations of trees, often trained using boosting techniques, also often performs well.
    - Unfortunately, ensembles of trees no longer easy to interprete.

Outline

Linear Predictors

- Given input $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, a linear function, parameterized by a weight vector $\mathbf{w} = (w_1, \ldots, w_d)$, is defined as

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i.$$

  We will also use the notation $\langle \mathbf{w}, \mathbf{x} \rangle$ instead of $\mathbf{w}^T\mathbf{x}$ in parts of the course.

- A linear function is necessarily zero when $\mathbf{x}$ is the zero vector. An affine function has an offset b, often called the *bias*:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{d} w_i x_i + b.$$

- We can augment the input $\mathbf{x} = (x_1, \ldots, x_d)$ with an additional component x_0 that always take a value 1, so that an affine function is the same as a linear function with the augmented input component.
  - When it is not necessary to differentiate, we will refer to both linear and affine functions as linear functions.

- Linear functions are often used for regression giving **linear regression**.
  - Appropriate when it is reasonable to assume that the output is a (weighted) sum of the input components.
  - If we double each input component, the output will also double.
  - If the input $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$, then the output $f(\mathbf{x}) = f(\mathbf{x}_1) + f(\mathbf{x}_2)$.

Linear Classifier

- We can use a linear function as a classifier by composing it with the sign function. Also called *linear threshold function*.

$$f(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} w_i x_i \geq -b) \\ 0 & \text{otherwise} \end{cases}.$$

  We call $-b$ the threshold.
- Positive region often called a *halfspace*. We call the function class *homogeneous halfspaces* when threshold zero and *nonhomogeneous halfspaces* when threshold nonzero.

- The decision boundary is called *hyperplane*, which separates the space into two halfspaces.
- If a dataset can be correctly classified by a linear function, we say that it is *linearly separable*.

Outline

Margin and Hard SVM

- Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_, y_m)$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.
- The training set is *separable* if there exists $(\mathbf{w}, b)$ such that $y_i = \mathrm{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ for all i.
  - We use $\langle \mathbf{w}, \mathbf{x}_i \rangle$ rather than $\mathbf{w}^T \mathbf{x}_i$ to be consistent with the inner product notation in kernel methods later.
- This can also be written as

$$\forall i \in [m], \qquad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0.$$

- There are many possible linear functions that satisfy this. Which should we select? What criterion should we use to choose a "good" function?

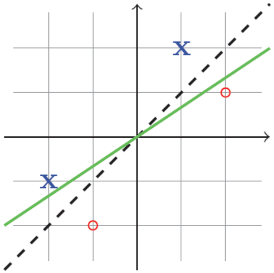


Figure from SSBD.

- Of the two lines shown, the dotted line appears better.
  - Even if we perturb the points, they will still be separated.
  - Robust to change in training to test set, to small change in data distribution.
- This suggests to maximize the margin (minimal distance from any point to the hyperplane).
  - Hard SVM returns the hyperplane that separates the points with the largest possible margin.

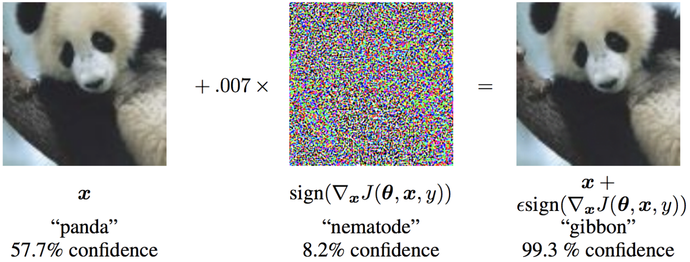

$+ .007 \times$
$=$
$\boldsymbol{x}$
“panda”
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Figure: From [1]. Example of a small perturbation changing the classification of a deep neural network. These examples are often known as adversarial examples.

Claim: (SSBD Claim 15.1) The distance between a point $\mathbf{x}$ and the hyperplane defined by $(\mathbf{w}, b)$ where $||\mathbf{w}|| = 1$ is $|\langle \mathbf{w}, \mathbf{x} \rangle + b|$.

Proof in the Appendix.

Consequences: We can transform the problem of maximizing distance to that of maximizing the magnitude of the output subject to $||\mathbf{w}|| = 1$.

The smallest magnitude on the data points subject to $||\mathbf{w}|| = 1$ is called the **margin**.

We will also refer to the value of $y(\langle \mathbf{w}, \mathbf{x} \rangle + b)$, where $y \in \{-1, 1\}$ is the label of $\mathbf{x}$ as the margin of the function at $\mathbf{x}$.

Hence Hard SVM solves for

$$\arg\max_{(\mathbf{w},b):||\mathbf{w}||=1} \min_{i\in[m]} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$$

When the problem is separable, this is equivalent to

$$\arg\max_{(\mathbf{w},b):||\mathbf{w}||=1} \min_{i\in[m]} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b). \tag{1}$$

We can turn the hard SVM problem into a quadratic optimization problem using the following lemma.

Lemma: (SSBD Lemma 5.2) Let

$$(\mathbf{w}_0, b_0) = \arg\min_{(\mathbf{w},b)} ||\mathbf{w}||^2 \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1.$$

Then $(\hat{\mathbf{w}}, \hat{b})$ where $\hat{\mathbf{w}} = \frac{\mathbf{w}_0}{||\mathbf{w}_0||}$ and $\hat{b} = \frac{b_0}{||\mathbf{w}_0||}$ is a solution to the Hard SVM problem.

Proof in the Appendix.

Quadratic programming can be solved in polynomial time, giving an efficient method for solving hard SVM when the problem is separable.

Soft SVM

- Soft SVM relaxes the hard constraints $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ for all i in order to handle non-separable cases.
- This is done by introducing slack variables $\xi_1, \ldots, \xi_m$ and replacing the constraints with $y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_i$.
- The slack variables ξ_i measure how much the constraints are being violated. We want the slack variables to be small. This leads to optimizing

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \left(\lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right)$$
$$\text{s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

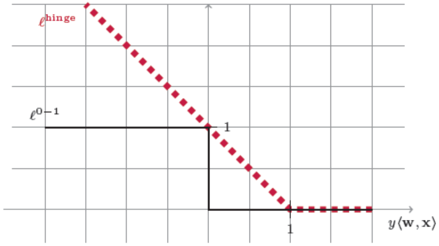


Figure from SSBD.

- We define the hinge loss as

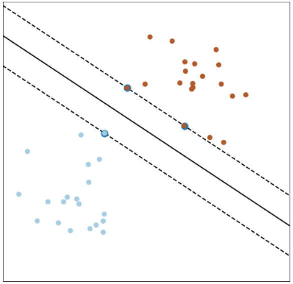
$$\ell^{hinge}((\mathbf{w}, b), (\mathbf{x}, y)) = \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}\rangle + b)\}.$$

- Let the average hinge loss on S be denoted $L_S^{hinge}((\mathbf{w}, b))$.
- The Soft SVM problem becomes a regularized loss minimization problem

$$\min_{\mathbf{w}, b} \left(\lambda ||\mathbf{w}||^2 + L_S^{hinge}((\mathbf{w}, b))\right).$$

Sparsity and Support Vectors

- The name Support Vector Machine comes from the fact that the solution of Hard SVM, $\mathbf{w}_0$ is supported by (i.e. in the span of) the examples that are exactly at distance $1/||\mathbf{w}_0||$ from the separating hyperplane.



- These vectors are called support vectors. Particularly useful for the case of non-linear kernels (to be discussed later).

We will need the following result.
Lemma: (Fritz John) Suppose that

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} f(\mathbf{w}) \text{ s.t. } \forall i \in [m], g_i(\mathbf{w}) \leq 0,$$

where $f, g_1, \ldots, g_m$ are differentiable. Then there exists $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $\alpha_0 \nabla f(\mathbf{w}^*) + \sum_{i \in I} \alpha_i \nabla g_i(\mathbf{w}^*) = \mathbf{0}$, where $I = \{i : g_i(\mathbf{w}^*) = 0\}$.

With additional constraints, we can ensure that $\alpha_0 \neq 0$. In particular, this holds when g_i are affine functions, giving a version of the Karush-Kuhn-Tucker (KKT) condition that we will use below.
Theorem: (SSBD Theorem 15.8) Let

$$(\mathbf{w}_0, b_0) = \arg\min_{(\mathbf{w}, b)} ||\mathbf{w}||^2 \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1.$$

and let $I = \{i : |\langle \mathbf{w}_0, \mathbf{x} \rangle + b_0| = 1\}$. Then there exists coefficients $\alpha_1, \ldots, \alpha_m$ such that $\mathbf{w}_0 = \sum_{i \in I} \alpha_i \mathbf{x}_i$.

Exercise 8:

In Scikit Learn, the SVC classifier optimizes $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$. In the experiment, we create a separable dataset from two Gaussians in 2 dimension centered at $(2,2)$ and $(-2,-2)$ and run the soft SVM with $C = 1$ and $C = 0.001$. In the plots

- The decision boundary is the solid line.
- The dotted lines are the lines where the function has magnitude 1.
- The larger circles show the points where $\alpha_i \neq 0$.

Write down some observations about the resulting plots.

Hinge Vs Logistic Loss

- Logistic regression uses the logistic loss

$$\ell_{log}(h,(x,y)) = -\log P(y|h(x)) = \log(1+\exp(-yh(x)),$$

  which is a composition of the logistic function to obtain probability from the linear function followed by the log loss.
  - Suitable when we want to use the output of the logistic function to represent probability.
  - With base 2, $\log_2(1+\exp(-yh(x))$ upper bounds the 0-1 loss (classification error).
  - All instances affect the solution (more global), does not have sparsity in the sense of support vectors.
  - Convex, gives a convex optimization problem with linear function.

- The hinge loss

$$\ell^{hinge}((\mathbf{w}, b), (\mathbf{x}, y)) = \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b)\}$$

  has the following properties:
  - Output not calibrated to be probability, although reasonable for ranking instances.
  - Upper bounds the 0-1 loss (classification error).
  - Only the support vector affect the solution (more local), correctly classified instances far from decision boundary has no effect.
  - Sparse representation in terms of using a small number of instances to represent solution.
  - Convex, gives a convex optimization problem with linear function.
- In practice, classification performance for hinge and logistic loss similar.

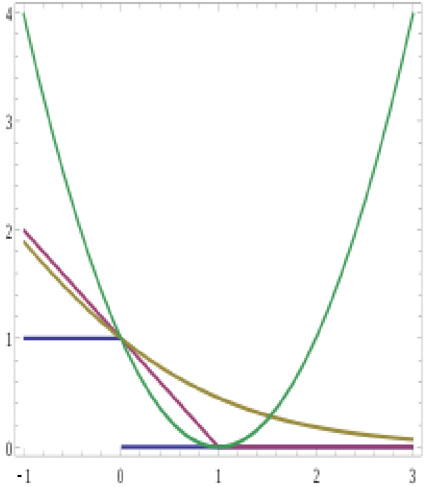


Image from Wikipedia. Various loss functions used with classification. All of them have value 1 at 0 and upper bound the 0-1 loss (blue). The logistic loss is yellow. Hinge loss is purple. Square loss is green.

Multiclass Problems

- For nearest neighbour and decision trees, multiclass problems are easily handled by encoding the classes using a categorical variable (e.g. an integer), and prediction is made using the majority class.
- Same approach does not work for linear functions.
- For linear functions, simplest way to solve multiclass classification is by reduction to binary classification.
- For k classes, the One-versus-All (also called One-versus-Rest) approach solves k binary problems. For problem i, where class i is positive and all other classes are negative, it produces a hypothesis h_i.
  - To do classification, the label corresponding to the function that has the largest valued output is selected $h(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} h_i(\mathbf{x})$..

- In the All-Pairs approach, we build a classifier $h_{i,j}$for every pair of classes (i, j) with i as the positive class.
  - To do classification, we use

$$h(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} \left(\sum_{j \neq i} h_{i,j}(\mathbf{x}) \right).$$

  - For each pair of classes, we usually only train one classifier $h_{i,j}$ and use $h_{j,i} = -h_{i,j}$ when we need $h_{j,i}$.

- Both One-vs-All and All-Pairs work well in practice.
    - One-vs-All solves k binary problems while All-Pairs solve $k(k-1)/2$ problems.
    - However, All-Pairs solve problems with smaller training sets.
    - Which method is faster depends on how the problems scale with training set size.

- Multi-class loss functions can also be used:
  - Composing the linear functions $h_i(\mathbf{x})$ with the softmax function to get $P(y = k|\mathbf{x}, h) = \frac{\exp(h_k(\mathbf{x}))}{\sum_{i=1}^{K} \exp(h_i(\mathbf{x}))}$ then using the log loss with the $P(y = k|\mathbf{x}, h)$ is commonly used in practice, often called *multiclass logistic regression*, *softmax regression* or *maximum entropy classifier*.
    - When the label set is very large, e.g. to label a sequence, graphical models is usually used to gain computational efficiency – called conditional random fields.
  - The hinge loss can also be extended into a multiclass hinge loss, also used for structured outputs such as label sequences.
  - Both multiclass logistic regression and learning with multiclass hinge loss are convex problems when used with linear functions.

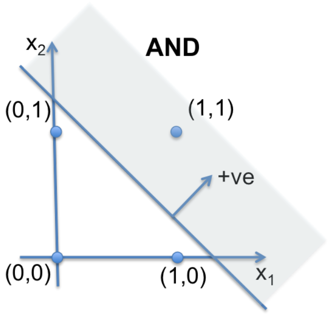
Representation Power of Linear Threshold Functions

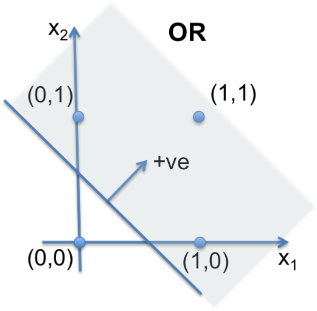
- What are the boolean functions representable by by the linear threshold function $\text{sign}(\sum_{i=1}^{d} w_i x_i + b)$?
  - AND function: Set $b = 0.5 - d$ and all other $w_i = 1$.
  - OR function: Set $b = -0.5$ and all other $w_i = 1$.
  - NOT function of a single variable: Set $b = 0.5$, $w_1 = -1$.

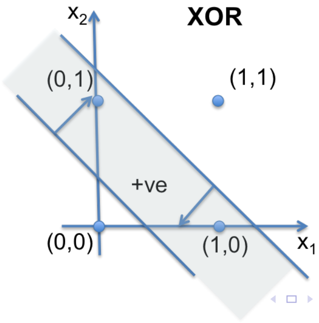
- The XOR function $f(x_1, x_2) = 1$ for $(1, 0)$ and $(0, 1)$ and 0 otherwise is not representable. To represent XOR, we need

$$\begin{aligned} b &< 0 \qquad \text{from } f(0,0) = 0 &(2)\\ w_2 + w_1 + b &< 0 \qquad \text{from } f(1,1) = 0 &(3)\\ w_1 + b &\geq 0 \qquad \text{from } f(1,0) = 1 &(4)\\ w_2 + b &\geq 0 \qquad \text{from } f(0,1) = 1 &(5)\end{aligned}$$

  Adding (2) and (3), we get $w_2 + w_1 + 2b < 0$ while adding (4) and (5), we get $w_2 + w_1 + 2b \geq 0$. Contradiction.
  More generally, the **parity** function (even or odd number of ones in the input) in higher dimensions cannot be represented by halfspaces.
- According to folklore, the inability of linear threshold functions (perceptron) to represent XOR (and the parity function) encouraged researcher to look into symbolic/logic representations instead of neural networks in the 1970.


AND
x_2
(0,1)
(1,1)
+ve
(0,0)
(1,0)
x_1


OR
x_2
(0,1)
(1,1)
+ve
(0,0)
(1,0)
x_1


XOR
x_2
(0,1)
(1,1)
+ve
(0,0)
(1,0)
x_1

When is a linear classifier appropriate?

- If the probability of the positive class does not change (increase or decrease) monotonically with the value of each input, then linear classifier may not be suitable.
    - If the class goes from negative to positive then back to negative as the value of an input increases, cannot be represented by linear classifier (e.g. XOR).

Exercise 9:
In the Adult Census dataset we are given information about individuals obtained from the US census and asked to predict whether the person earns more than US$50K a year. The features consist of

- Continuous features "Age", "fnlwgt", "Education-Num", "Capital Gain", "Capital Loss", "Hours per week", and
- Categorical features "Workclass", "Education", "Marital Status", "Occupation", "Relationship", "Race", "Sex", "Country".

Categorical features cannot be used directly in a linear classifier. One simple way to handle this is to simply map each category of a feature to an integer.

Before running the exercise, think of a potentially better way to handle categorical variable and write it down. Run the experiment, then discuss.

Exercise 10:

Linear classifiers work quite well for certain applications. They usually work quite well for text classification.

Run the experiment and suggest some reasons why linear classifiers may work well for text classification. Suggest possible ways to improve text classification (not necessarily for this dataset).

Practical Properties

- Linear classifiers are often practically effective.
- Fast to learn, usually convex optimization, scales to very large datasets.
- Often need to encode nominal categorical features with dummy/one hot encoding.
- For classification, ordinal features may sometimes be fine encoded as integers.
- Often need to scale/normalize features.
- With appropriate regularization, e.g. running SVM or regularized logistic regression, feature selection may be less important.
- Can often get sparse weight solution using ℓ_1 regularization.

How many functions can a linear threshold function represent?

- How many Boolean functions can halfspaces in $\mathbb{R}^d$ represent?
- More generally, how many functions can a hypothesis class $\mathcal{H}$ represent on an arbitrary set of m input points (not necessarily binary inputs) $c_1, \ldots, c_m$?
- We will answer the question in terms of a combinatorial parameter of function classes called the **VC-dimension**.

- **Restriction of $\mathcal{H}$ to C:** Let $\mathcal{H}$ be a class of functions from $\mathcal{X}$ to $\{-1, 1\}$ and let $C = \{c_1, \ldots, c_m\} \subset \mathcal{X}$. The restriction of $\mathcal{H}$ to C is the set of functions from C to $\{-1, 1\}$ that can be derived from $\mathcal{H}$, i.e.

$$\mathcal{H}_C = \{(h(c_1), \ldots, h(c_m) : h \in \mathcal{H}\},$$

  where each function from $\mathcal{H}$ is represented as a vector in $\{-1, 1\}^{|C|}$.

- **Shattering:** A hypothesis class $\mathcal{H}$ *shatters* a set $C \subset \mathcal{X}$ if the restriction of $\mathcal{H}$ to C is the set of all functions from C to $\{-1, 1\}$, i.e. $|\mathcal{H}_C| = 2^{|C|}$.

- **VC-dimension:** The VC-dimension of hypothesis class $\mathcal{H}$ is the size of the largest set $C \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$.

- **Growth function:** The growth function $\tau_{\mathcal{H}}(m)$ of $\mathcal{H}$ is the largest number of functions that can be obtained when the input is restricted to a set of size m:

$$\tau_{\mathcal{H}}(m) = \max_{C \subset \mathcal{X} : |C| = m} |\mathcal{H}_C|.$$

- **Sauer's Lemma:** Let $\mathcal{H}$ be a hypothesis class with VCdim($\mathcal{H}$)$\leq d < \infty$. Then, for all m, $\tau_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}$. In particular, if $m > d + 1$, then

$$\tau_{\mathcal{H}}(m) \leq (em/d)^d.$$

VC-dimension of Linear Classifiers

Theorem: (SSBD Theorem 9.2) The class of homogeneous halfspaces in $\mathbb{R}^d$ has VC-dimension d and the class of nonhomogeneous halfspaces in $\mathbb{R}^d$ has VC-dimension $d + 1$.

Proof in Appendix.

Observations:

- For linear functions, VCdim is the same as the number of parameters (not true in general).
- The number of functions is $O(m^d)$ when input set of size m. Small fraction of 2^m possible functions.
- The polynomial (with fixed d) growth function will have implications on the number of examples required to learn the function class (later in the course).

Exercise 11: Consider a linear threshold function in d dimensional space with binary weights (0 or 1). Further assume that the length of the weight vectors is at most $\sqrt{k}$. What is the VC-dimension of this function class?

A. VDdim$< k$

B. VCdim$= O(k \log d)$

C. VCdim$\geq d$

Reading

Some material are taken directly from SSBD.

1. SSBD Chapter 19 on nearest neighbour
2. SSBD section 9.1 on halfspaces.
3. SSBD sections 15.1, 15.2, 15.3 on SVM
4. SSBD section 6.2, 6.3 and 6.5.1 on VC dimension.
5. SSBD section 17.1, for multiclass problems.

References I

[1] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

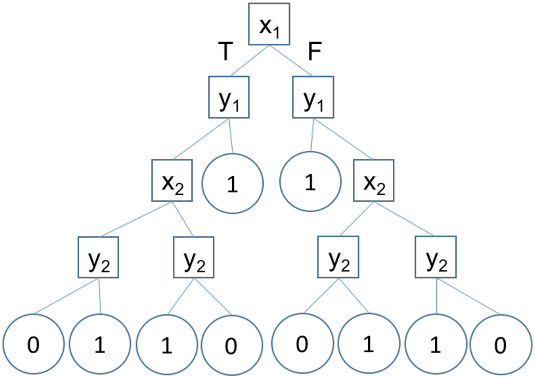
Outline

Decision Tree Proofs

- Recall the problem $DISTINCT : \{0,1\}^{2d} \rightarrow \{0,1\}$ where

  $DISTINCT(x,y) = (x_1 \neq y_1)\ OR\ (x_2 \neq y_2) \cdots\ OR\ (x_d \neq y_d)$.

- Represented as a decision tree:



Theorem: To represent *DISTINCT* a decision tree requires at least 2^d leaves.

Proof:

- First note that we can represent the negation of a decision tree function simply by negating each of the values of the leaves.
- Consider the function $EQUAL = NOT\ \ DISTINCT$.
  - We will argue that *EQUAL* requires a DNF with at least 2^d terms.
  - As a DNF with 2^d terms can always represent a tree with 2^d leaves, a tree will require at least 2^d leaves to represent *EQUAL*.
  - Since negation can be represented with a tree of the same size, *DISTINCT* will also require at least 2^d leaves.

- First we argue that any term in any DNF for *EQUAL* must include all $2d$ variables.
  - Assume a term has one variable missing.
  - Find an (x, y) with $x = y$ where the term evaluates TRUE.
  - Change the missing variable so that $x \neq y$.
  - Since the variable is missing, the term still evaluates to TRUE making the DNF evaluate TRUE, contradicting the fact that the DNF represents *EQUAL*.
- As a term in the DNF contains all $2d$ variables, it evaluates TRUE for exactly one value of (x, y).
- But at least 2^d values of (x, y) satisfies $x = y$.
- Hence the DNF must contain at least 2^d terms. $\square$

SVM Proofs

Claim: (SSBD Claim 15.1) The distance between a point $\mathbf{x}$ and the hyperplane defined by $(\mathbf{w}, b)$ where $||\mathbf{w}|| = 1$ is $|\langle \mathbf{w}, \mathbf{x} \rangle + b|$.

Proof:

- The distance from a point $\mathbf{x}$ to the hyperplane is

$$\min\{||\mathbf{x} - \mathbf{x}'|| : \langle \mathbf{w}, \mathbf{x}' \rangle + b = 0\}.$$

- We argue that the point $\mathbf{x}' = \mathbf{x} - (\langle \mathbf{w}, \mathbf{x} \rangle + b)\mathbf{w}$ is the point on the hyperplane that minimizes the distance to $\mathbf{x}$.
  - First we show that $\mathbf{x}'$ is on the hyperplane

$$\langle \mathbf{w}, \mathbf{x}' \rangle + b = \langle \mathbf{w}, \mathbf{x} \rangle - (\langle \mathbf{w}, \mathbf{x} \rangle + b)||\mathbf{w}||^2 + b = 0.$$

- (Continued)
  - Now we show that $||\mathbf{x} - \mathbf{x}''|| \geq ||\mathbf{x} - \mathbf{x}'||$ for any point x'' on the hyperplane, i.e. satisfying $\langle \mathbf{w}, \mathbf{x}'' \rangle + b = 0$.

$$\begin{aligned} ||\mathbf{x} - \mathbf{x}''||^2 &= ||\mathbf{x} - \mathbf{x}' + \mathbf{x}' - \mathbf{x}''||^2 \\ &= ||\mathbf{x} - \mathbf{x}'||^2 + ||\mathbf{x}' - \mathbf{x}''||^2 + 2\langle \mathbf{x} - \mathbf{x}', \mathbf{x}' - \mathbf{x}'' \rangle \\ &\geq ||\mathbf{x} - \mathbf{x}'||^2 + 2\langle \mathbf{x} - \mathbf{x}', \mathbf{x}' - \mathbf{x}'' \rangle \\ &= ||\mathbf{x} - \mathbf{x}'||^2 + 2(\langle \mathbf{w}, \mathbf{x} \rangle + b)\langle \mathbf{w}, \mathbf{x}' - \mathbf{x}'' \rangle \\ &= ||\mathbf{x} - \mathbf{x}'||^2. \end{aligned}$$

  where the last line is because $\langle \mathbf{w}, \mathbf{x}' \rangle = \langle \mathbf{w}, \mathbf{x}'' \rangle = -b$.

- Finally, note that

$$||\mathbf{x} - \mathbf{x}'|| = |\langle \mathbf{w}, \mathbf{x} \rangle + b|||\mathbf{w}|| = |\langle \mathbf{w}, \mathbf{x} \rangle + b|.$$

□

Lemma: (SSBD Lemma 5.2) Let

$$(\mathbf{w}_0, b_0) = \arg\min_{(\mathbf{w},b)} ||\mathbf{w}||^2 \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x}\rangle + b) \geq 1.$$

Then $(\hat{\mathbf{w}}, \hat{b})$ where $\hat{\mathbf{w}} = \frac{\mathbf{w}_0}{||\mathbf{w}_0||}$ and $\hat{b} = \frac{b_0}{||\mathbf{w}_0||}$ is a solution to finding

$$\arg\max_{(\mathbf{w},b):||\mathbf{w}||=1} \min_{i\in[m]} y_i(\langle \mathbf{w}, \mathbf{x}\rangle + b). \tag{6}$$

Proof:

- Let $(\mathbf{w}^*, b^*)$ be a solution to (6) with margin
$$\gamma^* = \min_{i\in[m]} y_i(\langle \mathbf{w}^*, \mathbf{x}\rangle + b^*).$$
- Dividing by γ^*, we have for all i
$$y_i\left(\left\langle \frac{\mathbf{w}^*}{\gamma^*}, \mathbf{x}\right\rangle + \frac{b^*}{\gamma^*}\right) \geq 1.$$
- The pair $(\frac{\mathbf{w}^*}{\gamma^*}, \frac{b^*}{\gamma^*})$ is a feasible solution to the quadratic optimization problem, hence $||\mathbf{w}_0|| \leq ||\frac{\mathbf{w}^*}{\gamma^*}|| \leq \frac{1}{\gamma^*}$.

- It follows that

$$y_i(\langle \hat{\mathbf{w}}, \mathbf{x} \rangle + \hat{b}) = \frac{1}{||\mathbf{w}_0||} y_i(\langle \mathbf{w}_0, \mathbf{x} \rangle + b_0) \geq \frac{1}{||\mathbf{w}_0||} \geq \gamma^*.$$

- Since $||\hat{\mathbf{w}}|| = 1$ and its margin is as large as the optimal margin it is a solution to (6). □

VCdim of Linear Classifiers

Theorem: (SSBD Theorem 9.2) The class of homogeneous halfspaces in $\mathbb{R}^d$ has VC-dimension d and the class of nonhomogeneous halfspaces in $\mathbb{R}^d$ has VC-dimension $d+1$.
Proof:

- Consider vectors $\mathbf{e}_1, \ldots, \mathbf{e}_d$ where every vector $\mathbf{e}_i$ is zero everywhere except at the i-th coordinate where it is 1.
  - This set is shattered by homogeneous halfspaces.
  - For each labeling $y_1, \ldots, y_d$, set $\mathbf{w} = (y_1, \ldots, y_d)$. Then $\mathbf{w}^T \mathbf{e}_i = y_i$ for all i.
  - For the case of nonhomogeneous halfspaces, add the zero vector $\mathbf{0}$ to the set of points. If the label for $\mathbf{0}$ is 1, set b to $1/2$, otherwise to $-1/2$. This does not change the label of any other points.

- Consider the homogeneous case. Let $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$ be $d+1$ vectors in $\mathbb{R}^d$.
- As they must be linearly dependent, there must be real numbers $a_1 \ldots, a_{d+1}$, not all of them zero, such that $\sum_{i=1}^{d+1} a_i \mathbf{x}_i = \mathbf{0}$.
- Let $I = \{i : a_i > 0\}$ and $J = \{j : a_j < 0\}$. Either I or J, or both are nonempty. Assume both are not empty. Then

$$\sum_{i \in I} a_i \mathbf{x}_i = \sum_{j \in J} |a_j| \mathbf{x}_j. \tag{7}$$

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$ is shattered. Then we can find a weight vector $\mathbf{w}$ such that $\mathbf{w}^T\mathbf{x}_i > 0$ for all $i \in I$ and $\mathbf{w}^T\mathbf{x}_j \leq 0$ for all $j \in J$. It follows that

$$0 < \sum_{i \in I} a_i \mathbf{x}_i^T \mathbf{w} = \sum_{j \in J} |a_j| \mathbf{x}_j^T \mathbf{w},$$

  where the equality follows from (7). But this contradicts $\mathbf{w}^T\mathbf{x}_j \leq 0$ for all $j \in J$.
- Can similarly derive contradiction for cases where I or J empty.
- Contradiction proof for homogeneous case also holds for nonhomogeneous case with $d + 2$ input vectors, as we can fix one dimension of input to 1 to simulate bias. $\square$