Lecture Note

CS5339 Theory and Algorithm for Machine Learning

Meng-Jiun Chiou
National University of Singapore
mengjiun.chiou@u.nus.edu

Contents

1 Formulation

1.1 Supervised Learning

1.1.1 Empirical Risk Minimization (ERM)

The learner does not know D and only have access to the training set S, a sample from D. For a predictor $h : X \to Y$, we can approximate the expected error by using the training set error

$$L_S(h) = \frac{|i \in 1, ..., m : h(x_i) \neq y_i|}{m}. \tag{1}$$

It can be rewritten using the 0-1 loss

$$L_S(h) = \frac{\sum_{i=1}^{m} l_{0-1}(h, (x_i, y_i))}{m}. \tag{2}$$

Training set error is often called the empirical error or empirical risk. Given a hypothesis class H, finding the hypothesis $h \in H$ that minimizes the empirical risk is a simple learning strategy, which is often called empirical risk minimization (ERM).

1.1.2 Maximum Likelihood Estimation (MLE)

Assume that the data distribution P is known up to some parameter $h \in H$, MLE selects the h that maximizes the probability of the data S being observed:

$$h_{ML} = \arg\max_{h \in H} P(S|h). \tag{3}$$

For i.i.d. data $S = (z_1, ..., z_m) \sim D^m$, this becomes:

$$h_{ML} = \arg\max_{h \in H} \prod_{i=1}^{m} D(z_i|h). \tag{4}$$

For supervised learning, $z_i = (x_i, y_i)$ and we have

$$h_{ML} = \arg\max_{h \in H} \prod_{i=1}^{m} D(y_i|x_i, h)D(x_i) \tag{5}$$

$$= \arg\max_{h \in H} \prod_{i=1}^{m} D(y_i|x_i, h) \tag{6}$$

$$= \arg\max_{h \in H} \sum_{i=1}^{m} logD(y_i|x_i, h). \tag{7}$$

With an equivalent distribution can be found, empirical risk minmization (ERM) is equal to maximum likelihood when for loss function

$$l(h, (x, y)) = -\ln p(y|x, h). \tag{8}$$

1.1.3 Classification Loss Functions

For binary classification using $h(x) \in (-\infty, \infty)$, we often compose the output of our function $h(x)$ with the logistic (sigmoid) function to get a probability model

$$P(y = 1|x, h) = \frac{\exp(h(x))}{1 + \exp(h(x))} = \frac{1}{1 + \exp(-h(x))} \tag{9}$$

$$P(y = -1|x, h) = 1 - \frac{\exp(h(x))}{1 + exp(h(x))} = \frac{1}{1 + \exp(h(x))}. \tag{10}$$

For $y \in \{1, -1\}$, can write $P(y|x,h) = \frac{1}{1+\exp(-yh(x))}$. Therefore log loss can be written as

$$l_{\log}(h,(x,y)) = -\log P(y|h(x)) = \log(1+\exp(-yh(x))). \tag{11}$$

When $h(x)$ is a linear function, this is often called *logistic regression*.

For multi-class classification,

$$P(y=k|x,h) = \frac{\exp(h_k(x))}{\sum_{i=1}^{K}\exp(h_i(x))}. \tag{12}$$

When $h_i(x)$ is linear, it is often called *multiclass logistic regression, softmax regression or maximum entropy classifier*.

1.1.4 Maximum A Posteriori Estimation (MAPE)

Instead of maximizing the likelihood, MAP find the parameter that maximizes the posterior probability $P(h|D)$:

$$\begin{aligned} h_{MAP} &= \arg\max_{h\in H} P(h|D) && (13)\\ &= \arg\max_{h\in H} P(D|h)P(h) && (14)\\ &= \arg\max_{h\in H} \prod_{i=1}^{m} D(z_i|h)P(h) && (15)\\ &= \arg\max_{h\in H} \sum_{i=1}^{m} \Big(\log D(y_i|x_i,h) + \log P(h)\Big). && (16) \end{aligned}$$

So balance between fitting the data (likelihood) and fitting the prior well. For example, we add an zero mean Gaussian prior on the weight vector $\lambda||\mathbf{w}||^2$ in loss function of linear regression. This is often called *ridge regression* or penalized least square. Minimizing a combination of empirical risk and a regularizer is also called Regularized Risk Minimization (RRM).

1.1.5 Bayesian Estimation

Instead of selecting a single h, we maintain the posterior distribution over the parameters $P(h|z_1,...,z_m)$. This can be used to make optimal prediction (assuming the prior is correct) for the variable of interest. For example,

$$\begin{aligned} P(y|x,z_1,...,z_m) &= \int_h P(y,h|x,z_1,...,z_m) && (17)\\ &= \int_h P(y|h,x,z_1,...,z_m)P(h|z_1,...,z_m). && (18) \end{aligned}$$

1.2 Unsupervised Learning

Many (but not all) unsupervised learning problems can be posed as density estimation problems, i.e. learning the distribution of the data. We can also pose the problem of learning the data distribution as a maximum likelihood (or maximum a posteriori) estimation problem.

For lossless compression, using $P(x_1,...,x_n)$ to compress $x_1,...,x_n$ using Hoffman coding or arithmetic coding, gives the shortest code length of $-\sum_{i=1}^{m}\log P(x_i|h)$ (ignoring rounding required for discrete lengths). Therefore, maximizing $P(x_i|h)$ is equal to minimizing number of bits required.

1.3 Discriminative/Generative Models

The approach of directly optimizing for the loss that we are interested in is called **discriminative learning**. Another approach is to ignore the fact that the data for supervised learning comes in pairs $z = (x, y)$ and learn the data distribution (density estimation) for z. After we have learned a model for $p(z) = p(x, y)$, we perform inference to get our estimate of y by computing $p(y|x)$. This approach is called **generative learning**. We illustrate this approach using two common learning models: naive Bayes and linear discriminant analysis (LDA).

1.3.1 Naive Bayes

The Bayes optimal classifier is

$$h_{Bayes}(x) = \arg\max_{y \in -1,1} P(Y = y|X = \mathbf{x}) \tag{19}$$

$$= \arg\max_{y \in -1,1} P(Y = y)P(X = \mathbf{x}|Y = y)/Z \tag{20}$$

and we make the naive generative assumption that, the features are independent of each other given the label, i.e.

$$P(X = \mathbf{x}|Y = y) = \prod_{i=1}^{d} P(X_i = x_i|Y = y). \tag{21}$$

Hence, the classifier becomes

$$h_{Bayes}(x) = \arg\max_{y \in -1,1} P(Y = y) \prod_{i=1}^{d} P(X_i = x_i|Y = y) \tag{22}$$

$$= \arg\max_{y \in -1,1} \log P(Y = y) + \sum_{i=1}^{d} \log P(X_i = x_i|Y = y) \tag{23}$$

$$= sign(w_0 + \sum_{i=1}^{d} (w_{i,0}x_{i,0} + wi, 1xi, 1)). \tag{24}$$

There comes a question: which is better, Naive Bayes or logistic regression? An answer [1] said it depends on the distribution of data. If the data set is large, discriminative model may perform better as it models directly $p(y|x)$. On the other hand, if there are some extra unlabeled data, generative model may do better as it can deal with missing data.

1.3.2 Linear Discriminant Analysis (LDA)

LDA and QDA classifiers are attractive because they have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice and have no hyperparameters to tune. Linear Discriminant Analysis can only learn linear boundaries, while Quadratic Discriminant Analysis can learn quadratic boundaries and is therefore more flexible.

1.3.3 Discriminative VS Generative Classifiers

Here are some comparisons between these two classes of models. Pros for **discriminative models**:

- Discriminative classifiers are more robust against misspecification of the model, while generative classifiers depend on the model being correct. If the model is wrong, the discriminative model can still converge to the best approximation within the model class as the data size increases.
- It is usually easier to add features to discriminative models. For generative model, we need to try to find the correct model for the feature, which can be difficult.

Pros for **generative models**:

- It is sometimes computationally cheaper to train generative classifiers, e.g. Naive Bayes mostly requires only counting.
- It may converge faster, e.g. Naive Bayes converges faster than logistic regression.
- It handles missing and unlabeled training data naturally. In doing inference, they are marginalized away. However, in discriminative models there is no standard way for handling missing/unlabeled variables.

1.4 Feature

1.4.1 Filter

Asses each feature independently of other features. We may select k features that achieves the highest score, using **Pearson correlation coefficient**:

$$\rho_(X_j, Y) = \frac{cov(X_j, Y)}{\sqrt{var(X_j)var(Y)}} = \frac{cov(X_j, Y)}{\sigma_{X_j}\sigma_Y} \tag{25}$$

For classification, **mutual information** (MI) is often used. Not limited to real-valued random variables like the correlation coefficient, MI is more general and determines how similar the joint distribution $p(X, Y)$ is to the products of factored marginal distribution $p(X)p(Y)$. MI is defined as:

$$I(X_i; Y) = \sum_{x_i}\sum_{y} p(x_i, y) \log \frac{p(x_i, y}{p(x_i)p(y)} \tag{26}$$

$$= H(Y) - H(Y|X), \tag{27}$$

where $p(x_i, y)$ is the joint probability function of X and Y, and $p(x_i)$ and $p(y)$ are the marginal probability distribution functions of X and Y respectively and $H(Y) - H(Y|X)$ is the difference between the entropy of the label Y and the conditional entropy of Y given input X and is also called the **information gain**. For feature selection, we rank features according to information gain, and select the k features with the largest information gain. Another commonly used feature selection method is **chi square** which tests whether the feature is independent of the label.

1.4.2 Wrapper

Filter approaches do not take into account dependence among the features. Wrapper approaches search for subset of variables that will perform well. Common greedy methods include:

- Forward selection: Features are progressively added as learning is done. e.g. Viola-Jones face detector is trained using Adaboost.
- Backward elimination: Starting with all variables and progressively eliminate the least promising ones.

1.4.3 Sparsity-Inducing Norms

We can formulate the feature selection problem as:

$$\min_w L_S(\mathbf{w}) s.t. ||\mathbf{w}||_0 \leq k, \tag{28}$$

where $||\mathbf{w}||_0 = |\{i : w_i \neq 0\}|$. is also called $l0$ norm. As this is often computationally hard, so we often relax into $l1$ norm:

$$\min_w L_S(\mathbf{w}) s.t. ||\mathbf{w}||_1 \leq k, \tag{29}$$

which can be solved efficiently if L_S is convex (convex optimization problem). We can also encourage a sparse solution to use the $l1$ norm of the weight as a regularizer,

$$\min_w (L_S(\mathbf{w}) + \lambda||\mathbf{w}||_1), \tag{30}$$

which again is convex if L_S is convex, e.g. in logistic regression.

We can derive Lasso as MAP with each w_i independently distributed with Laplacian prior $Pr(w_i) = \frac{\lambda}{2}\exp(-\lambda|w_i|)$. Refer to Figure 1. Compared to Lasso, Ridge regression performs poor for the data out

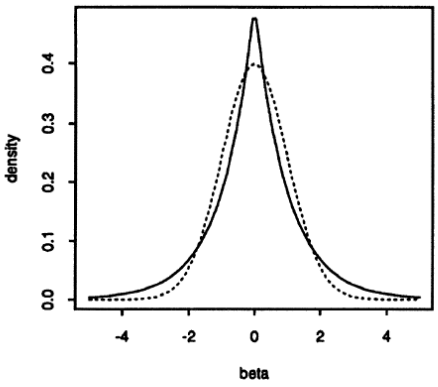

Fig. 7. Double-exponential density (———) and normal density (- - - - -): the former is the implicit prior used by the lasso; the latter by ridge regression

Figure 1: Laplacian prior (l1, double-exponential density) versus Gaussian prior (l2, normal density).

of range of training data. Lasso regression zeros out most of the coefficients.

1.4.4 Common Feature Transformation Methods

Let $\mathbf{f} = (\mathrm{f}_1, ..., \mathrm{f}_m) \in \mathbb{R}^m$ be the value of feature f over the m training examples and $\bar{\mathrm{f}} = \frac{1}{m}\sum_{i=1}^{m} \mathrm{f}_i$.

- Centering
- Unit Range
- Standardization (zero mean and unit variance)
- Clipping
- Sigmoidal Transformation
- Logarithmic Transformation

1.4.5 Feature Learning

Instead of selecting a subset of predefined features, we can also learn a feature mapping $\varphi : \mathcal{X} \to \mathbb{R}^d$ which maps instances in $\mathcal{X}$ to d dimensional feature space. If we have a lot of unlabeled data, may be useful to do unsupervised feature learning, called **dictionary learning**.

To do dictionary learning, it is common to construct an autoencoder consisting of a function pair: an encoder $\varphi : \mathbb{R}^d \mapsto \mathbb{R}^k$ and a decoder $\phi : \mathbb{R}^k \mapsto \mathbb{R}^d$. The goal of learning is to minimize the reconstruction error: $\sum_i ||\mathbf{x}_i - \phi(\varphi(\mathbf{x}_i))||^2$. Principal Component Analysis (PCA) is an example where $k < d$.

2 Representation

Some interesting functions can be represented/approximated:

- **Boolean functions**: arbitrary boolean functions from $\{0,1\}^d$ to $\{0,1\}$.
- **PARITY**: a boolean function from $\{0,1\}^d$ which returns 1 if the number of the occurences of 1 in the input is odd and returns 0 otherwise.
- **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 $|\mathrm{f}(\mathrm{x}) - f(x')| < \epsilon$.
- **Lipschitz functions**: A function f is c-Lipschitz for some $c > 0$ if for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $|\mathrm{f}(\mathbf{x}) - \mathrm{f}(\mathbf{x}')| \leq c||\mathbf{x} - \mathbf{x}'||$.

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 $\sup_{x \in [0,1]^d} |f(\mathbf{x}) - g(\mathbf{x})| < \epsilon$.

2.1 Nearest Neighbor (NN)

Among the simplest learning algorithm, it simply memorize the training set and predict with the label of the nearest neighbor (NN) in the training set. It requires a distance function to measure the distances. NN can represent any Boolean function – simply store the label for each possible input. NN is also an universal approximator – store the value of the function at a fine enough regular grid. NN has some properties:

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

NN suffers curse of dimensionality. (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,

$$E_{S \sim \mathcal{D}^m}[L_D(h_S)] \leq 2L_{\mathcal{D}}(h^*) + 4c\sqrt{d}m^{-\frac{1}{d+1}}, \tag{31}$$

where h^* is the Bayes decision rule. The first term is optimal error, and the last term represents curse of dimension. For the last term to be smaller than σ, $m > (4c\sqrt{d}/\epsilon)^{d+1}$ is sufficient. This means sample size grows exponentially with dimension d. Some properties:

- It can bound the expected error by twice the optimum plus c times the expected distance of the points from its nearest neighbors.
- It cover the space with boxes of sides ϵ.
- If point and nearest neighbor fall in the same box, distance at most $\sqrt{d}\epsilon$, otherwise upper bound with largest distance in $[0,1]^d$ which is $\sqrt{d}$.
- The probability of nearest neighbor falling into a different box can be bounded by a constant factor of the number of boxes, which is roughly proportional to volume which grows exponentially with dimension.
- Balanccing ϵ with probability gives second term in the bound.

SSBD Theorem 19.4 shows that the sample size necessarily grows exponentially with dimension if we only impose the Lipschitz condition. Some other 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. If all tests and final prediction are binary, the function represented by the decision tree can be interpreted as a DNF (disjunction of conjunctions).

2.2 Decision Tree

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.

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

- **Disjunctive Normal Form (DNF)**: Disjunction (OR) of conjunctions (AND) of variables or their negations, e.g. $(x_1 \wedge \neg x_3 \wedge x_4) \vee ... \vee (x_2 \wedge x_5 \wedge \neg x_8)$.
  - **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 width, is no more than k.
- **Conjunctive Normal Form (CNF)**: Conjunction (AND) of disjunctions (OR) of variables or their negations, e.g. $(x_2 \vee \neg x_5) \wedge ... \wedge (\neg x_4 \vee x_6 \vee x_9)$.
  - **k-clause CNF**: when the number of disjunctive clauses is no more than k.
  - **k-CNF**: when the number of literals, or width, is no more than k.

A decision tree representation 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**. On the other hand, a binary decision tree with k leaves can also be represented with a **k-clause CNF**, by representing each leaf as the conjunction of the literals representing the path to the leaves with label 0 from the root. A depth d decision tree can also be represented as a **d-CNF** – the tree depth corresponds to the width of the formula.

Trees are **universal approximators**, since it can

- represent any Boolean function
- approximate any continuous function

However, to represent PARITY and DISTINCT: $\{0,1\}^{2d} \to \{0,1\}$, it requires 2^d leaves: exponential in the number of inputs. **Restricting the rules tot take the form of a tree instead of disjunctions of them sometimes result in requiring an exponentially larger representation**.

Some practical properties of trees:

- Trees are usually more interpretable to humans
- Features do not need to be scaled/normalized.
- However, it 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
- 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 trained with random subsets of the training example to obtain a *random forest* and then output a combination of the outputs of the trees.
  - Weighted combinations of trees, often trained using boosting techniques also performs well.

2.3 Linear Predictors

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

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i. \tag{32}$$

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

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{d} w_i x_i + b = \sum_{i=0}^{d} w_i x_i. \ \ (\text{set } w_0 = 0) \tag{33}$$

Linear functions are often used for regression giving linear regression.

We can also use a linear function as a classifier by composing it with the sign function, which is also called *linear threshold function*:

$$f(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} w_i x_i \geq -b \\ 0 & \text{otherwise} \end{cases} . \tag{34}$$

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

2.4 Support Vector Machine

2.4.1 Margin and Hard SVM

Let $S = (x_1, y_1), ..., (x_m, y_m)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. The training set is separable if there exists $(\mathbf{w}, b)$ such that $y_i = \text{sign}(\langle \mathbf{w}, x_i \rangle + b)$ for all i. That is, $\forall i \in [m],\ y_i(\langle \mathbf{w}, x_i \rangle + b) > 0$. Hard SVM returns the hyperplane that separates the points with the largest possible margin.

(SSBD Claim 15.1) The distance between a point $\mathbf{x}$ and the hyperplane defined by $(\mathbf{w}, b)$ where $||\mathbf{w}|| = 1$ is $|\langle \mathbf{w}, \mathbf{x} \rangle + b|$. Therefore, 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**. Hence, when the problem is separable ($\forall i, y_i(\langle \mathbf{w}, x_i \rangle + b) > 0$), hard SVM solves for

$$\underset{(\mathbf{w},b):||\mathbf{w}||=1}{\arg\max} \min_{i \in [m]} y_i(\langle \mathbf{w}, x_i \rangle + b). \tag{35}$$

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

$$(w_0, b_0) = \underset{(\mathbf{w},b)}{\arg\min} ||\mathbf{w}||^2 \ \ \text{s.t. } \forall i, y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1. \tag{36}$$

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.

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

$$\min_{\mathbf{w},b,\xi} \left(\lambda ||\mathbf{w}^2|| + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \ \ \text{s.t. } \forall i, y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0. \tag{37}$$

The hinge loss is defined as $\ell^{hinge}(y, \hat{y}) = \max\{0, 1 - y_i(\langle \mathbf{w}, x_i \rangle + b)\}$, and is denoted as $L_S^{hinge}(y, \hat{y})$. The soft SVM becomes a regularized loss minimization problem:

$$\min_{\mathbf{w},b} \left(\lambda ||\mathbf{w}||^2 + L_S^{hinge}(y, (\mathbf{w}, b)) \right) \tag{38}$$

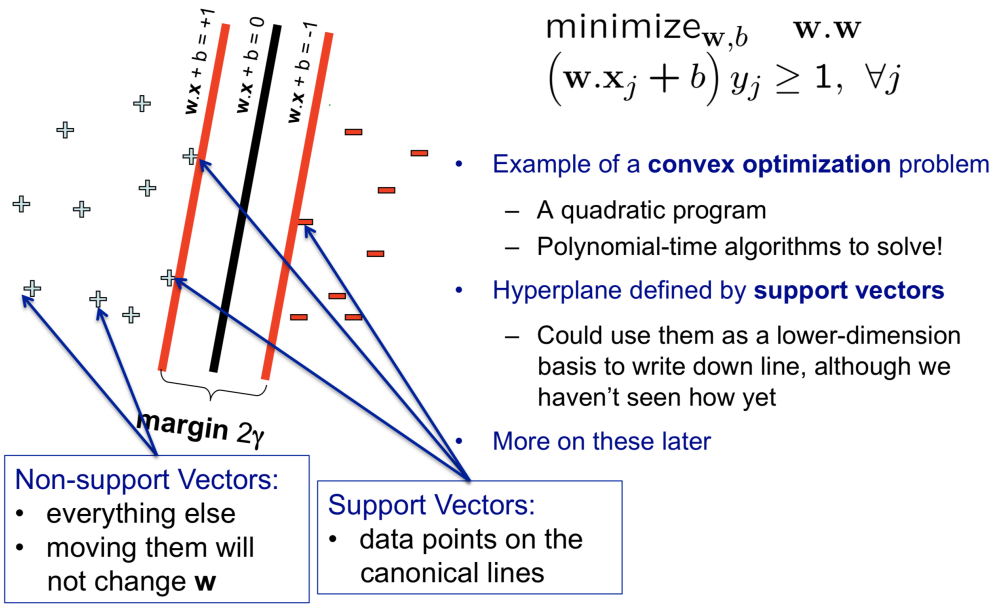

Figure 2: Hard SVM explained.

The solution of hard SVM, $\mathbf{w}_0$ is supported by (i.e. in the span of) the examples that are exactly at distance $\frac{1}{||\mathbf{w}_0||}$ from the separating hyperplane.

(SSBD Theorem 15.8) For hard SVM

$$(w_0, b_0) = \arg\min_{(\mathbf{w},b)} ||\mathbf{w}||^2 \quad \text{s.t. } \forall i, y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1, \tag{39}$$

and let $I = \{i : |\langle \mathbf{w}_0, \mathbf{x} \rangle| + b_0 = 1\}$ there exists coefficients $\alpha_1, ..., \alpha_m$ such that $w_0 = \sum_{i \in I} \alpha_i \mathbf{x}_i$.

2.4.2 Representation Power of Linear Threshold Functions

What are the boolean functions representable by the linear threshold function?

- AND function
- OR function
- NOT function

XOR (PARITY function), however, can not be represented by halfspaces. More generally, if the probability of the positive class does not change monotonically with the value of each input, then linear classifier may not be suitable.

2.4.3 How many functions can a linear threshold function represent?

More generally, how many functions can a hypothesis class $\mathcal{H}$ represent on an arbitrary set of m input points $c_1, ..., c_m$?

- VC dimension
- Growth function
- Shattering
- Sauer's lemma

Refer to slide *Representation I.*

2.5 Linear combination of Functions

The XOR that cannot be represented by a linear classifier can be represented by a thresholded quadratic function, which can represent an ellipse.

$$f(x_1, x_2) = \text{sign}(b + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2). \tag{40}$$

The quadratic function, or more generally polynomial functions, can also be viewed as a linear combination of features. In this case, the features are the monomials $(x_1, x_2, x_1 x_2, x_1^2, x_2^2)$. A feature is just a function of the input, e.g. $g_1(x_1, x_2) = x_1, ..., g_5(x_1, x_2) = x_2^2$ are the features in this example.

Linear combination of functions is very commonly used in machine learning:

- Kernel methods
- Single hidden layer neural networks
- Matrix/tensor factorization
- Ensemble methods

2.6 Kernel Method

Previously, XOR has been showed that it cannot be represented by a linear threshold function but can be represented by a threshold quadratic function:

$$f(x_1, x_2) = \text{sign}(b + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2). \tag{41}$$

We can construct a mapping $\psi : \mathbb{R}^2 \to \mathbb{R}^5$ as

$$\psi(x_1, x_2) = (x1, x2, x_1 x_2, x_1^2, x_2^2). \tag{42}$$

Then we have a linear function in the feature space:

$$f(x_1, x_2) = \text{sign}(b + \mathbf{w}^T \psi(x_1, x_2)). \tag{43}$$

In high dimensions, the number of monomials in a polynomial grows exponentially with degree, so polynomials that contains all monomials cannot be computed efficiently in high dimensions. We use the kernel trick to avoid having to compute the value of each feature.

Kernel refers to a function that computes the inner product between two elements in the feature space. Given an embedding ψ of the domain space $\mathcal{X}$ into some Hilbert space, we define the kernel function $K(x, x') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$. We use kernels that can be efficiently computed so that inner products in high or even infinite dimensional feature spaces can be efficiently computed.

- The d-th Polynomial kernel: $K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^d$.
- Gaussian (RBF) Kernel on $\mathcal{R}$: $K(\mathbf{x}, \mathbf{x}') = e^{-\frac{||x - x'||^2}{2\sigma}}$.

Generally, a function k(x,y) is a valid kernel function (in the sense of the kernel trick) if it satisfies two key properties:

- symmetry: $k(x, y) = k(y, x)$
- positive semi-definiteness.

From the experiment, we know non-linear kernel is helpful for handwritten digit problem.

2.6.1 Hilbert Space

Given a mapping ψ that maps to a finite feature space, we can then compute inner product and distances in the feature space like we do in $\mathbb{R}^d$. But what if the mapping ψ maps the instances to infinite length feature vectors? We would like to extend the notion of $\mathbb{R}^d$ to infinite dimensional vector and even function spaces: Hilbert spaces.

Some properties we need to define Hilbert Space:

- The norm of an element is $||v|| = \sqrt{\langle v, v\rangle}$.
- The distance between two elements is defined in terms of the norm: $d(u, v) = ||w-v|| = \sqrt{\langle u - v, u - v\rangle}$.
- The inner product allows us to define orthogonality: u and v are orthogonal if $\langle u, v\rangle = 0$.
- Cauchy-Schwarz inequality: $|\langle u, v\rangle| \leq ||u||||v||$.
- A Hilbert space $\mathcal{H}$ is complete inner product space, i.e. every Cauchy sequence converges to a point in $\mathcal{H}$.

2.6.2 Representer Theorem

The representer theorem shows how to use inner products for solving many learning tasks.

Representer Theorem. Let ψ be a mapping from $\mathcal{X}$ to a Hilbert space. Then there exists a vector $\alpha \in \mathbb{R}^m$ such that $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(x_i)$ is an optimal solution to

$$\min_{\mathbf{w}} \Big(f(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, ..., \langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle) + R||w|| \Big), \tag{44}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically nondecreasing function. The inner product can be written as

$$\langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle = \left\langle \sum_{j=1}^m \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \right\rangle \tag{45}$$

$$= \sum_{j=1}^m \langle \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i)\rangle \tag{46}$$

$$= \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}_i). \tag{47}$$

Computing the value of a linear function is just taking the inner product between the weight vector and the input features. In the hilbert space, the squared norm is

$$||\mathbf{w}||^2 = \langle \mathbf{w}, \mathbf{w}\rangle = \left\langle \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i), \sum_{j=1}^m \alpha_j \psi(\mathbf{x}_j) \right\rangle \tag{48}$$

$$= \sum_{i,j=1}^m \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j)\rangle = \sum_{i,j=1}^m \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{49}$$

Therefore, instead of optimizing equation (44), we can optimize

$$f\left(\sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}_1), .., \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}_m) \right) + R\left(\sqrt{\sum_{i,j=1}^m \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)} \right). \tag{50}$$

with respect to the α_j's instead of $\mathbf{w}$. (m dimensional instead of very large (or infinite) dimensional). For example, belows are examples of utilizing kernel methods to convert the objective function:

- Soft SVM
$$\sum_{i=1}^{m} \max\left\{0, 1 - y_i\left(\sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)\right\} + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{51}$$

- Regularized logistic regression
$$\sum_{i=1}^{m} \log\left\{1 + y_i\left(\sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)\right\} + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{52}$$

- ridge regression
$$\sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)^2 + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{53}$$

Often the objective function is convex, e.g. in the three cases, allowing efficient learning. We only need to know the value of the $m \times m$ matrix G such that $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, often called the Gram matrix for solving these problems, e.g. soft SVM can be written as

$$\min_{\alpha \in \mathbb{R}^m} \left(\lambda \alpha^T G \alpha + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i (G\alpha)_i\}\right), \tag{54}$$

where $(G\alpha)_i$ is the i-th element of the vector obtainned from nultiplying G with α.

Some representation properties of kernel method:

- With the use of kernel, many techniques for linear functions carry over to non-linear functions, e.g. polynomials.
- Certain kernels give rise to universal approximators. In particular, the Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = e^{-\frac{||x-x'||^2}{2\sigma}}$.
- Useful for moderate sized datasets. As computing kernels for all pairs grows quadratically, need further approximation techniques to scale to very large datasets.

2.6.3 Characterizing Kernel Functions

We have seen how we construct a kernel by constructing a mapping to feature space, then defining the kernel as the inner product between features. Assume we are given a similarity functino of the form $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. When is it a kernel, i.e. represents an inner product between $\psi(\mathbf{x})$ and $\psi(\mathbf{x}')$ for some feature mapping ψ?

Positive semidefinite matrix. A $n \times n$ matrix A is positive semidefinite if $\mathbf{x}^T A \mathbf{x} \geq 0$ for every nonzero vector $\mathbf{x}$.

Gram matrix. Given m data points $\mathbf{x}_1, ..., \mathbf{x}_m$, the $m \times m$ matrix G, where $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ is called *Gram matrix*.

Lemma. A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implements an inner product in some Hilbert space if and only if it is positive semidefinite; i.e. for all $\mathbf{x}_1, ..., \mathbf{x}_m$, the Gram matrix $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ is a positive semidefinite matrix.

The following properties allow valid kernels to be constructed from other valid kernels:

- If k_1, k_2 are kernels, $\alpha_1 k_1 + \alpha_2 k_2$ is a kernel for $\alpha_1, \alpha_2 \geq 0$.
- If k_1, k_2 are kernels, $(k_1, k_2)(x, x') = k_1(x, x') k_2(x, x')$ is also a kernel.
- For any mapping $h : \mathcal{X} \to \mathcal{X}$ and kernel k_1, $k(x, x') = k_1(h(x), h(x'))$ is a kernel.

- If g is a polynomial with positive coefficients and k_1 is a kernel, $k(x, x') = g(k_1(x, x'))$ is a kernel.
- If k_1 is a kernel, $k(x, x') = \exp(k_1(x, x'))$ is a kernel.

2.7 Neural Networks

Instead of fixed features, it is also natural to parameterize the features and learn the parameters. This can be represented as a single hidden layer neural network.

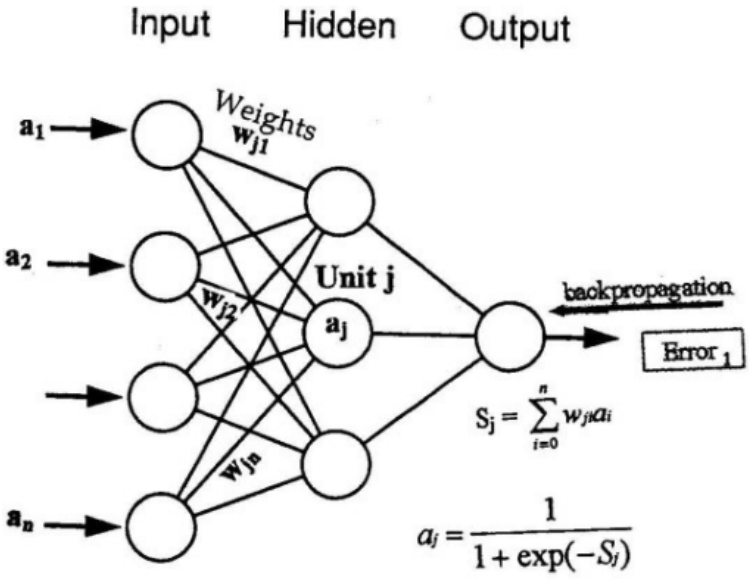

Figure 3: Single hidden layer neural network.

The functions are usually represented in the form $f(\mathbf{x}) = b + \sum_{i=1}^{d} w_i \sigma(\mathbf{v}_i^T \mathbf{x})$, where $\sigma(\mathbf{v}_i^T \mathbf{x})$ is called a hidden unit and $\sigma(\cdot)$ is called an activation function. If the output of the hidden unit depends only on the distance to a center $\phi(||\mathbf{x} - c||)$, where c is a parameter, it is often called a *radial basis function*. One common radial basis function is a Gaussian $\phi(||\mathbf{x} - c||) = \exp(\beta||\mathbf{x} - c||^2)$.

Commonly used functions for hidden units include:

- Linear threshold function: $\sigma(\mathbf{x}, \mathbf{v}) = \text{step}(\mathbf{v}^T \mathbf{x} + \mathbf{v}_0)$.
- Linear functions composed with sigmoid $\sigma(\mathbf{x}, \mathbf{v}) = \frac{1}{1+e^{-(\mathbf{v}^T \mathbf{x} + \mathbf{v}_0)}}$ or tanh function: $\sigma(\mathbf{x}, \mathbf{v}) = \tanh(\mathbf{v}^T \mathbf{x} + \mathbf{v}_0)$.
- Rectifier linear function: $\sigma(\mathbf{x}, \mathbf{v}) = \max(0, \mathbf{v}^T \mathbf{x} + \mathbf{v}_0)$.

2.7.1 Representing All Boolean Functions

For the case of using linear threshold activation and considering networks consisting of only AND, OR and NOT gates. A single hidden layer network is sufficient to represent all functions:

- Disjunctive Normal Form (DNF): an OR gate as output with AND gates as the first hidden layer.
- Conjunctive Normal Form (CNF): an AND gate as output with OR gates as the first hidden layer.

As a linear threshold unit can represent AND and OR gates, a single hidden layer neural network with linear threshold units can represent all Boolean functions as well (with negated inputs added).

2.7.2 Representing PARITY

It is know that constant depth circuits (DNF, CNF) using AND, OR and NOT gates requires size exponential in the input dimension d to represent PARITY. The following theorem show that allowing a more powerful linear threshold unit representation can reduce the size to lienar in d.

Theorem: A single hidden layer neural network with linear threshold hidden and output units can represent the PARITY function using $d + 1$ hidden units.

To represent the parity function, we simply set the second layer weights of $\mathbb{1}_k$ to 1 when k is odd and to -1 when k is even, for $0 \leq k \leq d$. The threshold for the output unit is set to -0.5.

To represent PARITY in d dimension using Gaussian kernels with centers fixed at $\{0, 1\}^d$, $2^d - 1$ non-zero coefficients are required. If the centers of the Gaussians and the bandwidth σ can be optimized, can represent PARITY using $d + 1$ Gaussians.

2.7.3 Approximating Real-valued Functions

A single hidden layer neural network is universal approximator under fairly mild conditions. The sigmoid function can approximate the linear threshold function to arbitrary accuracy by using large enough weights. For more detail, refer to page 52-54, *Representation II*.

2.7.4 Summary of Representational Properties

- Single hidden layer neural networks are universal approximators for many types of hidden units.
- When the hidden units are learnable, exponential gains in size of representation is sometimes achieved, compared to fixed features.

2.7.5 Single Hidden Layer NN In Practice

- Reasonable method when non-linear classifier/regressor required.
- Optimization is non-convex, so may depend on initialization. Harder to tune compared to kernel methods.

For other details, refer to page 57, *Representation II*.

2.8 Matrix/Tensor Factorization

In this section, we switch to unsupervised learning to learn features from unlabeled data. We will look at dimension reduction, which can help supervised learning methods that are sensitive to irrelevant features. Unsupervised learning can also bring in information from large amounts of unlabeled data to help supervised learning when the amount of labeled data is small.

2.8.1 Principal Component Analysis (PCA)

Consider an autoencoder with a function pair: an encoder $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ to compress the information and a decoder $\phi : \mathbb{R}^k \mapsto \mathbb{R}^d$ to reconstruct the data from the compressed representation. In PCA, both $\psi(\mathbf{x}) = V_k^T \mathbf{x}$ and $\phi(\mathbf{u}) = V_k \mathbf{u}$ are linear transformations. The aim is to minimize the reconstruction error:

$$\min_{\psi,\phi} \sum_i ||\mathbf{x}_i - \phi(\psi(\mathbf{x}_i))||^2. \tag{55}$$

Consider the data as a $m \times d$ matrix X where each row represents a data point and each column represents the values of an input dimension. X is firstly normalized (zero mean and unit variance), then be conducted SVD to become $X = U\Sigma V^T$, where the columns of U are the eigenvectors of XX^T and the columns of V are the eigenvectors of $X^T X$. Σ is a diagonal $m \times d$ matrix with each entry σ_i is called the singular values of X in descending order.

- $X_k = U_k \Sigma_k V_k^T$ gives the best rank k approximation of X.
- $XV = U\Sigma$ gives the projection of the data on the principle axes V, called the principle components.
- $F = XV_k$ is a $m \times k$ matrix that gives the first k principle components for each of the data points.
- (Whitening) We normalize each principle component (feature) to have same variance by $U = XV_k\Sigma^{-1}$.

2.8.2 Latent Semantic Analysis (LSA)

Given a term document matrix X where element $x_{i,j}$ describe the occurrence of term (word) i in document j, LSA finds a low-rank approximation of the matrix. Usually we use SVD to get $X = U\Sigma V^T$ without removing mean.

Keep the k largest singular values (zero the rest) to get $X_k = U_k\Sigma_k V_k^T$. X_k is the best rank k approximation for X.

- Column i of $V_k^T = \Sigma_k^{-1} U_k^T X$ is embedding of document i.
- The word embedding is represented in $U_k = XV_k\Sigma_k^{-1}$, row j of U_k represents embedding of word j.

Refer to slide 72-74, *RepresentationII* for more detail.

2.8.3 Representation as Linear Combination of Functions

We can write the factorization as a sum of rank 1 matrices $\mathbf{u}_i\mathbf{v}_i^T$.

$$\hat{X} = U\Sigma_k V^T = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \tag{56}$$

Absorbing the diagonal matrix Σ, it becomes:

$$\hat{X} = UV^T = \sum_{i=1}^{k} \mathbf{u}_i \mathbf{v}_i^T. \tag{57}$$

We can view the entries of $\hat{X}$ as a function of the rows and columns, and similarly the vector entries of $\mathbf{u}_i$ and $\mathbf{v}_i$, hence

$$\hat{X}(a,b) = \sum_{i=1}^{k} \mathbf{u}_i(a)\mathbf{v}_i(b) \tag{58}$$

is a linear combination of functions. The idea can also be extended to q-dimensional tables called tensors:

$$\hat{X}(a_1, ...a_q) = \sum_{i=1}^{k} \mathbf{u}_i^1(a_1)...\mathbf{u}_i^q(a_q). \tag{59}$$

Word2Vec[2] represents the probability $p(w_j|w_i)$ of a word j appearing in a window ($\pm$ c words) around a target word i in text collections using

$$p(w_j|w_i) = \sigma(X(u,j)) = \sigma\left(\sum_{r=1}^{k} \mathbf{u}_r(i)\mathbf{v}_r(j)\right) = \sigma(\mathbf{u}_i^T\mathbf{v}_j), \tag{60}$$

where σ is the logistic function.

2.8.4 Collaborative Filtering

By treating the matrix as a function $\hat{X}(a,b) = \sum_{i=1}^{k} \mathbf{u}_i(a)\mathbf{v}_i(b)$, we can also use matrix factorization for supervised learning. In a collaborative filtering task, we are given a partially filled $m \times d$ matrix X of ratings, where $x_{i,j}$ is the rating of user i on item (e.g. movie) j.

To predict the unknown entries in X, we try to learn a low rank matrix $\hat{X} = UV^T$ where u and V are $m \times k$ and $d \times k$ matrices. The number of parameters in U and V is $(m+d)k$ and is much smaller than md, so we hope to be able to generalize and fill in the missing values in X with good estimates. One common solution is to treat it as a regression problem and do supervised learning of observed ratings.

2.9 Ensemble Methods

Ensemble methods take a weighted average of outputs of learning algorithms. The most famous method is XGBoost [3]. XGBoost is a tree ensemble model (weighted average of trees).

We generalize ensembles into linear combinations with bounded ℓ_1 norm and look at approximation results of the larger class. Consider $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$ where $w_i > 0$ and $\sum_{i=1}^{\infty} w_i = 1$. This describes an ensemble of functions. We can generalize it by allowing w_i to also be negative and $\sum_{i=1}^{\infty} |w_i| = C$ for some constant C. We then call $C = \sum_{i=1}^{\infty} |w_i|$ the ℓ_1 norm of the linear combination.

2.9.1 Approximation with Square Loss

We denote ℓ_2 distance as $||f-g||_2 = \sqrt{\int_K (f(\mathbf{x}) - g(\mathbf{x}))^2 p(\mathbf{x}) d(\mathbf{x})}$. Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$, $C = \sum_{i=1}^{\infty} |w_i|$ and G include all the functions g_i used in f. Assume $||g_i|| \leq 1$.

Theorem. There is a k term linear combination f_k of functions from G such that $||f - f_k||_2^2 \leq \frac{C^2}{k}$.

- The theorem shows the approximation error for $||f - f_k||$ decays at a rate $O(\frac{C}{\sqrt{k}})$.
- To get an approximation error $||f - f_k|| \leq \epsilon$, a relatively small number $k = O(\frac{C}{\sqrt{\epsilon}})$ of functions is required.

2.9.2 Maximum Norm Approximation

We now look at approximation using the maximum norm $||f||_\infty = \max_K |f(\mathbf{x})|$. (also called supremum norm, Chebyshev norm or infinity norm).

Hoeffding's Inequality. Let $Z_1, ..., Z_m$ be a sequence of *i.i.d.* random variables and let $\bar{Z} = \frac{1}{m}\sum_{i=1}^{m} Z_i$. Assume that $E[\bar{Z}] = \mu$ and $Pr[a \leq Z_i \leq b] = 1$ for every i. Then for any $\epsilon > 0$,

$$Pr\left[\left|\frac{1}{m}\sum_{i=1}^{m} Z_i - \mu\right| > \epsilon\right] \leq 2\exp\left(\frac{-2m\epsilon^2}{(b-a)^2}\right). \tag{61}$$

Union bound.

$$Pr[A_1 \cup A_2, ..., A_n] \leq Pr[A-1] + ... + Pr[A_n]. \tag{62}$$

Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$ and $C = \sum_{i=1}^{\infty} |w_i|$. Further assume g_i has been normalized so that $\sup_{x \in K} |g_i(\mathbf{x})| = 1$, and G contains all the functions g_i used in f. sup means least upper bound.

Theorem. There is a k term linear combination f_k of functions in G such that

$$||f - f_k||_\infty \leq \frac{(C+1)\sqrt{\log 2|\mathcal{X}|}}{\sqrt{8k}} = O\left(\frac{C\sqrt{\log|\mathcal{X}|}}{\sqrt{k}}\right). \tag{63}$$

For the proof, refer to slide 13-14, *Representation III*.

2.9.3 Using Ensemble for Classification

Corollary. Consider a function $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$, assume $\max_{x \in K} |g_i(\mathbf{x})| = 1$ and let $\sum_i |w_i| = 1$. Assume f correctly classifies all $x \in \mathcal{X}$ with margin γ. Then it is possible to correctly classify all the instances in $\mathcal{X}$ using a linear combination of k terms from $\{g_i | i = 1, ..., \infty\}$ for $k > k_{min} = \frac{\log(2|\mathcal{X}|)}{2\gamma^2}$.

This corollary suggests that classifiers with large margins are easier to approximate.

2.9.4 Large Margin Classifier and Weak Learning

Definition of Weak Learning. For $\gamma > 0$, an algorithm A is called a γ-weak learning algorithm for a Boolean function F if for any distribution D over $\mathcal{X}$, the algorithm takes a set of examples from distribution D as input and outputs a hypothesis $h \in \mathcal{H}$ with error at most $\frac{1}{2} - \gamma$, i.e. $Pr_D(h(x) \neq f(x)) \leq \frac{1}{2} - \gamma$.

A weak learning algorithm is guaranteed to find a classifier that does slightly better than chance, regardless of the input distribution.

Assume that the componenet functions $\{-1, 1\} - valued$.

Theorem. There is a γ-weak learning algorithm using $\mathcal{H}$ if and only if there is an ensemble of functions from $\mathcal{H}$ with margin 2γ.

2.9.5 Ensembles in Practice

- Ensembles of decision trees are very successful in practive.
  - Boosting methods (decrease bias) explicitly optimize the weights and examples used to learn each component tree.
  - Bagging methods (decreases variance) subsample the examples to get a diverse collection of trees and give equal weighting. It is targeted at variance reduction but may also result in improved approximation.
- Appears to improve performance of trees making it competitive with other methods and yet scale quite well to large data sets.
- Beside trees, can also be used with other methods.
- Combining the output of various methods using ensembles often helps in practice, possibly for both variance reduction and improved approximation.

For detailed comparison, refer to the article on StackOverflow [4].

2.10 Deep Neural Network

2.10.1 VC Dimension of Neural Networks

We bound the number of functions that can be represented by a neural network with linear threshold units that has $|E|$ weights, and obtain a bound on the VC-dimension.

Theorem. The VC-dimension of a neural networks with linear threshold units is $O(|E| \log |E|)$ where $|E|$ is the number of weights in the network.

For the proof, refer to slide 31-34, *Representation III*.

Recall:

- **Definition**: a **dichotomy** of a set S is a partition of S into two disjoint subsets.

- **Definition**: a set of instances S is **shattered** by hypothesis space H if and only of for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.
- **Definition**: The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) = \infty$.

Using the bound on the VC-dimension, we can show that there are Boolean functions that requires an exponential sized neural network with linear threshold processubg units to represent, regardless of the architecture.

Theorem. For every d, let $s(d)$ be the minimal integer such that there is a graph (V, E) with $|V| = s(d)$ such that the hypothesis class H_G with linear threshold processing units contains all functions from $\{0,1\}^d$ to $\{0,1\}$. Then $s(d)$ is exponential in d.

On the other hand, any function that is computable in polynomial time also has a representation that is polynomial sized.

Theorem. Let $T : \mathbb{N} \to \mathbb{N}$ and for every N, let $\mathcal{F}_n$ be the class of functions that can be implemented using a Turing machine using a runtime of at most $T(n)$. Then, there exists constants $b, c \in \mathbb{R}_+$ such that for any n there is graph $G_n = (V_n, E_n)$ of size at most $cT(n)^2 + b$ such that H_{G_n} with linear threshold processing units contains $F + n$.

2.10.2 Deep Vs Shallow

Theorem. Every constant depth circuit with AND, OR, NOT gates for computing the PARITY function requires size exponential in d.

When logarithmic depth is allowed, PARITY can be represented in size polynomial in d. So for logic circuits, there are functions that can be represented compactly if depth is unrestricted but requires exponential size if depth is restricted.

2.10.3 CNN

We have earlier seen that PARITY can be represented using a single hidden layer threshold network with $d + 1$ hidden units – no need depth. So does the same phenomenon affect networks used practically?

- Shallow network requires exponential size to realize the same function as some polynomial sized deep networks.

2.10.4 Deep Learning in Practice

- Often useful when dataset is large
- When the dataset is not very large, methods such as SVM or ensemble of decision trees may work better.
- Often advantageous when there is structure in the data that we want to exploit. Can design architectures to take advantage of structure in the data.

2.11 Embedding Knowledge/Algorithms

Polynomial time algorithm can be encoded in a polynomial sized network: can be helpful in encoding prior knowledge.

Design heuristic:

- Design network architectures that encode knowledge about problem domain

- Can then learn the parameters for the network discriminatively (aka end-to-end learning) to get a robust function that fits the data well.
- Exploit both knowledge of problem as well as learning from data.

2.11.1 Properties of Convolutional Networks

- Feature detector (based on convolution) that detects the same feature in different locations.
- Same kernel at different locations utilize parameter tying – reduce data required for learning. Local kernels for efficiency.
- Approximate invariance to small translations and other transformations through pooling.
- CNN is known to be depth efficient.

2.11.2 RNN

Refer to slide 64, *Representation III.*

2.11.3 Value Iteration Network

Value iteration network encodes the value iteration process for Markov decision processes.

- Dynamic programming.
- Essentially a generalization of Bellman-Ford shortest path algorithm. However, in MDP the transition from one state to another is noisy, unlike in shortest path problems.

Definition. A Markov decision process (MDP) is described by a tuple $\langle S, A, T, R\rangle$ where S denotes the state space, A denotes the action space, T denotes the transition function $T : S \times A \times S \to \mathbb{R}$ and R denotes the reward function $R : S \times A \to \mathbb{R}$.

A policy $\pi(s)$ prescribes the action to take in state s. The aim of the policy is to maximize the expected sum of (discounted) reward $V^\pi(s) = E^\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|s_0 = s]$ where γ is the discount factor.

- A discount factor $\gamma < 1$ allows the sum to be finite when we sum to ∞.
- We may also be interested in the finite horizon case where we sum from $t = 0$ to K and set $\gamma = 1$. e.g. in the case of deterministic shortest path, this would give the shortest path to reach the goal using K edges.

Value iteration is a DP algorithm for computing the value/policy.

$$V_{t+1}(s) = \max_a Q_t(s, a), \text{ where} \tag{64}$$

$$Q_t(s, a) = \begin{cases} R(s, a) & \text{if } t = 0 \\ R(s, a) + \gamma \sum_{s'} T(s', a, s) V_t(s') & \text{otherwise.} \end{cases} \tag{65}$$

For application of value iteration for Bellman-Ford algorithm in graph, refer to slide 79-81, *Representation III.*

3 Estimation

In this part, we will study how much data is required to learn.

3.1 Finite Class

Theorem. Let $\mathbb{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$ and $\epsilon > 0$, and let m be an integer that satisfies

$$m \geq \frac{log(|\mathcal{H}|/\delta)}{\epsilon}. \tag{66}$$

Then for any labeling function f, and for any distribution $\mathcal{D}$ for which the realizablity assumption holds, with probability at least $1-\delta$ over the choice of an i.i.d. sample $\mathcal{S}$ of size m, we have that for every empirical risk minimization hypothesis $h_{\mathcal{S}}$, it holds that

$$L_{(D,f)}(h_{\mathcal{S}}) \leq \epsilon. \tag{67}$$

3.2 PAC Learning

We cannot require the model to make perfect prediction every time. Instead, we hope the event that deviation is less than ϵ happens with high probability. In other words, machine learning can produce a good model with high probability.

Definition (PAC Learnability). A hypothesis class $\mathcal{H}$ is Probably Approximately Correct (PAC) learnable if there exists a function $m_{\mathcal{H}}(\epsilon,\delta)$ and a learning algorithm with the following property: For every $\epsilon,\delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function f, if the realizability assumption holds with respect to $\mathcal{H},\mathcal{D},f$ then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon,\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability at least $1-\delta$ (over the choice of examples), $L_{(\mathcal{D},f)}(h) \leq \epsilon$.

- The accuracy parameter ϵ determines how far the classifier is allowed to be from optimal (approximately correct).
- The confidence parameter δ indicates how likely the classifier is to meet the accuracy requirement (probably part of PAC).

3.2.1 sample Complexity

The function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{R}$ determines the sample complexity of learning $\mathcal{H}$. Sample complexity is the minimum size sample needed to obtain the required results.

Corollary: Every finite hypothesis class is PAC learnable with sample complexity

$$m_{\mathcal{H}}(\epsilon,\delta) \leq \left\lceil \frac{log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil. \tag{68}$$

We can relax the definition of PAC learning in the following ways:

- Realizability assumption
- Agnostic learning
- Loss function

Definition (Agnostic PAC Learnability for General Loss Functions). A hypothesis class $\mathcal{H}$ is agnostic PAC learnable with respect to a set $\mathbb{Z}$ and a loss function $\ell : \mathcal{H}times\mathcal{Z} \to \mathbb{R}_+$, if there exists a function $m_{\mathcal{H}}(\epsilon,\delta)$ and a learning algorithm with the following property: For every $\epsilon,\delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{Z}$, when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon,\delta)$ i.i.d. examples generated by

$\mathcal{D}$, the algorithm returns $h \in \mathcal{H}$ such that, with probability at least $1-\delta$ (over the choice of m training examples),

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon, \tag{69}$$

where $L_{\mathcal{D}}(h) = E_{z \sim \mathbb{D}}[\ell(h, z)]$.

3.2.2 Error Decomposition

Let $h_{\mathcal{S}}$ be a $ERM_{\mathcal{H}}$ hypothesis. By using agnostic learning, we can decompose the error into two components:

$$L_{\mathcal{D}}(h_{\mathcal{S}}) \leq \epsilon_{app} + \epsilon_{est}. \tag{70}$$

- The approximation error $\epsilon_{app} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ is the minimum risk achievable by hypotheses in the class.
- The estimation error $\epsilon_{est} \geq L_{\mathcal{D}}(h_{\mathcal{S}}) - \epsilon_{app}$ is an upper bound on the difference between the error achieved by the $ERM_{\mathcal{H}}$ predictor and the minimum risk achievable by hypotheses in the class.

3.3 Uniform Convergence

Uniform convergence is a general tool for showing learnability, including agnostic learning.

Definition (ϵ-representative sample). A training sample $\mathcal{S}$ is called ϵ-representative (w.r.t. domain $\mathcal{Z}$, hypothesis class $\mathcal{H}$, loss function ℓ and distribution $\mathcal{D}$) if

$$\forall h \in H, |L_S(h) - L_D(h)| \leq \epsilon. \tag{71}$$

Definition (Uniform Convergence): We say that a hypothesis class $\mathcal{H}$ has the uniform convergence property (w.r.t. domain $\mathcal{Z}$ and loss function ℓ) if there exists a function $m_{\mathcal{H}}^{UC} : (0,1)^2 \to \mathbb{N}$ such that for every $\epsilon, \delta \in (0,1)$ and every probability distribution $\mathcal{D}$ over $\mathcal{Z}$, if $\mathcal{S}$ is a sample of $m \geq m_{\mathcal{H}}^{UC}(\epsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, then with probability at least $1-\delta$, $\mathcal{S}$ is ϵ-representative.

In other words, We bound the difference between the training error and generalization error for all functions in $\mathcal{H}$.

Lemma. Assume that training set S is $\epsilon/2$-representative (w.r.t. domain $\mathcal{Z}$, hypothesis class $\mathcal{H}$, loss function ℓ and distribution $\mathcal{D}$). Then any output of $ERM_H(S)$ (empirical risk minimizer), namely, any $h_s \in \arg\min_{h \in H} L_S(h)$ satisfies

$$L_D(h_S) \leq \min_{h \in H} L_D(h) + \epsilon. \tag{72}$$

3.4 No Free Lunch

Theorem (No-Free-Lunch). Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain $\mathcal{X}$. Let m be any number smaller than $\frac{|\mathcal{X}|}{2}$, representing a training set size. Then there exists a distribution D over $\mathcal{X} \times \{0,1\}$ such that:

- There exists a function $f : \mathcal{X} \to \{0,1\}$ with $L_D(f) = 0$.
- With probability of at least $\frac{1}{7}$ over the choise of $\mathcal{S} \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq \frac{1}{8}$.

Implication: For every learner, there exists a task on which it fails.

3.5 Fundamental Theorem

Theorem. (Fundamental Theorem) Let $\mathcal{H}$ be hypothesis class of functions from a domain $\mathcal{X}$ to $\{0, 1\}$ and let the loss function be the 0-1 loss. Then the following are equivalent:

- $\mathcal{H}$ has the uniform convergence property.
- Any ERM rule is a successful agnostic PAC learner for $\mathcal{H}$.
- $\mathcal{H}$ is agnostic PAC learnable.
- $\mathcal{H}$ is PAC learnable.
- Any ERM rule is a successful PAC learner for $\mathcal{H}$.
- $\mathcal{H}$ has finite VC-dimension.

3.5.1 Rademacher Complexity

The bounds we derived based on VC dimension were distribution independent. In some sense, distribution independence is a nice property because it guarantees the bounds to hold for any data distribution. On the other hand, the bounds may not be tight for some specific distributions that are more benign than the worst case. Furthermore, the concepts used in defining VC dimension apply only to binary classification, but we are often interested in generalization bounds for multiclass classification and regression as well. *Rademacher complexity* is a more modern notion of complexity that is distribution dependent and defined for any class real-valued functions (not only discrete-valued functions).

To simplify notation, we will compose our hypothesis class with the loss function. Denote

$$\mathcal{F} = \ell \; o \; \mathcal{H} = \{z \to \ell(h, z) : h \in \mathcal{H}\}. \tag{73}$$

We will use the empirical and expected loss of $f \; in \mathcal{F}$:

$$L_D(f) = E_{z\sim\mathcal{D}}[f(z)], \quad L_S(f) = \frac{1}{m}\sum_{i=1}^{m} f(z_i). \tag{74}$$

Recall that we used the notion of representativeness when we studied uniform convergence: we have uniform convergence if the samples are ϵ-representative with high probability for all distributions.

For this section, it suffices to look at one-sided representativeness of S with respect of $\mathcal{F}$ as

$$\text{Rep}_{\mathcal{D}}(\mathcal{F}, \mathcal{S}) = \sup_{f\in\mathcal{F}}(L_D(f) - L_S(f)). \tag{75}$$

This means, if S has good representativeness (small), then functions with small empirical risk will also have small expected risk. However, we do not know $\mathcal{D}$ and would like to estimate or bound the representativeness error from data. Given S, one possible way to estimate its representativeness is by randomly splitting it into disjoint sets S_1 and S_2 and measuring

$$\frac{1}{m}\sup_{f\in\mathcal{F}}(m_1 L_{S_1}(f) - m_2 L_{S_2}(f)), \tag{76}$$

where m_1 and m_2 are the sizes of S_1 and S_2. Rademacher complexity averages the estimates over the random splits generated from M coin tosses: heads goes into S_1 and tail into S_2.

Let $\mathcal{F} \; o \; \mathcal{S}$ be the set of all possible evaluation of function $f \in \mathcal{F}$ on $\mathcal{S}$:

$$\mathcal{F} \; o \; \mathcal{S} = \{(f(z_1), ..., f(z_m)) : f \in \mathcal{F}\}. \tag{77}$$

The Rademacher complexity of $\mathcal{F}$ with respect to $\mathcal{S}$ is:

$$R(F\ o\ S) = E_{\sigma \sim \{\pm 1\}^m}\left[\sup_{f \in \mathcal{F}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i f(z_i)\right], \tag{78}$$

where σ_i are i.i.d. sampled with $P(\sigma_i = 1) = P(\sigma_i = -1) = -1/2$. We can also define the Rademacher complexity of a set $A \in \mathbb{R}^m$ as

$$R(A) = E_{\sigma}\left[\sup_{a \in A} \frac{1}{m}\sum_{i=1}^{m} \sigma_i a_i\right]. \tag{79}$$

It turns out that, on average, the Rademacher complexity can be use to upper bound the representativeness value. So small Rademacher complexity implies small representativeness value.

Lemma:

$$E_{\mathcal{S} \sim \mathcal{D}^m}[\mathrm{Rep}_{\mathcal{D}}(\mathcal{F}, \mathcal{S})] \leq 2E_{\mathcal{S} \sim \mathcal{D}^m} R(F\ o\ S). \tag{80}$$

References

[1] ebony1 (https://stats.stackexchange.com/users/881/ebony1), "Generative vs discriminative models (in bayesian context)." Cross Validated. URL:https://stats.stackexchange.com/q/4690 (version: 2017-06-26).

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[3] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016.

[4] A. G. (https://stats.stackexchange.com/users/7647/alexander galkin), "Bagging, boosting and stacking in machine learning." Cross Validated. URL:https://stats.stackexchange.com/q/19053 (version: 2017-10-26).